Zelig: Everyone's Statistical Software[1]

Kosuke Imai[2] Gary King[3] Olivia Lau[4]

Version 3.3-0
June 13, 2008

[1]The current version of this software is available at `http://gking.harvard.edu/zelig/`, free of charge and open-source (under the terms of the GNU GPL, v. 2).

[2]Assistant Professor, Department of Politics, Princeton University (Corwin Hall, Department of Politics, Princeton University, Princeton NJ 08544; `http://Imai.Princeton.Edu`, `KImai@Princeton.Edu`).

[3]David Florence Professor of Government, Harvard University (Institute for Quantitative Social Sciences, 1737 Cambridge Street, Harvard University, Cambridge MA 02138; `http://GKing.Harvard.Edu`, `King@Harvard.Edu`, (617) 495-2027).

[4]Ph.D. Candidate, Department of Government, Harvard University (1737 Cambridge Street, Cambridge MA 02138; `http://www.people.fas.harvard.edu/~olau`, `OLau@Fas.Harvard.Edu`).

Contents

III Reference Manual 88

12.26 `logit.gee`: Generalized Estimating Equation for Logistic Regression . . . 276

12.28 `logit.mixed`: Mixed effects logistic Regression . . . 290

Acknowledgments

The Zelig project would not have been possible without considerable help from many sources. Our special thanks go to the R core team for providing an excellent modular, open-source platform for the entire statistics and methodological community.

The authors of the following R packages have provided some of the models available through Zelig: `MASS` by William N. Venables and Brian D. Ripley; `MCMCpack` by Andrew D. Martin and Kevin M. Quinn; `survival` by Terry Therneau and Brian D. Ripley; and `VGAM` by Thomas Yee.

The authors of the following R packages have provided some of the auxiliary statistical procedures and methods available through Zelig: `boot` by Angelo Canty and Brian Ripley; `coda` by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines; `sandwich` by Achim Zeileis; and `zoo` by Achim Zeileis and Gabor Grothendieck.

Our appreciation also goes to Ferdinand Alimadhi, Skyler J. Cranmer, Ben Goodrich, Justin Grimmer, Yang Lin, and Ying Lu, who have contributed either code or documentation; Dan Hopkins, Ian Yohai, and others who have given valuable feedback from the courses in which they have used Zelig as a teaching tool.

For research support, we thank the National Institutes of Aging (P01 AG17625-01), the National Science Foundation (SES-0318275, SES-0550873, IIS-9874747, SES-0112072), the Mexican Ministry of Health, the U.S. Library of Congress (PA# NDP03-1), and the Princeton University Committee on Research in the Humanities and Social Sciences.

As usual, all errors are our responsibility.

Chapter 1

Introduction

1.1 What Zelig and R Do

Zelig[1] is an easy-to-use program that can estimate and help interpret the results of an enormous and growing range of statistical models. It literally *is* "everyone's statistical software" because Zelig's unified framework incorporates everyone else's (R) code. We also hope it will *become* "everyone's statistical software" for applications, and we have designed Zelig so that anyone can use it or add their models to it.

When you are using Zelig, you are also using R, a powerful statistical software language. You do not need to learn R separately, however, since this manual introduces you to R through Zelig, which simplifies R and reduces the amount of programming knowledge you need to get started. Because so many individuals contribute different packages to R (each with their own syntax and documentation), estimating a statistical model can be a frustrating experience. Users need to know which package contains the model, find the modeling command within the package, and refer to the manual page for the model-specific arguments. In contrast, Zelig users can skip these start-up costs and move directly to data analyses. Using Zelig's unified command syntax, you gain the convenience of a packaged program, without losing any of the power of R's underlying statistical procedures.

In addition to generalizing R packages and making existing methods easier to use, Zelig includes infrastructure that can improve all existing methods and R programs. Even if you know R, using Zelig greatly simplifies your work. It mimics the popular Clarify program for Stata (and thus the suggestions of King, Tomz, and Wittenberg, 2000) by translating the raw output of existing statistical procedures into quantities that are of direct interest to researchers. Instead of trying to interpret coefficients parameterized for modeling convenience, Zelig makes it easy to compute quantities of real interest: probabilities, predicted values, expected values, first differences, and risk ratios, along with confidence intervals, standard errors, or full posterior (or sampling) densities for all quantities. Zelig extends Clarify by

[1]Zelig is named after a Woody Allen movie about a man who had the strange ability to become the physical reflection of anyone he met — Scottish, African-American, Indian, Chinese, thin, obese, medical doctor, Hassidic rabbi, anything — and thus to fit well in any situation.

seamlessly integrating an option for bootstrapping into the simulation of quantities of interest. It also integrates a full suite of nonparametric matching methods as a preprocessing step to improve the performance of any parametric model for causal inference (see MatchIt). For missing data, Zelig accepts multiply imputed datasets created by Amelia (see King, Honaker, Joseph, and Scheve, 2001) and other programs, allowing users to analyze them as if they were a single, fully observed dataset. Zelig outputs replication data sets so that you (and if you wish, anyone else) will always be able to replicate the results of your analyses (see King, 1995). Several powerful Zelig commands also make running multiple analyses and recoding variables simple.

Using R in combination with Zelig has several advantages over commercial statistical software. R and Zelig are part of the open source movement, which is roughly based on the principles of science. That is, anyone who adds functionality to open source software or wishes to redistribute it (legally) must provide the software accompanied by its source free of charge.[2] If you find a bug in open source software and post a note to the appropriate mailing list, a solution you can use will likely be posted quickly by one of the thousands of people using the program all over the world. Since you can see the source code, you might even be able to fix it yourself. In contrast, if something goes wrong with commercial software, you have to wait for the programmers at the company to fix it (and speaking with them is probably out of the question), and wait for a new version to be released.

We find that Zelig makes students and colleagues more amenable to using R, since the startup costs are lower, and since the manual and software are relatively self-contained. This manual even includes an appendix devoted to the basics of advanced R programming, although you will not need it to run most procedures in Zelig. A large and growing fraction of the world's quantitative methodologists and statisticians are moving to R, and the base of programs available for R is quickly surpassing all alternatives. In addition to built-in functions, R is a complete programming language, which allows you to design new functions to suit your needs. R has the dual advantage that you do not need to understand how to program to use it, but if it turns out that you want to do something more complicated, you do not need to learn another program. In addition, methodologists all over the world add new functions all the time, so if the function you need wasn't there yesterday, it may be available today.

1.2 Getting Help

You may find documentation for Zelig on-line (and hence must be on-line to access it). If you are unable to connect to the Internet, we recommend that you print the pdf version of this document for your reference.

If you are on-line, you may access comprehensive help files for Zelig commands and for each of the models. For example, load the Zelig library and then type at the R prompt:

```
> help.zelig(command)                 # For help with all zelig commands.
```

[2] As specified in the GNU General License v. 2 http://www.gnu.org/copyleft.

```
> help.zelig(logit)                  # For help with the logit model.
```

In addition, `help.zelig()` searches the manual pages for R in addition to the Zelig specific pages. On certain rare occasions, the name of the help topic in Zelig and in R are identical. In these cases, `help.zelig()` will return the Zelig help page by default. If you wish to access the R help page, you should use `help(topic)`.

In addition, built-in examples with sample data and plots are available for each model. For example, type `demo(logit)` to view the demo for the logit model. Commented code for each model is available under the examples section of each model reference page.

Please direct inquiries and problems about Zelig to our listserv at zelig@lists.gking.harvard.edu. We suggest you subscribe to this mailing list while learning and using Zelig: go to `http://lists.hmdc.harvard.edu/index.cgi?info=zelig`. (You can choose to receive email in digest form, so that you will never receive more than one message per day.) You can also browse or search our archive of previous messages before posting your query.

1.3 How to Cite Zelig

Please cite Zelig with reference to these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.
>
> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

To refer to a particular Zelig model, please refer to the "how to cite" portion at the end of each model documentation section.

Part I

User's Guide

Chapter 2

Installation

To use Zelig, you must install the statistical program R (if it is not already installed), the Zelig package, and some R libraries (coda, MCMCpack, sandwich, VGAM, and zoo).

Note: In this document, `>` denotes the R prompt.

If You Know R

We recommend that you launch R and type

```
> source("http://gking.harvard.edu/zelig/install.R")
> library(Zelig)
```

then proceed to Section 4.1.1. For Windows R, you may edit the `Rprofile` file to load Zelig automatically at launch (after which you will no longer need to type `library(Zelig)` at startup). Simply add the line:

```
options(defaultPackages = c(getOption("defaultPackages"), "Zelig"))
```

If You Are New to R

If you are new to R, we recommend that you read the following section on installation procedures as well as the overview of R syntax and usage in Section 6.

This distribution works on a variety of platforms, including Windows (see Section 2.1), MacOSX (see Section 2.2), and Linux (see Section 2.3). Alternatively, you may access R from your PC using a terminal window or an X-windows tunnel to a Linux or Unix server (see Section 2.3). Most servers have R installed; if not, contact your network administrator.

There are advantages and disadvantages to each type of installation. On a personal computer, R is easier to install and launch. Using R remotely on a server requires a bit more set-up, but does not tie up your local CPU, and allows you to take advantage of the server's speed.

2.1 Windows

Installing R

Go to the Comprehensive R Archive Network website (`http://www.r-project.org`) and download the latest installer for Windows at `http://cran.us.r-project.org/bin/windows/base/` Double-click the `.exe` file to launch the R installer. We recommend that you accept the default installation options if this your first installation.

Installing Zelig

Once R is installed, you must install the Zelig and VGAM packages. There are three ways to do this.

1. We recommend that you start R and then type:

   ```
   > source("http://gking.harvard.edu/zelig/install.R")
   > library(Zelig)
   ```

2. Alternatively, you may install each component package individually in R:

   ```
   > install.packages("Zelig")
   > install.packages("zoo")
   > install.packages("sandwich")
   > install.packages("MCMCpack")
   > install.packages("coda")
   > install.packages("lattice")
   > install.packages("mvtnorm")
   > install.packages("VGAM")
   > install.packages("sna")
   > install.packages("systemfit")
   > install.packages("nnet")
   > install.packages("gee")
   > install.packages("mgcv")
   > library(Zelig)
   ```

   Zelig will load the optional libraries whenever their functions are needed; it is not necessary to load any package other than Zelig at startup.

3. Alternatively, you may use the drop down menus to install Zelig. This requires four steps.

   (a) Go to the Zelig website and and download the latest release of Zelig. The VGAM, MCMCpack, coda, zoo, and sandwich packages are available from CRAN. Store these `.zip` files in your R program directory. For example, the default R program directory is `C:\Program Files\R\R-2.5.1\`.[1]

(b) Start R. From the drop-down menus, select the "Packages" menu and then the "Install Files from Local Zip Files" option.

(c) A window will pop up, allowing you to select one of the downloaded files for installation. There is no need to unzip the files prior to installation. Repeat and select the other downloaded file for installation.

(d) At the R prompt, type `library(Zelig)` to load the functionality described in this manual. Note that Zelig will automatically load the other libraries as necessary.

4. An additional *recommended but optional step* is to set up R to load Zelig automatically at launch. (If you skip this step, you must type `library(Zelig)` at the beginning of every R session.) To automate this process, edit the `Rprofile` file located in the R program subdirectory (`C:\Program Files\R\R-2.5.1\etc\` in our example). Using a text editor such as Windows notepad, add the following line to the `Rprofile` file:

```
options(defaultPackages = c(getOption("defaultPackages"), "Zelig"))
```

Zelig is distributed under the GNU General Public License, Version 2. After installation, the source code is located in your R library directory, which is by default `C:\Program Files\R\R-2.5.1\library\Zelig\`.

Updating Zelig

There are two ways to update Zelig.

1. We recommend that you periodically update Zelig at the R prompt by typing:

```
> update.packages()
> library(Zelig)
```

2. Alternatively, you may use the procedure outlined in Section 3a to periodically update Zelig. Simply download the latest `.zip` file and follow the four steps.

2.2 MacOS X

Installing R

If you are using MacOS X, you may install the latest version of R (2.5.1 at this time) from the CRAN website http://cran.us.r-project.org/bin/macosx/. At this time, Zelig is not supported for R on MacOS 8.6 through 9.x.

[1]Note that when updating R to the latest release, the installer does not delete previous versions from your `C:\Program Files\R\` directory. In this example, the subdirectory `\R-2.5.1\` stores R version 2.5.1. Thus, if you have a different version of R installed, you should change the last part of the R program directory file path accordingly.

Installing Zelig

Once R is installed, you must install the Zelig and VGAM packages. There are several ways to do this.

1. **For RAqua**:

   (a) We recommend that you start R, and then type:

   ```
   > source("http://gking.harvard.edu/zelig/install.R")
   > library(Zelig)
   ```

   (You may ignore the warning messages, unless they say "Non-zero exit status".)

   (b) Alternatively, to avoid the warning messages, you need to install each package individually and specify the specific installation path:

   ```
   > install.packages("Zelig", lib = "~/Library/R/library")
   > install.packages("zoo", lib = "~/Library/R/library")
   > install.packages("sandwich", lib = "~/Library/R/library")
   > install.packages("MCMCpack", lib = "~/Library/R/library")
   > install.packages("coda", lib = "~/Library/R/library")
   > install.packages("lattice", lib = "~/Library/R/library")
   > install.packages("mvtnorm", lib = "~/Library/R/library")
   > install.packages("VGAM", lib = "~/Library/R/library")
   > install.packages("sna", lib = "~/Library/R/library")
   > install.packages("systemfit", lib = "~/Library/R/library")
   > install.packages("nnet", lib = "~/Library/R/library")
   > install.packages("gee", lib = "~/Library/R/library")
   > install.packages("mgcv", lib = "~/Library/R/library")
   > library(Zelig)
   ```

   where `~/Library/R/library` is the default local library directory. Zelig will load the other libraries whenever their functions are needed; it is not necessary to load these packages at startup.

   (c) Alternatively, you may use the drop down menus to install Zelig. This requires three steps.

      i. Go to the Zelig website and download the latest release of Zelig. The VGAM, MCMCpack, coda, zoo, and sandwich packages are available from CRAN. Save these `.tar.gz` files in a convenient place.

      ii. Start R. From the drop-down menus, select the "Packages" menu and then the "Install Files from Local Files" option.

      iii. A window will pop up, allowing you to select the one of the downloaded files for installation. There is no need to unzip the files prior to installation. Repeat and select the other downloaded file for installation.

2. **For command line R:**

   (a) Before installing command line R, you need to create a local R library directory. If you have done so already, you may skip to the next step. Otherwise, at the terminal prompt in your home directory, type:

   ```
   % mkdir ~/Library/R ~/Library/R/library
   ```

   (b) Modify your configuration file to identify `~/Library/R/library` as your R library directory. There are two ways of doing this:

      i. Open the `.Renviron` file (or create one, if you don't have one) and add the following line:

      ```
      R_LIBS = "~/Library/R/library"
      ```

      ii. *Alternatively*, you may modify your shell configuration file. For a Bash shell, open your `.bashrc` file and add the following line:

      ```
      export R_LIBS="$HOME/Library/R/library"
      ```

   (c) Start R and at the prompt, type:

   ```
   > source("http://gking.harvard.edu/zelig/install.R")
   > library(Zelig)
   ```

   (You may ignore the warning messages, unless they say "Non-zero exit status".)

   (d) Alternatively, to avoid the warning messages, you need to install each component package separately and specify the installation path:

   ```
   > install.packages("Zelig", lib = "~/Library/R/library")
   > install.packages("zoo", lib = "~/Library/R/library")
   > install.packages("sandwich", lib = "~/Library/R/library")
   > install.packages("MCMCpack", lib = "~/Library/R/library")
   > install.packages("coda", lib = "~/Library/R/library")
   > install.packages("lattice", lib = "~/Library/R/library")
   > install.packages("mvtnorm", lib = "~/Library/R/library")
   > install.packages("VGAM", lib = "~/Library/R/library")
   > install.packages("sna", lib = "~/Library/R/library")
   > install.packages("systemfit", lib = "~/Library/R/library")
   > install.packages("nnet", lib = "~/Library/R/library")
   > install.packages("gee", lib = "~/Library/R/library")
   > install.packages("mgcv", lib = "~/Library/R/library")
   > library(Zelig)
   ```

   Although the `lib` argument is optional, we recommend that you set it to the default RAqua directory (`"~/Library/R/library"`), in case you later decide to install the RAqua GUI (which has a different default directory).

At the R prompt, type `library(Zelig)` to load the functionality described in this manual. Note that Zelig will automatically load the other packages as necessary.

Zelig is distributed under the GNU General Public License, Version 2. After installation, the source code is located in your R library directory, `~/Library/R/library/Zelig/`.

Updating Zelig

There are two ways to update Zelig.

1. We recommend that you start R and, at the R prompt, type:

   ```
   > update.packages()
   ```

2. Alternatively, you may remove an old version by command by typing `R CMD REMOVE Zelig` at the terminal prompt. Then download and reinstall the package using the installation procedures Section 2.2 outlined above.

2.3 UNIX and Linux

Installing R

Type `R` at the terminal prompt (which we denote as `%` in this section) to see if R is available. (Typing `q()` will enable you to quit.) If it is installed, proceed to the next section. If it is not installed and you are not the administrator, contact that individual, kindly request that they install R on the server, and continue to the next section. If you have administrator privileges, you may download the latest release at the CRAN website. Although installation varies according to your Linux distribution, we provide an example for Red Hat Linux 9.0 as a guide:

1. Log in as root.

2. Download the appropriate binary file for Red Hat 9 from CRAN. For example, for Red Hat 9 running on the Intel 386 platform, go to `http://cran.r-project.org/bin/linux/`.

3. Type the following command at the terminal prompt:
   `% rpm -ivh R-2.5.1-1.i386.rpm`

Installing Zelig

Before installing Zelig, you need to create a local R library directory. If you have done so already, you can skip to Section 2.3. If not, you must do so before proceeding because most users do not have authorization to install programs globally. Suppose we want the directory to be `~/.R/library`. At the terminal prompt in your home directory, type:

```
% mkdir ~/.R ~/.R/library
```

Now you are ready to install Zelig. There are two ways to proceed.

1. Recommended procedure:

   (a) Open the `~/.Renviron` file (or create it if it does not exist) and add the following line:

   ```
   R_LIBS = "~/.R/library"
   ```

   You only need to perform this step once.

   (b) Start R. At the R prompt, type:

   ```
   > source("http://gking.harvard.edu/zelig/install.R")
   > library(Zelig)
   ```

   (You may ignore the warning messages, unless they say "Non-zero exit status".)

   (c) Alternatively, you can avoid the warning messages by installing each component package separately and specifying the installation path:

   ```
   > install.packages("Zelig", lib = "~/Library/R/library")
   > install.packages("zoo", lib = "~/Library/R/library")
   > install.packages("sandwich", lib = "~/Library/R/library")
   > install.packages("MCMCpack", lib = "~/Library/R/library")
   > install.packages("coda", lib = "~/Library/R/library")
   > install.packages("lattice", lib = "~/Library/R/library")
   > install.packages("mvtnorm", lib = "~/Library/R/library")
   > install.packages("VGAM", lib = "~/Library/R/library")
   > install.packages("sna", lib = "~/Library/R/library")
   > install.packages("systemfit", lib = "~/Library/R/library")
   > install.packages("nnet", lib = "~/Library/R/library")
   > install.packages("gee", lib = "~/Library/R/library")
   > install.packages("mgcv", lib = "~/Library/R/library")
   > library(Zelig)
   ```

   (d) Finally, create a `.Rprofile` file in your home directory, containing the line:

   ```
   library(Zelig)
   ```

   This will load Zelig every time you start R.

2. Alternatively:

   (a) Add the local R library directory that you created above (`~/.R/library` in the example) to the environmental variable `R_LIBS`.

   (b) Download the latest bundles for Unix from the Zelig website, and (for the VGAM, MCMCpack, coda, sandwich, and zoo packages) from the CRAN website.

(c) If XX is the current version number, at the terminal prompt, type:

```
% R CMD INSTALL Zelig_XX.tar.gz
% R CMD INSTALL zoo_XX.tar.gz
% R CMD INSTALL sandwich_XX.tar.gz
% R CMD INSTALL MCMCpack_XX.tar.gz
% R CMD INSTALL coda_XX.tar.gz
% R CMD INSTALL lattice_XX.tar.gz
% R CMD INSTALL mvtnorm_XX.tar.gz
% R CMD INSTALL VGAM_XX.tar.gz
% R CMD INSTALL sna_XX.tar.gz
% R CMD INSTALL systemfit_XX.tar.gz
% R CMD INSTALL nnet_XX.tar.gz
% R CMD INSTALL gee_XX.tar.gz
% R CMD INSTALL mgcv_XX.tar.gz

% rm Zelig_XX.tar.gz zoo_XX.tar.gz sandwich_XX.tar.gz MCMCpack_XX.tar.gz coda_X
```

(d) Create a .Rprofile file in your home directory, containing the line:

```
library(Zelig)
```

This will load Zelig every time you start R.

Zelig is distributed under the GNU General Public License, Version 2. After installation, the source code is located in your R library directory. If you followed the example above, this is /.R/library/Zelig/.

Updating Zelig

There are two ways to update Zelig.

1. We recommend that you start R and, at the R prompt, type:

```
> update.packages()
```

2. Alternatively, you may remove an old version by command by typing R CMD REMOVE Zelig at the terminal prompt. Then download and reinstall the package using the installation procedure Section 2.3 outlined above.

2.4 Version Compatability

In addition to R itself, Zelig also depends on several R packages maintained by other development teams. Although we make every effort to keep the latest version of Zelig up-to-date with the latest version of those packages, there may occasionally be incompatabilities. See A.1 in the Appendix for a list of packages tested to be compatabile with a given Zelig release. You may obtain older versions of most packages at http://www.r-project.org.

Chapter 3

Data Analysis Commands

3.1 Command Syntax

Once R is installed, you only need to know a few basic elements to get started. It's important to remember that R, like any spoken language, has rules for proper syntax. Unlike English, however, the rules for intelligible R are small in number and quite precise (see Section 3.1.2).

3.1.1 Getting Started

1. To start R under Linux or Unix, type R at the terminal prompt or M-x R under ESS.
2. The R prompt is `>`.
3. Type commands and hit enter to execute. (No additional characters, such as semicolons or commas, are necessary at the end of lines.)
4. To quit from R, type `q()` and press enter.
5. The `#` character makes R ignore the rest of the line, and is used in this document to comment R code.
6. We highly recommend that you make a separate working directory or folder for each project.
7. Each R session has a workspace, or working memory, to store the *objects* that you create or input. These objects may be:
   (a) *values*, which include numerical, integer, character, and logical values;
   (b) *data structures* made up of variables (vectors), matrices, and data frames; or
   (c) *functions* that perform the desired tasks on user-specified values or data structures.

After starting R, you may at any time use Zelig's built-in help function to access on-line help for any command. To see help for all Zelig commands, type `help.zelig(command)`, which will take you to the help page for all Zelig commands. For help with a specific Zelig or R command substitute the name of the command for the generic `command`. For example, type `help.zelig(logit)` to view help for the logit model.

3.1.2 Details

.

Zelig uses the syntax of R, which has several essential elements:

1. R is case sensitive. `Zelig`, the package or library, is not the same as `zelig`, the command.

2. R functions accept user-defined arguments: while some arguments are required, other optional arguments modify the function's default behavior. Enclose arguments in parentheses and separate multiple arguments with commas. For example, `print(x)` or `print(x, digits = 2)` prints the contents of the object `x` using the default number of digits or rounds to two digits to the right of the decimal point, respectively. You may nest commands as long as each has its own set of parentheses: `log(sqrt(5))` takes the square root of 5 and then takes the natural log.

3. The `<-` operator takes the output of the function on the right and saves them in the named object on the left. For example, `z.out <- zelig(...)` stores the output from `zelig()` as the object `z.out` in your working memory. You may use `z.out` as an argument in other functions, view the output by typing `z.out` at the R prompt, or save `z.out` to a file using the procedures described in Section 3.2.3.

4. You may name your objects anything, within a few constraints:

   - You may only use letters (in upper or lower case) and periods to punctuate your variable names.
   - You may *not* use any special characters (aside from the period) or spaces to punctuate your variable names.
   - Names cannot begin with numbers. For example, R will not let you save an object as `1997.election` but will let you save `election.1997`.

5. Use the `names()` command to see the contents of R objects, and the `$` operator to extract elements from R objects. For example:

```
# Run least squares regression and save the output in working memory:
> z.out <- zelig(y ~ x1 + x2, model = "ls", data = mydata)
# See what's in the R object:
> names(z.out)
```

```
 [1] "coefficients"  "residuals"  "effects"  "rank"
# Extract and display the coefficients in z.out:
> z.out$coefficients
```

6. All objects have a class designation which tells R how to treat it in subsequent commands. An object's class is generated by the function or mathematical operation that created it.

7. To see a list of all objects in your current workspace, type: `ls()`. You can remove an object permanently from memory by typing `remove(goo)` (which deletes the object `goo`), or remove all the objects with `remove(list = ls())`.

8. To run commands in a batch, use a text editor (such as the Windows R script editor or emacs) to compose your R commands, and save the file with a `.R` file extension in your working directory. To run the file, type `source("Code.R")` at the R prompt.

If you encounter a syntax error, check your spelling, case, parentheses, and commas. These are the most common syntax errors, and are easy to detect and correct with a little practice. If you encounter a syntax error in batch mode, R will tell you the line on which the syntax error occurred.

3.2 Data Sets

3.2.1 Data Structures

Zelig uses only three of R's many data structures:

1. A **variable** is a one-dimensional vector of length n.

2. A **data frame** is a rectangular matrix with n rows and k columns. Each column represents a variable and each row an observation. Each variable may have a different class. (See Section 3.3.1 for a list of classes.) You may refer to specific variables from a data frame using, for example, `data$variable`.

3. A **list** is a combination of different data structures. For example, `z.out` contains both `coefficients` (a vector) and `data` (a data frame). Use `names()` to view the elements available within a list, and the `$` operator to refer to an element in a list.

For a more comprehensive introduction, including ways to manipulate these data structures, please refer to Chapter 6.

3.2.2 Loading Data

Datasets in Zelig are stored in "data frames." In this section, we explain the standard ways to load data from disk into memory, how to handle special cases, and how to verify that the data you loaded is what you think it is.

Standard Ways to Load Data

Make sure that the data file is saved in your working directory. You can check to see what your working directory is by starting R, and typing `getwd()`. If you wish to use a different directory as your starting directory, use `setwd("dirpath")`, where `"dirpath"` is the full directory path of the directory you would like to use as your working directory.

After setting your working directory, load data using one of the following methods:

1. If your dataset is in a **tab- or space-delimited .txt file**, use `read.table("mydata.txt")`

2. If your dataset is **a comma separated table**, use `read.csv("mydata.csv")`.

3. To import **SPSS, Stata, and other data files**, use the foreign package, which automatically preserves field characteristics for each variable. Thus, variables classed as dates in Stata are automatically translated into values in the date class for R. For example:

```
> library(foreign)                             # Load the foreign package.
> stata.data <- read.dta("mydata.dta")         # For Stata data.
> spss.data <- read.spss("mydata.sav", to.data.frame = TRUE) # For SPSS.
```

4. To load data in R format, use `load("mydata.RData")`.

5. For sample data sets included with R packages such as Zelig, you may use the `data()` command, which is a shortcut for loading data from the sample data directories. Because the locations of these directories vary by installation, it is extremely difficult to locate sample data sets and use one of the three preceding methods; `data()` searches all of the currently used packages and loads sample data automatically. For example:

```
> library(Zelig)                                # Loads the Zelig library.
> data(turnout)                                 # Loads the turnout data.
```

Special Cases When Loading Data

These procedures apply to any of the above `read` commands:

1. If your file uses the **first row to identify variable names**, you should use the option `header = TRUE` to import those field names. For example,

```
> read.csv("mydata.csv", header = TRUE)
```

   will read the words in the first row as the variable names and the subsequent rows (each with the same number of values as the first) as observations for each of those variables. If you have additional characters on the last line of the file or fewer values in one of the rows, you need to edit the file before attempting to read the data.

2. The R missing value code is `NA`. If this value is in your data, R will recognize your missing values as such. If you have instead used a place-holder value (such as -9) to represent missing data, you need to tell R this on loading the data:

   ```
   > read.table("mydata.tab", header = TRUE, na.strings = "-9")
   ```

   Note: You must enclose your place holder values in quotes.

3. Unlike Windows, the file extension in R does not determine the default method for dealing with the file. For example, if your data is tab-delimited, but saved as a `.sav` file, `read.table("mydata.sav")` will load your data into R.

Verifying You Loaded The Data Correctly

Whichever method you use, try the `names()`, `dim()`, and `summary()` commands to verify that the data was properly loaded. For example,

```
> data <- read.csv("mydata.csv", header = TRUE)            # Read the data.
> dim(data)                         # Displays the dimensions of the data frame
[1] 16000  8                        #  in rows then columns.
> data[1:10,]                       # Display rows 1-10 and all columns.
> names(data)                       # Check the variable names.
[1] "V1" "V2" "V3"                  # These values indicate that the variables
                                    #  weren't named, and took default values.
> names(data) <- c("income", "educate", "year")     # Assign variable names.
> summary(data)                     # Returning a summary for each variable.
```

In this case, the `summary()` command will return the maximum, minimum, mean, median, first and third quartiles, as well as the number of missing values for each variable.

3.2.3 Saving Data

Use `save()` to write data or any object to a file in your working directory. For example,

```
> save(mydata, file = "mydata.RData")      # Saves `mydata' to `mydata.RData'
                                           #  in your working directory.
> save.image()                             # Saves your entire workspace to
                                           #  the default `.RData' file.
```

R will also prompt you to save your workspace when you use the `q()` command to quit. When you start R again, it will load the previously saved workspace. Restarting R will not, however, load previously used packages. You must remember to load Zelig at the beginning of every R session.

Alternatively, you can recall individually saved objects from .RData files using the `load()` command. For example,

```
> load("mydata.RData")
```

loads the objects saved in the `mydata.RData` file. You may save a data frame, a data frame and associated functions, or other R objects to file.

3.3 Variables

3.3.1 Classes of Variables

R variables come in several types. Certain Zelig models require dependent variables of a certain class of variable. (These are documented under the manual pages for each model.) Use `class(variable)` to determine the class of a variable or `class(data$variable)` for a variable within a data frame.

Types of Variables

For all types of variable (vectors), you may use the `c()` command to "concatenate" elements into a vector, the `:` operator to generate a sequence of integer values, the `seq()` command to generate a sequence of non-integer values, or the `rep()` function to repeat a value to a specified length. In addition, you may use the `<-` operator to save variables (or any other objects) to the workspace. For example:

```
> logic <- c(TRUE, FALSE, TRUE, TRUE, TRUE) # Creates `logic' (5 T/F values).
> var1 <- 10:20                             # All integers between 10 and 20.
> var2 <- seq(from = 5, to = 10, by = 0.5)  # Sequence from 5 to 10 by
                                            #  intervals of 0.5.
> var3 <- rep(NA, length = 20)              # 20 `NA' values.
> var4 <- c(rep(1, 15), rep(0, 15))         # 15 `1's followed by 15 `0's.
```

For the `seq()` command, you may alternatively specify `length` instead of `by` to create a variable with a specific number (denoted by the `length` argument) of evenly spaced elements.

1. **Numeric** variables are real numbers and the default variable class for most dataset values. You can perform any type of math or logical operation on numeric values. If `var1` and `var2` are numeric variables, we can compute

   ```
   > var3 <- log(var2) - 2*var1      # Create `var3' using math operations.
   ```

   `Inf` (infinity), `-Inf` (negative infinity), `NA` (missing value), and `NaN` (not a number) are special numeric values on which most math operations will fail. (Logical operations will work, however.) Use `as.numeric()` to transform variables into numeric variables. Integers are a special class of numeric variable.

2. **Logical** variables contain values of either `TRUE` or `FALSE`. R supports the following logical operators: `==`, exactly equals; `>`, greater than; `<`, less than; `>=`, greater than or equals; `<=`, less than or equals; and `!=`, not equals. The `=` symbol is *not* a logical operator. Refer to Section 3.3.2 for more detail on logical operators. If `var1` and `var2` both have n observations, commands such as

   ```
   > var3 <- var1 < var2
   > var3 <- var1 == var2
   ```

   create n `TRUE`/`FALSE` observations such that the ith observation in `var3` evaluates whether the logical statement is true for the ith value of `var1` with respect to the ith value of `var2`. Logical variables should usually be converted to integer values prior to analysis; use the `as.integer()` command.

3. **Character** variables are sets of text strings. Note that text strings are always enclosed in quotes to denote that the string is a value, not an object in the workspace or an argument for a function (neither of which take quotes). Variables of class character are not normally used in data analysis, but used as descriptive fields. If a character variable is used in a statistical operation, it must first be transformed into a factored variable.

4. **Factor** variables may contain values consisting of either integers or character strings. Use `factor()` or `as.factor()` to convert character or integer variables into factor variables. Factor variables separate unique values into levels. These levels may either be ordered or unordered. In practice, this means that including a factor variable among the explanatory variables is equivalent to creating dummy variables for each level. In addition, some models (ordinal logit, ordinal probit, and multinomial logit), require that the dependent variable be a factor variable.

3.3.2 Recoding Variables

Researchers spend a significant amount of time cleaning and recoding data prior to beginning their analyses. R has several procedures to facilitate the process.

Extracting, Replacing, and Generating New Variables

While it is not difficult to recode variables, the process is prone to human error. Thus, we recommend that before altering the data, you save your existing data frame using the procedures described in Section 3.2.3, that you only recode one variable at a time, and that you recode the variable outside the data frame and then return it to the data frame.

To extract the variable you wish to recode, type:

```
> var <- data$var1               # Copies `var1' from `data', creating `var'.
```

Do *not* sort the extracted variable or delete observations from it. If you do, the ith observation in `var` will no longer match the ith observation in `data`.

To replace the variable or generate a new variable in the data frame, type:

```
> data$var1 <- var                # Replace `var1' in `data' with `var'.
> data$new.var <- var             # Generate `new.var' in `data' using `var'.
```

To remove a variable from a data frame (rather than replacing one variable with another):

```
> data$var1 <- NULL
```

Logical Operators

R has an intuitive method for recoding variables, which relies on logical operators that return statements of `TRUE` and `FALSE`. A mathematical operator (such as `==`, `!=`, `>`, `>=` `<`, and `<=`) takes two objects of equal dimensions (scalars, vectors of the same length, matrices with the same number of rows and columns, or similarly dimensioned arrays) and compares every element in the first object to its counterpart in the second object.

- `==`: checks that one variable "exactly equals" another in a list-wise manner. For example:

  ```
  > x <- c(1, 2, 3, 4, 5)                   # Creates the object `x'.
  > y <- c(2, 3, 3, 5, 1)                   # Creates the object `y'.
  > x == y                                  # Only the 3rd `x' exactly equals
    [1] FALSE FALSE  TRUE FALSE FALSE       #  its counterpart in `y'.
  ```

  (The `=` symbol is *not* a logical operator.)

- `!=`: checks that one variable does not equal the other in a list-wise manner. Continuing the example:

  ```
  > x != y
    [1]  TRUE  TRUE FALSE  TRUE  TRUE
  ```

- `>` (`>=`): checks whether each element in the left-hand object is greater than (or equal to) every element in the right-hand object. Continuing the example from above:

  ```
  > x > y                                      # Only the 5th `x' is greater
    [1] FALSE FALSE FALSE FALSE  TRUE          #  than its counterpart in `y'.
  > x >= y                                     # The 3rd `x' is equal to the
    [1] FALSE FALSE  TRUE FALSE  TRUE          #  3rd `y' and becomes TRUE.
  ```

- `<` (`<=`): checks whether each element in the left-hand object is less than (or equal to) every object in the right-hand object. Continuing the example from above:

```
> x < y                               # The elements 1, 2, and 4 of `x' are
[1]  TRUE  TRUE FALSE  TRUE FALSE #  less than their counterparts in `y'.
> x <= y                              # The 3rd `x' is equal to the 3rd `y'
[1]  TRUE  TRUE  TRUE  TRUE FALSE #  and becomes TRUE.
```

For two vectors of five elements, the mathematical operators compare the first element in `x` to the first element in `y`, the second to the second and so forth. Thus, a mathematical comparison of `x` and `y` returns a vector of five `TRUE`/`FALSE` statements. Similarly, for two matrices with 3 rows and 20 columns each, the mathematical operators will return a 3×20 matrix of logical values.

There are additional logical operators which allow you to combine and compare logical statements:

- `&`: is the logical equivalent of "and", and evaluates one array of logical statements against another in a list-wise manner, returning a `TRUE` only if both are true in the same location. For example:

  ```
  > a <- matrix(c(1:12), nrow = 3, ncol = 4)      # Creates a matrix `a'.
  > a
       [,1] [,2] [,3] [,4]
  [1,]    1    4    7   10
  [2,]    2    5    8   11
  [3,]    3    6    9   12
  > b <- matrix(c(12:1), nrow = 3, ncol = 4)      # Creates a matrix `b'.
  > b
       [,1] [,2] [,3] [,4]
  [1,]   12    9    6    3
  [2,]   11    8    5    2
  [3,]   10    7    4    1
  > v1 <- a > 3                     # Creates the matrix `v1' (T/F values).
  > v2 <- b > 3                     # Creates the matrix `v2' (T/F values).
  > v1 & v2                         # Checks if the (i,j) value in `v1' and
        [,1] [,2] [,3]  [,4]        #  `v2' are both TRUE.  Because columns
  [1,] FALSE TRUE TRUE FALSE        #  2-4 of `v1' are TRUE, and columns 1-3
  [2,] FALSE TRUE TRUE FALSE        #  of `var2' are TRUE, columns 2-3 are
  [3,] FALSE TRUE TRUE FALSE        #  TRUE here.
  > (a > 3) & (b > 3)               # The same, in one step.
  ```

  For more complex comparisons, parentheses may be necessary to delimit logical statements.

- `|`: is the logical equivalent of "or", and evaluates in a list-wise manner whether either of the values are `TRUE`. Continuing the example from above:

```
> (a < 3) | (b < 3)              # (1,1) and (2,1) in `a' are less
      [,1]  [,2]  [,3]  [,4]     #  than 3, and (2,4) and (3,4) in
[1,]  TRUE FALSE FALSE FALSE     #  `b' are less than 3; | returns
[2,]  TRUE FALSE FALSE  TRUE     #  a matrix with `TRUE' in (1,1),
[3,] FALSE FALSE FALSE  TRUE     #  (2,1), (2,4), and (3,4).
```

The `&&` (if and only if) and `||` (either or) operators are used to control the command flow within functions. The `&&` operator returns a `TRUE` only if every element in the comparison statement is true; the `||` operator returns a `TRUE` if any of the elements are true. Unlike the `&` and `|` operators, which return arrays of logical values, the `&&` and `||` operators return only one logical statement irrespective of the dimensions of the objects under consideration. Hence, `&&` and `||` are logical operators which are *not* appropriate for recoding variables.

Coding and Recoding Variables

R uses vectors of logical statements to indicate how a variable should be coded or recoded. For example, to create a new variable `var3` equal to 1 if `var1` < `var2` and 0 otherwise:

```
> var3 <- var1 < var2              # Creates a vector of n T/F observations.
> var3 <- as.integer(var3)         # Replaces the T/F values in `var3' with
                                   #  1's for TRUE and 0's for FALSE.
> var3 <- as.integer(var1 < var2)  # Combine the two steps above into one.
```

In addition to generating a vector of dummy variables, you can also refer to specific values using logical operators defined in Section 3.3.2. For example:

```
> v1 <- var1 == 5                    # Creates a vector of T/F statements.
> var1[v1] <- 4                      # For every TRUE in `v1', replaces the
                                     #  value in `var1' with a 4.
> var1[var1 == 5] <- 4               # The same, in one step.
```

The index (inside the square brackets) can be created with reference to other variables. For example,

```
> var1[var2 == var3] <- 1
```

replaces the ith value in `var1` with a 1 when the ith value in `var2` equals the ith value in `var3`. If you use `=` in place of `==`, however, you will replace all the values in `var1` with 1's because `=` is another way to assign variables. Thus, the statement `var2 = var3` is of course true.

Finally, you may also replace any (character, numerical, or logical) values with special values (most commonly, `NA`).

```
> var1[var1 == "don't know"] <- NA   # Replaces all "don't know"'s with NA's.
```

After recoding the `var1` replace the old `data$var1` with the recoded `var1`: `data$var1 <- var1`. You may combine the recoding and replacement procedures into one step. For example:

```
> data$var1[data$var1 =< 0] <- -1
```

Alternatively, rather than recoding just specific values in variables, you may calculate new variables from existing variables. For example,

```
> var3 <- var1 + 2 * var2
> var3 <- log(var1)
```

After generating the new variables, use the assignment mechanism `<-` to insert the new variable into the data frame.

In addition to generating vectors of dummy variables, you may transform a vector into a matrix of dummy indicator variables. For example, see Section 7.3 to transform a vector of k unique values (with n observations in the complete vector) into a $n \times k$ matrix.

Missing Data

To deal with missing values in some of your variables:

1. You may generate multiply imputed datasets using Amelia (or other programs).

2. You may omit missing values. Zelig models automatically apply list-wise deletion, so no action is required to run a model. To obtain the total number of observations or produce other summary statistics using the analytic dataset, you may manually omit incomplete observations. To do so, first create a data frame containing only the variables in your analysis. For example:

   ```
   > new.data <- cbind(data$dep.var, data$var1, data$var2, data$var3)
   ```

   The `cbind()` command "column binds" variables into a data frame. (A similar command `rbind()` "row binds" observations with the same number of variables into a data frame.) To omit missing values from this new data frame:

   ```
   > new.data <- na.omit(new.data)
   ```

   If you perform `na.omit()` on the full data frame, you risk deleting observations that are fully observed in your experimental variables, but missing values in other variables. Creating a new data frame containing only your experimental variables usually increases the number of observations retained after `na.omit()`.

Chapter 4

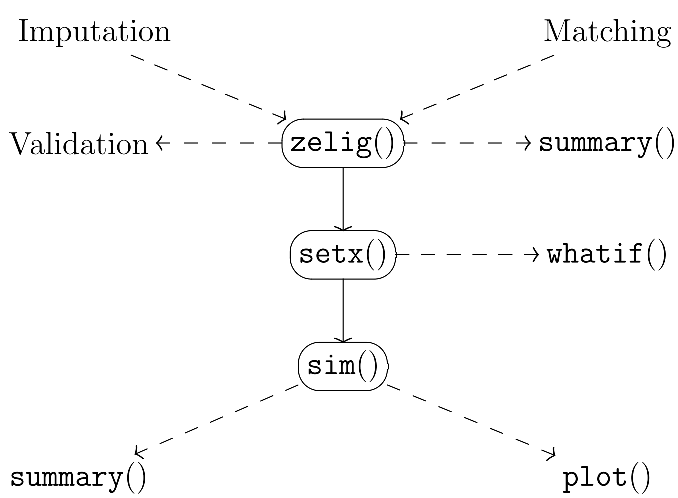
Statistical Commands

4.1 Zelig Commands

4.1.1 Quick Overview

For any statistical model, Zelig does its work with a combination of three commands.

Figure 4.1: Main Zelig commands (solid arrows) and some options (dashed arrows)



1. Use `zelig()` to run the chosen statistical model on a given data set, with a specific set of variables. For standard likelihood models, for example, this step estimates the coefficients, other model parameters, and a variance-covariance matrix. In addition, you may choose from a variety of options:

- Pre-process data: Prior to calling `zelig()`, you may choose from a variety of data pre-processing commands (matching or multiple imputation, for example) to make your statistical inferences more accurate.
- Summarize model: After calling `zelig()`, you may summarize the fitted model output using `summary()`.
- Validate model: After calling `zelig()`, you may choose to validate the fitted model. This can be done, for example, by using cross-validation procedures and diagnostics tools.

2. Use `setx()` to set each of the explanatory variables to chosen (actual or counterfactual) values in preparation for calculating quantities of interest. After calling `setx()`, you may use WhatIf to evaluate these choices by determining whether they involve interpolation (i.e., are inside the convex hull of the observed data) or extrapolation, as well as how far these counterfactuals are from the data. Counterfactuals chosen in `setx()` that involve extrapolation far from the data can generate considerably more model dependence (see King and Zeng (2006), King and Zeng (2007), Stoll et al. (2005)).

3. Use `sim()` to draw simulations of your quantity of interest (such as a predicted value, predicted probability, risk ratio, or first difference) from the model. (These simulations may be drawn using an asymptotic normal approximation (the default), bootstrapping, or other methods when available, such as directly from a Bayesian posterior.) After calling `sim()`, use any of the following to summarize the simulations:

   - The `summary()` function gives a numerical display. For multiple `setx()` values, `summary()` lets you summarize simulations by choosing one or a subset of observations.
   - If the `setx()` values consist of only one observation, `plot()` produces density plots for each quantity of interest.

Whenever possible, we use `z.out` as the `zelig()` output object, `x.out` as the `setx()` output object, and `s.out` as the `sim()` output object, but you may choose other names.

4.1.2 Examples

- Use the `turnout` data set included with Zelig to estimate a logit model of an individual's probability of voting as function of race and age. Simulate the predicted probability of voting for a white individual, with age held at its mean:

```
> data(turnout)
> z.out <- zelig(vote ~ race + age, model = "logit", data = turnout)
> x.out <- setx(z.out, race = "white")
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
```

- Compute a first difference and risk ratio, changing education from 12 to 16 years, with other variables held at their means in the data:

```
> data(turnout)
> z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout)
> x.low <- setx(z.out, educate = 12)
> x.high <- setx(z.out, educate = 16)
> s.out <- sim(z.out, x = x.low, x1 = x.high)
> summary(s.out)                                     # Numerical summary.
> plot(s.out)                                        # Graphical summary.
```

- Calculate expected values for every observation in your data set:

```
> data(turnout)
> z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout)
> x.out <- setx(z.out, fn = NULL)
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
```

- Use five multiply imputed data sets from Scheve and Slaughter (2001) in an ordered logit model:

```
> data(immi1, immi2, immi3, immi4, immi5)
> z.out <- zelig(as.factor(ipip) ~ wage1992 + prtyid + ideol,
                 model = "ologit",
                 data = mi(immi1, immi2, immi3, immi4, immi5))
```

- Use the nearest propensity score matching via *MatchIt* package, and then calculate the conditional average treatment effect of the job training program based on the linear regression model:

```
> library(MatchIt)
> data(lalonde)
> m.out <- matchit(treat ~ re74 + re75 + educ + black + hispan + age,
                   data = lalonde, method = "nearest")
> m.data <- match.data(m.out)
> z.out <- zelig(re78 ~ treat + distance + re74 + re75 + educ + black +
                 hispan + age, data = m.data, model = "ls")
> x.out0 <- setx(z.out, fn = NULL, treat = 0)
> x.out1 <- setx(z.out, fn = NULL, treat = 1)
> s.out <- sim(z.out, x=x.out0, x1=x.out1)
> summary(s.out)
```

- Validate the fitted model using the leave-one-out cross validation procedure and calculating the average squared prediction error via *boot* package. For example:

```
> library(boot)
> data(turnout)
> z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout)
> cv.out <- cv.glm(z.out, data = turnout)
> print(cv.out$delta)
```

4.1.3 Details

1. `z.out <- zelig(formula, model, data, by = NULL, ...)`

   The `zelig()` command estimates a selected statistical model given the specified data. You may name the output object (`z.out` above) anything you desire. You must include three required arguments, in the following order:

   (a) `formula` takes the form `y ~ x1 + x2`, where `y` is the dependent variable and `x1` and `x2` are the explanatory variables, and `y`, `x1`, and `x2` are contained in the same dataset. The `+` symbol means "inclusion" not "addition." You may include interaction terms in the form of `x1*x2` without having to compute them in prior steps or include the main effects separately. For example, R treats the formula `y ~ x1*x2` as `y ~ x1 + x2 + x1*x2`. To prevent R from automatically including the separate main effect terms, use the `I()` function, thus: `y ~ I(x1 * x2)`.

   (b) `model` lets you choose which statistical model to run. You must put the name of the model in quotation marks, in the form `model = "ls"`, for example. See Section 4.2 for a list of currently supported models.

   (c) `data` specifies the data frame containing the variables called in the formula, in the form `data = mydata`. Alternatively, you may input multiply imputed datasets in the form `data = mi(data1, data2, ...)`.[1] If you are working with matched data created using MatchIt, you may create a data frame within the `zelig()` statement by using `data = match.data(...)`. In all cases, the data frame or MatchIt object must have been previously loaded into the working memory.

   (d) `by` (an optional argument which is by default `NULL`) allows you to choose a factor variable (see Section 2) in the data frame as a subsetting variable. For each of the unique strata defined in the `by` variable, `zelig()` does a separate run of the specified model. The variable chosen should *not* be in the formula, because there will be no variance in the `by` variable in the subsets. If you have one data set for all 191 countries in the UN, for example, you may use the `by` option to run the

[1] Multiple imputation is a method of dealing with missing values in your data which is more powerful than the usual list-wise deletion approach. You can create multiply imputed datasets with a program such as Amelia; see King, Honaker, Joseph, Scheve (2000).

same model 191 times, once on each country, all with a single `zelig()` statement. You may also use the `by` option to run models on MatchIt subclasses.

(e) The output object, `z.out`, contains all of the options chosen, including the name of the data set. Because data sets may be large, Zelig does not store the full data set, but only the name of the dataset. Every time you use a Zelig function, it looks for the dataset with the appropriate name in working memory. (Thus, it is critical that you do *not* change the name of your data set, or perform any additional operations on your selected variables between calling `zelig()` and `setx()`, or between `setx()` and `sim()`.)

(f) If you would like to view the regression output at this intermediate step, type `summary(z.out)` to return the coefficients, standard errors, t-statistics and p-values. We recommend instead that you calculate quantities of interest; creating `z.out` is only the first of three steps in this task.

2. `x.out <- setx(z.out, fn = list(numeric = mean, ordered = median, others = mode), data = NULL, cond = FALSE, ...)`

   The `setx()` command lets you choose values for the explanatory variables, with which `sim()` will simulate quantities of interest. There are two types of `setx()` procedures:

   - You may perform the usual *unconditional* prediction (by default, `cond = FALSE`), by explicitly choosing the values of each explanatory variable yourself or letting `setx()` compute them, either from the data used to create `z.out` or from a new data set specified in the optional `data` argument. You may also compute predictions for all observed values of your explanatory variables using `fn = NULL`.
   - Alternatively, for advanced uses, you may perform *conditional* prediction (`cond = TRUE`), which predicts certain quantities of interest by conditioning on the observed value of the dependent variable. In a simple linear regression model, this procedure is not particularly interesting, since the conditional prediction is merely the observed value of the dependent variable for that observation. However, conditional prediction is extremely useful for other models and methods, including the following:
     - In a matched sampling design, the sample average treatment effect for the treated can be estimated by computing the difference between the observed dependent variable for the treated group and their expected or predicted values of the dependent variable under no treatment (Ho et al. 2007).
     - With censored data, conditional prediction will ensure that all predicted values are greater than the censored observed values (King et al. 1990a).
     - In ecological inference models, conditional prediction guarantees that the predicted values are on the tomography line and thus restricted to the known bounds (King 1997; Adolph et al. 2003).

– The conditional prediction in many linear random effects (or Bayesian hierarchical) models is a weighted average of the unconditional prediction and the value of the dependent variable for that observation, with the weight being an estimable function of the accuracy of the unconditional prediction (see **?**). When the unconditional prediction is highly certain, the weight on the value of the dependent variable for this observation is very small, hence reducing inefficiency; when the unconditional prediction is highly uncertain, the relative weight on the unconditional prediction is very small, hence reducing bias. Although the simple weighted average expression no longer holds in nonlinear models, the general logic still holds and the mean square error of the measurement is typically reduced (see King et al. 2004).

In these and other models, conditioning on the observed value of the dependent variable can vastly increase the accuracy of prediction and measurement.

The `setx()` arguments for **unconditional** prediction are as follows:

(a) `z.out`, the `zelig()` output object, must be included first.

(b) You can set particular explanatory variables to specified values. For example:

```
> z.out <- zelig(vote ~ age + race, model = "logit", data = turnout)
> x.out <- setx(z.out, age = 30)
```

`setx()` sets the variables *not* explicitly listed to their mean if numeric, and their median if ordered factors, and their mode if unordered factors, logical values, or character strings. Alternatively, you may specify one explanatory variable as a range of values, creating one observation for every unique value in the range of values:[2]

```
> x.out <- setx(z.out, age = 18:95)
```

This creates 78 observations with with age set to 18 in the first observation, 19 in the second observation, up to 95 in the 78th observation. The other variables are set to their default values, but this may be changed by setting `fn`, as described next.

(c) Optionally, `fn` is a list which lets you to choose a different function to apply to explanatory variables of class

- `numeric`, which is `mean` by default,
- `ordered` factor, which is `median` by default, and
- `other` variables, which consist of logical variables, character string, and unordered factors, and are set to their `mode` by default.

[2]If you allow more than one variable to vary at a time, you risk confounding the predictive effect of the variables in question.

While any function may be applied to numeric variables, `mean` will default to median for ordered factors, and mode is the only available option for other types of variables. In the special case, `fn = NULL`, `setx()` returns all of the observations.

(d) You cannot perform other math operations within the `fn` argument, but can use the output from one call of `setx` to create new values for the explanatory variables. For example, to set the explanatory variables to one standard deviation below their mean:

```
> X.sd <- setx(z.out, fn = list(numeric = sd))
> X.mean <- setx(z.out, fn = list(numeric = mean))
> x.out <- X.mean - X.sd
```

(e) Optionally, `data` identifies a new data frame (rather than the one used to create `z.out`) from which the `setx()` values are calculated. You can use this argument to set values of the explanatory variables for hold-out or out-of-sample fit tests.

(f) The `cond` is always `FALSE` for unconditional prediction.

If you wish to calculate risk ratios or first differences, call `setx()` a second time to create an additional set of the values for the explanatory variables. For example, continuing from the example above, you may create an alternative set of explanatory variables values one standard deviation above their mean:

```
> x.alt <- X.mean + X.sd
```

The required arguments for **conditional** prediction are as follows:

(a) `z.out`, the `zelig()` output object, must be included first.

(b) `fn`, which equals `NULL` to indicate that all of the observations are selected. You may only perform conditional inference on actual observations, not the mean of observations or any other function applied to the observations. Thus, if `fn` is missing, but `cond = TRUE`, `setx()` coerces `fn = NULL`.

(c) `data`, the data for conditional prediction.

(d) `cond`, which equals `TRUE` for conditional prediction.

Additional arguments, such as any of the variable names, are ignored in conditional prediction since the actual values of that observation are used.

3. `s.out <- sim(z.out, x = x.out, x1 = NULL, num = c(1000, 100), bootstrap = FALSE, bootfn = NULL, ...)`

   The `sim()` command simulates quantities of interest given the output objects from `zelig()` and `setx()`. This procedure uses only the assumptions of the statistical model. The `sim()` command performs either unconditional or conditional prediction depending on the options chosen in `setx()`.

   The arguments are as follows for **unconditional** prediction:

(a) `z.out`, the model output from `zelig()`.

(b) `x`, the output from the `setx()` procedure performed on the model output.

(c) Optionally, you may calculate first differences by specifying `x1`, an additional `setx()` object. For example, using the `x.out` and `x.alt`, you may generate first differences using:

```
> s.out <- sim(z.out, x = x.out, x1 = x.alt)
```

(d) By default, the number of simulations, `num`, equals 1000 (or 100 simulations if bootstrap is selected), but this may be decreased to increase computational speed, or increased for additional precision.

(e) Zelig simulates parameters from classical *maximum likelihood* models using asymptotic normal approximation to the log-likelihood. This is the same assumption as used for frequentist hypothesis testing (which is of course equivalent to the asymptotic approximation of a Bayesian posterior with improper uniform priors). See King, Tomz, and Wittenberg (2000). For *Bayesian models*, Zelig simulates quantities of interest from the posterior density, whenever possible. For *robust Bayesian models*, simulations are drawn from the identified class of Bayesian posteriors.

(f) Alternatively, you may set `bootstrap = TRUE` to simulate parameters using bootstrapped data sets. If your dataset is large, bootstrap procedures will usually be more memory intensive and time-consuming than simulation using asymptotic normal approximation. The type of bootstrapping (including the sampling method) is determined by the optional argument `bootfn`, described below.

(g) If `bootstrap = TRUE` is selected, `sim()` will bootstrap parameters using the default `bootfn`, which re-samples from the data frame with replacement to create a sampled data frame of the same number of observations, and then re-runs `zelig()` (inside `sim()`) to create one set of bootstrapped parameters. Alternatively, you may create a function outside the `sim()` procedure to handle different bootstrap procedures. Please consult `help(boot)` for more details.[3]

For **conditional** prediction, `sim()` takes only two required arguments:

(a) `z.out`, the model output from `zelig()`.

(b) `x`, the conditional output from `setx()`.

(c) Optionally, for duration models, `cond.data`, which is the `data` argument from `setx()`. For models for duration dependent variables (see Section 6), `sim()` must impute the uncensored dependent variables before calculating the average treatment effect. Inputting the `cond.data` allows `sim()` to generate appropriate values.

Additional arguments are ignored or generate error messages.

[3]If you choose to create your own `bootfn`, it must include the the following three arguments: `data`, the original data frame; one of the sampling methods described in `help(boot)`; and `object`, the original `zelig()` output object. The alternative bootstrapping function must sample the data, fit the model, and extract the model-specific parameters.

Presenting Results

1. Use `summary(s.out)` to print a summary of your simulated quantities. You may specify the number of significant digits as:

   ```
   > print(summary(s.out), digits = 2)
   ```

2. Alternatively, you can plot your results using `plot(s.out)`.

3. You can also use `names(s.out)` to see the names and a description of the elements in this object and the `$` operator to extract particular results. For most models, these are: `s.out$qi$pr` (for predicted values), `s.out$qi$ev` (for expected values), and `s.out$qi$fd` (for first differences in expected values). For the logit, probit, multinomial logit, ordinal logit, and ordinal probit models, quantities of interest also include `s.out$qi$rr` (the risk ratio).

4.2 Supported Models

We list here all models implemented in Zelig, organized by the nature of the dependent variable(s) to be predicted, explained, or described.

1. **Continuous Unbounded** dependent variables can take any real value in the range $(-\infty, \infty)$. While most of these models take a continuous dependent variable, Bayesian factor analysis takes multiple continuous dependent variables.

   (a) `"ls"`: The *linear least-squares* (see Section 12.32) calculates the coefficients that minimize the sum of squared residuals. This is the usual method of computing linear regression coefficients, and returns unbiased estimates of β and σ^2 (conditional on the specified model).

   (b) `"normal"`: The *Normal* (see Section 12.39) model computes the maximum-likelihood estimator for a Normal stochastic component and linear systematic component. The coefficients are identical to `ls`, but the maximum likelihood estimator for σ^2 is consistent but biased.

   (c) `"normal.bayes"`: The *Bayesian Normal* regression model (Section 12.40) is similar to maximum likelihood Gaussian regression, but makes valid small sample inferences via draws from the exact posterior and also allows for priors.

   (d) `"netls"`: The *network least squares* regression (Section **??**) is similar to least squares regression for continuous-valued proximity matrix dependent variables. Proximity matrices are also known as sociomatrices, adjacency matrices, and matrix representations of directed graphs.

   (e) `"tobit"`: The *tobit* regression model (see Section 12.65) is a Normal distribution with left-censored observations.

(f) `"tobit.bayes"`: The *Bayesian tobit* distribution (see Section 12.66) is a Normal distribution that has either left and/or right censored observations.

(g) `"arima"`: Use *auto-regressive, integrated, moving-average* (ARIMA) models for time series data (see Section **??**.

(h) `"factor.bayes"`: The *Bayesian factor analysis* model (see Section 12.12) estimates multiple observed continuous dependent variables as a function of latent explanatory variables.

2. **Dichotomous** dependent variables consist of two discrete values, usually $(0, 1)$.

(a) `"logit"`: *Logistic regression* (see Section 12.22) specifies $\Pr(Y = 1)$ to be a(n inverse) logistic transformation of a linear function of a set of explanatory variables.

(b) `"relogit"`: The *rare events logistic* regression option (see Section 12.62) estimates the same model as the logit, but corrects for bias due to rare events (when one of the outcomes is much more prevalent than the other). It also optionally uses prior correction to correct for choice-based (case-control) sampling designs.

(c) `"logit.bayes"`: *Bayesian logistic regression* (see Section 12.23) is similar to maximum likelihood logistic regression, but makes valid small sample inferences via draws from the exact posterior and also allows for priors.

(d) `"probit"`: *Probit regression* (see Section 12.55) Specifies $\Pr(Y = 1)$ to be a(n inverse) CDF normal transformation as a linear function of a set of explanatory variables.

(e) `"probit.bayes"`: *Bayesian probit* regression (see Section 12.56) is similar to maximum likelihood probit regression, but makes valid small sample inferences via draws from the exact posterior and also allows for priors.

(f) `"netlogit"`: The *network logistic* regression (Section **??**) is similar to logistic regression for binary-valued proximity matrix dependent variables. Proximity matrices are also known as sociomatrices, adjacency matrices, and matrix representations of directed graphs.

(g) `"blogit"`: The *bivariate logistic* model (see Section 12.3) models $\Pr(Y_{i1} = y_1, Y_{i2} = y_2)$ for $(y_1, y_2) = (0, 0), (0, 1), (1, 0), (1, 1)$ according to a bivariate logistic density.

(h) `"bprobit"`: The *bivariate probit* model (see Section 12.4) models $\Pr(Y_{i1} = y_1, Y_{i2} = y_2)$ for $(y_1, y_2) = (0, 0), (0, 1), (1, 0), (1, 1)$ according to a bivariate normal density.

(i) `"irt1d"`: The *one-dimensional item response* model (see Section 12.20) takes multiple dichotomous dependent variables and models them as a function of *one* latent (unobserved) explanatory variable.

(j) `"irtkd"`: The *k-dimensional item response* model (see Section 12.21) takes multiple dichotomous dependent variables and models them as a function of k latent (unobserved) explanatory variables.

3. **Ordinal** are used to model ordered, discrete dependent variables. The values of the outcome variables (such as kill, punch, tap, bump) are ordered, but the distance between any two successive categories is not known exactly. Each dependent variable may be thought of as linear, with one continuous, unobserved dependent variable observed through a mechanism that only returns the ordinal choice.

   (a) `"ologit"`: The *ordinal logistic* model (see Section 12.45) specifies the stochastic component of the unobserved variable to be a standard logistic distribution.

   (b) `"oprobit"`: The *ordinal probit* distribution (see Section 12.46) specifies the stochastic component of the unobserved variable to be standardized normal.

   (c) `"oprobit.bayes"`: *Bayesian ordinal probit* model (see Section 12.47) is similar to ordinal probit regression, but makes valid small sample inferences via draws from the exact posterior and also allows for priors.

   (d) `"factor.ord"`: *Bayesian ordered factor analysis* (see Section 12.14) models observed, ordinal dependent variables as a function of latent explanatory variables.

4. **Multinomial** dependent variables are unordered, discrete categorical responses. For example, you could model an individual's choice among brands of orange juice or among candidates in an election.

   (a) `"mlogit"`: The *multinomial logistic* model (see Section 12.35) specifies categorical responses distributed according to the multinomial stochastic component and logistic systematic component.

   (b) `"mlogit.bayes"`: *Bayesian multinomial logistic* regression (see Section 12.36) is similar to maximum likelihood multinomial logistic regression, but makes valid small sample inferences via draws from the exact posterior and also allows for priors.

5. **Count** dependent variables are non-negative integer values, such as the number of presidential vetoes or the number of photons that hit a detector.

   (a) `"poisson"`: The *Poisson* model (see Section 12.48) specifies the expected number of events that occur in a given observation period to be an exponential function of the explanatory variables. The Poisson stochastic component has the property that, $\lambda = \mathrm{E}(Y_i|X_i) = \mathrm{V}(Y_i|X_i)$.

   (b) `"poisson.bayes"`: *Bayesian Poisson* regression (see Section 12.49) is similar to maximum likelihood Poisson regression, but makes valid small sample inferences via draws from the exact posterior and also allows for priors.

   (c) `"negbin"`: The *negative binomial* model (see Section 12.38) has the same systematic component as the Poisson, but allows event counts to be over-dispersed, such that $\mathrm{V}(Y_i|X_i) > \mathrm{E}(Y_i|X_i)$.

6. **Continuous Bounded** dependent variables that are continuous only over a certain range, usually $(0, \infty)$. In addition, some models (exponential, lognormal, and Weibull) are also censored for values greater than some censoring point, such that the dependent variable has some units fully observed and others that are only partially observed (censored).

   (a) `"gamma"`: The *Gamma* model (see Section 12.15) for positively-valued, continuous dependent variables that are fully observed (no censoring).

   (b) `"exp"`: The *exponential* model (see Section 12.11) for right-censored dependent variables assumes that the hazard function is constant over time. For some variables, this may be an unrealistic assumption as subjects are more or less likely to fail the longer they have been exposed to the explanatory variables.

   (c) `"weibull"`: The *Weibull* model (see Section 12.68) for right-censored dependent variables relaxes the assumption of constant hazard by including an additional scale parameter α: If $\alpha > 1$, the risk of failure increases the longer the subject has survived; if $\alpha < 1$, the risk of failure decreases the longer the subject has survived. While `zelig()` estimates α by default, you may optionally fix α at any value greater than 0. Fixing $\alpha = 1$ results in an exponential model.

   (d) `"lognorm"`: The *log-normal* model (see Section 12.31) for right-censored duration dependent variables specifies the hazard function non-monotonically, with increasing hazard over part of the observation period and decreasing hazard over another.

7. **Mixed** dependent variables include models that take more than one dependent variable, where the dependent variables come from two or more of categories above. (They do not need to be of a homogeneous type.)

   (a) The *Bayesian mixed factor analysis* model, in contrast to the Bayesian factor analysis model and ordinal factor analysis model, can model both types of dependent variables as a function of latent explanatory variables.

8. **Ecological inference** models estimate unobserved internal cell values given contingency tables with observed row and column marginals.

   (a) `ei.hier`: The *hierarchical* EI model (see Section 12.9) produces estimates for a cross-section of 2×2 tables.

   (b) `ei.dynamic`: *Quinn's dynamic Bayesian* EI model (see Section 12.8) estimates a dynamic Bayesian model for 2×2 tables with temporal dependence across tables.

   (c) `ei.RxC`: The $R \times C$ EI model (see Section **??**) estimates a hierarchical Multinomial-Dirichlet EI model for contingency tables with more than 2 rows or columns.

4.3 Replication Procedures

A large part of any statistical analysis is documenting your work such that given the same data, anyone may replicate your results. In addition, many journals require the creation and dissemination of "replication data sets" in order that others may replicate your results (see King, 1995). Whether you wish to create replication materials for your own records, or contribute data to others as a companion to your published work, Zelig makes this process easy.

4.3.1 Saving Replication Materials

Let `mydata` be your final data set, `z.out` be your `zelig()` output, and `s.out` your `sim()` output. To save all of this in one file, type:

```
> save(mydata, z.out, s.out, file = "replication.RData")
```

This creates the file replication.RData in your working directory. You may compress this file using zip or gzip tools.

If you have run several specifications, all of these estimates may be saved in one .RData file. Even if you only created quantities of interest from one of these models, you may still save all the specifications in one file. For example:

```
> save(mydata, z.out1, z.out2, s.out, file = "replication.RData")
```

Although the .RData format can contain data sets as well as output objects, it is not the most space-efficient way of saving large data sets. In an uncompressed format, ASCII text files take up less space than data in .RData format. (When compressed, text-formatted data is still smaller than .RData-formatted data.) Thus, if you have more than 100,000 observations, you may wish to save the data set separately from the Zelig output objects. To do this, use the `write.table()` command. For example, if `mydata` is a data frame in your workspace, use `write.table(mydata, file = "mydata.tab")` to save this as a tab-delimited ASCII text file. You may specify other delimiters as well; see `help.zelig("write.table")` for options.

4.3.2 Replicating Analyses

If the data set and analyses are all saved in one .RData file, located in your working directory, you may simply type:

```
> load("replication.RData")              # Loads the replication file.
> z.rep <- repl(z.out)                   # To replicate the model only.
> s.rep <- repl(s.out)                   # To replicate the model and
                                         #  quantities of interest.
```

By default, `repl()` uses the same options used to create the original output object. Thus, if the original `s.out` object used bootstrapping with 245 simulations, the `s.rep` object will similarly have 245 bootstrapped simulations. In addition, you may use the `prev` option when replicating quantities of interest to reuse rather than recreate simulated parameters. Type `help.zelig("repl")` to view the complete list of options for `repl()`.

If the data were saved in a text file, use `read.table()` to load the data, and then replicate the analysis:

```
> dat <- read.table("mydata.tab", header = TRUE)  # Where `dat' is the same
> load("replication.RData")                       #   as the name used in
> z.rep <- repl(z.out)                            #   `z.out'.
> s.rep <- repl(s.out)
```

If you have problems loading the data, please refer to Section 3.2.2.

Finally, you may use the `identical()` command to ensure that the replicated regression output is in every way identical to the original `zelig()` output.[4] For example:

```
> identical(z.out$coef, z.rep$coef)              # Checks the coefficients.
```

Simulated quantities of interest will vary from the original quantities if parameters are re-simulated or re-sampled. If you wish to use `identical()` to verify that the quantities of interest are identical, you may use

```
# Re-use the parameters simulated (and stored) in the original sim() output.
> s.rep <- repl(s.out, prev = s.out$par)

# Check that the expected values are identical.  You may do this for each qi.
> identical(s.out$qi$ev, s.rep$qi$ev)
```

[4]The `identical()` command checks that numeric values are identical to the maximum number of decimal places (usually 16), and also checks that the the two objects have the same class (numeric, character, integer, logical, or factor). Refer to `help(identical)` for more information.

Chapter 5

Graphing Commands

R, and thus Zelig, can produce exceptionally beautiful plots. Many built-in plotting functions exist, including scatter plots, line charts, histograms, bar charts, pie charts, ternary diagrams, contour plots, and a variety of three-dimensional graphs. If you desire, you can exercise a high degree of control to generate just the right graphic. Zelig includes several default plots for one-observation simulations for each model. To view these plots on-screen, simply type `plot(s.out)`, where `s.out` is the output from `sim()`. Depending on the model chosen, `plot()` will return different plots.

If you wish to create your own plots, this section reviews the most basic procedures for creating and saving two-dimensional plots. R plots material in two steps:

1. You must call an output device (discussed in Section 5.3), select a type of plot, draw a plotting region, draw axes, and plot the given data. At this stage, you may also define axes labels, the plot title, and colors for the plotted data. Step one is described in Section 5.1 below.

2. Optionally, you may add points, lines, text, or a legend to the existing plot. These commands are described in Section 5.2.

5.1 Drawing Plots

The most generic plotting command is `plot()`, which automatically recognizes the type of R object(s) you are trying to plot and selects the best type of plot. The most common graphs returned by `plot()` are as follows:

1. If `X` is a variable of length n, `plot(X)` returns a scatter plot of (x_i, i) for $i = 1, \ldots n$. If X is unsorted, this procedure produces a messy graph. Use `plot(sort(X))` to arrange the plotted values of (x_i, i) from smallest to largest.

2. With two numeric vectors `X` and `Y`, both of length n, `plot(X, Y)` plots a scatter plot of each point (x_i, y_i) for $i = 1, \ldots n$. Alternatively, if Z is an object with two vectors, `plot(Z)` also creates a scatter plot.

Optional arguments specific to `plot` include:

- `main` creates a title for the graph, and `xlab` and `ylab` label the x and y axes, respectively. For example,

  ```
  plot(x, y, main = "My Lovely Plot", xlab = "Explanatory Variable",
       ylab = "Dependent Variable")
  ```

- `type` controls the type of plot you request. The default is `plot(x, y, type = "p")`, but you may choose among the following types:

`"p"`	points
`"l"`	lines
`"b"`	both points and lines
`"c"`	lines drawn up to but not including the points
`"h"`	histogram
`"s"`	a step function
`"n"`	a blank plotting region (with the axes specified)

- If you choose `type = "p"`, R plots open circles by default. You can change the type of point by specifying the `pch` argument. For example, `plot(x, y, type = "p", pch = 19)` creates a scatter-plot of filled circles. Other options for `pch` include:

19	solid circle (a disk)
20	smaller solid circle
21	circle
22	square
23	diamond
24	triangle pointed up
25	triangle pointed down

  In addition, you can specify your own symbols by using, for example, `pch = "*"` or `pch = "."`.

- If you choose `type = "l"`, R plots solid lines by default. Use the optional `lty` argument to set the line type. For example, `plot(x, y, type = "l", lty = "dashed")` plots a dashed line. Other options are dotted, dotdash, longdash, and twodash.

- `col` sets the color of the points, lines, or bars. For example, `plot(x, y, type = "b", pch = 20, lty = "dotted", col = "violet")` plots small circles connected by a dotted line, both of which are violet. (The axes and labels remain black.) Use `colors()` to see the full list of available colors.

- `xlim` and `ylim` set the limits to the x-axis and y-axis. For example, `plot(x, y, xlim = c(0, 25), ylim = c(-15, 5))` sets range of the x-axis to [0, 25] and the range of the y-axis to $[-15, 5]$.

For additional plotting options, refer to `help(par)`.

5.2 Adding Points, Lines, and Legends to Existing Plots

Once you have created a plot, you can *add* points, lines, text, or a legend. To place each of these elements, R uses coordinates defined in terms of the x-axes and y-axes of the plot area, not coordinates defined in terms of the the plotting window or device. For example, if your plot has an x-axis with values between $[0, 100]$, and a y-axis with values between $[50, 75]$, you may add a point at $(55, 55)$.

- **points()** plots one or more sets of points. Use `pch` with `points` to add points to an existing plot. For example, `points(P, Q, pch = ".", col = "forest green")` plots each (p_i, q_i) as tiny green dots.

- **lines()** joins the specified points with line segments. The arguments `col` and `lty` may also be used. For example, `lines(X, Y, col = "blue", lty = "dotted")` draws a blue dotted line from each set of points (x_i, y_i) to the next. Alternatively, `lines` also takes command output which specifies (x, y) coordinates. For example, `density(Z)` creates a vector of x and a vector of y, and `plot(density(Z))` draws the kernel density function.

- **text()** adds a character string at the specified set of (x, y) coordinates. For example, `text(5, 5, labels = "Key Point")` adds the label "Key Point" at the plot location $(5, 5)$. You may also choose the font using the `font` option, the size of the font relative to the axis labels using the `cex` option, and choose a color using the `col` option. The full list of options may be accessed using `help(text)`.

- **legend()** places a legend at a specified set of (x, y) coordinates. Type `demo(vertci)` to see an example for `legend()`.

5.3 Saving Graphs to Files

By default, R displays graphs in a window on your screen. To save R plots to file (to include them in a paper, for example), preface your plotting commands with:

```
> ps.options(family = c("Times"), pointsize = 12)
> postscript(file = "mygraph.eps", horizontal = FALSE, paper = "special",
             width = 6.25, height = 4)
```

where the `ps.options()` command sets the font type and size in the output file, and the `postscript` command allows you to specify the name of the file as well as several additional options. Using `paper = special` allows you to specify the width and height of the encapsulated postscript region in inches (6.25 inches long and 4 inches high, in this case), and the statement `horizontal = FALSE` suppresses R's default landscape orientation. Alternatively, you may use `pdf()` instead of `postscript()`. If you wish to select postscript options for .pdf output, you may do so using options in `pdf()`. For example:

```
> pdf(file = "mygraph.pdf", width = 6.25, height = 4, family = "Times",
+     pointsize = 12)
```

At the end of every plot, you should close your output device. The command `dev.off()` stops writing and saves the `.eps` or `.pdf` file to your working directory. If you forget to close the file, you will write all subsequent plots to the same file, overwriting previous plots. You may also use `dev.off()` to close on-screen plot windows.

To write multiple plots to the same file, you can use the following options:

- For plots on separate pages in the same `.pdf` document, use

  ```
  > pdf(file = "mygraph.pdf", width = 6.25, height = 4, family = "Times",
  +     pointsize = 12, onefile = TRUE)
  ```

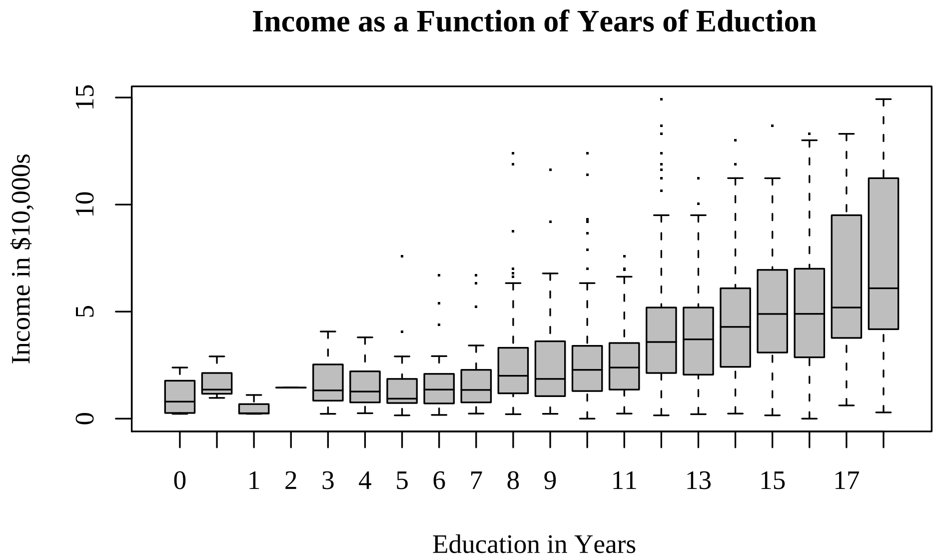
- For multiple plots on one page, initialize either a `.pdf` or `.eps` file, then (before any plotting commands) type:

  ```
  par(mfrow = c(2, 4))
  ```

  This creates a grid that has two rows and four columns. Your plot statements will populate the grid going across the first row, then the second row, from left to right.

5.4 Examples

5.4.1 Descriptive Plots: Box-plots

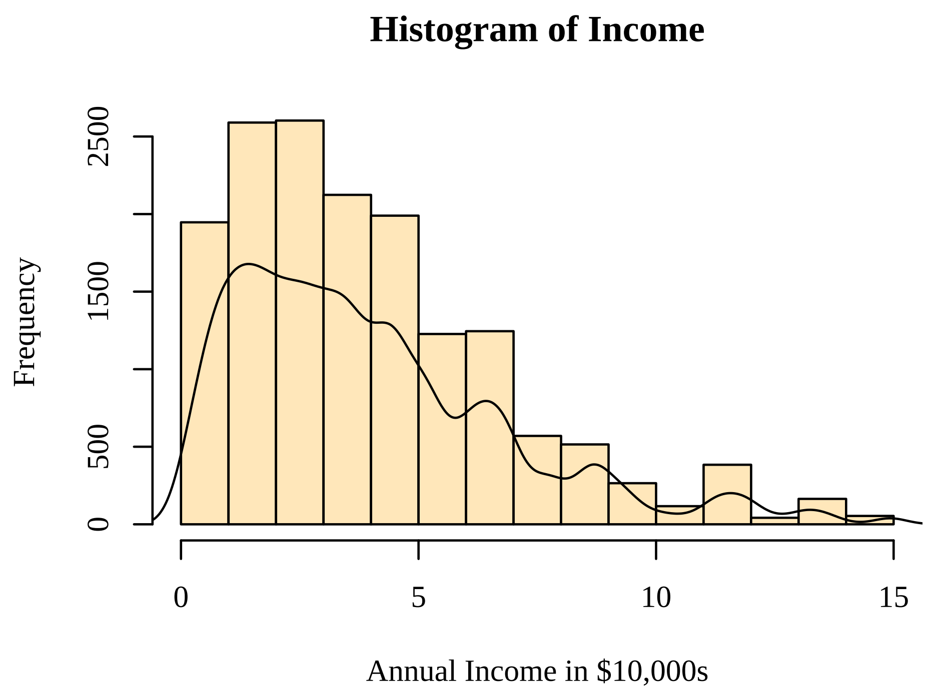


Using the sample `turnout` data set included with Zelig, the following commands will produce the graph above.

```
> library(Zelig)                              # Loads the Zelig package.
> data (turnout)                              # Loads the sample data.
> boxplot(income ~ educate,                   # Creates a boxplot with income
+  data = turnout, col = "grey", pch = ".",   #  as a function of education.
+  main = "Income as a Function of Years of Education",
+  xlab = "Education in Years", ylab = "Income in \$10,000s")
```

5.4.2 Density Plots: A Histogram

Histograms are easy ways to evaluate the density of a quantity of interest.



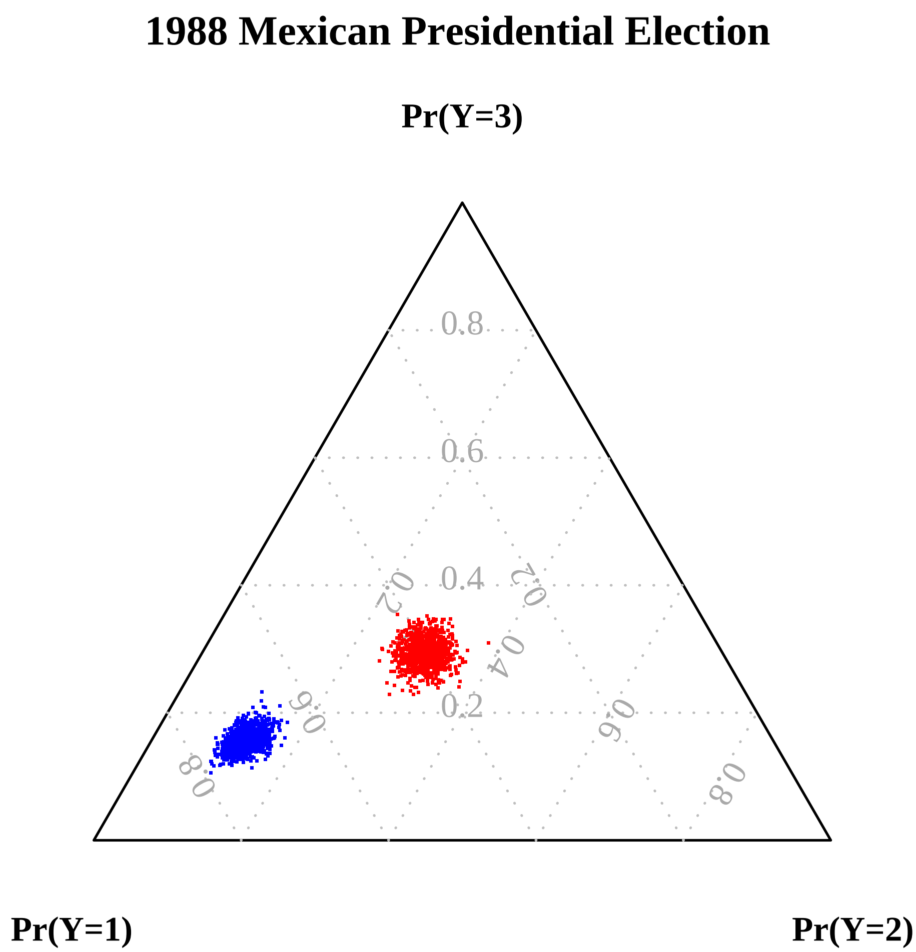
Here's the code to create this graph:

```
> library(Zelig)                                # Loads the Zelig package.
> data(turnout)                                 # Loads the sample data set.
> truehist(turnout$income,  col = "wheat1",     # Calls the main plot, with
+      xlab = "Annual Income in $10,000s",      #  options.
+      main = "Histogram of Income")
> lines(density(turnout$income))                # Adds the kernel density line.
```

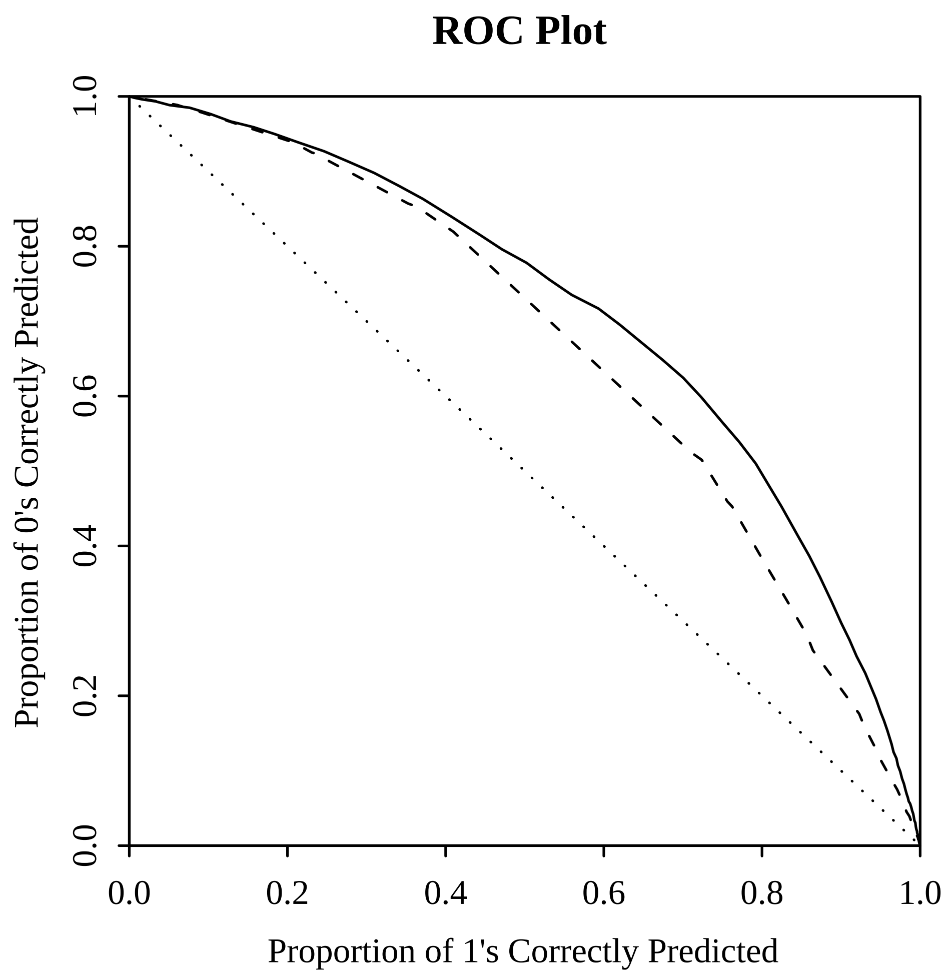
5.4.3 Advanced Examples

The examples above are simple examples which only skim the surface of R's plotting potential. We include more advanced, model-specific plots in the Zelig demo scripts, and have created functions that automate some of these plots, including:

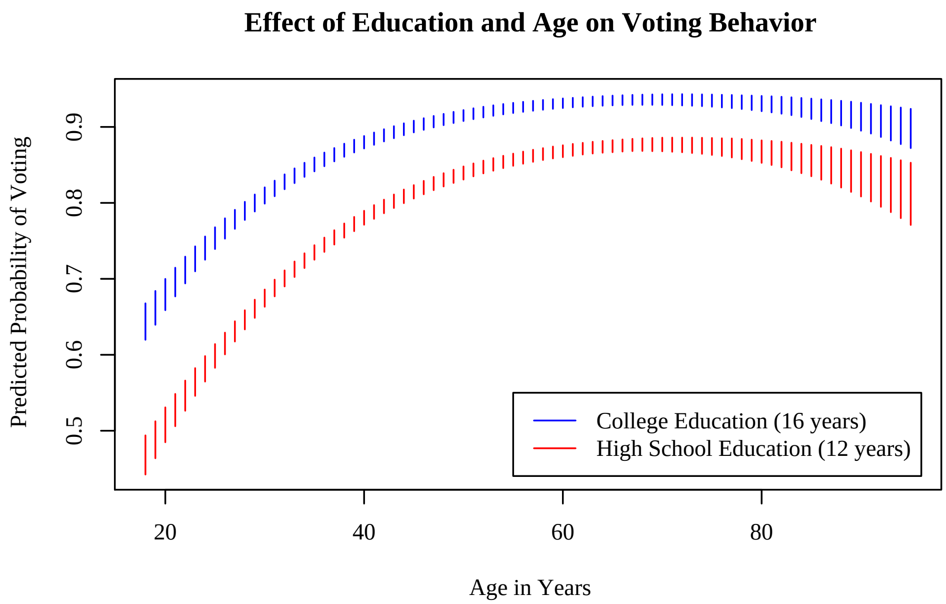
1. **Ternary Diagrams** describe the predicted probability of a categorical dependent variable that has three observed outcomes. You may choose to use this plot with the multinomial logit, the ordinal logit, or the ordinal probit models (Katz and King, 1999). See Section 12.35 for the sample code, type `demo(mlogit)` at the R prompt to run the example, and refer to Section 12.35 to add points to a ternary diagram.



2. **ROC Plots** summarize how well models for binary dependent variables (logit, probit, and relogit) fit the data. The ROC plot evaluates the fraction of 0's and 1's correctly predicted for every possible threshold value at which the continuous Prob($Y = 1$) may be realized as a dichotomous prediction. The closer the ROC curve is to the upper right corner of the plot, the better the fit of the model specification (King and Zeng, 2002*b*). See Section 3 for the sample code, and type `demo(roc)` at the R prompt to run the example.



3. **Vertical Confidence Intervals** may be used for almost any model, and describe simulated confidence intervals for any quantity of interest while allowing one of the explanatory variables to vary over a given range of values (King, Tomz and Wittenberg, 2000). Type `demo(vertci)` at the R prompt to run the example, and `help.zelig(plot.ci)` for the manual page.



Part II

Advanced Zelig Uses

Chapter 6

R Objects

In R, objects can have one or more classes, consisting of the class of the scalar value and the class of the data structure holding the scalar value. Use the `is()` command to determine what an object *is*. If you are already familiar with R objects, you may skip to Section 3.2.2 for loading data, or Section 4.1 for a description of Zelig commands.

6.1 Scalar Values

R uses several classes of scalar values, from which it constructs larger data structures. R is highly class-dependent: certain operations will only work on certain types of values or certain types of data structures. We list the three basic types of scalar values here for your reference:

1. **Numeric** is the default value type for most numbers. An `integer` is a subset of the `numeric` class, and may be used as a `numeric` value. You can perform any type of math or logical operation on numeric values, including:

```
> log(3 * 4 * (2 + pi))          # Note that pi is a built-in constant,
   [1] 4.122270                  #   and log() the natural log function.
> 2 > 3                          # Basic logical operations, including >,
   [1] FALSE                     #   <, >= (greater than or equals),
                                 #   <= (less than or equals), == (exactly
                                 #   equals), and != (not equals).
> 3 >= 2 && 100 == 1000/10       # Advanced logical operations, including
   [1] TRUE                      #   & (and), && (if and only if), | (or),
                                 #   and || (either or).
```

   Note that `Inf` (infinity), `-Inf` (negative infinity), `NA` (missing value), and `NaN` (not a number) are special numeric values on which most math operations will fail. (Logical operations will work, however.)

2. **Logical** operations create logical values of either `TRUE` or `FALSE`. To convert logical values to numerical values, use the `as.integer()` command:

```
> as.integer(TRUE)
   [1] 1
> as.integer(FALSE)
   [1] 0
```

3. **Character** values are text strings. For example,

```
> text <- "supercalafragilisticxpaladocious"
> text
[1] "supercalafragilisticxpaladocious"
```

   assigns the text string on the right-hand side of the `<-` to the named object in your workspace. Text strings are primarily used with data frames, described in the next section. R always returns character strings in quotes.

6.2 Data Structures

6.2.1 Arrays

Arrays are data structures that consist of only one type of scalar value (e.g., a vector of character strings, or a matrix of numeric values). The most common versions, one-dimensional and two-dimensional arrays, are known as *vectors* and *matrices*, respectively.

Ways to create arrays

1. Common ways to create **vectors** (or one-dimensional arrays) include:

```
> a <- c(3, 7, 9, 11)    # Concatenates numeric values into a vector
> a <- c("a", "b", "c")  # Concatenates character strings into a vector
> a <- 1:5               # Creates a vector of integers from 1 to 5 inclusive
> a <- rep(1, times = 5) # Creates a vector of 5 repeated `1's
```

   To manipulate a vector:

```
> a[10]                 # Extracts the 10th value from the vector `a'
> a[5] <- 3.14          # Inserts 3.14 as the 5th value in the vector `a'
> a[5:7] <- c(2, 4, 7) # Replaces the 5th through 7th values with 2, 4, and 7
```

   Unlike larger arrays, vectors can be extended without first creating another vector of the correct length. Hence,

```
> a <- c(4, 6, 8)
> a[5] <- 9       # Inserts a 9 in the 5th position of the vector,
                  #  automatically inserting an `NA' values position 4
```

2. A **factor vector** is a special type of vector that allows users to create j indicator variables in one vector, rather than using j dummy variables (as in Stata or SPSS). R creates this special class of vector from a pre-existing vector `x` using the `factor()` command, which separates `x` into levels based on the discrete values observed in `x`. These values may be either integer value or character strings. For example,

   ```
   > x <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 9, 9, 9, 9)
   > factor(x)
      [1] 1 1 1 1 1 2 2 2 2 9 9 9 9
      Levels: 1 2 9
   ```

   By default, `factor()` creates unordered factors, which are treated as discrete, rather than ordered, levels. Add the optional argument `ordered = TRUE` to order the factors in the vector:

   ```
   > x <- c("like", "dislike", "hate", "like", "don't know", "like", "dislike")
   > factor(x, levels = c("hate", "dislike", "like", "don't know"),
   +        ordered = TRUE)
     [1] like    dislike    hate    like   don't know   like   dislike
   Levels: hate < dislike < like < don't know
   ```

   The `factor()` command orders the levels according to the order in the optional argument `levels`. If you omit the levels command, R will order the values as they occur in the vector. Thus, omitting the `levels` argument sorts the levels as `like < dislike < hate < don't know` in the example above. If you omit one or more of the levels in the list of levels, R returns levels values of `NA` for the missing level(s):

   ```
   > factor(x, levels = c("hate", "dislike", "like"), ordered = TRUE)
     [1] like    dislike hate    like    <NA>    like    dislike
   Levels: hate < dislike < like
   ```

   Use factored vectors within data frames for plotting (see Section 5.1), to set the values of the explanatory variables using `setx` (see Section 10) and in the ordinal logit and multinomial logit models (see Section 4.2).

3. Build **matrices** (or two-dimensional arrays) from vectors (one-dimensional arrays). You can create a matrix in two ways:

   (a) From a vector: Use the command `matrix(vector, nrow =` k`, ncol =` n`)` to create a $k \times n$ matrix from the vector by filling in the columns from left to right. For example,

```
> matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
      [,1] [,2] [,3]       # Note that when assigning a vector to a
 [1,]    1    3    5       #  matrix, none of the rows or columns
 [2,]    2    4    6       #  have names.
```

(b) From two or more vectors of length k: Use `cbind()` to combine n vectors vertically to form a $k \times n$ matrix, or `rbind()` to combine n vectors horizontally to form a $n \times k$ matrix. For example:

```
> x <- c(11, 12, 13)         # Creates a vector `x' of 3 values.
> y <- c(55, 33, 12)         # Creates another vector `y' of 3 values.
> rbind(x, y)                # Creates a 2 x 3 matrix.  Note that row
    [,1] [,2] [,3]           #  1 is named x and row 2 is named y,
 x   11   12   13            #  according to the order in which the
 y   55   33   12            #  arguments were passed to rbind().
> cbind(x, y)                # Creates a 3 x 2 matrix.  Note that the
       x  y                  #  columns are named according to the
 [1,] 11 55                  #  order in which they were passed to
 [2,] 12 33                  #  cbind().
 [3,] 13 12
```

R supports a variety of matrix functions, including: `det()`, which returns the matrix's determinant; `t()`, which transposes the matrix; `solve()`, which inverts the the matrix; and `%*%`, which multiplies two matricies. In addition, the `dim()` command returns the dimensions of your matrix. As with vectors, square brackets extract specific values from a matrix and the assignment mechanism `<-` replaces values. For example:

```
> loo[,3]         # Extracts the third column of loo.
> loo[1,]          # Extracts the first row of loo.
> loo[1,3] <- 13          # Inserts 13 as the value for row 1, column 3.
> loo[1,] <- c(2,2,3)     # Replaces the first row of loo.
```

If you encounter problems replacing rows or columns, make sure that the `dims()` of the vector matches the `dims()` of the matrix you are trying to replace.

4. An **n-dimensional array** is a set of stacked matrices of identical dimensions. For example, you may create a three dimensional array with dimensions (x, y, z) by stacking z matrices each with x rows and y columns.

```
> a <- matrix(8, 2, 3)       # Creates a 2 x 3 matrix populated with 8's.
> b <- matrix(9, 2, 3)       # Creates a 2 x 3 matrix populated with 9's.
> array(c(a, b), c(2, 3, 2)) # Creates a 2 x 3 x 2 array with the first
   , , 1                     #  level [,,1] populated with matrix a (8's),
                             #  and the second level [,,2] populated
```

```
     [,1] [,2] [,3]       #  with matrix b (9's).
[1,]    8    8    8
[2,]    8    8    8       # Use square brackets to extract values.  For
                          #  example, [1, 2, 2] extracts the second
, , 2                     #  value in the first row of the second level.
                          # You may also use the <- operator to
     [,1] [,2] [,3]       #  replace values.
[1,]    9    9    9
[2,]    9    9    9
```

If an array is a one-dimensional vector or two-dimensional matrix, R will treat the array using the more specific method.

Three functions especially helpful for arrays:

- `is()` returns both the type of scalar value that populates the array, as well as the specific type of array (vector, matrix, or array more generally).
- `dims()` returns the size of an array, where

  ```
  > dims(b)
   [1]  33  5
  ```

  indicates that the array is two-dimensional (a matrix), and has 33 rows and 5 columns.
- The single bracket `[ ]` indicates specific values in the array. Use commas to indicate the index of the specific values you would like to pull out or replace:

  ```
  > dims(a)
   [1]  14
  > a[10]       # Pull out the 10th value in the vector `a'
  > dims(b)
   [1]  33  5
  > b[1:12, ]   # Pull out the first 12 rows of `b'
  > c[1, 2]     # Pull out the value in the first row, second column of `c'
  > dims(d)
   [1]  1000  4  5
  > d[ , 3, 1]  # Pulls out a vector of 1,000 values
  ```

6.2.2 Lists

Unlike arrays, which contain only one type of scalar value, lists are flexible data structures that can contain heterogeneous value types and heterogeneous data structures. Lists are so flexible that one list can contain another list. For example, the list `output` can contain `coef`, a vector of regression coefficients; `variance`, the variance-covariance matrix; and another list `terms` that describes the data using character strings. Use the `names()` function to view the named elements in a list, and to extract a named element, use

```
> names(output)
 [1] coefficients   variance   terms
> output$coefficients
```

For lists where the elements are not named, use double square brackets `[[ ]]` to extract elements:

```
> L[[4]]      # Extracts the 4th element from the list `L'
> L[[4]] <- b # Replaces the 4th element of the list `L' with a matrix `b'
```

Like vectors, lists are flexible data structures that can be extended without first creating another list of with the correct number of elements:

```
> L <- list()                        # Creates an empty list
> L$coefficients <- c(1, 4, 6, 8)  # Inserts a vector into the list, and
                                     #  names that vector `coefficients'
   #  within the list
> L[[4]] <- c(1, 4, 6, 8)          # Inserts the vector into the 4th position
                                     #  in the list.  If this list doesn't
                                     #  already have 4 elements, the empty
                                     #  elements will be `NULL' values
```

Alternatively, you can easily create a list using objects that already exist in your workspace:

```
> L <- list(coefficients = k, variance = v) # Where `k' is a vector and
                                            #   `v' is a matrix
```

6.2.3 Data Frames

A data frame (or data set) is a special type of list in which each variable is constrained to have the same number of observations. A data frame may contain variables of different types (numeric, integer, logical, character, and factor), so long as each variable has the same number of observations.

Thus, a data frame can use both matrix commands and list commands to manipulate variables and observations.

```
> dat[1:10,]          # Extracts observations 1-10 and all associated variables
> dat[dat$grp == 1,] # Extracts all observations that belong to group 1
> group <- dat$grp   # Saves the variable `grp' as a vector `group' in
                      #   the workspace, not in the data frame
> var4 <- dat[[4]]   # Saves the 4th variable as a `var4' in the workspace
```

For a comprehensive introduction to data frames and recoding data, see Section 3.2.2.

6.2.4 Identifying Objects and Data Structures

Each data structure has several *attributes* which describe it. Although these attributes are normally invisible to users (e.g., not printed to the screen when one types the name of the object), there are several helpful functions that display particular attributes:

- For arrays, `dims()` returns the size of each dimension.
- For arrays, `is()` returns the scalar value type and specific type of array (vector, matrix, array). For more complex data structures, `is()` returns the default methods (classes) for that object.
- For lists and data frames, `names()` returns the variable names, and `str()` returns the variable names and a short description of each element.

For almost all data types, you may use `summary()` to get summary statistics.

Chapter 7

Programming Statements

This chapter introduces the main programming commands. These include functions, if-else statements, for-loops, and special procedures for managing the inputs to statistical models.

7.1 Functions

Functions are either built-in or user-defined sets of encapsulated commands which may take any number of arguments. Preface a function with the `function` statement and use the `<-` operator to assign functions to objects in your workspace.

You may use functions to run the same procedure on different objects in your workspace. For example,

```
check <- function(p, q) {
 result <- (p - q)/q
 result
 }
```

is a simple function with arguments `p` and `q` which calculates the difference between the ith elements of the vector `p` and the ith element of the vector `q` as a proportion of the ith element of `q`, and returns the resulting vector. For example, `check(p = 10, q = 2)` returns 4. You may omit the descriptors as long as you keep the arguments in the correct order: `check(10, 2)` also returns 4. You may also use other objects as inputs to the function. If `again = 10` and `really = 2`, then `check(p = again, q = really)` and `check(again, really)` also returns 4.

Because functions run commands as a set, you should make sure that each command in your function works by testing each line of the function at the R prompt.

7.2 If-Statements

Use `if` (and optionally, `else`) to control the flow of R functions. For example, let `x` and `y` be scalar numerical values:

```
if (x == y) {                  # If the logical statement in the ()'s is true,
  x <- NA                      #  then `x' is changed to `NA' (missing value).
}
else {                         # The `else' statement tells R what to do if
  x <- x^2                     #  the if-statement is false.
}
```

As with a function, use `{` and `}` to define the set of commands associated with each if and else statement. (If you include if statements inside functions, you may have multiple sets of nested curly braces.)

7.3 For-Loops

Use `for` to repeat (loop) operations. Avoiding loops by using matrix or vector commands is usually faster and more elegant, but loops are sometimes necessary to assign values. If you are using a loop to assign values to a data structure, you must first initialize an empty data structure to hold the values you are assigning.

Select a data structure compatible with the type of output your loop will generate. If your loop generates a scalar, store it in a vector (with the ith value in the vector corresponding to the the ith run of the loop). If your loop generates vector output, store them as rows (or columns) in a matrix, where the ith row (or column) corresponds to the ith iteration of the loop. If your output consists of matrices, stack them into an array. For list output (such as regression output) or output that changes dimensions in each iteration, use a list. To initialize these data structures, use:

```
> x <- vector()                         # An empty vector of any length.
> x <- list()                           # An empty list of any length.
```

The `vector()` and `list()` commands create a vector or list of any length, such that assigning `x[5] <- 15` automatically creates a vector with 5 elements, the first four of which are empty values (`NA`). In contrast, the `matrix()` and `array()` commands create data structures that are restricted to their original dimensions.

```
> x <- matrix(nrow = 5, ncol = 2)  # A matrix with 5 rows and 2 columns.
> x <- array(dim = c(5,2,3))       # A 3D array of 3 stacked 5 by 2 matrices.
```

If you attempt to assign a value at $(100, 200, 20)$ to either of these data structures, R will return an error message ("subscript is out of bounds"). R does not automatically extend the dimensions of either a matrix or an array to accommodate additional values.

Example 1: Creating a vector with a logical statement

```
x <- array()             # Initializes an empty data structure.
for (i in 1:10) {        # Loops through every value from 1 to 10, replacing
```

```
  if (is.integer(i/2)) { #  the even values in `x' with i+5.
    x[i] <- i + 5
  }
}                        # Enclose multiple commands in {}.
```

You may use `for()` inside or outside of functions.

Example 2: Creating dummy variables by hand You may also use a loop to create a matrix of dummy variables to append to a data frame. For example, to generate fixed effects for each state, let's say that you have `mydata` which contains `y`, `x1`, `x2`, `x3`, and `state`, with `state` a character variable with 50 unique values. There are three ways to create dummy variables: 1) with a built-in R command; 2) with one loop; or 3) with 2 for loops.

1. R will create dummy variables on the fly from a single variable with distinct values.

   ```
   > z.out <- zelig(y ~ x1 + x2 + x3 + as.factor(state),
                    data = mydata, model = "ls")
   ```

   This method returns $k-1$ indicators for k states.

2. Alternatively, you can use a loop to create dummy variables by hand. There are two ways to do this, but both start with the same initial commands. Using vector commands, first create an index of for the states, and initialize a matrix to hold the dummy variables:

   ```
   idx <- sort(unique(mydata$state))
   dummy <- matrix(NA, nrow = nrow(mydata), ncol = length(idx))
   ```

   Now choose between the two methods.

   (a) The first method is computationally inefficient, but more intuitive for users not accustomed to vector operations. The first loop uses `i` as in index to loop through all the rows, and the second loop uses `j` to loop through all 50 values in the vector `idx`, which correspond to columns 1 through 50 in the matrix `dummy`.

   ```
   for (i in 1:nrow(mydata)) {
     for (j in 1:length(idx)) {
       if (mydata$state[i,j] == idx[j]) {
         dummy[i,j] <- 1
       }
       else {
         dummy[i,j] <- 0
       }
     }
   }
   ```

Then add the new matrix of dummy variables to your data frame:

```
names(dummy) <- idx
mydata <- cbind(mydata, dummy)
```

(b) As you become more comfortable with vector operations, you can replace the double loop procedure above with one loop:

```
for (j in 1:length(idx)) {
  dummy[,j] <- as.integer(mydata$state == idx[j])
}
```

The single loop procedure evaluates each element in `idx` against the vector `mydata$state`. This creates a vector of n `TRUE`/`FALSE` observations, which you may transform to `1`'s and `0`'s using `as.integer()`. Assign the resulting vector to the appropriate column in `dummy`. Combine the `dummy` matrix with the data frame as above to complete the procedure.

Example 3: Weighted regression with subsets Selecting the `by` option in `zelig()` partitions the data frame and then automatically loops the specified model through each partition. Suppose that `mydata` is a data frame with variables `y`, `x1`, `x2`, `x3`, and `state`, with `state` a factor variable with 50 unique values. Let's say that you would like to run a weighted regression where each observation is weighted by the inverse of the standard error on `x1`, estimated for that observation's state. In other words, we need to first estimate the model for each of the 50 states, calculate 1 / $\textsc{se}(\texttt{x1}_j)$ for each state $j = 1, \ldots, 50$, and then assign these weights to each observation in `mydata`.

- Estimate the model separate for each state using the `by` option in `zelig()`:

  ```
  z.out <- zelig(y ~ x1 + x2 + x3, by = "state", data = mydata, model = "ls")
  ```

  Now `z.out` is a list of 50 regression outputs.

- Extract the standard error on `x1` for each of the state level regressions.

  ```
  se <- array()                           # Initalize the empty data structure.
  for (i in 1:50) {                       # vcov() creates the variance matrix
    se[i] <- sqrt(vcov(z.out[[i]])[2,2]) # Since we have an intercept, the 2nd
  }                                       # diagonal value corresponds to x1.
  ```

- Create the vector of weights.

  ```
  wts <- 1 / se
  ```

  This vector `wts` has 50 values that correspond to the 50 sets of state-level regression output in `z.out`.

- To assign the vector of weights to each observation, we need to match each observation's state designation to the appropriate state. For simplicity, assume that the states are numbered 1 through 50.

  ```
  mydata$w <- NA            # Initalizing the empty variable
  for (i in 1:50) {
    mydata$w[mydata$state == i] <- wts[i]
  }
  ```

  We use `mydata$state` as the index (inside the square brackets) to assign values to `mydata$w`. Thus, whenever state equals 5 for an observation, the loop assigns the fifth value in the vector `wts` to the variable `w` in `mydata`. If we had 500 observations in `mydata`, we could use this method to match each of the 500 observations to the appropriate `wts`.

  If the states are character strings instead of integers, we can use a slightly more complex version

  ```
  mydata$w <- NA
  idx <- sort(unique(mydata$state))
  for (i in 1:length(idx) {
    mydata$w[mydata$state == idx[i]] <- wts[i]
  }
  ```

- Now we can run our weighted regression:

  ```
  z.wtd <- zelig(y ~ x1 + x2 + x3, weights = w, data = mydata,
                 model = "ls")
  ```

Chapter 8

Writing New Models

With Zelig, writing a new model in R is straightforward. (If you already have a model, see Chapter 9 for how to include it in Zelig.) With tools to streamline user inputs, writing a new model does not require a lot of programming knowledge, but lets developers focus on the model's math. Generally, writing a new statistical procedure or model comes in orderly steps:

1. Write down the mathematical model. Define the parameters that you need, grouping parameters into convenient vectors or matrices whenever possible (this will make your code clearer).

2. Write the code.

3. Test the code (usually using Monte Carlo data, where you know the true values being estimated) and make sure that it works as expected.

4. Write some documentation explaining your model and the functions that run your model.

Somewhere between steps [1] and [2], you will need to translate input data into the mathematical notation that you used to write down the model. Rather than repeating whole blocks of code, use functions to streamline the number of commands that users will need to run your model.

With more steps being performed by fewer commands, the inputs to these commands become more sophisticated. The structure of those inputs actually matters quite a lot. If your function has a convoluted syntax, it will be difficult to use, difficult to explain, and difficult to document. If your function is easy to use and has an intuitive syntax, however, it will be easy to explain and document, which will make your procedure more accessible to all users.

8.1 Managing Statistical Model Inputs

Most statistical models require a matrix of explanatory variables and a matrix of dependent variables. Rather than have users create matrices themselves, R has a convenient user interface to create matrices of response and explanatory variables on the fly. Users simply specify a `formula` in the form of `dependent ~ explanatory variables`, and developers use the following functions to transform the formula into the appropriate matrices. Let `mydata` be a data frame.

```
> formula <- y ~ x1 + x2                     # User input

# Given the formula above, programmers can use the following standard commands
> D <- model.frame(formula, data = mydata) # Subset & listwise deletion
> X <- model.matrix(formula, data = D)     # Creates X matrix
> Y <- model.response(D)                   # Creates Y matrix
```

where

- `D` is a subset of `mydata` that contains only the variables specified in the formula (`y`, `x1`, and `x2`) with listwise deletion performed on the subset data frame;
- `X` is a matrix that contains a column of 1's, and the explanatory variables `x1` and `x2` from `D`; and
- `Y` is a matrix containing the dependent variable(s) from `D`.

Depending on the model, `Y` may be a column vector, matrix, or other data structure.

8.1.1 Describe the Statistical Model

After setting up the X matrix, the next step for most models will be to identify the corresponding vector of parameters. For a single response variable model with no ancillary parameters, the standard R interface is quite convenient: given X, the model's parameters are simply β.

There are very few models, however, that fall into this category. Even Normal regression, for example, has two sets of parameters β and σ^2. In order to make the R formula format more flexible, Zelig has an additional set of tools that lets you describe the inputs to your model (for multiple sets of parameters).

After you have written down the statistical model, identify the parameters in your model. With these parameters in mind, the first step is to write a `describe.*()` function for your model. If your model is called `mymodel`, then the `describe.mymodel()` function takes no arguments and returns a list with the following information:

- `category`: a character string that describes the dependent variable. See Section **??** for the current list of available categories.

- `parameters`: a list containing parameter sets used in your model. For each parameter (e.g., theta), you need to provide the following information:
  - `equations`: an integer number of equations for the parameter. For parameters that can take, for example, two to four equations, use `c(2, 4)`.
  - `tagsAllowed`: a logical value (`TRUE`/`FALSE`) specifying whether a given parameter allows constraints.
  - `depVar`: a logical value (`TRUE`/`FALSE`) specifying whether a parameter requires a corresponding dependent variable.
  - `expVar`: a logical value (`TRUE`/`FALSE`) specifying whether a parameter allows explanatory variables.

(See Section **??** for examples and additional arguments output by `describe.mymodel()`.)

8.1.2 Single Response Variable Models: Normal Regression Model

Let's say that you are trying to write a Normal regression model with stochastic component

$$\text{Normal}(y_i \mid \mu_i, \sigma^2) \;=\; \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\left(\frac{(y_i-\mu_i)^2}{2\sigma^2}\right)\right)$$

with scalar variance parameter $\sigma^2 > 0$, and systematic component $E(Y_i) = \mu_i = x_i\beta$. This implies two sets of parameters in your model, and the following `describe.normal.regression()` function:

```
describe.normal.regression <- function() {
  category <- "continuous"
  mu <- list(equations = 1,              # Systematic component
             tagsAllowed = FALSE,
             depVar = TRUE,
             expVar = TRUE)
  sigma2 <- list(equations = 1,          # Scalar ancillary parameter
                 tagsAllowed = FALSE,
                 depVar = FALSE,
                 expVar = FALSE)
  pars <- list(mu = mu, sigma2 = sigma2)
  list(category = category, parameters = pars)
}
```

To find the log-likelihood:

$$
\begin{aligned}
\mathrm{L}\ (\beta, \sigma^2 \mid y) &= \prod_{1=1}^{n} \mathrm{Normal}(y_i \mid \mu_i, \sigma^2) \\
&= \prod_{i=1}^{n} (2\pi\sigma^2)^{-1/2} \exp\left(\frac{-(y_i - \mu_i)^2}{2\sigma^2}\right) \\
&= (2\pi\sigma^2)^{-n/2} \prod_{i=1}^{n} \exp\left(\frac{-(y_i - \mu_i)^2}{2\sigma^2}\right) \\
&= (2\pi\sigma^2)^{-n/2} \prod_{i=1}^{n} \exp\left(\frac{-(y_i - x_i\beta)^2}{2\sigma^2}\right) \\
\ln \mathrm{L}\ (\beta, \sigma^2 \mid y) &= -\frac{n}{2} \ln(2\pi\sigma^2) - \sum_{i=1}^{n} \frac{(y_i - x_i\beta)^2}{2\sigma^2} \\
&= -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - x_i\beta)^2 \\
&\propto -\frac{1}{2}\left(n \ln \sigma^2 + \frac{\sum_{i=1}^{n}(y_i - x_i\beta)^2}{\sigma^2}\right)
\end{aligned}
$$

In R code, this translates to:

```
ll.normal <- function(par, X, Y, n, terms) {
  beta <- parse.par(par, terms, eqn = "mu")              # [1]
  gamma <- parse.par(par, terms, eqn = "sigma2")         # [2]
  sigma2 <- exp(gamma)
  -0.5 * (n * log(sigma2) + sum((Y - X %*% beta)^2 / sigma2))
}
```

At Comment [1] above, we use the function `parse.par()` to pull out the vector of parameters `beta` (which relate the systematic component μ_i to the explanatory variables x_i). No matter how many covariates there are, the `parse.par()` function can use `terms` to pull out the appropriate parameters from `par`. We also use `parse.par()` at Comment [2] to pull out the scalar ancillary parameter that (after transformation) corresponds to the σ^2 parameter.

To optimize this function, simply type:

```
out <- optim(start.val, ll.normal, control = list(fnscale = -1),
             method = "BFGS", hessian = TRUE, X = X, Y = Y, terms = terms)
```

where

- `start.val` is a vector of starting values for `par`. Use `set.start()` to create starting values for all parameters, systematic and ancillary, in one step.
- `ll.normal` is the log-likelihood function derived above.

- "BFGS" specifies unconstrained optimization using a quasi-Newton method.
- control = list(fnscale = -1) specifies that R should maximize the function (omitting this causes R to minimize the function by default).
- hessian = TRUE instructs R to return the Hessian matrix (from which you may calculate the variance-covariance matrix).
- X and Y are the matrix of explanatory variables and vector of dependent variables, used in the ll.normal() function.
- terms are meta-data constructed from the model.frame() command.

Please refer to the R-help for optim() for more options.

To make this procedure generalizable, we can write a function that takes a user-specified data frame and formula, and optional starting values for the optimization procedure:

```
normal.regression <- function(formula, data, start.val = NULL, ...) {

  fml <- parse.formula(formula, model = "normal.regression") # [1]
  D <- model.frame(fml, data = data)
  X <- model.matrix(fml, data = D)
  Y <- model.response(D)
  terms <- attr(D, "terms")
  n <- nrow(X)

  start.val <- set.start(start.val, terms)

  res <- optim(start.val, ll.normal, method = "BFGS",
               hessian = TRUE, control = list(fnscale = -1),
               X = X, Y = Y, n = n, terms = terms, ...)      # [2]

  fit <- model.end(res, D)                                  # [3]
  fit$n <- n
  class(fit) <- "normal"                                    # [4]
  fit
}
```

The following comments correspond to the bracketed numbers above:

1. The parse.formula() command looks for the describe.normal.regression() function, which changes the user-specified formula into the following format:

   ```
   list(mu = formula,         # where `formula' was specified by the user
        sigma = ~ 1)
   ```

2. The `...` here indicate that if the user enters any additional arguments when calling `normal.regression()`, that those arguments should go to the `optim()` function.

3. The `model.end()` function takes the optimized output and the listwise deleted data frame `D` and creates an object that will work with `setx()`.

4. Choose a class for your model output so that you will be able to write an appropriate `summary()`, `param()`, and `qi()` function for your model.

8.1.3 Multivariate models: Bivariate Normal example

Most common models have one systematic component. For n observations, the systematic component varies over observations $i = 1, \ldots, n$. In the case of the Normal regression model, the systematic component is μ_i (σ^2 is not estimated as a function of covariates).

In some cases, however, your model may have more than one systematic component. In the case of bivariate probit, we have a dependent variable $Y_i = (Y_{i1}, Y_{i2})$ observed as (0,0), (1,0), (0,1), or (1,1) for $i = 1, \ldots, n$. Similar to a single-response probit model, the stochastic component is described by two latent (unobserved) continuous variables (Y^*_{i1}, Y^*_{i2}) which follow the bivariate Normal distribution:

$$\begin{pmatrix} Y^*_{i1} \\ Y^*_{i2} \end{pmatrix} \sim \text{Normal}\left\{ \begin{pmatrix} \mu_{i1} \\ \mu_{i2} \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right\},$$

where for $j = 1, 2$, μ_{ij} is the mean for Y^*_{ij} and ρ is a correlation parameter. The following observation mechanism links the observed dependent variables, Y_{ij}, with these latent variables

$$Y_{ij} = \begin{cases} 1 & \text{if } Y^*_{ij} \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

The systemic components for each observation are

$$\begin{aligned} \mu_{ij} &= x_{ij}\beta_j \quad \text{for} \quad j = 1, 2, \\ \rho &= \frac{\exp(x_{i3}\beta_3) - 1}{\exp(x_{i3}\beta_3) + 1}. \end{aligned}$$

In the default specification, ρ is a scalar (such that x_{i3} only contains an intercept term).

If so, we have two sets of parameters: $\mu_i = (\mu_{i1}, \mu_{i2})$ and ρ. This implies the following `describe.bivariate.probit()` function:

```
describe.bivariate.probit <- function() {
  category <- "dichotomous"
  package <- list(name = "mvtnorm",       # Required package and
                  version = "0.7")        #  minimum version number
  mu <- list(equations = 2,               # Systematic component has 2
             tagsAllowed = TRUE,          #  required equations
```

```
             depVar = TRUE,
             expVar = TRUE),
  rho <- list(equations = 1,                # Optional systematic component
             tagsAllowed = FALSE,           #   (estimated as an ancillary
             depVar = FALSE,                #    parameter by default)
             expVar = TRUE),
  pars <- parameters(mu = mu, rho = rho)
  list(category = category, package = package, parameters = pars)
}
```

Since users may choose different explanatory variables to parameterize μ_{i1} and μ_{i2} (and sometimes ρ), the model requires a minimum of *two* formulas. For example,

```
formulae <- list(mu1 = y1 ~ x1 + x2,                             # User input
                 mu2 = y2 ~ x2 + x3)
fml <- parse.formula(formulae, model = "bivariate.probit")   # [1]
D <- model.frame(fml, data = mydata)
X <- model.matrix(fml, data = D)
Y <- model.response(D)
```

At comment [1], `parse.formula()` finds the `describe.bivariate.probit()` function and parses the formulas accordingly.

If ρ takes covariates (and becomes a systematic component rather than an ancillary parameter), there can be three sets of explanatory variables:

```
formulae <- list(mu1 = y1 ~ x1 + x2,
                 mu2 = y2 ~ x2 + x3,
                 rho = ~ x4 + x5)
```

From the perspective of the programmer, a nearly identical framework works for both single and multiple equation models. The (`parse.formula()`) line changes the class of `fml` from `"list"` to `"multiple"` and hence ensures that `model.frame()` and `model.matrix()` go to the appropriate methods. `D`, `X` , and `Y` are analogous to their single equation counterparts above:

- `D` is the subset of `mydata` containing the variables `y1`, `y2`, `x1`, `x2`, and `x3` with listwise deletion performed on the subset;
- `X` is a matrix corresponding to the explanatory variables, in one of three forms discussed below (see Section 8.2).
- `Y` is an $n \times J$ matrix (where $J = 2$ here) with columns (`y1`, `y2`) corresponding to the outcome variables on the left-hand sides of the formulas.

Given for the bivariate probit probability density described above, the likelihood is:

$$L(\pi|Y_i) = \prod_{i=1}^{n} \pi_{00}^{\mathrm{I}\{Y_i=(0,0)\}} \pi_{10}^{\mathrm{I}\{Y_i=(1,0)\}} \pi_{01}^{\mathrm{I}\{Y_i=(0,1)\}} \pi_{11}^{\mathrm{I}\{Y_i=(1,1)\}}$$

where I is an indicator function and

- $\pi_{00} = \int_{-\infty}^{0} \int_{-\infty}^{0} \text{Normal}(Y_{i1}^*, Y_{i2}^* \mid \mu_{i1}, \mu_{i2}, \rho) dY_{i2}^* dY_{i1}^*$
- $\pi_{10} = \int_{0}^{\infty} \int_{-\infty}^{0} \text{Normal}(Y_{i1}^*, Y_{i2}^* \mid \mu_{i1}, \mu_{i2}, \rho) dY_{i2}^* dY_{i1}^*$
- $\pi_{01} = \int_{-\infty}^{0} \int_{0}^{\infty} \text{Normal}(Y_{i1}^*, Y_{i2}^* \mid \mu_{i1}, \mu_{i2}, \rho) dY_{i2}^* dY_{i1}^*$
- $\pi_{11} = 1 - \pi_{00} - \pi_{10} - \pi_{01}$

This implies the following log-likelihood:

$$\begin{aligned}\log L(\pi|Y_i) &= \sum_{i=1}^{n} \mathrm{I}\{Y_i = (0,0)\} \log \pi_{00} + \mathrm{I}\{Y_i = (1,0)\} \log \pi_{10} \\ &\quad + \mathrm{I}\{Y_i = (0,1)\} \log \pi_{01} + \mathrm{I}\{Y_i = (1,1)\} \log \pi_{11}\end{aligned}$$

(For the corresponding R code, see Section 8.2.4 below.)

8.2 Easy Ways to Manage Matrices

Most statistical methods relate explanatory variables x_i to a dependent variable of interest y_i for each observation $i = 1, \ldots, n$. Let β be a set of parameters that correspond to each column in X, which is an $n \times k$ matrix with rows x_i. For a single equation model, the linear predictor is

$$\eta_i = x_i \beta = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik}$$

Thus, η is the set of η_i for $i = 1, \ldots, n$ and is usually represented as an $n \times 1$ matrix.

For a two equation model such as bivariate probit, the linear predictor becomes a matrix with columns corresponding to each dependent variable (y_{1i}, y_{2i}):

$$\eta_i = (\eta_{i1}, \eta_{i2}) = (x_{i1}\beta_1, x_{i2}\beta_2)$$

With η as an $n \times 2$ matrix, we now have a few choices as to how to create the linear predictor:

1. An **intuitive** layout, which stacks matrices of explanatory variables, provides an easy visual representation of the relationship between explanatory variables and coefficients;

2. A **computationally-efficient** layout, which takes advantage of computational vectorization; and

3. A **memory-saving** layout, which reduces the overall size of the X and β matrices.

Using the simple tools described in this section, you can pick the best matrix management method for your model.

In addition, the way in which η is created also affects the way parameters are estimated. Let's say that you want two parameters to have the same effect in different equations. By setting up X and β in a certain way, you can let users set constraints across parameters. Continuing the bivariate probit example above, let the model specification be:

```
formulae <- list(mu1 = y1 ~ x1 + x2 + tag(x3, "land"),
                 mu2 = y2 ~ x3 + tag(x4, "land"))
```

where `tag()` is a special function that constrains variables to have the same effect across equations. Thus, the coefficient for `x3` in equation `mu1` is constrained to be equal to the coefficient for `x4` in equation `mu2`, and this effect is identified as the "land" effect in both equations. In order to consider constraints across equations, the structure of both X and β matter.

8.2.1 The Intuitive Layout

A stacked matrix of X and vector β is probably the most visually intuitive configuration. Let $J = 2$ be the number of equations in the bivariate probit model, and let v_t be the total number of unique covariates in both equations. Choosing `model.matrix(..., shape = "stacked")` yields a $(Jn \times v_t) = (2n \times 6)$ matrix of explanatory variables. Again, let x_1 be an $n \times 1$ vector representing variable `x1`, x_2 `x2`, and so forth. Then

$$X = \begin{pmatrix} 1 & 0 & x_1 & x_2 & 0 & x_3 \\ 0 & 1 & 0 & 0 & x_3 & x_4 \end{pmatrix}$$

Correspondingly, β is a vector with elements

$$(\beta_0^{\mu_1} \ \beta_0^{\mu_2} \ \beta_{x_1}^{\mu_1} \ \beta_{x_2}^{\mu_1} \ \beta_{x_3}^{\mu_2} \ \beta_{\text{land}})'$$

where β_0^j are the intercept terms for equation $j = \{\mu_1, \mu_2\}$. Since X is $(2n \times 6)$ and β is (6×1), the resulting linear predictor η is also stacked into a $(2n \times 1)$ matrix. Although difficult to manipulate (since observations are indexed by i and $2i$ for each $i = 1, \ldots, n$ rather than just i), it is easy to see that we have turned the two equations into one big X matrix and one long vector β, which is directly analogous to the familiar single-equation η.

8.2.2 The Computationally-Efficient Layout

Choosing array X and vector β is probably the the most computationally-efficient configuration: `model.matrix(..., shape = "array")` produces an $n \times k_t \times J$ array where J is the total number of equations and k_t is the total number of parameters across all the equations. Since some parameter values may be constrained across equations, $k_t \leq \sum_{j=1}^{J} k_j$. If a variable is not in a certain equation, it is observed as a vector of 0s. With this option, each $i = 1, \ldots, n$ x_i matrix becomes:

$$\begin{pmatrix} 1 & 0 & x_{i1} & x_{i2} & 0 & x_{i3} \\ 0 & 1 & 0 & 0 & x_{i3} & x_{i4} \end{pmatrix}$$

By stacking each of these x_i matrices along the first dimension, we get X as an array with dimensions $n \times k_t \times J$.

Correspondingly, β is a vector with elements

$$(\beta_0^{\mu_1}\ \beta_0^{\mu_2}\ \beta_{x_1}^{\mu_1}\ \beta_{x_2}^{\mu_1}\ \beta_{x_3}^{\mu_2}\ \beta_{\text{land}})'$$

To multiply the X array with dimensions $(n \times 6 \times 2)$ and the (6×1) β vector, we *vectorize* over equations as follows:

```
eta <- apply(X, 3, '%*%', beta)
```

The linear predictor `eta` is therefore a $(n \times 2)$ matrix.

8.2.3 The Memory-Efficient Layout

Choosing a "compact" X matrix and matrix β is probably the most memory-efficient configuration: `model.matrix(..., shape = "compact")` (the default) produces an $n \times v$ matrix, where v is the number of unique variables (5 in this case)[1] in all of the equations. Let x_1 be an $n \times 1$ vector representing variable `x1`, x_2 `x2`, and so forth.

$$X = (1\ x_1\ x_2\ x_3\ x_4) \qquad \beta = \begin{pmatrix} \beta_0^{\mu_1} & \beta_0^{\mu_2} \\ \beta_{x_1}^{\mu_1} & 0 \\ \beta_{x_2}^{\mu_1} & 0 \\ \beta_{\text{land}} & \beta_{x_3}^{\mu_2} \\ 0 & \beta_{\text{land}} \end{pmatrix}$$

The β_{land} parameter is used twice to implement the constraint, and the number of empty cells is minimized by implementing the constraints in β rather than X. Furthermore, since X is $(n \times 5)$ and β is (5×2), $X\beta = \eta$ is $n \times 2$.

8.2.4 Interchanging the Three Methods

Continuing the bivariate probit example above, we only need to modify a few lines of code to put these different schemes into effect. Using the default (memory-efficient) options, the log-likelihood is:

[1]Why 5? In addition to the intercept term (a variable which is the same in either equation, and so counts only as one variable), the *unique* variables are x_1, x_2, x_3, and x_4.

```
bivariate.probit <- function(formula, data, start.val = NULL, ...) {
  fml <- parse.formula(formula, model = "bivariate.probit")
  D <- model.frame(fml, data = data)
  X <- model.matrix(fml, data = D, eqn = c("mu1", "mu2"))       # [1]
  Xrho <- model.matrix(fml, data = D, eqn = "rho")
  Y <- model.response(D)
  terms <- attr(D, "terms")
  start.val <- set.start(start.val, terms)
  start.val <- put.start(start.val, 1, terms, eqn = "rho")

  log.lik <- function(par, X, Y, terms) {
    Beta <- parse.par(par, terms, eqn = c("mu1", "mu2"))         # [2]
    gamma <- parse.par(par, terms, eqn = "rho")
    rho <- (exp(Xrho %*% gamma) - 1) / (1 + exp(Xrho %*% gamma))
    mu <- X %*% Beta                                             # [3]
    llik <- 0
    for (i in 1:nrow(mu)){
      Sigma <- matrix(c(1, rho[i,], rho[i,], 1), 2, 2)
      if (Y[i,1]==1)
        if (Y[i,2]==1)
          llik <- llik + log(pmvnorm(lower = c(0, 0), upper = c(Inf, Inf),
                                     mean = mu[i,], corr = Sigma))
        else
          llik <- llik + log(pmvnorm(lower = c(0, -Inf), upper = c(Inf, 0),
                                     mean = mu[i,], corr = Sigma))
      else
        if (Y[i,2]==1)
          llik <- llik + log(pmvnorm(lower = c(-Inf, 0), upper = c(0, Inf),
                                     mean = mu[i,], corr = Sigma))
        else
          llik <- llik + log(pmvnorm(lower = c(-Inf, -Inf), upper = c(0, 0),
                                     mean = mu[i,], corr = Sigma))
        }
    return(llik)
  }
  res <- optim(start.val, log.lik, method = "BFGS",
               hessian = TRUE, control = list(fnscale = -1),
               X = X, Y = Y, terms = terms, ...)
  fit <- model.end(res, D)
  class(fit) <- "bivariate.probit"
  fit
}
```

If you find that the default (memory-efficient) method isn't the best way to run your model, you can use either the intuitive option or the computationally-efficient option by changing just a few lines of code as follows:

- **Intuitive option** At Comment [1]:

  ```
  X <- model.matrix(fml, data = D, shape = "stacked", eqn = c("mu1", "mu2"))
  ```

  and at Comment [2],

  ```
  Beta <- parse.par(par, terms, shape = "vector", eqn = c("mu1", "mu2"))
  ```

  The line at Comment [3] remains the same as in the original version.

- **Computationally-efficient option** Replace the line at Comment [1] with

  ```
  X <- model.matrix(fml, data = D, shape = "array", eqn = c("mu1", "mu2"))
  ```

  At Comment [2]:

  ```
  Beta <- parse.par(par, terms, shape = "vector", eqn = c("mu1", "mu2"))
  ```

  At Comment [3]:

  ```
  mu <- apply(X, 3, '%*%', Beta)
  ```

Even if your optimizer calls a C or FORTRAN routine, you can use combinations of `model.matrix()` and `parse.par()` to set up the data structures that you need to obtain the linear predictor (or your model's equivalent) before passing these data structures to your optimization routine.

Chapter 9

Adding Models and Methods to Zelig

Zelig is highly modular. You can add methods to Zelig *and*, if you wish, release your programs as a stand-alone package. By making your package compatible with Zelig, you will advertise your package and help it achieve a widespread distribution.

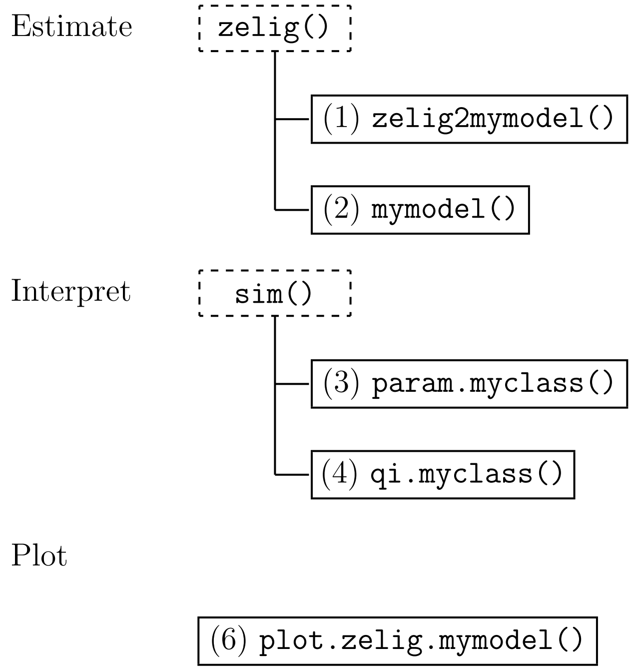
This chapter assumes that your model is written as a function that takes a user-defined formula and data set (see Chapter 8), and returns a list of output that includes (at the very least) the estimated parameters and terms that describe the data used to fit the model. You should choose a class (either S3 or S4 class) for this list of output, and provide appropriate methods for generic functions such as `summary()`, `print()`, `coef()` and `vcov()`.

To add new models to Zelig, you need to provide six R functions, illustrated in Figure 9.1. Let `mymodel` be a new model with class `"myclass"`.

These functions are as follows:

1. `zelig2mymodel()` translates `zelig()` arguments into the arguments for `mymodel()`.
2. `mymodel()` estimates your statistical procedure.
3. `param.myclass()` simulates parameters for your model. Alternatively, if your model's parameters consist of one vector with a correspondingly observed variance-covariance matrix, you may write *two* simple functions to substitute for `param.myclass()`:
   (a) `coef.myclass()` to extract the coefficients from your model output, and
   (b) `vcov.myclass()` to extract the variance-covariance matrix from your model.
4. `qi.myclass()` calculates expected values, simulates predicted values, and generates other quantities of interest for your model (applicable only to models that take explanatory variables).
5. `plot.zelig.mymodel()` to plot the simulated quantities of interest from your model.
6. A **reference manual page** to document the model. (See Section 9.3)
7. A function (`describe.mymodel()`) describing the inputs to your model, for use with a graphical user interface. (See Section **??**).

Figure 9.1: Six functions (solid boxes) to implement a new Zelig model



8. An optional **demo script** `mymodel.R` which contains commented code for the models contained in the example section of your reference manual page.

9.1 Making the Model Compatible with Zelig

You can develop a model, write the model-fitting function, and test it within the Zelig framework without explicit intervention from the Zelig team. (We are, of course, happy to respond to any questions or suggestions for improvement.)

Zelig's modularity relies on two R programming conventions:

1. **wrappers**, which pass arguments from R functions to other R functions or to foreign function calls (such as C, C++, or Fortran functions); and

2. **classes**, which tell generic functions how to handle objects of a given class.

Specific methods for R generic functions take the general form: `method.class()`, where `method` is the name of the generic procedure to be performed and `class` is the class of the object. You may define, for example, `summary.contrib()` to summarize the output of your model. Note that for S4 classes, the name of generic functions does not have to be `method.class()` so long as users can call them via `method()`.

To Work with `zelig()`

Zelig has implemented a unique method for incorporating new models which lets contributors test their models *within* the Zelig framework *without* any modification of the `zelig()` function itself.

Using a wrapper function `zelig2contrib()` (where `contrib` is the name of your new model), `zelig2contrib()` redefines the inputs to `zelig()` to work with the inputs you need for your function `contrib()`. For example, if you type

```
zelig(..., model = "normal.regression")
```

`zelig()` looks for a `zelig2normal.regression()` wrapper in any environment (either attached libraries or your workspace). If the wrapper exists, then `zelig()` runs the model.

If you have a pre-existing model, writing a `zelig2contrib()` function is quite easy. Let's say that your model is `contrib()`, and takes the following arguments: `formula`, `data`, `weights`, and `start`. The `zelig()` function, in contrast, only takes the `formula`, `data`, `model`, and `by` arguments. You may use the `...` to pass additional arguments from `zelig()` to `zelig2contrib()`, and `<- NULL` to omit the elements you do not need. Continuing the Normal regression example from Section 8.1.2, let `formula`, `model`, and `data` be the inputs to `zelig()`, `M` is the number of subsets, and `...` are the additional arguments not defined in the `zelig()` call, but passed to `normal.regression()`.

```
zelig2normal.regression <- function(formula, model, data, M, ...) {
  mf <- match.call(expand.dots = TRUE)                    # [1]
  mf$model <- mf$M <- NULL                                # [2]
  mf[[1]] <- as.name("normal.regression")                 # [3]
  as.call(mf)                                             # [4]
}
```

The bracketed numbers above correspond to the comments below:

1. Create a call (an expression to be evaluated) by creating a list of the arguments in `zelig2normal.regression()`, including the extra arguments taken by `normal.regression()`, but not by `zelig()`. All wrappers must take the same standardized arguments (`formula`, `model`, `data`, and `M`), which may be used in the wrapper function to manipulate the `zelig()` call into the `normal.regression()` call. Additional arguments to `normal.regression()`, such as `start.val` are passed implicitly from `zelig()` using the `...` operator.

2. Erase extraneous information from the call object `mf`. In this wrapper, `model` and `M` are not used. In other models, these are used to further manipulate the call, and so are included in the standard inputs to all wrappers.

3. Reassign the first element of the call (currently `zelig2normal.regression`) with the name of the function to be evaluated, `normal.regression()`.

4. Return the call to `zelig()`, which will evaluate the call for each multiply-imputed data set, each subset defined in `by`, or simply `data`.

If you use an S4 class to represent your model, say `mymodel`, within `zelig.default()`, Zelig's internal function, `create.ZeligS4()`, automatically creates a new S4 class called `ZeligS4mymodel` in the global environment with two additional slots. These include `zelig`, which stores the name of the model, and `zelig.data`, which stores the data frame if `save.data=TRUE` and is empty otherwise. These names are taken from the original call. This new output inherits the original class `mymodel` so all the generic functions associated with `mymodel` should still work. If you would like to see an example, see the models implemented using the VGAM package, such as multinomial probit.

To Work with `setx()`

In the case of `setx()`, most models will use `setx.default()`, which in turn relies on the generic R function `model.matrix()`. For this procedure to work, your list of output must include:

- `terms`, created by `model.frame()`, or manually;
- `formula`, the formula object input by the user;
- `xlevels`, which define the strata in the explanatory variables; and
- `contrasts`, an optional element which defines the type of factor variables used in the explanatory variables. See `help(contrasts)` for more information.

If your model output does not work with `setx.default()`, you must write your own `setx.contrib()` function. For example, models fit to multiply-imputed data sets have output from `zelig()` of class `"MI"`. The special `setx.MI()` wrapper pre-processes the `zelig()` output object and passes the appropriate arguments to `setx.default()`.

Compatibility with `sim()`

Simulating quantities of interest is an integral part of interpreting model results. To use the functionality built into the Zelig `sim()` procedure, you need to provide a way to simulate parameters (called a `param()` function), and a method for calculating or drawing quantities of interest from the simulated parameters (called a `qi()` function).

Simulating Parameters Whether you choose to use the default method, or write a model-specific method for simulating parameters, these functions require the same three inputs:

- `object`: the estimated model or `zelig()` output.
- `num`: the number of simulations.
- `bootstrap`: either `TRUE` or `FALSE`.

The output from `param()` should be either

- If `bootstrap = FALSE` (default), an matrix with rows corresponding to simulations and columns corresponding to model parameters. Any ancillary parameters should be included in the output matrix.
- If `bootstrap = TRUE`, a vector containing all model parameters, including ancillary parameters.

There are two ways to simulate parameters:

1. Use the `param.default()` function to extract parameters from the model and, if bootstrapping is not selected, simulate coefficients using asymptotic normal approximation. The `param.default()` function relies on two R functions:

   (a) `coef()`: extracts the coefficients. Continuing the Normal regression example from above, the appropriate `coef.normal()` function is simply:

   ```
   coef.normal <- function(object)
     object$coefficients
   ```

   (b) `vcov()`: extracts the variance-covariance matrix. Again continuing the Poisson example from above:

   ```
   vcov.normal <- function(object)
     object$variance
   ```

2. Alternatively, you can write your own `param.contrib()` function. This is appropriate when:

   (a) Your model has auxiliary parameters, such as σ in the case of the Normal distribution.

   (b) Your model performs some sort of correction to the coefficients or the variance-covariance matrix, which cannot be performed in either the `coef.contrib()` or the `vcov.contrib()` functions.

   (c) Your model does not rely on asymptotic approximation to the log-likelihood. For Bayesian Markov-chain monte carlo models, for example, the `param.contrib()` function (`param.MCMCzelig()` in this case) simply extracts the model parameters simulated in the model-fitting function.

   Continuing the Normal example,

   ```
   param.normal <- function(object, num = NULL, bootstrap = FALSE,
                        terms = NULL) {
     if (!bootstrap) {
       par <- mvrnorm(num, mu = coef(object), Sigma = vcov(object))
       Beta <- parse.par(par, terms = terms, eqn = "mu")
   ```

```
    sigma2 <- exp(parse.par(par, terms = terms, eqn = "sigma2"))
    res <- cbind(Beta, sigma2)
  }
  else {
    par <- coef(object)
    Beta <- parse.par(par, terms = terms,  eqn = "mu")
    sigma2 <- exp(parse.par(par, terms = terms, eqn = "sigma2"))
    res <- c(coef, sigma2)
  }
  res
}
```

Calculating Quantities of Interest All models require a model-specific method for calculating quantities of interest from the simulated parameters. For a model of class `contrib`, the appropriate `qi()` function is `qi.contrib()`. This function should calculate, at the bare minimum, the following quantities of interest:

- `ev`: the expected values, calculated from the analytic solution for the expected value as a function of the systematic component and ancillary parameters.
- `pr`: the predicted values, drawn from a distribution defined by the predicted values. If R does not have a built-in random generator for your function, you may take a random draw from the uniform distribution and use the inverse CDF method to calculate predicted values.
- `fd`: first differences in the expected value, calculated by subtracting the expected values given the specified `x` from the expected values given `x1`.
- `ate.ev`: the average treatment effect calculated using the expected values `ev`. This is simply `y - ev`, averaged across simulations for each observation.
- `ate.pr`: the average treatment effect calculated using the predicted values `pr`. This is simply `y - pr`, averaged across simulations for each observation.

The required arguments for the `qi()` function are:

- `object`: the zelig output object.
- `par`: the simulated parameters.
- `x`: the matrix of explanatory variables (created using `setx()`).
- `x1`: the optional matrix of alternative values for first differences (also created using `setx()`). If first differences are inappropriate for your model, you should put in a `warning()` or `stop()` if `x1` is not `NULL`.

- y: the optional vector or matrix of dependent variables (for calculating average treatment effects). If average treatment effects are inappropriate for your model, you should put in a `warning()` or `stop()` if conditional prediction has been selected in the `setx()` step.

Continuing the Normal regression example from above, the appropriate `qi.normal()` function is as follows:

```
qi.normal <- function(object, par, x, x1 = NULL, y = NULL) {
  Beta <- parse.par(par, eqn = "mu")                          # [1]
  sigma2 <- parse.par(par, eqn = "sigma2")                    # [2]
  ev <- Beta %*% t(x)                                         # [3a]
  pr <- matrix(NA, ncol = ncol(ev), nrow = nrow(ev))
  for (i in 1:ncol(ev))
    pr[,i] <- rnorm(length(ev[,i]), mean = ev[,i],            # [4]
                    sigma = sd(sigma2[i]))
  qi <- list(ev = ev, pr = pr)
  qi.name <- list(ev = "Expected Values: E(Y|X)",
                  pr = "Predicted Values: Y|X")
  if (!is.null(x1)){
    ev1 <- par %*% t(x1)                                      # [3b]
    qi$fd <- ev1 - ev
    qi.name$fd <- "First Differences in Expected Values: E(Y|X1)-E(Y|X)"
  }
  if (!is.null(y)) {
    yvar <- matrix(rep(y, nrow(par)), nrow = nrow(par), byrow = TRUE)
    tmp.ev <- yvar - qi$ev
    tmp.pr <- yvar - qi$pr
    qi$ate.ev <- matrix(apply(tmp.ev, 1, mean), nrow = nrow(par))
    qi$ate.pr <- matrix(apply(tmp.pr, 1, mean), nrow = nrow(par))
    qi.name$ate.ev <- "Average Treatment Effect: Y - EV"
    qi.name$ate.pr <- "Average Treatment Effect: Y - PR"
  }
  list(qi=qi, qi.name=qi.name)
}
```

There are five lines of code commented above. By changing these five lines in the following *four* ways, you can write `qi()` function appropriate to almost any model:

1. Extract any systematic parameters by substituting the name of your systematic parameter (defined in `describe.mymodel()`).

2. Extract any ancillary parameters (defined in `describe.mymodel()`) by substituting their names here.

3. Calculate the expected value using the inverse link function and $\eta = X\beta$. (For the normal model, this is linear.) You will need to make this change in two places, at Comment [3a] and [3b].

4. Replace `rnorm()` with a function that takes random draws from the stochastic component of your model.

9.2 Getting Ready for the GUI

Zelig can work with a variety of graphical user interfaces (GUIs). GUIs work by knowing *a priori* what a particular model accepts, and presenting only those options to the user in some sort of graphical interface. Thus, in order for your model to work with a GUI, you must describe your model in terms that the GUI can understand. For models written using the guidelines in Chapter 8, your model will be compatible with (at least) the Virtual Data Center GUI. For pre-existing models, you will need to create a `describe.*()` function for your model following the examples in Section **??**.

9.3 Formatting Reference Manual Pages

One of the primary advantages of Zelig is that it fully documents the included models, in contrast to the programming-orientation of R documentation which is organized by function. Thus, we ask that Zelig contributors provide similar documentation, including the syntax and arguments passed to `zelig()`, the systematic and stochastic components to the model, the quantities of interest, the output values, and further information (including references). There are several ways to provide this information:

- If you have an existing package documented using the .Rd help format, `help.zelig()` will automatically search R-help in addition to Zelig help.

- If you have an existing package documented using on-line HTML files with static URLs (like Zelig or MatchIt), you need to provide a `PACKAGE.url.tab` file which is a two-column table containing the name of the function in the first column and the url in the second. (Even though the file extension is `.url.tab`, the file should be a tab- or space-delimited text file.) For example:

  ```
  command       http://gking.harvard.edu/zelig/docs/Main_Commands.html
  model         http://gking.harvard.edu/zelig/docs/Specific_Models.html
  ```

  If you wish to test to see if your `.url.tab` files works, simply place it in your R library/Zelig/data/ directory. (You do not need to reinstall Zelig to test your `.url.tab` file.)

- Preferred method: You may provide a LaTeX 2_ε `.tex` file. This document uses the book style and supports commands from the following packages: `graphicx`, `natbib`, `amsmath`, `amssymb`, `verbatim`, `epsf`, and `html`. Because model pages are incorporated into this document using `\include{}`, you should make sure that your document compiles before submitting it. Please adhere to the following conventions for your model page:

  1. All mathematical formula should be typeset using the `equation*` and `array`, `eqnarray*`, or `align` environments. Please avoid `displaymath`. (It looks funny in html.)
  2. All commands or R objects should use the `texttt` environment.
  3. The model begins as a subsection of a larger document, and sections within the model page are of sub-subsection level.
  4. For stylistic consistency, please avoid using the `description` environment.

  Each LaTeX model page should include the following elements. Let `contrib` specify the new model.

Help File Template

```
\subsection{{\tt contrib}: Full Name for [type] Dependent Variables}
\label{contrib}

\subsubsection{Syntax}

\subsubsection{Examples}
\begin{enumerate}
\item First Example
\item Second Example
\end{enumerate}

\subsubsection{Model}
\begin{itemize}
\item The observation mechanism, if applicable.
\item The stochastic component.
\item The systematic component.
\end{itemize}

\subsubsection{Quantities of Interest}
\begin{itemize}
\item The expected value of your distribution, including the formula
  for the expected value as a function of the systemic component and
```

```
  ancillary paramters.
\item The predicted value drawn from the distribution defined by the
       corresponding expected value.
\item The first difference in expected values, given when x1 is specified.
\item Other quantities of interest.
\end{itemize}

\subsubsection{Output Values}
\begin{itemize}
\item From the {\tt zelig()} output stored in {\tt z.out}, you may
  extract:
   \begin{itemize}
   \item
   \item
   \end{itemize}
\item From {\tt summary(z.out)}, you may extract:
   \begin{itemize}
   \item
   \item
   \end{itemize}
\item From the {\tt sim()} output stored in {\tt s.out}:
   \begin{itemize}
   \item
   \item
   \end{itemize}
\end{itemize}

\subsubsection{Further Information}

\subsubsection{Contributors}
```

Part III

Reference Manual

Chapter 10

Main Commands

Help for each command in Zelig and R is available through `help.zelig()`. For example, typing `help.zelig(setx)` will launch a web browser with the appropriate reference manual page for the `setx()` command. (Occasionally, you may need to use, for example, `help(print)` rather than `help.zelig(print)`, to access the R help page instead of the default Zelig help page.)

10.1 `setx`: Setting Explanatory Variable Values

Description

The `setx` command uses the variables identified in the `formula` generated by `zelig` and sets the values of the explanatory variables to the selected values. Use `setx` after `zelig` and before `sim` to simulate quantities of interest.

Usage

```
x.out <- setx(object, fn = list(numeric = mean, ordered = median,
                                others = mode),
              data = NULL, cond = FALSE, ...)
```

Arguments

- `object` the saved output from `zelig`.
- `fn` a list of functions to apply to three types of variables:
- numeric `numeric` variables are set to their mean by default, but you may select any mathematical function to apply to numeric variables.
- ordered `ordered` factors are set to their meidan by default, and most mathematical operations will work on them. If you select `ordered = mean`, however, `setx` will default to median with a warning.
- other variables may consist of unordered factors, character strings, or logical variables. The `other` variables may only be set to their mode. If you wish to set one of the other variables to a specific value, you may do so using `...` below. In the special case `fn = NULL`, `setx` will return all of the observations without applying any function to the data.
- `data` a new data frame used to set the values of explanatory variables. If `data = NULL` (the default), the data frame called in `zelig` is used.
- `cond` a logical value indicating whether unconditional (default) or conditional (choose `cond = TRUE`) prediction should be performed. If you choose `cond = TRUE`, `setx` will coerce `fn = NULL` and ignore the additional arguments in `...`. If `cond = TRUE` and `data = NULL`, `setx` will prompt you for a data frame.
- `...` user-defined values of specific variables overwriting the default values set by the function `fn`. For example, adding `var1 = mean(data$var1)` or `x1 = 12` explicitly sets the value of `x1` to 12. In addition, you may specify one explanatory variable as a range of values, creating one observation for every unique value in the range of values.

Value

For unconditional prediction, `x.out` is a model matrix based on the specified values for the explanatory variables. For multiple analyses (i.e., when choosing the `by` option in `zelig`, `setx` returns the selected values calculated over the entire data frame. If you wish to calculate values over just one subset of the data frame, the 5th subset for example, you may use: `x.out <- setx(z.out[[5]])`

For conditional prediction, `x.out` includes the model matrix and the dependent variables. For multiple analyses (when choosing the `by` option in `zelig`), `setx` returns the observed explanatory variables in each subset.

Author(s)

Kosuke Imai <⟨kimai@princeton.edu⟩>; Gary King <⟨king@harvard.edu⟩>; Olivia Lau <⟨olau@fas.harvard.edu⟩>

See Also

The full Zelig manual may be accessed online at `http://gking.harvard.edu/zelig`.

Examples

```
# Unconditional prediction:
data(turnout)
z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout)
x.out <- setx(z.out)
s.out <- sim(z.out, x = x.out)

# Unconditional prediction with all observations:
x.out <- setx(z.out, fn = NULL)
s.out <- sim(z.out, x = x.out)

# Unconditional prediction with out of sample data:
z.out <- zelig(vote ~ race + educate, model = "logit",
               data = turnout[1:1000,])
x.out <- setx(z.out, data = turnout[1001:2000,])
s.out <- sim(z.out, x = x.out)

# Using a user-defined function in fn:
## Not run:
quants <- function(x)
  quantile(x, 0.25)
x.out <- setx(z.out, fn = list(numeric = quants))
## End(Not run)
```

```
# Conditional prediction:
## Not run:
library(MatchIt)
data(lalonde)
match.out <- matchit(treat ~ age + educ + black + hispan + married +
                     nodegree + re74 + re75, data = lalonde)
z.out <- zelig(re78 ~ distance, data = match.data(match.out, "control"),
               model = "ls")
x.out <- setx(z.out, fn = NULL, data = match.data(match.out, "treat"),
              cond = TRUE)
s.out <- sim(z.out, x = x.out)
## End(Not run)
```

10.2 `sim`: Simulating Quantities of Interest

Description

Simulate quantities of interest from the estimated model output from `zelig()` given specified values of explanatory variables established in `setx()`. For classical *maximum likelihood* models, `sim()` uses asymptotic normal approximation to the log-likelihood. For *Bayesian models*, Zelig simulates quantities of interest from the posterior density, whenever possible. For *robust Bayesian models*, simulations are drawn from the identified class of Bayesian posteriors. Alternatively, you may generate quantities of interest using bootstrapped parameters.

Usage

```
s.out <- sim(object, x, x1 = NULL, num = c(1000, 100), prev = NULL,
             bootstrap = FALSE,  bootfn = NULL, ...)
```

Arguments

`object` the output object from `zelig`.

`x` values of explanatory variables used for simulation, generated by `setx`.

`x1` optional values of explanatory variables (generated by a second call of `setx`), used to simulate first differences and risk ratios. (Not available for conditional prediction.)

`num` the number of simulations, i.e., posterior draws. If the `num` argument is omitted, `sim` draws 1,000 simulations by if `bootstrap = FALSE` (the default), or 100 simulations if `bootstrap = TRUE`. You may increase this value to improve accuracy. (Not available for conditional prediction.)

`bootstrap` a logical value indicating if parameters should be generated by re-fitting the model for bootstrapped data, rather than from the likelihood or posterior. (Not available for conditional prediction.)

`bootfn` a function which governs how the data is sampled, re-fits the model, and returns the bootstrapped model parameters. If `bootstrap = TRUE` and `bootfn = NULL`, `sim` will sample observations from the original data (with replacement) until it creates a sampled dataset with the same number of observations as the original data. Alternative bootstrap methods include sampling the residuals rather than the observations, weighted sampling, and parametric bootstrapping. (Not available for conditional prediction.)

`...` additional optional arguments passed to `boot`.

Value

The output stored in `s.out` varies by model. Use the `names` command to view the output stored in `s.out`. Common elements include: normal-bracket109bracket-normal

- `x` — the `setx` values for the explanatory variables, used to calculate the quantities of interest (expected values, predicted values, etc.).
- `x1` — the optional `setx` object used to simulate first differences, and other model-specific quantities of interest, such as risk-ratios.
- `call` — the options selected for `sim`, used to replicate quantities of interest.
- `zelig.call` — the original command and options for `zelig`, used to replicate analyses.
- `num` — the number of simulations requested.
- `par` — the parameters (coefficients, and additional model-specific parameters). You may wish to use the same set of simulated parameters to calculate quantities of interest rather than simulating another set.
- `qi$ev` — simulations of the expected values given the model and `x`.
- `qi$pr` — simulations of the predicted values given by the fitted values.
- `qi$fd` — simulations of the first differences (or risk difference for binary models) for the given `x` and `x1`. The difference is calculated by subtracting the expected values given `x` from the expected values given `x1`. (If do not specify `x1`, you will not get first differences or risk ratios.)
- `qi$rr` — simulations of the risk ratios for binary and multinomial models. See specific models for details.
- `qi$ate.ev` — simulations of the average expected treatment effect for the treatment group, using conditional prediction. Let t_i be a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Then the average expected treatment effect for the treatment group is

$$\frac{1}{n}\sum_{i=1}^{n}[\, Y_i(t_i = 1) - E[Y_i(t_i = 0)] \mid t_i = 1\,],$$

  where $Y_i(t_i = 1)$ is the value of the dependent variable for observation i in the treatment group. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.
- `qi$ate.pr` — simulations of the average predicted treatment effect for the treatment group, using conditional prediction. Let t_i be a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Then

the average predicted treatment effect for the treatment group is

$$\frac{1}{n}\sum_{i=1}^{n}[\,Y_i(t_i=1) - \widehat{Y_i(t_i=0)} \mid t_i = 1\,],$$

where $Y_i(t_i=1)$ is the value of the dependent variable for observation i in the treatment group. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i=0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

normal-bracket109bracket-normal

In the case of censored Y in the exponential, Weibull, and lognormal models, `sim` first imputes the uncensored values for Y before calculating the ATE.

You may use the `$` operator to extract any of the above from `s.out`. For example, `s.out$qi$ev` extracts the simulated expected values.

Author(s)

Kosuke Imai <⟨kimai@princeton.edu⟩>; Gary King <⟨king@harvard.edu⟩>; Olivia Lau <⟨olau@fas.harvard.edu⟩>

See Also

The full Zelig at `http://gking.harvard.edu/zelig`, and `boot`.

10.3 `plot.zelig`: Graphing Quantities of Interest

Description

The `zelig` method for the generic `plot` command generates default plots for `sim` output with one-observation values in `x` and `x1`.

Usage

```
## S3 method for class 'zelig':
plot(x, xlab = "", user.par = FALSE, ...)
```

Arguments

`x`	stored output from `sim`. If the `x$x` or `x$x1` values stored in the object contain more than one observation, `plot.zelig` will return an error. For linear or generalized linear models with more than one observation in `x$x` and optionally `x$x1`, you may use `plot.ci`.
`xlab`	a character string for the x-axis label for all graphs.
`user.par`	a logical value indicating whether to use the default Zelig plotting parameters (`user.par = FALSE`) or user-defined parameters (`user.par = TRUE`), set using the `par` function prior to plotting.
`...`	Additional parameters passed to `plot.default`. Because `plot.zelig` primarily produces diagnostic plots, many of these parameters are hard-coded for convenience and presentation.

Value

Depending on the class of model selected, `plot.zelig` will return an on-screen window with graphs of the various quantities of interest. You may save these plots using the commands described in the Zelig manual (available at `http://gking.harvard.edu/zelig`).

Author(s)

Kosuke Imai <⟨kimai@princeton.edu⟩>; Gary King <⟨king@harvard.edu⟩>; Olivia Lau <⟨olau@fas.harvard.edu⟩>

See Also

The full Zelig manual at `http://gking.harvard.edu/zelig` and `plot`, `lines`, and `par`.

10.4 `print`: Printing Quantities of Interest

Description

The `print` command formats summary output and allows users to specify the number of decimal places as an optional argument.

Syntax

```
> print(x, digits = 3, print.x = FALSE)
```

Arguments

- `x`: the object to be printed may be `z.out` output from `zelig()`, `x.out` output from `setx()`, `s.out` output from `sim()`, or other R data structures.
- `digits`: the minimum number of significant digits to return for all elements of $x < 0$. By default, `print()` avoids scientific notation, but setting the number of digits to 1 will frequently force output in scientific notation. The number of `digits` is not the number of significant digits for all output values, but the minimum number of significant digits for the smallest value in `x` between -1 and 1; this governs the number of significant digits in the rest of the values with decimal output.
- `print.x`: a logical value for `sim()` output, which specifies whether to print a summary (`print.x = FALSE`, the default) of the `x` and `x1` *inputs* to `sim()`, or the complete set of inputs (optionally, `print.x = TRUE`).

Examples

```
> print(summary(z.out), digits = 2)
> print(summary(s.out), digits = 3, print.x = TRUE)
```

See Also

Advanced users may wish to refer to `help(print)`.

Contributors

Kosuke Imai, Gary King, and Olivia Lau added print methods for `sim()` output, and `summary()` output for Zelig objects.

10.5 `repl`: Replicating Analyses

Description

The generic function `repl` command takes `zelig` or `sim` output objects and replicates (literally, re-runs) the entire analysis. The results should be an output object identical to the original input object in the case of `zelig` output. In the case of `sim` output, the replicated analyses may differ slightly due to stochastic randomness in the simulation procedure.

Usage

```
repl(object, data, ...)
## Default S3 method:
repl(object, data = NULL, ...)
## S3 method for class 'zelig':
repl(object, data = NULL, prev = NULL, x = NULL, x1 = NULL,
     bootfn = NULL, ...)
```

Arguments

- `object` — Stored output from either `zelig` or `sim`.
- `data` — You may manually input the data frame name rather than allowing `repl` to draw the data frame name from the object to be replicated.
- `prev` — When replicating `sim` output, you may optionally use the previously simulated parameters to calculate the quantities of interest rather than simulating a new set of parameters. For all models, this should produce identical quantities of interest. In addition, for if the parameters were bootstrapped in the original analysis, this will save a considerable amount of time.
- `x` — When replicating `sim` output, you may optionally use an alternative `setx` value for the `x` input.
- `x1` — When replicating `sim` output, you may optionally use an alternative `setx` object for the `x1` input to replicating the `sim` object.
- `bootfn` — When replicating `sim` output with bootstrapped parameters, you should manually specify the `bootfn` if a non-default option was used.
- `...` — Additional arguments passed to either `zelig` or `sim`.

Value

For `zelig` output, `repl` will create output that is in every way identical to the original input. You may check to see whether they are identical by using the `identical` command.

For `sim` output, `repl` output will be will be identical to the original object if you choose not to simulate new parameters, and instead choose to calculate quantities of interest using the previously simulated parameters (using the `prev` option. If you choose to simulate new parameters, the summary statistics for each quantity of interest should be identical, up to a random approximation error. As the number of simulations increases, this error decreases.

Author(s)

Kosuke Imai <⟨kimai@princeton.edu⟩>; Gary King <⟨king@harvard.edu⟩>; Olivia Lau <⟨olau@fas.harvard.edu⟩>

See Also

`zelig`, `setx`, and `sim`. In addition, the full Zelig manual may be accessed online at `http://gking.harvard.edu/zelig`.

Examples

```
data(turnout)
z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout[1:1000,])
x.out <- setx(z.out)
s.out <- sim(z.out, x = x.out)
z.rep <- repl(z.out)
identical(z.out$coef, z.rep$coef)
z.alt <- repl(z.out, data = turnout[1001:2000,])
s.rep <- repl(s.out, prev = s.out$par)
identical(s.out$ev, s.rep$ev)
```

Chapter 11

Supplementary Commands

11.1 `mi`: Bundle multiply imputed data sets as a list

Description

The code `mi` bundles multiply imputed data sets as a list for further analysis.

Usage

```
mi(...)
```

Arguments

...	multiply imputed data sets, separated by commas. The arguments can be tagged by `name=data` where `name` is the element named used for the data set `data`.

Value

The list containing each multiply imputed data set as an element. The class name is `mi`. The list can be inputted into `zelig` for statistical analysis with multiply imputed data sets. See `zelig` for details.

Author(s)

Kosuke Imai <⟨kimai@princeton.edu⟩>; Gary King <⟨king@harvard.edu⟩>; Olivia Lau <⟨olau@fas.harvard.edu⟩>

See Also

The full Zelig manual is available at `http://gking.harvard.edu/zelig`.

Examples

```
data(immi1, immi2, immi3, immi4, immi5)
mi(immi1, immi2, immi3, immi4, immi5)
```

11.2 `network`: Format matricies into a data frame for social network analysis

Description

This function accepts individual matricies as its inputs, combining the input matricies into a single data frame which can then be used in the `data` argument for social network analysis (models `"netlm"` and `"netlogit"`) in Zelig.

Usage

```
network(...)
```

Arguments

...	matricies representing variables, with rows and columns corresponding to individuals. These can be given as named arguments and should be given in the order the in which the user wishes them to appear in the output data frame.

Value

The `network` function creates a data frame which contains matricies instead of vectors as its variables. Inputs to the function should all be square matricies and can be given as named arguments.

Author(s)

Skyler J. Cranmer

See Also

The full Zelig manual is available at `http://gking.harvard.edu/zelig`.

Examples

```
## Not run:
## Let Var1, Var2, Var3, Var4, and Var5 be matrices
friendship <- network(Var1, Var2, Var3, Var4, Var5)
## End(Not run)
```

11.3 `plot.ci`: Plotting Vertical confidence Intervals

Description

The `plot.ci` command generates vertical confidence intervals for linear or generalized linear univariate response models.

Usage

```
plot.ci(x, CI = 95, qi = "ev", main = "", ylab = NULL, xlab = NULL,
        xlim = NULL, ylim = NULL, col = c("red", "blue"), ...)
```

Arguments

`x`	stored output from `sim`. The `x$x` and optional `x$x1` values used to generate the `sim` output object must have more than one observation.
`CI`	the selected confidence interval. Defaults to 95 percent.
`qi`	the selected quantity of interest. Defaults to expected values.
`main`	a title for the plot.
`ylab`	label for the y-axis.
`xlab`	label for the x-axis.
`xlim`	limits on the x-axis.
`ylim`	limits on the y-axis.
`col`	a vector of at most two colors for plotting the expected value given by `x` and the alternative set of expected values given by `x1` in `sim`. If the quantity of interest selected is not the expected value, or `x1 = NULL`, only the first color will be used.
`...`	Additional parameters passed to `plot`.

Value

For all univariate response models, `plot.ci()` returns vertical confidence intervals over a specified range of one explanatory variable. You may save this plot using the commands described in the Zelig manual (`http://gking.harvard.edu/zelig`).

Author(s)

Kosuke Imai <⟨kimai@princeton.edu⟩>; Gary King <⟨king@harvard.edu⟩>; Olivia Lau <⟨olau@fas.harvard.edu⟩>

See Also

The full Zelig manual is available at `http://gking.harvard.edu/zelig`, and users may also wish to see `plot`, `lines`.

Examples

```
data(turnout)
z.out <- zelig(vote ~ race + educate + age + I(age^2) + income,
               model = "logit", data = turnout)
age.range <- 18:95
x.low <- setx(z.out, educate = 12, age = age.range)
x.high <- setx(z.out, educate = 16, age = age.range)
s.out <- sim(z.out, x = x.low, x1 = x.high)
plot.ci(s.out, xlab = "Age in Years",
        ylab = "Predicted Probability of Voting",
        main = "Effect of Education and Age on Voting Behavior")
legend(45, 0.52, legend = c("College Education (16 years)",
       "High School Education (12 years)"), col = c("blue","red"),
       lty = c("solid"))
```

11.4 `plot.surv`: Plotting Confidence Intervals for Survival Curves

Description

The `plot.surv` command generates confidence intervals for Kaplan-Meier survival curves

Usage

```
plot.surv(x, duration, censor, type = "line", plotcensor=TRUE,
          plottimes = FALSE, int = c(0.025,0.975), ...)
```

Arguments

`x`	output from `sim` stored as a list. Each element of the list is the `sim` output for a particular survival curve.
`duration`	the duration variable (e.g. lifetime, survival, etc.).
`censor`	the censored data
`type`	the type of confidence interval. Defaults to `"line"`, which draws vertical confidence intervals at observed event times. `"poly"` draws confidence regions using polygons.
`plotcensor`	default is `TRUE`. Plots censoring times as a `rug` object.
`plottimes`	default is `FALSE`. Plots step function with indicators at observed event times.
`int`	vector of quantile limits for the confidence interval. Default is 95% interval.
`...`	Additional parameters passed to `plot`.

Value

For survival models, `plot.surv()` returns vertical confidence intervals or polygon survival regions for Kaplan-Meier survival curves. You may save this plot using the commands described in the Zelig manual (`http://gking.harvard.edu/zelig`).

Author(s)

John A. Graves <⟨graveja0@gmail.com⟩>

See Also

The full Zelig manual is available at `http://gking.harvard.edu/zelig`, and users may also wish to see `plot`, `lines`.

Examples

```
## Not run:
data(coalition)
z.out1 <- zelig(Surv(duration,ciep12)~invest+numst2+crisis,
robust=TRUE,cluster="polar",model="coxph",data=coalition)
low <- setx(z.out1,numst2=0)
high <- setx(z.out1,numst2=1
# Simulate Survival Curves for Each Group
s.out1 <- sim(z.out1,x=low)
s.out2 <- sim(z.out1,x=high)

# Organize simulated output as a list
out <- list(s.out1,s.out2)

plot.surv(x = out, duration = coalition$duration, censor=coalition$ciep12,
          type="line", plottimes=FALSE, plotcensor=FALSE,
          main="Survival", xlab="Time", ylab="Survival")
## End(Not run)
```

11.5 `rocplot`: Receiver Operator Characteristic Plots

Description

The `rocplot` command generates a receiver operator characteristic plot to compare the in-sample (default) or out-of-sample fit for two logit or probit regressions.

Usage

```
rocplot(y1, y2, fitted1, fitted2, cutoff = seq(from=0, to=1, length=100),
        lty1 = "solid", lty2 = "dashed", lwd1 = par("lwd"), lwd2 = par("lwd"),
        col1 = par("col"), col2 = par("col"), main, xlab, ylab,
        plot = TRUE, ...)
```

Arguments

`y1`	Response variable for the first model.
`y2`	Response variable for the second model.
`fitted1`	Fitted values for the first model. These values may represent either the in-sample or out-of-sample fitted values.
`fitted2`	Fitted values for the second model.
`cutoff`	A vector of cut-off values between 0 and 1, at which to evaluate the proportion of 0s and 1s correctly predicted by the first and second model. By default, this is 100 increments between 0 and 1, inclusive.
`lty1, lty2`	The line type for the first model (`lty1`) and the second model (`lty2`), defaulting to solid and dashed, respectively.
`lwd1, lwd2`	The width of the line for the first model (`lwd1`) and the second model (`lwd2`), defaulting to 1 for both.
`col1, col2`	The colors of the line for the first model (`col1`) and the second model (`col2`), defaulting to black for both.
`main`	a title for the plot. Defaults to `ROC Curve`.
`xlab`	a label for the x-axis. Defaults to `Proportion of 1's Correctly Predicted`.
`ylab`	a label for the y-axis. Defaults to `Proportion of 0's Correctly Predicted`.
`plot`	defaults to `TRUE`, which generates a plot to the selected device. If `FALSE`, returns a list of items (see below).
`...`	Additional parameters passed to plot, including `xlab`, `ylab`, and `main`.

Value

If `plot = TRUE`, `rocplot` generates an ROC plot for two logit or probit models. If `plot = FALSE`, `rocplot` returns a list with the following elements: normal-bracket63bracket-normal

- `roc1` — a matrix containing a vector of x-coordinates and y-coordinates corresponding to the number of ones and zeros correctly predicted for the first model.
- `roc2` — a matrix containing a vector of x-coordinates and y-coordinates corresponding to the number of ones and zeros correctly predicted for the second model.
- `area1` — the area under the first ROC curve, calculated using Reimann sums.
- `area2` — the area under the second ROC curve, calculated using Reimann sums.

normal-bracket63bracket-normal

Author(s)

Kosuke Imai <⟨kimai@princeton.edu⟩>; Gary King <⟨king@harvard.edu⟩>; Olivia Lau <⟨olau@fas.harvard.edu⟩>

See Also

The full Zelig manual (available at `http://gking.harvard.edu/zelig`), `plot`, `lines`.

Examples

```
data(turnout)
z.out1 <- zelig(vote ~ race + educate + age, model = "logit",
  data = turnout)
z.out2 <- zelig(vote ~ race + educate, model = "logit",
  data = turnout)
rocplot(z.out1$y, z.out2$y, fitted(z.out1), fitted(z.out2))
```

11.6 `ternaryplot`: Ternary diagram

Description

Visualizes compositional, 3-dimensional data in an equilateral triangle (from the vcd library, Version 0.1-3.3, Date 2004-04-21), using plot graphics. Differs from implementation in vcd (0.9-7), which uses grid graphics.

Usage

```
ternaryplot(x, scale = 1, dimnames = NULL, dimnames.position = c("corner","edge","non
            dimnames.color = "black", id = NULL, id.color = "black", coordinates = FA
            grid = TRUE, grid.color = "gray", labels = c("inside", "outside", "none")
            labels.color = "darkgray", border = "black", bg = "white", pch = 19, cex
            prop.size = FALSE, col = "red", main = "ternary plot", ...)
```

Arguments

`x` a matrix with three columns.

`scale` row sums scale to be used.

`dimnames` dimension labels (defaults to the column names of `x`).

`dimnames.position, dimnames.color`
: position and color of dimension labels.

`id` optional labels to be plotted below the plot symbols. `coordinates` and `id` are mutual exclusive.

`id.color` color of these labels.

`coordinates` if `TRUE`, the coordinates of the points are plotted below them. `coordinates` and `id` are mutual exclusive.

`grid` if `TRUE`, a grid is plotted. May optionally be a string indicating the line type (default: `"dotted"`).

`grid.color` grid color.

`labels, labels.color`
: position and color of the grid labels.

`border` color of the triangle border.

`bg` triangle background.

`pch` plotting character. Defaults to filled dots.

`cex` a numerical value giving the amount by which plotting text and symbols should be scaled relative to the default. Ignored for the symbol size if `prop.size` is not `FALSE`.

`prop.size`	if `TRUE`, the symbol size is plotted proportional to the row sum of the three variables, i.e. represents the weight of the observation.
`col`	plotting color.
`main`	main title.
`...`	additional graphics parameters (see `par`)

Details

A points' coordinates are found by computing the gravity center of mass points using the data entries as weights. Thus, the coordinates of a point P(a,b,c), $a+b+c=1$, are: P(b + c/2, c * sqrt(3)/2).

Author(s)

David Meyer
⟨david.meyer@ci.tuwien.ac.at⟩

References

M. Friendly (2000), *Visualizing Categorical Data.* SAS Institute, Cary, NC.

See Also

`ternarypoints`

Examples

```
data(mexico)
if (require(VGAM)) {
z.out <- zelig(as.factor(vote88) ~ pristr + othcok + othsocok,
                model = "mlogit", data = mexico)
x.out <- setx(z.out)
s.out <- sim(z.out, x = x.out)

ternaryplot(s.out$qi$ev, pch = ".", col = "blue",
            main = "1988 Mexican Presidential Election")
}
```

11.7 `ternarypoints`: Adding Points to Ternary Diagrams

Description

Use `ternarypoints` to add points to a ternary diagram generated using the `ternaryplot` function in the vcd library. Use ternary diagrams to plot expected values for multinomial choice models with three categories in the dependent variable.

Usage

```
ternarypoints(object, pch = 19, col = "blue", ...)
```

Arguments

`object`	The input object must be a matrix with three columns.
`pch`	The selected type of point. By default, `pch = 19`, solid disks.
`col`	The color of the points. By default, `col = "blue"`.
`...`	Additional parameters passed to `points`.

Value

The `ternarypoints` command adds points to a previously existing ternary diagram. Use `ternaryplot` in the `vcd` library to generate the main ternary diagram.

Author(s)

Kosuke Imai <⟨kimai@princeton.edu⟩>; Gary King <⟨king@harvard.edu⟩>; Olivia Lau <⟨olau@fas.harvard.edu⟩>

See Also

The full Zelig manual at `http://gking.harvard.edu/zelig`, `points`, and `ternaryplot`.

Chapter 12

Models Zelig Can Run

This section describes the mathematical components of the models supported by Zelig, using whenever possible the classification and notation of King (1989). Most models have a *stochastic component* (probability density given certain parameters) and a *systematic component* (deterministic functional form that specifies how one or more of the parameters varies over the observed values y_i as a function of the explanatory variables x_i).

Let Y_i be a random outcome variable, realized as $i = 1, \ldots, n$ observations y_i. For the probability density $f(\cdot)$ with systematic feature θ_i varying over i and a scalar ancillary parameter α (constant over i), the stochastic component is given by

$$Y_i \sim f(y_i \mid \theta_i, \alpha).$$

For a functional form $g(\cdot)$, k explanatory variables X_i, and effect parameters β, the systematic component is:

$$\theta_i = g(x_i, \beta).$$

Using the definitions of King, Tomz, and Wittenberg, 2000, Zelig generates at least two quantities of interest:

- The predicted value is a random draw from the stochastic component given random draws of β and α from their sampling (or posterior) distribution.
- The expected value is the *mean* of the stochastic component given random draws of β and α from their sampling (or posterior) distributions. For computational efficiency, Zelig deterministically calculates the expected values from the simulated parameters whenever possible.

Both the predicted values and expected values produced by Zelig can be displayed as histograms or density estimates (to summarize the full sampling or posterior density), or summarized with confidence intervals (by sorting the simulations and taking the 5th and 95th percentile values for a 90% confidence interval for example), standard errors (by taking the standard deviation of the simulations), or point estimates (by averaging the simulations). The point estimate of predicted and expected values are the same only in linear models. In almost all situations, simulations from predicted values have more variance than expected values. As the number of simulations increases the distribution of the expected values tends toward a constant; the distribution of the predicted values does not collapse as the number of simulations increases.

12.1 aov: Analysis of Variance for Continuous Dependent Variables

Model "aov" uses least squares regression to estimate the residual sum of squares and degrees of freedom for each explanatory variable around the best linear predictor for the specified dependent variables. Model "aov" is particularly useful for the analysis of randomized experiments with more than one strata or group (e.g., balanced incomplete block design). For the model with only one strata, "aov" reduces to "ls".

Syntax

```
> z.out <- zelig(Y ~ X1 + X2 + Error(Z), model = "aov", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where the `Error()` term is optional and takes strata formula.

Examples

1. Basic Example of aov.

   Attach sample data and set orthogonal contrasts:

   ```
   > data(npk, package = "MASS")
   > op <- options(contrasts = c("contr.helmert", "contr.poly"))
   ```

   Estimate the model (Venables and Ripley 2002, p.165):

   ```
   > z.out1 <- zelig(yield ~ block + N * P + K, model = "aov", data = npk)
   ```

   Summarize the fitted model:

   ```
   > summary(z.out1)
   ```

   Set explanatory variables to their default (mean/mode) values

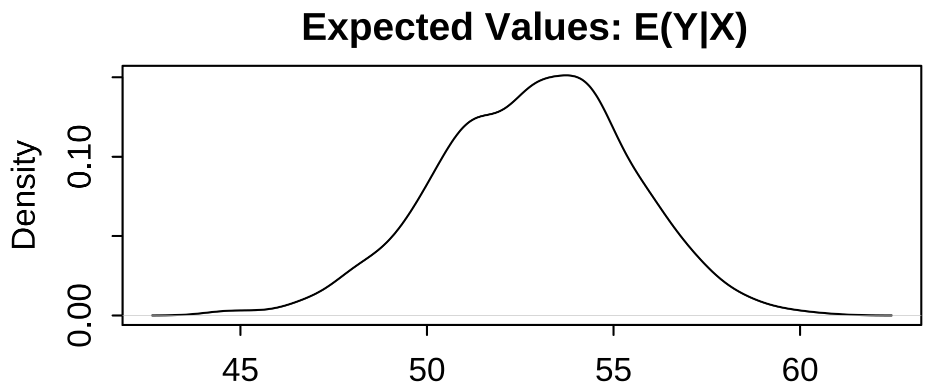
   ```
   > x <- setx(z.out1)
   ```

   Simulate model at values explanatory variables as in x

   ```
   > s.out1 <- sim(z.out1, x = x)
   ```

   ```
   > summary(s.out1)
   ```

   ```
   > plot(s.out1)
   ```



2. Example with `Error()` term allowing for more than one source of random variation in an experiment (multistratum model).

   Estimate the model:

```
> z.out2 <- zelig(yield ~ N * P * K + Error(block), model = "aov",
+     data = npk)
```

   Summarize regression coefficients:

```
> summary(z.out2)
```

   Set explanatory variables to their default (mean/mode) values

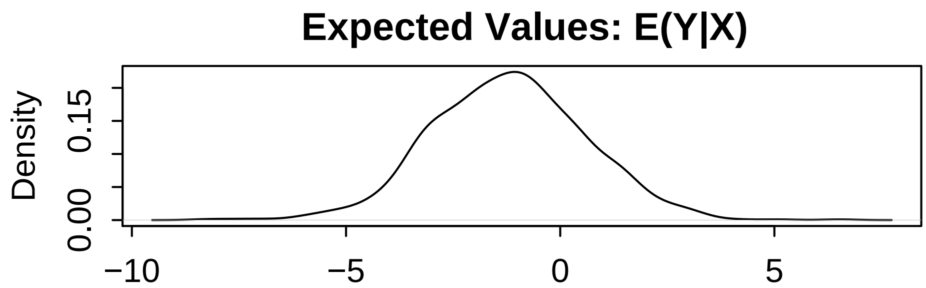
```
> x <- setx(z.out2)
```

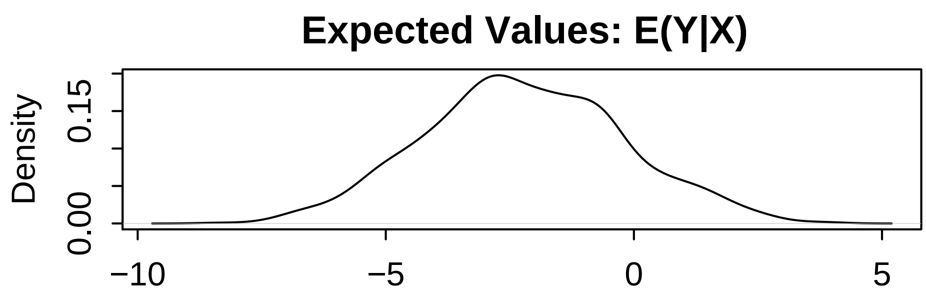
   Simulate model at values explanatory variables as in x

```
> s.out2 <- sim(z.out2, x = x)
```

```
> summary(s.out2)
```

```
> plot(s.out2)
```





3. Example with *Error()* term (multistratum model) and first differences.

   Reset to previous contrasts

```
> options(op)
```

   Estimate model (Venables and Ripley 2002, p.283):

```
> z.out3 <- zelig(Y ~ N * V + Error(B/V), model = "aov", data = oats)
```

   Summarize regression coefficients:

```
> summary(z.out3)
```

   Set explanatory variables using mode

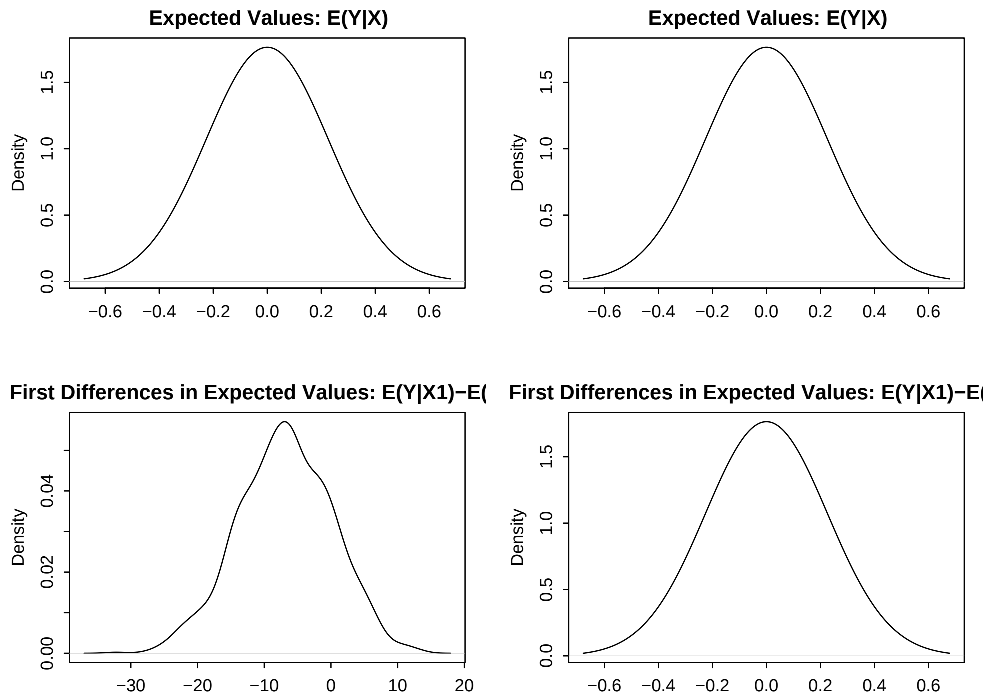
```
> x.out <- setx(z.out3, N = "0.0cwt", V = "Golden.rain")
> x.out1 <- setx(z.out3, N = "0.0cwt", V = "Victory")
```

   Simulate model at values explanatory variables as in x

```
> s.out3 <- sim(z.out3, x = x.out, x1 = x.out1)
```

```
> summary(s.out3)
```

```
> plot(s.out3)
```



Model

- The *stochastic component* is described by a normal density with mean μ_i and the common variance σ^2
$$Y_i \sim f(y_i \mid \mu_i, \sigma^2).$$
- The *systematic component* models the conditional mean as
$$\mu_i = x_i \beta$$
where x_i is the vector of covariates, and β is the vector of coefficients.

  The least squares estimator is the best linear predictor of a dependent variable given x_i, and minimizes the sum of squared residuals, $\sum_{i=1}^{n}(Y_i - x_i\beta)^2$. The output of `aov` model is the sum of squared residuals. Note that `aov` is the same model as `ls` but the output values of function call `zelig` are different. You may check that `name(z.out)` returns the same arguments for the two models.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the stochastic component,
$$E(Y) = x_i\beta,$$
given a draw of β from its sampling distribution.

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "aov", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: fitted values.
  - `df.residual`: the residual degrees of freedom.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the residuals sum of squares estimated with their associated degree of freedom, their mean squares, F-values, and F-statistics for all explanatory variables.
  - `residuals`: the sum of square, mean, degree of freedom, F-values, and F-statistics for the vector of residuals or standard errors that check the adequecy of the fit for the dependent variable versus the true values or data points.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.

- `qi$fd`: the simulated first differences (or differences in expected values) for the specified values of `x` and `x1`.
- `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *aov* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "aov: Analysis of Variance for Continuous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`. Elena Villalon implemented the software.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The analysis of variance model is part of the stats package by William N. Venables and Brian D. Ripley (Venables and Ripley 2002). In addition, advanced users may wish to refer to `help(aov)` and `help(lm)`.

12.2 `ARIMA`: ARIMA Models for Time Series Data

Use auto-regressive, integrated, moving-average (ARIMA) models for time series data. A time series is a set of observations ordered according to the time they were observed. Because the value observed at time t may depend on values observed at previous time points, time series data may violate independence assumptions. An ARIMA(p, d, q) model can account for temporal dependence in several ways. First, the time series is differenced to render it stationary, by taking d differences. Second, the time dependence of the stationary process is modeled by including p auto-regressive and q moving-average terms, in addition to any time-varying covariates. For a cyclical time series, these steps can be repeated according to the period of the cycle, whether quarterly or monthly or another time interval. ARIMA models are extremely flexible for continuous data. Common formulations include, ARIMA(0, 0, 0) for least squares regression (see Section 12.32), ARIMA(1, 0, 0), for an AR1 model, and ARIMA(0, 0, 1) for an MA1 model. For a more comprehensive review of ARIMA models, see Enders (2004).

Syntax

```
> z.out <- zelig(Diff(Y, d, ds=NULL, per=NULL) ~ lag.y(p, ps=NULL)
                 + lag.eps(q, qs=NULL) + X1 + X2,
                 model="arima", data=mydata, ...)
> x.out <- setx(z.out, X1 = list(time, value), cond = FALSE)
> s.out <- sim(z.out, x=x.out, x1=NULL)
```

Inputs

In addition to independent variables, `zelig()` accepts the following arguments to specify the `ARIMA` model:

- `Diff(Y, d, ds, per)` for a dependent variable Y sets the number of non-seasonal differences (`d`), the number of seasonal differences (`ds`), and the period of the season (`per`).
- `lag.y(p, ps)` sets the number of lagged observations of the dependent variable for non-seasonal (`p`) and seasonal (`ps`) components.
- `lag.eps(q, qs)` sets the number of lagged innovations, or differences between the observed value of the time series and the expected value of the time series for non-seasonal (`q`) and seasonal (`qs`) components.

In addition the user can control the estimation of the time series with the following terms:

- ...: Additional inputs. See `help(arima)` in the stats library for further information.

Stationarity

A stationary time series has finite variance, correlations between observations that are not time-dependent, and a constant expected value for all components of the time series (Brockwell and Davis 1991, p. 12). Users should ensure that the time series being analyzed is stationary before specifying a model. The following commands provide diagnostics to determine if a time series `Y` is stationary.

- `pp.test(Y)`: Tests the null hypothesis that the time series is non-stationary.
- `kpss.test(Y)`: Tests the null hypothesis that the time series model is stationary.

The following commands provide graphical means of diagnosing whether a given time series is stationary.

- `ts.plot(Y)`: Plots the observed time series.
- `acf(Y)`: Provides the sample auto-correlation function (correlogram) for the time series.
- `pacf(Y)`: Provides the sample partial auto-correlation function (PACF) for the time series.

These latter two plots are also useful in determining the p autoregressive terms and the q lagged error terms. See Enders (2004) for a complete description of how to utilize ACF and PACF plots to determine the order of an ARIMA model.

Examples

1. No covariates

   Estimate the ARIMA model, and summarize the results.

   ```
   > data(approval)

   > z.out1 <- zelig(Diff(approve, 1) ~ lag.eps(2) + lag.y(2), data = approval,
   +     model = "arima")
   > summary(z.out1)
   ```

   Set the number of time periods (ahead) for the prediction to run. for which you would like the prediction to run:

   ```
   > x.out1 <- setx(z.out1, pred.ahead = 10)
   ```

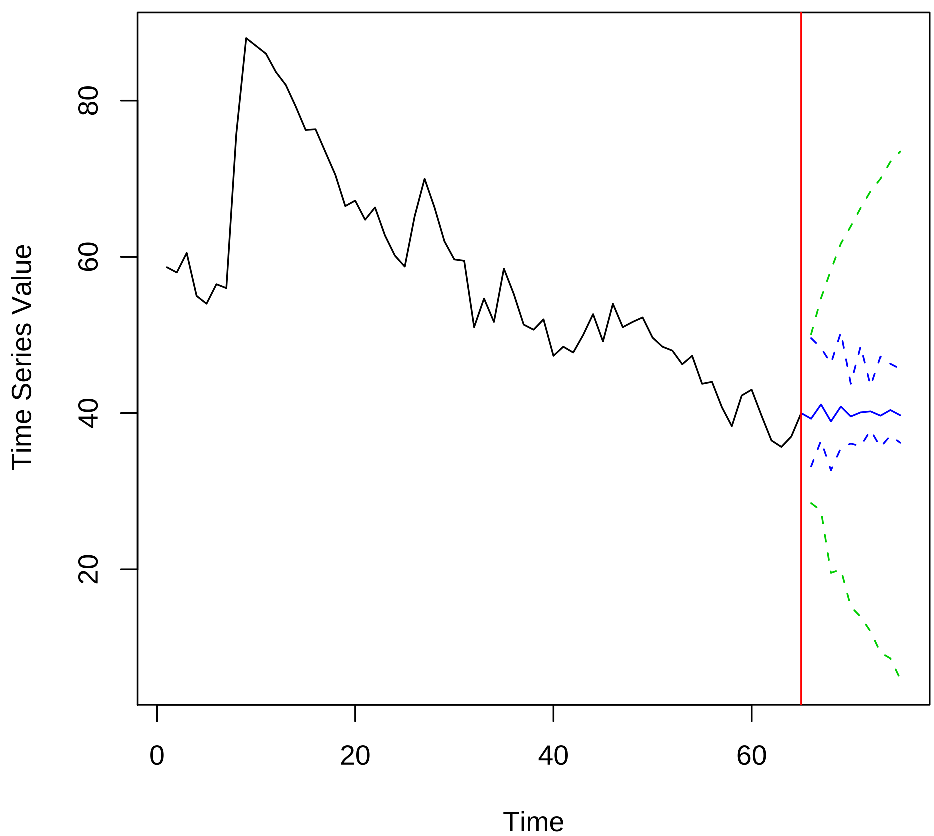
   Simulate the predicted quantities of interest:

   ```
   > s.out1 <- sim(z.out1, x = x.out1)
   ```

Summarize and plot the results:

```
> summary(s.out1)

> plot(s.out1, lty.set = 2)
```



2. Calculating a treatment effect

   Estimate an ARIMA model with exogenous regressors, in addition to lagged errors and lagged values of the dependent variable.

```
> z.out2 <- zelig(Diff(approve, 1) ~ iraq.war + sept.oct.2001 +
+      avg.price + lag.eps(1) + lag.y(2), data = approval, model = "arima")
```

   To calculate a treatment effect, provide one counterfactual value for one time period for one of the exogenous regressors (this is the counterfactual treatment).

```
> x.out2 <- setx(z.out2, sept.oct.2001 = list(time = 45, value = 0),
+      cond = T)
```

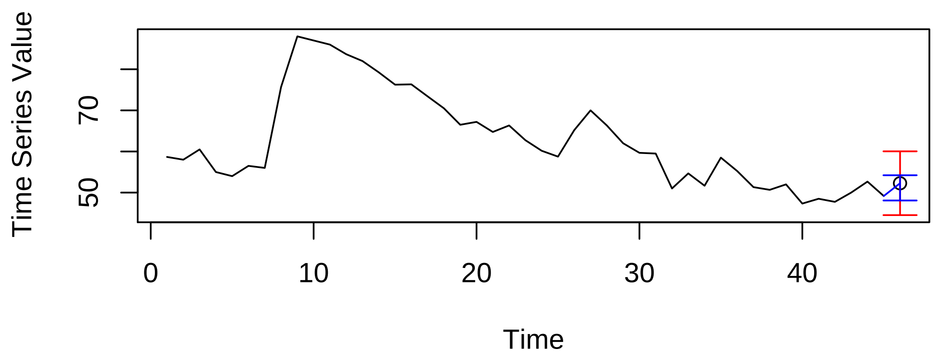
Simulate the quantities of interes

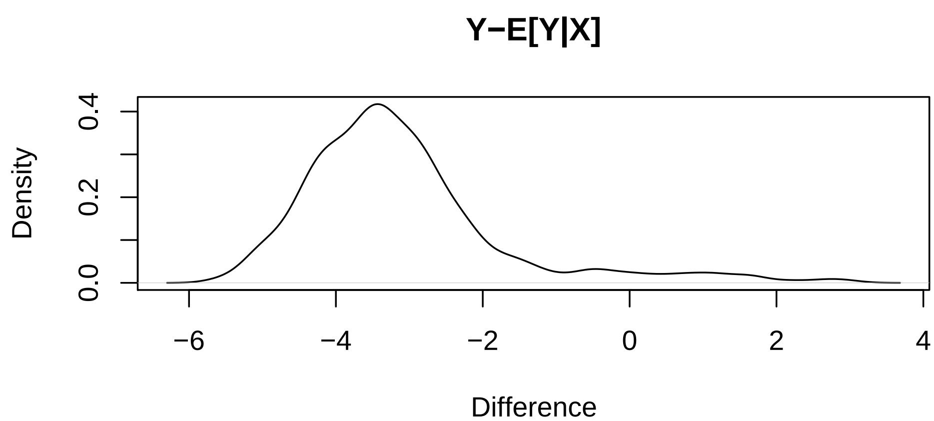
```
> s.out2 <- sim(z.out2, x = x.out2)
```

Summarizing and plotting the quantities of interest.

```
> summary(s.out2)
```

```
> plot(s.out2)
```





3. Calculating first differences

Continuing the example from above, calculate first differences by selecting several counterfactual values for one of the exogenous regressors.

```
> x.out3 <- setx(z.out2, sept.oct.2001 = list(time = 45:50, value = 0))
> x1.out3 <- setx(z.out2, sept.oct.2001 = list(time = 45:50, value = 1))
```

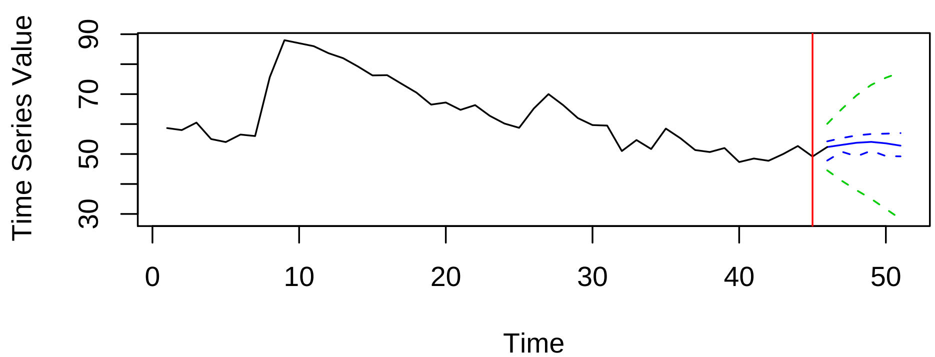
Simulating the quantities of interest

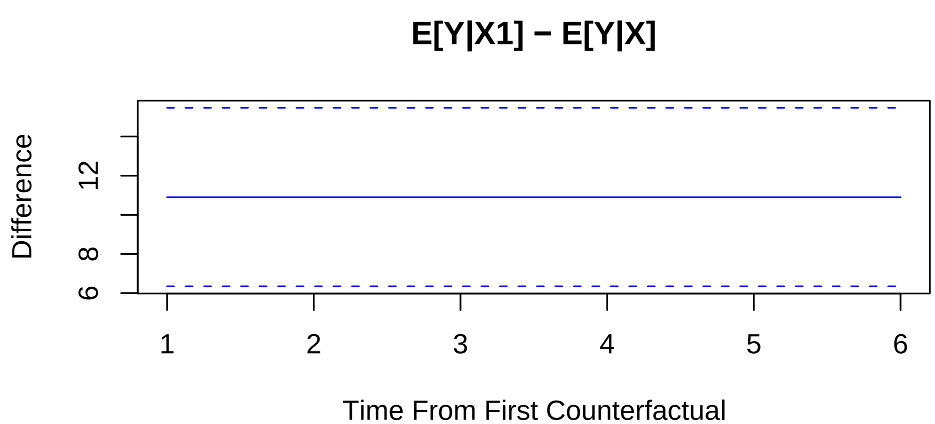
```
> s.out3 <- sim(z.out2, x = x.out3, x1 = x1.out3)
```

Summarizing and plotting the quantities of interest. Choosing `pred.se = TRUE` only displays the uncertainty resulting from parameter estimation.

```
> summary(s.out3)
```

```
> plot(s.out3, pred.se = TRUE)
```





Model

Suppose we observe a time series Y, with observations Y_i where i denotes the time at which the observation was recorded. The first step in the ARIMA procedure is to ensure that this series is stationary. If initial diagnostics indicate non-stationarity, then we take additional differences until the diagnostics indicate stationarity. Formally, define the difference operator, ∇^d, as follows. When $d = 1$, $\nabla^1 Y = Y_i - Y_{i-1}$, for all observations in the series. When $d = 2$, $\nabla^2 Y = (Y_i - Y_{i-1}) - (Y_{i-1} - Y_{i-2})$. This is analogous to a polynomial expansion, $Y_i - 2Y_{i-1} + Y_{i-2}$. Higher orders of differencing ($d > 2$) following the same function. Let Y^* represent the stationary time series derived from the initial time series by differencing Y d times. In the second step, lagged values of Y^* and errors $\mu - Y_i^*$ are used to model the time series. ARIMA utilizes a state space representation of the ARIMA model to assemble the likelihood and then utilizes maximum likelihood to estimate the parameters of the model. See Brockwell and Davis (1991) Chapter 12 for further details.

- A stationary time series Y_i^* that has been differenced d times has *stochastic component*:

$$Y_i^* \sim \text{Normal}(\mu_i, \sigma^2),$$

  where μ_i and σ^2 are the mean and variance of the Normal distribution, respectively.

- The *systematic component*, μ_i is modeled as

$$\mu_i = x_i\beta + \alpha_1 Y_{i-1}^* + \ldots + \alpha_p Y_{i-p}^* + \gamma_1 \epsilon_{i-1} + \ldots + \gamma_q \epsilon_{i-q}$$

  where x_i are the explanatory variables with associated parameter vector β; Y^* the lag-p observations from the stationary time series with associated parameter vector α; and ϵ_i the lagged errors or innovations of order q, with associated parameter vector γ.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the stochastic component,

$$\text{E}(\text{Y}_\text{i}) = \mu_i = x_i\beta + \alpha_1 Y_{i-1}^* + \ldots + \alpha_p Y_{i-p}^* + \gamma_1 \epsilon_{i-1} + \ldots + \gamma_q \epsilon_{i-q}$$

  given draws of β, α, and γ from their estimated distribution.

- The first difference (`qi$fd`) is:

$$\text{FD}_i = E(Y|x_{1i}) - E(Y|x_i)$$

- The treatment effect (`qi$t.eff`), obtained with `setx(..., cond = TRUE)`, represents the difference between the observed time series and the expected value of a time series with counterfactual values of the external regressors. Formally,

$$\text{t.eff}_\text{i} = Y_i - E[Y_i|x_i]$$

  Zelig will not estimate both first differences and treatment effects.

Output Values

The output of each Zelig command contains useful information which the user may view. For example, if the user runs `z.out <- zelig(Diff(Y,1) + lag.y(1) + lag.eps(1) + X1, model = "arima", data)` then the user may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `z.out$coef` and a default summary of information through `summary(z.out)`. `tsdiag(z.out)` returns a plot of the residuals, the ACF of the residuals, and a plot displaying the p-values for the Ljung-Box statistic. Other elements, available through the $ operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coef`: parameter estimates for the explanatory variables, lagged observations of the time series, and lagged innovations.
  - `sigma2`: maximum likelihood estimate of the variance of the stationary time series.
  - `var.coef`: variance-covariance matrix for the parameters.
  - `loglik`: maximized log-likelihood.
  - `aic`: Akaike Information Criterion (AIC) for the maximized log-likelihood.
  - `residuals`: Residuals from the fitted model.
  - `arma`: A vector with seven elements corresponding to the AR and MA, the seasonal AR and MA, the period of the seasonal component, and the number of non-seasonal and seasonal differences of the dependent variable.
  - `data`: the name of the input data frame.
- From the `sim()` output object `s.out` you may extract quantities of interest arranged as matrices, with the rows indicating the number of the simulations, and the columns representing the simulated value of the dependent variable for the counterfactual value at that time period. `summary(s.out)` provides a summary of the simulated values, while `plot(s.out)` provides a graphical representation of the simulations. Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$fd` : the simulated first difference for the values that are specified in `x` and `x1`.
  - `qi$t.eff`: the simulated treatment effect, difference between the observed `y` and the expected values given the counterfactual values specified in `x`.

How to Cite

To cite the ARIMA Zelig module:

Justin Grimmer. 2007. "ARIMA: Models for Time Series Data," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The ARIMA function is part of the stats package (Venables and Ripley 2002) For an accessible introduction to identifying the order of an ARIMA model consult Enders (2004) In addition, advanced users may wish to become more familiar with the state-space representation of an ARIMA process (Brockwell and Davis 1991) Additional options for ARIMA models may be found using `help(arima)`.

12.3 `blogit`: Bivariate Logistic Regression for Two Dichotomous Dependent Variables

Use the bivariate logistic regression model if you have two binary dependent variables (Y_1, Y_2), and wish to model them jointly as a function of some explanatory variables. Each pair of dependent variables (Y_{i1}, Y_{i2}) has four potential outcomes, $(Y_{i1} = 1, Y_{i2} = 1)$, $(Y_{i1} = 1, Y_{i2} = 0)$, $(Y_{i1} = 0, Y_{i2} = 1)$, and $(Y_{i1} = 0, Y_{i2} = 0)$. The joint probability for each of these four outcomes is modeled with three systematic components: the marginal $\Pr(Y_{i1} = 1)$ and $\Pr(Y_{i2} = 1)$, and the odds ratio ψ, which describes the dependence of one marginal on the other. Each of these systematic components may be modeled as functions of (possibly different) sets of explanatory variables.

Syntax

```
> z.out <- zelig(list(mu1 = Y1 ~ X1 + X2 ,
                      mu2 = Y2 ~ X1 + X3),
                 model = "blogit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Input Values

In every bivariate logit specification, there are three equations which correspond to each dependent variable (Y_1, Y_2), and ψ, the odds ratio. You should provide a list of formulas for each equation or, you may use `cbind()` if the right hand side is the same for both equations

```
> formulae <- list(cbind(Y1, Y2) ~ X1 + X2)
```

which means that all the explanatory variables in equations 1 and 2 (corresponding to Y_1 and Y_2) are included, but only an intercept is estimated (all explanatory variables are omitted) for equation 3 (ψ).

You may use the function `tag()` to constrain variables across equations:

```
> formulae <- list(mu1 = y1 ~ x1 + tag(x3, "x3"), mu2 = y2 ~ x2 +
+     tag(x3, "x3"))
```

where `tag()` is a special function that constrains variables to have the same effect across equations. Thus, the coefficient for `x3` in equation `mu1` is constrained to be equal to the coefficient for `x3` in equation `mu2`.

Examples

1. Basic Example

   Load the data and estimate the model:

```
> data(sanction)

> z.out1 <- zelig(cbind(import, export) ~ coop + cost + target,
+     model = "blogit", data = sanction)
```

By default, `zelig()` estimates two effect parameters for each explanatory variable in addition to the odds ratio parameter; this formulation is parametrically independent (estimating unconstrained effects for each explanatory variable), but stochastically dependent because the models share an odds ratio.

Generate baseline values for the explanatory variables (with cost set to 1, net gain to sender) and alternative values (with cost set to 4, major loss to sender):

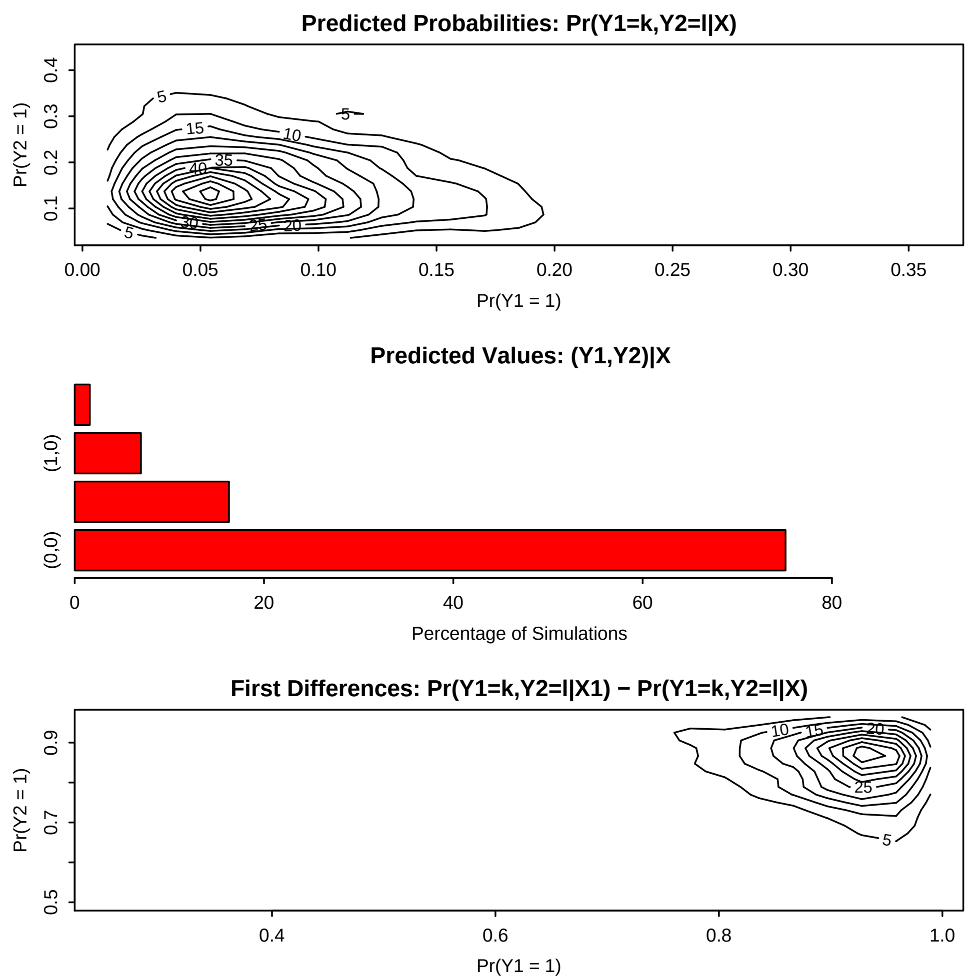
```
> x.low <- setx(z.out1, cost = 1)

> x.high <- setx(z.out1, cost = 4)
```

Simulate fitted values and first differences:

```
> s.out1 <- sim(z.out1, x = x.low, x1 = x.high)
> summary(s.out1)

> plot(s.out1)
```



2. Joint Estimation of a Model with Different Sets of Explanatory Variables

   Using sample data `sanction`, estimate the statistical model, with `import` a function of `coop` in the first equation and `export` a function of `cost` and `target` in the second equation:

```
> z.out2 <- zelig(list(import ~ coop, export ~ cost + target),
+      model = "blogit", data = sanction)
> summary(z.out2)
```

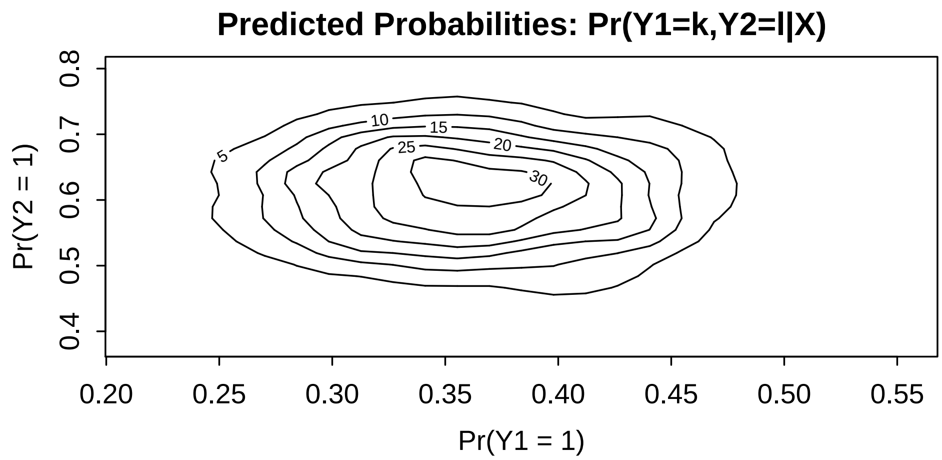
   Set the explanatory variables to their means:

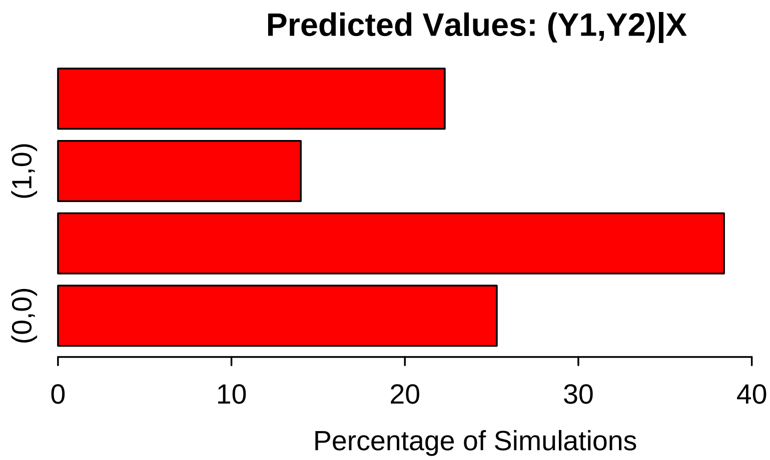
```
> x.out2 <- setx(z.out2)
```

   Simulate draws from the posterior distribution:

```
> s.out2 <- sim(z.out2, x = x.out2)
> summary(s.out2)
```

```
> plot(s.out2)
```





3. Joint Estimation of a Parametrically and Stochastically Dependent Model

   Using the sample data `sanction` The bivariate model is parametrically dependent if Y_1 and Y_2 share some or all explanatory variables, *and* the effects of the shared explanatory variables are jointly estimated. For example,

```
> z.out3 <- zelig(list(import ~ tag(coop, "coop") + tag(cost, "cost") +
+      tag(target, "target"), export ~ tag(coop, "coop") + tag(cost,
+      "cost") + tag(target, "target")), model = "blogit", data = sanction)
> summary(z.out3)
```

   Note that this model only returns one parameter estimate for each of `coop`, `cost`, and `target`. Contrast this to Example 1 which returns two parameter estimates for each of the explanatory variables.

Set values for the explanatory variables:

```
> x.out3 <- setx(z.out3, cost = 1:4)
```

Draw simulated expected values:

```
> s.out3 <- sim(z.out3, x = x.out3)
> summary(s.out3)
```

Model

For each observation, define two binary dependent variables, Y_1 and Y_2, each of which take the value of either 0 or 1 (in the following, we suppress the observation index). We model the joint outcome (Y_1, Y_2) using a marginal probability for each dependent variable, and the odds ratio, which parameterizes the relationship between the two dependent variables. Define Y_{rs} such that it is equal to 1 when $Y_1 = r$ and $Y_2 = s$ and is 0 otherwise, where r and s take a value of either 0 or 1. Then, the model is defined as follows,

- The *stochastic component* is

$$\begin{aligned} Y_{11} &\sim \text{Bernoulli}(y_{11} \mid \pi_{11}) \\ Y_{10} &\sim \text{Bernoulli}(y_{10} \mid \pi_{10}) \\ Y_{01} &\sim \text{Bernoulli}(y_{01} \mid \pi_{01}) \end{aligned}$$

  where $\pi_{rs} = \Pr(Y_1 = r, Y_2 = s)$ is the joint probability, and $\pi_{00} = 1 - \pi_{11} - \pi_{10} - \pi_{01}$.

- The *systematic components* model the marginal probabilities, $\pi_j = \Pr(Y_j = 1)$, as well as the odds ratio. The odds ratio is defined as $\psi = \pi_{00}\pi_{01}/\pi_{10}\pi_{11}$ and describes the relationship between the two outcomes. Thus, for each observation we have

$$\begin{aligned} \pi_j &= \frac{1}{1 + \exp(-x_j \beta_j)} \quad \text{for} \quad j = 1, 2, \\ \psi &= \exp(x_3 \beta_3). \end{aligned}$$

Quantities of Interest

- The expected values (`qi$ev`) for the bivariate logit model are the predicted joint probabilities. Simulations of β_1, β_2, and β_3 (drawn from their sampling distributions) are substituted into the systematic components (π_1, π_2, ψ) to find simulations of the predicted joint probabilities:

$$\begin{aligned} \pi_{11} &= \begin{cases} \frac{1}{2}(\psi - 1)^{-1} - a - \sqrt{a^2 + b} & \text{for } \psi \neq 1 \\ \pi_1 \pi_2 & \text{for } \psi = 1 \end{cases}, \\ \pi_{10} &= \pi_1 - \pi_{11}, \\ \pi_{01} &= \pi_2 - \pi_{11}, \\ \pi_{00} &= 1 - \pi_{10} - \pi_{01} - \pi_{11}, \end{aligned}$$

where $a = 1 + (\pi_1 + \pi_2)(\psi - 1)$, $b = -4\psi(\psi - 1)\pi_1\pi_2$, and the joint probabilities for each observation must sum to one. For n simulations, the expected values form an $n \times 4$ matrix for each observation in `x`.

- The predicted values (`qi$pr`) are draws from the multinomial distribution given the expected joint probabilities.

- The first differences (`qi$fd`) for each of the predicted joint probabilities are given by

$$\text{FD}_{rs} = \Pr(Y_1 = r, Y_2 = s \mid x_1) - \Pr(Y_1 = r, Y_2 = s \mid x).$$

- The risk ratio (`qi$rr`) for each of the predicted joint probabilities are given by

$$\text{RR}_{rs} = \frac{\Pr(Y_1 = r, Y_2 = s \mid x_1)}{\Pr(Y_1 = r, Y_2 = s \mid x)}$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \{Y_{ij}(t_i = 1) - E[Y_{ij}(t_i = 0)]\} \text{ for } j = 1, 2,$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{ij}(t_i = 0)]$, the counterfactual expected value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \left\{Y_{ij}(t_i = 1) - \widehat{Y_{ij}(t_i = 0)}\right\} \text{ for } j = 1, 2,$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_{ij}(t_i = 0)}$, the counterfactual predicted value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "blogit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and obtain a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: the named vector of coefficients.
  - `fitted.values`: an $n \times 4$ matrix of the in-sample fitted values.
  - `predictors`: an $n \times 3$ matrix of the linear predictors $x_j \beta_j$.
  - `residuals`: an $n \times 3$ matrix of the residuals.
  - `df.residual`: the residual degrees of freedom.
  - `df.total`: the total degrees of freedom.
  - `rss`: the residual sum of squares.
  - `y`: an $n \times 2$ matrix of the dependent variables.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coef3`: a table of the coefficients with their associated standard errors and t-statistics.
  - `cov.unscaled`: the variance-covariance matrix.
  - `pearson.resid`: an $n \times 3$ matrix of the Pearson residuals.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation; otherwise the quantities are matrices). Available quantities are:
  - `qi$ev`: the simulated expected joint probabilities (or expected values) for the specified values of `x`.
  - `qi$pr`: the simulated predicted outcomes drawn from a distribution defined by the expected joint probabilities.
  - `qi$fd`: the simulated first difference in the expected joint probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio in the predicted probabilities for given `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *blogit* Zelig model use:

> Kosuke Imai,Gary King, and Olivia Lau. 2007. "blogit: Bivariate Logistic Regression for Two Dichotomous Dependent Variable," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The bivariate logit function is part of the VGAM package by Thomas Yee (Yee and Hastie 2003). In addition, advanced users may wish to refer to `help(vglm)` in the VGAM library. Additional documentation is available at http://www.stat.auckland.ac.nz/˜ yee.Sample data are from Martin (1992)

12.4 `bprobit`: Bivariate Logistic Regression for Two Dichotomous Dependent Variables

Use the bivariate probit regression model if you have two binaryrun dependent variables (Y_1, Y_2), and wish to model them jointly as a function of some explanatory variables. Each pair of dependent variables (Y_{i1}, Y_{i2}) has four potential outcomes, $(Y_{i1} = 1, Y_{i2} = 1)$, $(Y_{i1} = 1, Y_{i2} = 0)$, $(Y_{i1} = 0, Y_{i2} = 1)$, and $(Y_{i1} = 0, Y_{i2} = 0)$. The joint probability for each of these four outcomes is modeled with three systematic components: the marginal $\Pr(Y_{i1} = 1)$ and $\Pr(Y_{i2} = 1)$, and the correlation parameter ρ for the two marginal distributions. Each of these systematic components may be modeled as functions of (possibly different) sets of explanatory variables.

Syntax

```
> z.out <- zelig(list(mu1 = Y1 ~ X1 + X2,
                      mu2 = Y2 ~ X1 + X3,
                      rho = ~ 1),
                 model = "bprobit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Input Values

In every bivariate probit specification, there are three equations which correspond to each dependent variable (Y_1, Y_2), and the correlation parameter ρ. Since the correlation parameter does not correspond to one of the dependent variables, the model estimates ρ as a constant by default. Hence, only two formulas (for μ_1 and μ_2) are required. If the explanatory variables for μ_1 and μ_2 are the same and effects are estimated separately for each parameter, you may use the following short hand:

```
> fml <- list(cbind(Y1, Y2) ~ X1 + X2)
```

which has the same meaning as:

```
> fml <- list(mu1 = Y1 ~ X1 + X2, mu2 = Y2 ~ X1 + X2, rho = ~1)
```

You may use the function `tag()` to constrain variables across equations. The `tag()` function takes a variable and a label for the effect parameter. Below, the constrained effect of `x3` in both equations is called the `age` parameter:

```
> fml <- list(mu1 = y1 ~ x1 + tag(x3, "age"), mu2 = y2 ~ x2 + tag(x3,
+     "age"))
```

You may also constrain different variables across different equations to have the same effect.

Examples

1. Basic Example

   Load the data and estimate the model:

```
> data(sanction)

> z.out1 <- zelig(cbind(import, export) ~ coop + cost + target,
+      model = "bprobit", data = sanction)
```

   By default, `zelig()` estimates two effect parameters for each explanatory variable in addition to the correlation coefficient; this formulation is parametrically independent (estimating unconstrained effects for each explanatory variable), but stochastically dependent because the models share a correlation parameter.

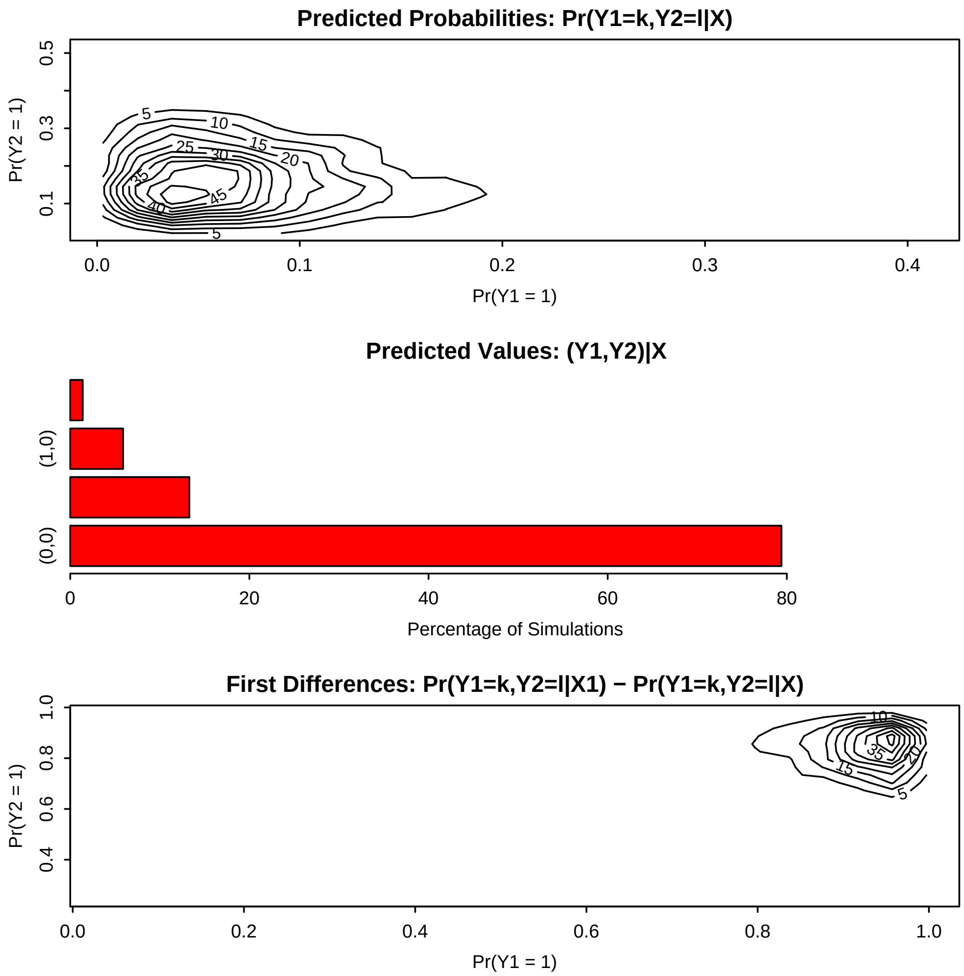
   Generate baseline values for the explanatory variables (with cost set to 1, net gain to sender) and alternative values (with cost set to 4, major loss to sender):

```
> x.low <- setx(z.out1, cost = 1)
> x.high <- setx(z.out1, cost = 4)
```

   Simulate fitted values and first differences:

```
> s.out1 <- sim(z.out1, x = x.low, x1 = x.high)
> summary(s.out1)

> plot(s.out1)
```



2. Joint Estimation of a Model with Different Sets of Explanatory Variables

   Using the sample data `sanction`, estimate the statistical model, with `import` a function of `coop` in the first equation and `export` a function of `cost` and `target` in the second equation:

```
> fml2 <- list(mu1 = import ~ coop, mu2 = export ~ cost + target)

> z.out2 <- zelig(fml2, model = "bprobit", data = sanction)
> summary(z.out2)
```

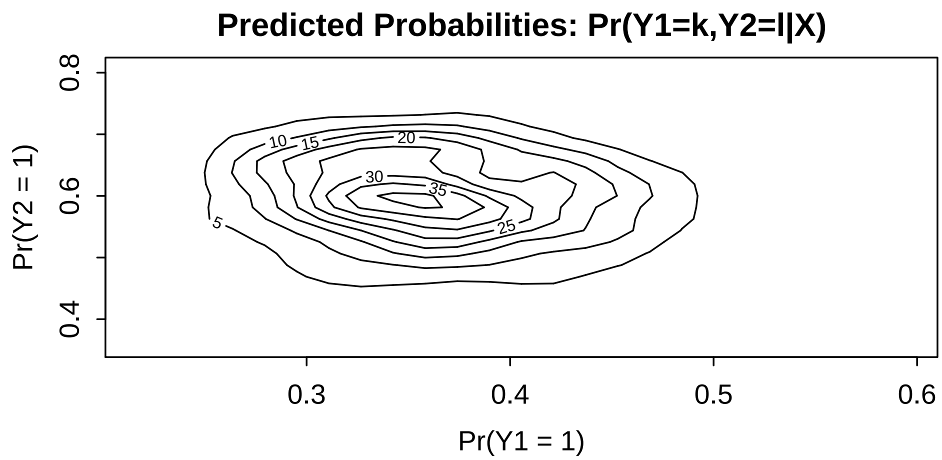
   Set the explanatory variables to their means:

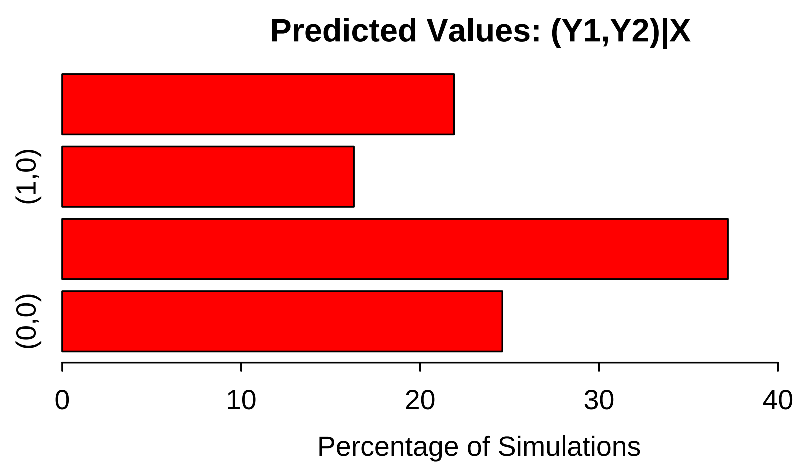
```
> x.out2 <- setx(z.out2)
```

   Simulate draws from the posterior distribution:

```
> s.out2 <- sim(z.out2, x = x.out2)
> summary(s.out2)

> plot(s.out2)
```





3. Joint Estimation of a Parametrically and Stochastically Dependent Model

   Using the sample data `sanction`. The bivariate model is parametrically dependent if Y_1 and Y_2 share some or all explanatory variables, *and* the effects of the shared explanatory variables are jointly estimated. For example,

```
> fml3 <- list(mu1 = import ~ tag(coop, "coop") + tag(cost, "cost") +
+      tag(target, "target"), mu2 = export ~ tag(coop, "coop") +
+      tag(cost, "cost") + tag(target, "target"))

> z.out3 <- zelig(fml3, model = "bprobit", data = sanction)
> summary(z.out3)
```

Note that this model only returns one parameter estimate for each of `coop`, `cost`, and `target`. Contrast this to Example 1 which returns two parameter estimates for each of the explanatory variables.

Set values for the explanatory variables:

```
> x.out3 <- setx(z.out3, cost = 1:4)
```

Draw simulated expected values:

```
> s.out3 <- sim(z.out3, x = x.out3)
> summary(s.out3)
```

Model

For each observation, define two binary dependent variables, Y_1 and Y_2, each of which take the value of either 0 or 1 (in the following, we suppress the observation index i). We model the joint outcome (Y_1, Y_2) using two marginal probabilities for each dependent variable, and the correlation parameter, which describes how the two dependent variables are related.

- The *stochastic component* is described by two latent (unobserved) continuous variables which follow the bivariate Normal distribution:

$$\begin{pmatrix} Y_1^* \\ Y_2^* \end{pmatrix} \sim N_2 \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right\},$$

  where μ_j is a mean for Y_j^* and ρ is a scalar correlation parameter. The following observation mechanism links the observed dependent variables, Y_j, with these latent variables

$$Y_j = \begin{cases} 1 & \text{if } Y_j^* \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

- The *systemic components* for each observation are

$$\begin{aligned} \mu_j &= x_j \beta_j \quad \text{for} \quad j = 1, 2, \\ \rho &= \frac{\exp(x_3 \beta_3) - 1}{\exp(x_3 \beta_3) + 1}. \end{aligned}$$

Quantities of Interest

For n simulations, expected values form an $n \times 4$ matrix.

- The expected values (`qi$ev`) for the binomial probit model are the predicted joint probabilities. Simulations of β_1, β_2, and β_3 (drawn form their sampling distributions) are substituted into the systematic components, to find simulations of the predicted joint probabilities $\pi_{rs} = \Pr(Y_1 = r, Y_2 = s)$:

$$
\begin{aligned}
\pi_{11} &= \Pr(Y_1^* \geq 0, Y_2^* \geq 0) &= \int_0^\infty \int_0^\infty \phi_2(\mu_1, \mu_2, \rho)\, dY_2^*\, dY_1^* \\
\pi_{10} &= \Pr(Y_1^* \geq 0, Y_2^* < 0) &= \int_0^\infty \int_{-\infty}^0 \phi_2(\mu_1, \mu_2, \rho)\, dY_2^*\, dY_1^* \\
\pi_{01} &= \Pr(Y_1^* < 0, Y_2^* \geq 0) &= \int_{-\infty}^0 \int_0^\infty \phi_2(\mu_1, \mu_2, \rho)\, dY_2^*\, dY_1^* \\
\pi_{11} &= \Pr(Y_1^* < 0, Y_2^* < 0) &= \int_{-\infty}^0 \int_{-\infty}^0 \phi_2(\mu_1, \mu_2, \rho)\, dY_2^*\, dY_1^*
\end{aligned}
$$

where r and s may take a value of either 0 or 1, ϕ_2 is the bivariate Normal density.

- The predicted values (`qi$pr`) are draws from the multinomial distribution given the expected joint probabilities.

- The first difference (`qi$fd`) in each of the predicted joint probabilities are given by

$$\text{FD}_{rs} = \Pr(Y_1 = r, Y_2 = s \mid x_1) - \Pr(Y_1 = r, Y_2 = s \mid x).$$

- The risk ratio (`qi$rr`) for each of the predicted joint probabilities are given by

$$\text{RR}_{rs} = \frac{\Pr(Y_1 = r, Y_2 = s \mid x_1)}{\Pr(Y_1 = r, Y_2 = s \mid x)}.$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \left\{Y_{ij}(t_i = 1) - E[Y_{ij}(t_i = 0)]\right\} \text{ for } j = 1, 2,$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{ij}(t_i = 0)]$, the counterfactual expected value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \left\{Y_{ij}(t_i = 1) - \widehat{Y_{ij}(t_i = 0)}\right\} \text{ for } j = 1, 2,$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_{ij}(t_i = 0)}$, the counterfactual predicted value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "bprobit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and obtain a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: the named vector of coefficients.
  - `fitted.values`: an $n \times 4$ matrix of the in-sample fitted values.
  - `predictors`: an $n \times 3$ matrix of the linear predictors $x_j \beta_j$.
  - `residuals`: an $n \times 3$ matrix of the residuals.
  - `df.residual`: the residual degrees of freedom.
  - `df.total`: the total degrees of freedom.
  - `rss`: the residual sum of squares.
  - `y`: an $n \times 2$ matrix of the dependent variables.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coef3`: a table of the coefficients with their associated standard errors and t-statistics.
  - `cov.unscaled`: the variance-covariance matrix.
  - `pearson.resid`: an $n \times 3$ matrix of the Pearson residuals.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation; otherwise the quantities are matrices). Available quantities are:
  - `qi$ev`: the simulated expected values (joint predicted probabilities) for the specified values of `x`.
  - `qi$pr`: the simulated predicted outcomes drawn from a distribution defined by the joint predicted probabilities.

- `qi$fd`: the simulated first difference in the predicted probabilities for the values specified in `x` and `x1`.
- `qi$rr`: the simulated risk ratio in the predicted probabilities for given `x` and `x1`.
- `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
- `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *bprobit* Zelig model use:

> Kosuke Imai,Gary King, and Olivia Lau. 2007. "bprobit: Bivariate Probit Regression for Two Dichotomous Dependent Variable," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The bivariate probit function is part of the VGAM package by Thomas Yee (Yee and Hastie 2003). In addition, advanced users may wish to refer to `help(vglm)` in the VGAM library. Additional documentation is available at http://www.stat.auckland.ac.nz/˜ yee.Sample data are from Martin (1992)

12.5 `chopit`: Compound Hierarchical Ordered Probit for Survey Vignettes

The Compound Hierarchical Ordered Probit (CHOPIT) model corrects for "differential item functioning" or "interpersonal comparability" in ordinal survey responses. Given a self-assessment question (such as, "How healthy are you? Excellent, good, fair, or poor."), different respondents may interpret the response categories in different ways, such that excellent health to one individual may be fair health to a hypochondriac. For each ordinal self-assessment to be corrected, the CHOPIT model requires one or more vignette question (such as a description of a hypothetical person's health, followed by the same response categories as the self-assessment), and a set of associated explanatory variables for the respondent. The key assumption of the approach is that the thresholds (which determine how respondents translate their views into the response categories) have the same effect for different questions asked of the same respondent, but may differ across respondents; the model uses a parametric specification to predict the thresholds associated with an individual. The self-assessment and vignette questions may be taken from different surveys, so long as both surveys include the same explanatory variable questions to predict the thresholds. For ordinal data (without vignettes), see Section 12.46, Section 12.45, and Section 12.47.

Syntax

```
> fml <- list(self = Y ~ X1 + X2,
              vign = cbind(Z1, Z2, Z3) ~ 1,
              tau  = ~ X1 + X2)
> z.out <- zelig(fml, data = list(self = data1, vign = data2),
                 model = "chopit")
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out, x1 = NULL)
```

Inputs

In this hierarchical model, the `formula` and `data` inputs to `zelig()` are lists with the following structure:

- The `formula` is a list with three `formula` objects corresponding to:
  - `self`: Specifies the self-response question (`Y`) as a function of a set of explanatory variables.
  - `vign`: Specifies the vignette questions on the left-hand side as a matrix in the form `cbind(Z1, Z2, Z3)`.
  - `tau`: Specifies explanatory variables that constrain the cut points across both the vignette and self-response questions. These explanatory variables do not necessarily need to overlap with the set of explanatory variables specified in the `self`

formula, but must be observed in both the `vign` and `self` data frames, described below.

- The `data` argument is a list of two data frames with
  - `self`: A data frame containing the self-response question(s) specified in the `self` formula and associated explanatory variables listed in the `self` and `tau` formulas.
  - `vign`: A data frame containing the vignette questions specified in the `vign` formula and associated explanatory variables listed in the `tau` formula.

Additional Inputs

In addition to the standard inputs, `zelig()` takes many additional options for compound hierarchical ordered probit regression, see `help(chopit)` and Wand et al. (2007, forthcoming) for details.

Examples

1. Basic Example

   Setting up the formula as a list for the self-response, vignettes, and the cut points (drawn from both the self-response and vignette data sets).

   ```
   > formula <- list(self = y ~ sex + age + educ + factor(country),
   +      vign = cbind(v1, v2, v3, v4, v5) ~ 1, tau = ~sex + age +
   +          educ + factor(country))
   ```

   Attaching the sample data sets. The `free1` data correspond to the self-response data, and the `free2` data correspond to the vignette subset. Note that the variables specified in the `tau` formula must be in both data sets.

   ```
   > data(free1, free2)
   ```

   ```
   > data <- list(self = free1, vign = free2)
   ```

   Estimating parameter values for the CHOPIT regression:

   ```
   > z.out <- zelig(formula, data = data, model = "chopit")
   ```

   Setting values for the explanatory variables to their default values:

   ```
   > x.out1 <- setx(z.out)
   ```

   Simulating quantities of interest from the sampling distribution.

   ```
   > s.out1 <- sim(z.out, x = x.out1)
   ```

```
> summary(s.out1)
```

2. Simulating First Differences

   Estimate the first difference in expected values between the average age (about 40 years old) and a 25 year old individual, with the other explanatory variables held at their default values:

   ```
   > x.out2 <- setx(z.out, age = 25)
   ```

   ```
   > s.out2 <- sim(z.out, x = x.out1, x1 = x.out2)
   ```

   ```
   > summary(s.out2)
   ```

3. Conditional prediction

   Conditional prediction generates expected values that are conditional on the observed self-response.

   ```
   > x.out3 <- setx(z.out, cond = TRUE)
   ```

   Since conditional prediction involves numeric integration, the procedure takes approximately one second per observation in `x.out3` on 64-bit R.

   ```
   > s.out3 <- sim(z.out, x = x.out3)
   ```

   ```
   > summary(s.out3)
   ```

Model

This model has two sets of response variables, one for the self-assessment and one for the vignettes. Let Y_i be the observed ordinal self-assessment for respondents $i = 1, \ldots, n$, and Z_{lj} be the ordinal vignette responses for individuals $l = 1, \ldots, L$ in the vignette subset for $j = 1, \ldots, J$ vignette questions, such that both $\{Y_i, Z_{lj}\}$ take integer values $k = 1, \ldots, K$ corresponding to the same ordinal assessment response categories.

- The *stochastic components* are described by unobserved continuous variables, Y_i^* and Z_{lj}^*, which follows normal distributions with mean μ_i and variance σ^2 in the case of Y_i^*, and mean θ_j and variance σ_j^2 in the case of each Z_{lj}^*. Using the default identification mechanism, the variance σ^2 for the self-assessment is fixed to 1. Thus,

  $$\begin{aligned} Y_i^* &\sim N(\mu_i, 1) \\ Z_{lj}^* &\sim N(\theta_j, \sigma_j^2) \end{aligned}$$

  such that each vignette response j has a scalar mean θ_j and variance σ_j^2 that does not vary over observations l. In cases where more than one self-response was administered to the same subject, an additional random effect may be included in the distribution of the latent Y_i^* in the form

$$Y_i^* \sim N(\mu_i, 1+\omega^2)$$

where the variance term is obtained via the proof described in Appendix A of King et al. (2004).

The observation mechanisms that divide the continuous $\{Y_i^*, Z_{lj}^*\}$ into the discrete $\{Y_i, Z_{lj}\}$ are

$$\begin{aligned} Y_i &= k \quad \text{if} \quad \tau_i^{k-1} \leq Y_i^* \leq \tau_i^k \text{ for } k = 1, \ldots, K \\ Z_{lj} &= k \quad \text{if} \quad \tau_l^{k-1} \leq Z_{lj}^* \leq \tau_l^k \text{ for } k = 1, \ldots, K \end{aligned}$$

where the threshold parameters τ vary over individuals $\{i, l\}$, but are subject to the following constraints within each individual: $\tau^p < \tau^q$ for all $p < q$ and $\tau_0 = -\infty$ and $\tau_K = \infty$.

- There are three *systematic components* in the model.
  - For the self-assessment component, let

    $$\mu_i = x_i \beta$$

    where x_i is the vector of q explanatory variables for observation i, and β is the associated vector of coefficients.
  - In addition, the threshold parameters also vary over individuals in the self-assessment component as follows

    $$\begin{aligned} \tau_i^1 &= v_i \gamma^1 \\ \tau_i^k &= \tau_i^{k-1} + \exp(v_i \gamma^k) \text{ for } k = 2, \ldots, K \end{aligned}$$

    where v_i is the vector of p explanatory variables for observation i, and γ^k for $k = 1, \ldots, K$ are the vectors of coefficients associated with each categorical response. Thus, the threshold parameters vary over individuals since v_i vary, and over response categories since the γ^k vary over the threshold parameters.
  - Similarly, the threshold parameters vary over individuals in the vignette component as follows

    $$\begin{aligned} \tau_l^1 &= v_l \gamma^1 \\ \tau_l^k &= \tau_l^{k-1} + \exp(v_l \gamma^k) \text{ for } k = 2, \ldots, K \end{aligned}$$

    where v_l is a vector of p explanatory variables for observation l in the vignette subset, and γ^k are restricted to be the same γ^k used to parameterize the threshold parameters for the self-assessment component.

As King et al. (2004) note, the interpersonal comparability of responses (or response consistency) is achieved by constraining γ^k to be the same in both the self-assessment and vignette components of the model. Note that the variables included in v_i and v_l are the same, but the observed values of those variables differ across the vignette and self-response samples.

Quantities of Interest

- The expected value (`qi$ev`) for the CHOPIT model is the expected value of the posterior density for the systematic component μ_i,

$$\text{EV} \;=\; E(\mu_i \mid x_i) \;=\; x_i\beta$$

  given draws of β from its sampling distribution.

- The first difference is the difference in the expected value of the posterior density for the systematic component μ_i given x_1 and x_0:

$$\text{FD} = E(\mu_i \mid x_1) - E(\mu_i \mid x_0).$$

- In conditional prediction models, the conditional expected values (`qi$cev`) are the expected value of the distribution of μ_i conditional on the observed self-assessment response Y_i, where

$$P(\mu_i \mid \tau_i, \beta, x_i, Y_i) \;=\; \prod_{k=1}^{K}[\Phi(\tau_i^k - \mu_i) - \Phi(\tau_i^{k-1} - \mu_i)] \times N(x_i\beta, x_i\widehat{V}(\widehat{\beta})x_i' + \widehat{\omega}^2)$$

  given the simulations of the threshold parameters calculated above, draws of β from its sampling distribution, and the estimated variance-covariance matrix for $\widehat{\beta}$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(..., model = "chopit")`, then you may examine the available information in `z.out` by using `names(z.out)`, see the estimated parameters by using `z.out$par`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `par`: the maximum likelihood parameter estimates for $\widehat{\gamma}^k$ for $k = 1, \ldots, K$ response categories, $\log(\widehat{\omega})$ (if estimated), $\log(\widehat{\sigma})$ (if estimated), $\log(\widehat{\sigma_j})$ for $j = 1, \ldots, J$ vignette questions, $\widehat{\theta}_j$, and $\widehat{\beta}$.
  - `chopit.hessian`: the estimated Hessian matrix, with rows and columns corresponding to the elements in `par`.

- `value`: the value of the log-likelihood at its maximum
- `counts`: the number of function and gradient calls to reach the maximum.
- `formula`: the formula for `self`, `vign`, and `tau` selected by the user.
- `call`: the call to `zelig()`.
- `...`: additional outputs described in `help(chopit)`.

- Typing `summary(z.out)` provides a useful summary of the output from `zelig()`, but no items can be extracted.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected values for the values specified in `x` and `x1`.
  - `qi$cev`: the simulated conditional expected value given `x`.

How to Cite

To cite the *chopit* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "chopit: Compound Hierarchical Ordered Probit for Survey Vignettes" in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The CHOPIT model is part of the anchors package by Jonathan Wand, Gary King, and Olivia Lau (Wand et al. 2007, forthcoming). Advanced users may wish to refer to `help(chopit)`, as well as King et al. (2004) and King and Wand (2007).

12.6 `cloglog.net`: Network Complementary Log Log Regression for Dichotomous Proximity Matrix Dependent Variables

Use network complementary log log regression analysis for a dependent variable that is a binary valued proximity matrix (a.k.a. sociomatricies, adjacency matrices, or matrix representations of directed graphs).

Syntax

```
> z.out <- zelig(y ~ x1 + x2, model = "cloglog.net", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Examples

1. Basic Example

   Load the sample data (see `?friendship` for details on the structure of the network dataframe):

   ```
   > data(friendship)
   ```

   Estimate model:

   ```
   > z.out <- zelig(friends ~ advice + prestige + perpower, model = "cloglog.net",
   +     data = friendship)
   > summary(z.out)
   ```

   Setting values for the explanatory variables to their default values:

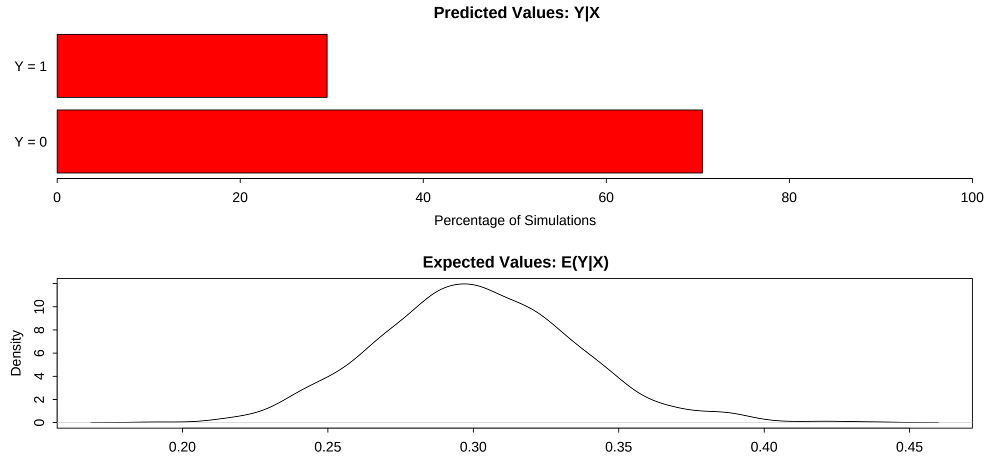
   ```
   > x.out <- setx(z.out)
   ```

   Simulating quantities of interest from the posterior distribution.

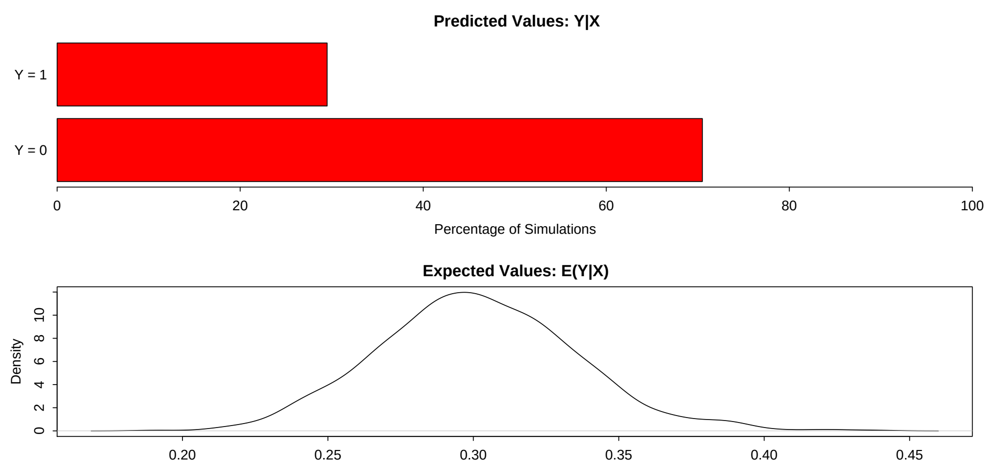
   ```
   > s.out <- sim(z.out, x = x.out)
   > summary(s.out)
   > plot(s.out)
   ```

2. Simulating First Differences

   Estimating the risk difference (and risk ratio) between low personal power (25th percentile) and high personal power (75th percentile) while all the other variables are held at their default values.




```
> x.high <- setx(z.out, perpower = quantile(friendship$perpower,
+      prob = 0.75))
> x.low <- setx(z.out, perpower = quantile(friendship$perpower,
+      prob = 0.25))
> s.out2 <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out2)
> plot(s.out2)
```



Model

The `cloglog.net` model performs a complementary log log regression of the proximity matrix $\mathbf{Y}$, a $m \times m$ matrix representing network ties, on a set of proximity matrices $\mathbf{X}$. This network regression model is directly analogous to standard complementary log log regression element-wise on the appropriately vectorized matrices. Proximity matrices are vectorized by creating

Y, a $m^2 \times 1$ vector to represent the proximity matrix. The vectorization which produces the Y vector from the $\mathbf{Y}$ matrix is performed by simple row-concatenation of $\mathbf{Y}$. For example, if $\mathbf{Y}$ is a 15×15 matrix, the $\mathbf{Y}_{1,1}$ element is the first element of Y, and the $\mathbf{Y}_{2,1}$ element is the second element of Y and so on. Once the input matrices are vectorized, standard complementary log log regression is performed.

Let Y_i be the binary dependent variable, produced by vectorizing a binary proximity matrix, for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$Y_i \sim \text{Bernoulli}(\pi_i)$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = 1 - \exp[\exp(-x_i\beta)]$$

  where x_i the vector of k explanatory variables for observation i and β is the vector of coefficients.

Quantities of Interest

The quantities of interest for the network complementary log log regression are the same as those for the standard complementary log log regression.

- The expected values (`qi$ev`) for the `cloglog.nett` model are simulations of the predicted probability of a success:

$$E(Y) = \pi_i = 1 - \exp[\exp(-x_i\beta)],$$

  given draws of β from its sampling distribution.

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the network complementary log log model is defined as

$$FD = \Pr(Y = 1|x_1) - \Pr(Y = 1|x)$$

Output Values

The output of each Zelig command contains useful information which you may view. For example, you run `z.out <- zelig(y ~ x, model = "cloglog.net", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `linear.predictors`: the vector of $x_i \beta$.
  - `aic`: Akaikeś Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `bic`: the Bayesian Information Criterion (minus twice the maximized log-likelihood plus the number of coefficients times $\log n$).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`
- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `mod.coefficients`: the parameter estimates with their associated standard errors, p-values, and t statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite

To cite the *cloglog.net* Zelig model:

> Skyler J. Cranmer. 2007. "cloglog.net: Social Network Complementary Log Log Regression for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The network complementary log log regression is part of the `netglm` package by Skyler J. Cranmer and is built using some of the functionality of the `sna` package by Carter T. Butts (Butts and Carley 2001).In addition, advanced users may wish to refer to `help(netbinom)`. Sample data are fictional.

12.7 `coxph`: Cox Proportional Hazards Regression for Duration Dependent Variables

Choose the Cox proportional hazards regression model if the values in your dependent variable are duration observations. The advantage of the semi-parametric Cox proportional hazards model over fully parametric models such as the exponential or Weibull models is that it makes no assumptions about the shape of the baseline hazard. The model only requires the proportional hazards assumption that the baseline hazard does not vary across observations. The baseline hazard can be estimated from the model via post-hoc analysis.

Syntax

```
> z.out <- zelig(Surv(Y, C) ~ X1 + X2, model = "coxph", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Cox proportional hazards models require that the dependent variable be in the form `Surv(Y, C)`, where `Y` and `C` are vectors of length n. For each observation i in 1, ..., n, the value y_i is the duration (lifetime, for example), and the associated c_i is a binary variable such that $c_i = 1$ if the duration is not censored (*e.g.*, the subject dies during the study) or $c_i = 0$ if the duration is censored (*e.g.*, the subject is still alive at the end of the study). If c_i is omitted, all Y are assumed to be completed; that is, c_i defaults to 1 for all observations.

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for Cox proportional hazards regression:

- `robust`: defaults to `FALSE`. If `TRUE, zelig()` computes robust standard errors based on sandwich estimators (see Huber (1981) and White (1980)) based on the options in `cluster`.
- `cluster`: if `robust = TRUE`, you may select a variable to define groups of correlated observations. Let `X3` be a variable that consists of either discrete numeric values, character strings, or factors that define the clusters. Then

  ```
  > z.out <- zelig(Surv(Y,C) ~ X1 + X2, robust = TRUE, cluster = "X3",
                   model = "coxph", data = mydata)
  ```

  means that the observations can be correlated within the clusters defined by the variable `X3`, and that robust standard errors should be calculated according to those clusters. If `robust = TRUE` but `cluster` is not specified, `zelig()` assumes that each observation falls into its own cluster.

- `method`: defaults to `"efron"`. Use this argument to specify how to handle ties within event times. The model assumes that no two event times should theoretically ever be the same, and any ties that occur are simply because the observation mechanism is not precise enough. In practice, ties often exist in the data so the model commonly uses one of three methods to deal with ties.

  - **Breslow method** (`method = "breslow"`): This method is the simplest computationally but also the least precise, especially as the number of tied events increases.
  - **Efron method** (`method = "efron"`): This is the default method and is more intensive computationally but also more precise than the Breslow method.
  - **Exact discrete method** (`method = "exact"`): This is the preferred method if the number of distinct events is rather small due to a large number of ties. Although it can be very computationally intensive, the exact discrete method, which computes the exact partial likelihood, is the most precise method when there are many ties.

Stratified Cox Model

In addition, `zelig()` also supports the stratified Cox model, where the baseline hazards are assumed to be different across different strata but the coefficients are restricted to be the same across strata. Let `id` be a variable that consists of either discrete numeric values, character strings, or factors that define the strata. Then the stratified Cox model can be estimated using `strata()` in the formula. The user can then find quantities of interest for a specific stratum by defining the stratum of choice in `setx()`. If no strata are defined, `setx` takes the mode. Strata on `setx` are defined as followed:

- If strata were defined by a variable (`strata(id)`), then strata should be defined as `strata = "id=5"`.
- If strata were defined by a mathematical expression (`strata(id>10)`), then strata should be defined as `strata = "id>10=TRUE"` or `strata = "id>10=FALSE"`.

```
> z.out <- zelig(Surv(Y,C) ~ X1 + X2 + strata(id), model = "coxph",
                 data = mydata)
> x.out <- setx(z.out, strata = "id=5")
> s.out <- sim(z.out, x = x.out)
```

Time-Varying Covariates

`zelig()` also supports the use of time-varying covariates for the Cox model, where some or all of the covariates change over time for each case. Let "case" refer to each unit in the data. Then each case can have one or more "observations", where each observation has a different value for one or more covariates for a specific case.

Estimating a time-varying covariate model with `zelig()` involves setting up the data differently to reflect a counting process. In the typical non-time-varying covariate model, the cases include a duration time (Y), a censoring mechanism (C), and covariates (X). A typical dataset would look like this:

Case	Y	C	X1	X2
1	35	0	4	7
2	56	1	6	11

The user would then estimate the model and find quantities of interest using the following syntax:

```
> z.out <- zelig(Surv(Y,C) ~ X1 + X2, model = "coxph", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

With time-varying covariates, each case is composed of multiple observations with start times, stop times, censoring (event) mechanisms, and covariates. The covariates are assumed to be constant within the intervals defined by the start and stop times. The covariates change only between intervals. Thus, the covariates are constant at each observation. The censoring mechanism equals 1 when an event occurs at the stop time and equals 0 if the observation is censored or if no event occurs at the stop time. A typical time-varying dataset would look like this:

Case	Start	Stop	C	X1	X2
1	0	26	0	4	7
1	26	35	0	4	10
2	0	39	0	6	11
2	39	56	1	9	5

The user would then estimate the model and find quantities of interest using the following syntax:

```
> z.out <- zelig(Surv(Start,Stop,C) ~ X1 + X2, model = "coxph",
                 data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Examples

1. Example 1: Basic Example

   Attaching the sample dataset:

   ```
   > data(coalition)
   ```

   Estimating parameter values for the coxph regression:

   ```
   > z.out1 <- zelig(Surv(duration, ciep12) ~ invest + numst2 + crisis,
   +     robust = TRUE, cluster = "polar", model = "coxph", data = coalition)
   ```

   Setting values for the explanatory variables:

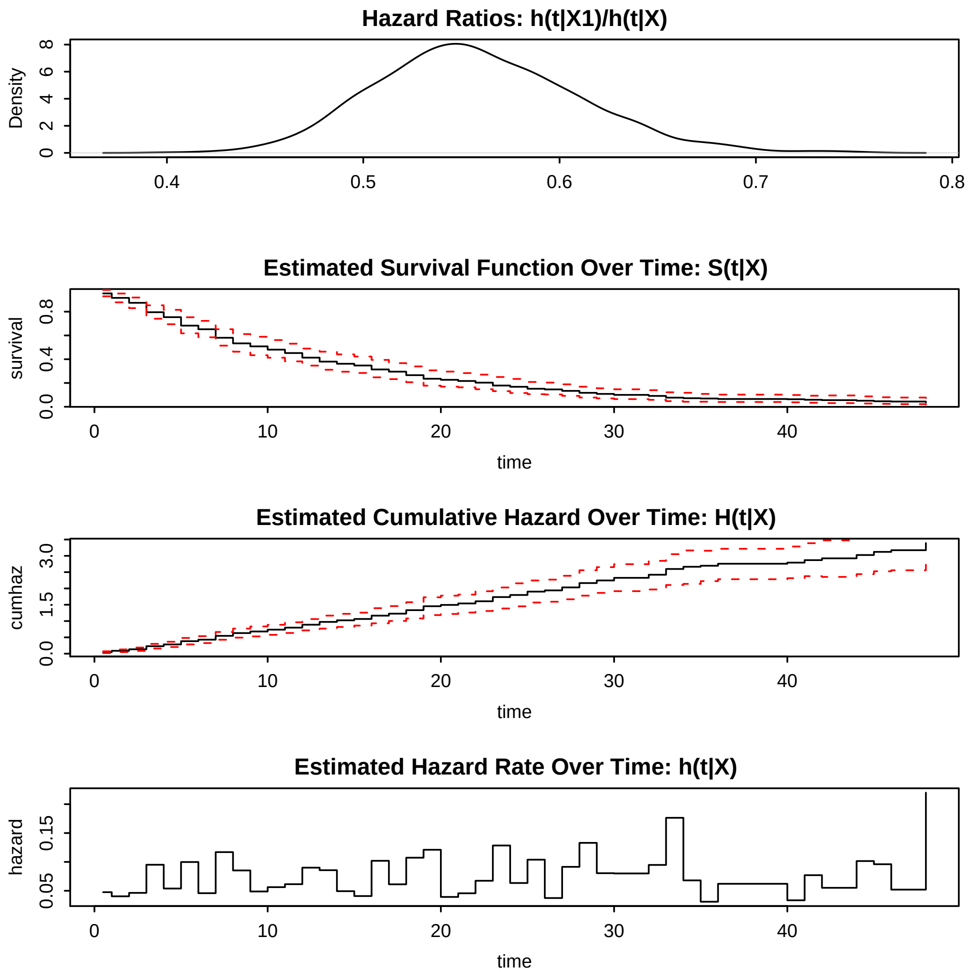
   ```
   > x.low1 <- setx(z.out1, numst2 = 0)
   > x.high1 <- setx(z.out1, numst2 = 1)
   ```

   Simulating quantities of interest:

   ```
   > s.out1 <- sim(z.out1, x = x.low1, x1 = x.high1)
   ```

   ```
   > summary(s.out1)
   ```

   ```
   > plot(s.out1)
   ```



2. Example 2: Example with Stratified Cox Model

   Estimating parameter values for the stratified coxph regression:

```
> z.out2 <- zelig(Surv(duration, ciep12) ~ invest + strata(polar) +
+      numst2 + crisis, model = "coxph", data = coalition)
```

   Setting values for the explanatory variables:

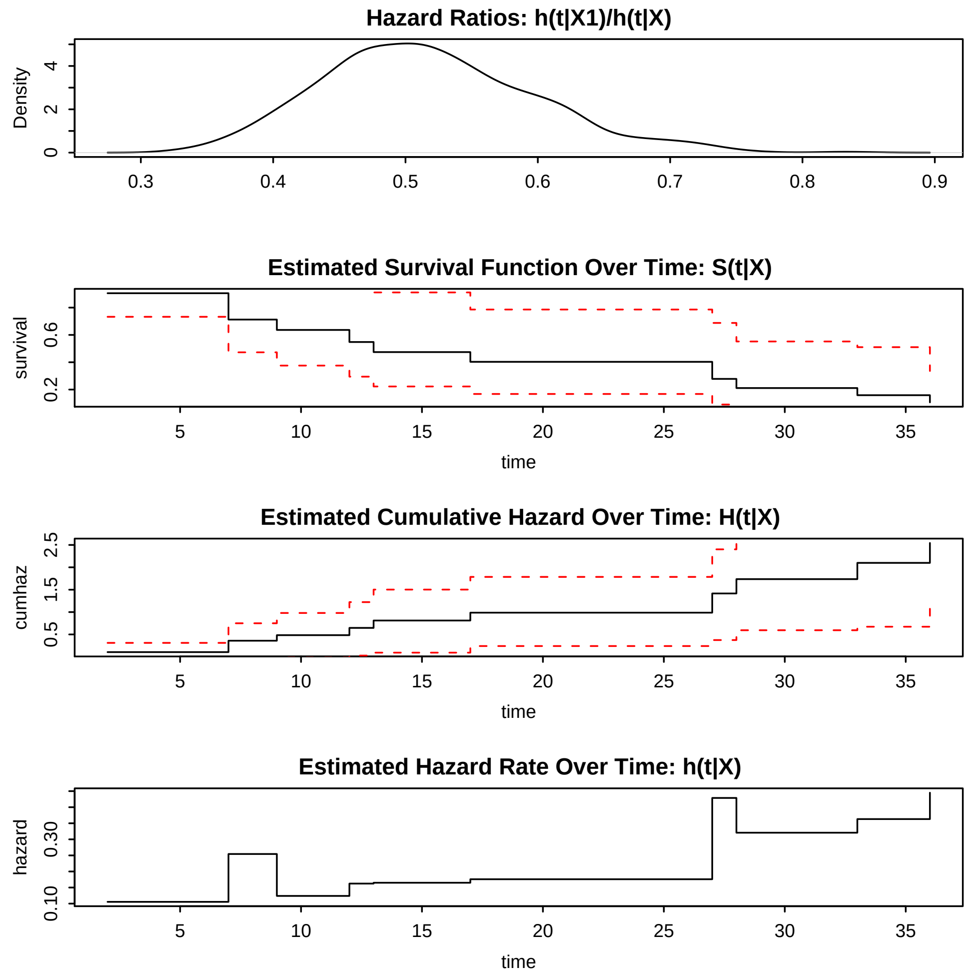
```
> x.low2 <- setx(z.out2, numst2 = 0, strata = "polar=3")
> x.high2 <- setx(z.out2, numst2 = 1, strata = "polar=3")
```

   Simulating quantities of interest:

```
> s.out2 <- sim(z.out2, x = x.low2, x1 = x.high2)

> summary(s.out2)
```

```
> plot(s.out2)
```



3. Example 3: Example with Time-Varying Covariates

   Create sample toy dataset (from `survival` package):

```
> toy <- as.data.frame(list(start = c(1, 2, 5, 2, 1, 7, 3, 4, 8,
+      8), stop = c(2, 3, 6, 7, 8, 9, 9, 9, 14, 17), event = c(1,
+      1, 1, 1, 1, 1, 1, 0, 0, 0), x = c(1, 0, 0, 1, 0, 1, 1, 1,
+      0, 0), x1 = c(5, 5, 7, 4, 5, 6, 3, 2, 7, 4)))
```

   Estimating parameter values for the coxph regression:

```
> z.out3 <- zelig(Surv(start, stop, event) ~ x + x1, model = "coxph",
+      data = toy)
```

   Setting values for the explanatory variables:

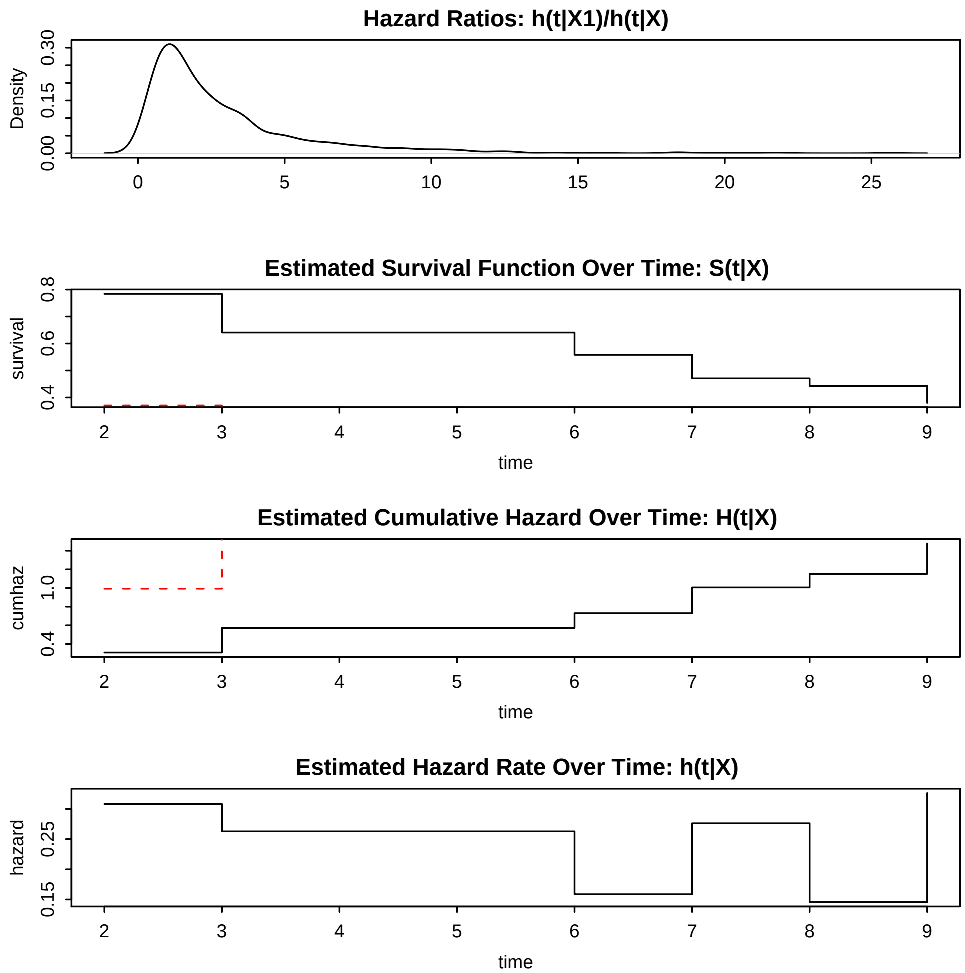
```
> x.low3 <- setx(z.out3, x = 0)
> x.high3 <- setx(z.out3, x = 1)
```

Simulating quantities of interest:

```
> s.out3 <- sim(z.out3, x = x.low3, x1 = x.high3)
```

```
> summary(s.out3)
```

```
> plot(s.out3)
```



The Model

Let Y_i^* be the survival time for observation i. This variable might be censored for some observations at a fixed time y_c such that the fully observed dependent variable, Y_i, is defined as

$$Y_i = \begin{cases} Y_i^* & \text{if } Y_i^* \leq y_c \\ y_c & \text{if } Y_i^* > y_c \end{cases}$$

- The *stochastic component* is described by the distribution of the partially observed variable Y^*:

$$Y_i^* \sim f(y_i^*|\mu_i, \alpha)$$

where f is an unspecified distribution with some mean μ_i and shape α. In the Cox proportional hazards model, the distributional form of the duration times is unknown and left unparameterized. Instead it uses the proportional hazards assumption to model the set of (ordered) event times on particular covariates.

An important component of all survival models is the hazard function $h(t)$, which measures the probability of an observation not surviving past time t given survival up to t. The hazard function is given by

$$h_i(t) = \lambda(t) \times \lambda_i$$

where $\lambda(t)$ is the baseline hazard (when all covariates equal 0), which varies over t but not over i, and λ_i is the parameterized part of the hazard function, which varies over i but not over t (the proportional hazards assumption).

The model estimates the parameters without a distributional assumption on the duration times by focusing on the occurrence of events and ignoring the time between events. The data are reconceptualized from duration times to K discrete event times such that each y_i corresponds to exactly one event time t_i. The model assumes that no two y_i have the same event times.

For each event time, denote $R(t_i)$ as the set of all observations j that are at risk at t_i. Given that an event occurred at t_i, we are interested in the conditional probability that the event occurred in observation i. The conditional probability is given by

$$\begin{aligned}
\Pr(y_i = t_i \mid \text{an event at } t_i) &= \frac{h_i(t_i)}{\sum_{j \in R(t_i)} h_j(t_i)} \\
&= \frac{\lambda(t_i)\,\lambda_i}{\sum_{j \in R(t_i)} \lambda(t_i)\,\lambda_j} \\
&= \frac{\lambda_i}{\sum_{j \in R(t_i)} \lambda_j}
\end{aligned}$$

where the numerator denotes the probability of observation i experiencing the event at t_i and the denominator denotes the probability that an event occurred at t_i.

- The *systematic component* λ_i is modeled as

$$\lambda_i = \exp(x_i\beta)$$

where x_i is the vector of explanatory variables, and β is the vector of coefficients.

- Each risk set (and thus each event time) contributes one conditional probability to the partial likelihood function, given by

$$L(\beta|y) = \prod_{i=1}^{K} \left[\frac{\exp(x_i \beta)}{\sum_{j \in R(t_i)} \exp(x_j \beta)} \right]^{c_i}$$

  where c_i is the binary censoring variable. Note that event times corresponding to censored observations are not counted since their corresponding terms for the partial likelihood are exponentiated to 0. However, all censored observations are considered part of the risk sets $R(t_i)$ for all event times prior to their censoring, but otherwise do not contribute to the partial likelihood. For an example, see Box-Steffensmeier and Jones (2004, 53).

- In the case of the Cox model with time-varying covariates, the partial likelihood function is similarly given by

$$L(\beta|y) = \prod_{i=1}^{K} \left[\frac{\exp(x_i(t_i)\ \beta)}{\sum_{j \in R(t_i)} \exp(x_j(t_i)\ \beta)} \right]^{c_i}$$

  where $x_i(t_i)$ is the value of the covariates at time t_i. Denote "cases" as the units in our data. Each case is composed of one or more observations corresponding to different values in one or more covariates. At each event time t_i, the partial likelihood evaluates the hazard of the case in which the event occurred in with its covariate values at t_i (the numerator) and the hazard of all the other cases at risk at t_i (risk set $R(t_i)$) with their covariate values at t_i (the denominator). See previous section for more information.

- Although the model assumes that there are no tied event times, in practice, data often have tied event times due to imprecise measurement. There are three commonly used methods to deal with tied event times.

  - **Breslow method**: The Breslow method simply treats the risk set as the same for all tied events in the risk set. Suppose observations 1 and 3 are tied in a risk set of observations 1, 2, 3, and 4. Theoretically, if the event occurred in 1 before in 3, then the risk set for observation 3 would have dropped observation 1. However, since we cannot tell which event occurred first, in the partial likelihood, the risk set for observation 1 and observation 3 are the same, consisting of both observations 1 and 3 as well as 2 and 4. For each risk set $R(t_i)$, let d_i equal the number of tied events in the ith risk set and let D_i denote the set of d_i tied events. For risk sets with no tied events, $d_i = 1$. The approximate partial likelihood for the Breslow method is given by

$$L(\beta|y) = \prod_{i=1}^{K} \left[\frac{\prod_{i \in D_i} \exp(x_i \beta)}{\left[\sum_{j \in R(t_i)} \exp(x_j \beta)\right]^{d_i}} \right]^{c_i}$$

- **Efron method**: The Efron method is more precise because it tries to account for how the risk set changes depending on the sequence of tied events. For an intuition behind the Efron approximation, suppose as in the previous example that observations 1 and 3 are tied in a risk set of observations 1, 2, 3, and 4. If the event occurred in 1 before 3, then the risk set for the second event would consist of observations $\{2, 3, 4\}$. On the other hand, if the event occurred in 3 before 1, then the risk set for the second event would consist of observations $\{1, 2, 4\}$. Since both cases are equally plausible with the tied event times, the Efron approximation suggests that the second risk set would consist of $\{2, 3, 4\}$ with 0.5 probability and $\{1, 2, 4\}$ with 0.5 probability. The Efron approximate partial likelihood is then given by

$$L(\beta|y) = \prod_{i=1}^{K} \left[\frac{\prod_{i \in D_i} \exp(x_i\beta)}{\prod_{r=1}^{d_i} \left[\sum_{j \in R(t_i)} \exp(x_j\beta) - \frac{r-1}{d_i} \sum_{j \in D_i} \exp(x_j\beta) \right]} \right]^{c_i}$$

  where r indexes D_i, which is the set of d_i tied events for the ith risk set.

- **Exact discrete method**: Unlike the Breslow and Efron methods, which assume a continuous time process, the exact discrete method assumes a discrete time process where the tied events actually do occur at exactly the same time. The method begins by assuming that the data are grouped into risk sets $R(t_i)$. In each risk set and for each observation, denote a binary dependent variable which takes on the value of 1 for each observation that experiences the event and 0 for each observation that does not experience the event. Denote d_i as the number of 1s in $R(t_i)$ and D_i as the set of observations with 1s in $R(t_i)$. D_i represents a specific pattern of 0s and 1s (in our previous example, the specific pattern of 0s and 1s is that observations 1 and 3 experienced an event while 2 and 4 did not, so D_i is the set $\{1, 3\}$). Then for each $R(t_i)$, we are interested in the conditional probability of getting the specific pattern of 0s and 1s given the total number of 1s in $R(t_i)$. Thus, the conditional probability for each risk set is given as

$$\Pr(D_i|d_i) = \frac{\prod_{i \in D_i} \exp(x_i\beta)}{\sum_{m=1}^{M} \left[\prod_{j \in A_{im}} \exp(x_j\beta) \right]}$$

  where A_{im} is a set of observations that represents one combination of d_i number of 1s in $R(t_i)$. There are M possible combinations for each risk set. The partial likelihood then takes the conditional probability over each i risk set. Note that the exact discrete approximation method is equivalent to a conditional logit model.

Quantities of Interest

- The hazard ratio (`qi$hr`) is defined as

$$\text{HR} = \frac{h(t \mid x_1)}{h(t \mid x)} = \frac{\lambda(t)\exp(x_1\beta)}{\lambda(t)\exp(x\beta)} = \frac{\exp(x_1\beta)}{\exp(x\beta)}$$

given draws of β from its sampling distribution, where x and x_1 are values of the independent variables chosen by the user. Typically, x and x_1 should only differ over one independent variable to interpret the effect of that variable on the hazard rate. In a stratified Cox model, the strata should be the same in both x and x_1.

- The survival function (`qi$survival`) is defined as the fraction of observations surviving past time t. It is derived from the cumulative hazard function (`exp(-cumhaz)`). The confidence interval of the survival function is drawn on the `log(survival)` scale.
- The cumulative hazard function (`qi$cumhaz`) is defined as `-log(survival)`. Although there is no direct interpretation, the cumulative hazard function is estimated from the data and then other quantities of interest are derived from the cumulative hazard function.
- The hazard function (`qi$hazard`) is defined as the probability of an observation not surviving past time t given survival up to t. It is derived directly from the cumulative hazard function.
- For MI data, if survival times are multiply imputed, we suggest having a larger number of imputed datasets. Because the quantities of interest are derived semi-parametrically, there may be instances in which survival times appear only in one or a small fraction of the multiply imputed datasets, which may bias the results.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(Surv(y,c) ~ x, model = "coxph", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `var`: the variance-covariance matrix.
  - `residuals`: the working residuals of the fit.
  - `loglik`: the log-likelihood for the baseline and full models
  - `linear.predictors`: a mean-adjusted linear predictor $x_i\beta$, where $x_i = x_i - \text{mean}(x)$.
- From `summary(z.out)`, you may extract:
  - `coef`: the parameter estimates with their associated standard errors, p-values, and z-statistics.

- `conf.int`: $\exp(\beta)$ and their associated confidence intervals.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation × `x`-observation (for more than one `x`-observation). Available quantities are:

  - `qi$hr`: the simulated hazard ratios for the specified values of `x` and `x1`.
  - `qi$survival`: the estimated survival function for the values specified in `x`.
  - `qi$cumhaz`: the estimated cumulative hazard function for the values specified in `x`.
  - `qi$hazard`: the estimated hazard function for the values specified in `x`.

How To Cite

To cite the *coxph* Zelig model:

> Patrick Lam. 2007. "coxph: Cox Proportional Hazards Regression for Duration Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

See also

The Cox proportional hazards model is part of the survival library by Terry Therneau (Therneau and Grambsch 2000), ported to R by Thomas Lumley. Advanced users may wish to refer to `help(coxph)` and `help(survfit)` in the survival library. Sample data are from King et al. (1990b)

12.8 `ei.dynamic`: Quinn's Dynamic Ecological Inference Model

Given contingency tables with observed marginals, ecological inference (EI) models estimate each internal cell value for each table. Quinn's dynamic EI model estimates a dynamic Bayesian model for 2×2 tables with temporal dependence across tables (units). The model is implemented using a Markov Chain Monte Carlo algorithm (via a combination of slice and Gibbs sampling). For a hierarchical Bayesian implementation of EI see Quinn's dynamic EI model (Section 12.9). For contingency tables larger than 2 rows by 2 columns, see R×C EI (Section ??).

Syntax

```
> z.out <- zelig(cbind(t0, t1) ~ x0 + x1, N = NULL,
                 model = "MCMCei.dynamic", data = mydata)
> x.out <- setx(z.out, fn = NULL, cond = TRUE)
> s.out <- sim(z.out, x = x.out)
```

Inputs

- `t0`, `t1`: numeric vectors (either counts or proportions) containing the column margins of the units to be analyzed.
- `x0`, `x1`: numeric vectors (either counts or proportions) containing the row margins of the units to be analyzed.
- `N`: total counts in each contingency table (unit). If `t0`,`t1`, `x0` and `x1` are proportions, you must specify `N`.

Additional Inputs

In addition, `zelig()` accepts the following additional inputs for `ei.dynamic` to monitor the convergence of the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 5,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults to 50,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.
- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.

- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed of 12345.

The model also accepts the following additional arguments to specify priors and other parameters:

- `W`: a $p \times p$ numeric matrix describing the structure of the temporal dependence among elements of θ_0 and θ_1. The default value is 0, which constructs a weight matrix corresponding to random walk priors for θ_0 and θ_1 (assuming that the tables are equally spaced throughout time, and that the elements of `t0`, `t1`,`x0`,`x1` are temporally ordered).
- `a0`: $a_0/2$ is the shape parameter for the Inverse Gamma prior on σ_0^2. The default is 0.825.
- `b0`: $b_0/2$ is the scale parameter for the Inverse Gamma prior on σ_0^2. The default is 0.0105.
- `a1`: $a_1/2$ is the shape parameter for the Inverse Gamma prior on σ_1^2. The default is 0.825.
- `b1`: $b_1/2$ is the scale parameter for the Inverse Gamma prior on σ_1^2. The default is 0.0105.

Users may wish to refer to `help(MCMCdynamicEI)` for more options.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic examples

   Attaching the example dataset:

   ```
   > data(eidat)
   ```

   Estimating the model using `ei.dynamic`:

   ```
   > z.out <- zelig(cbind(t0, t1) ~ x0 + x1, model = "ei.dynamic",
   +      data = eidat, mcmc = 40000, thin = 10, burnin = 10000, verbose = TRUE)
   > summary(z.out)
   ```

   Setting values for in-sample simulations given the marginal values of `t0`, `t1`, `x0`, and `x1`:

   ```
   > x.out <- setx(z.out, fn = NULL, cond = TRUE)
   ```

   In-sample simulations from the posterior distribution:

   ```
   > s.out <- sim(z.out, x = x.out)
   ```

   Summarizing in-sample simulations at aggregate level weighted by the count in each unit:

   ```
   > summary(s.out)
   ```

   Summarizing in-sample simulations at unit level for the first 5 units:

   ```
   > summary(s.out, subset = 1:5)
   ```

Model

Consider the following 2×2 contingency table for the racial voting example. For each geographical unit $i = 1, \ldots, p$, the marginals t_i^0, t_i^1, x_i^0, and x_i^1 are known, and we would like to estimate n_i^{00}, n_i^{01}, n_i^{10}, and n_i^{11}.

	No Vote	Vote	
Black	n_i^{00}	n_i^{01}	x_i^0
White	n_i^{10}	n_i^{11}	x_i^1
	t_i^0	t_i^1	N_i

The marginal values x_i^0, x_i^1, t_i^0, t_i^1 are observed as either counts or fractions. If fractions, the counts can be obtained by multiplying by the total counts per table $N_i = n_i^{00}+n_i^{01}+n_i^{10}+n_i^{11}$, and rounding to the nearest integer. Although there are four internal cells, only two unknowns are modeled since $n_i^{01} = x_i^0 - n_i^{00}$ and $n_i^{11} = s_i^1 - n_i^{10}$.

The hierarchical Bayesian model for ecological inference in 2×2 is illustrated as following:

- The *stochastic component* of the model assumes that

$$\begin{aligned} n_i^{00} \mid x_i^0, \beta_i^b &\sim \text{Binomial}\left(x_i^0, \beta_i^b\right), \\ n_i^{10} \mid x_i^1, \beta_i^w &\sim \text{Binomial}\left(x_i^1, \beta_i^w\right) \end{aligned}$$

  where β_i^b is the fraction of the black voters who vote and β_i^w is the fraction of the white voters who vote. β_i^b and β_i^w as well as their aggregate summaries are the focus of inference.

- The *systematic component* of the model is

$$\begin{aligned} \beta_i^b &= \frac{\exp \theta_i^0}{1 - \exp \theta_i^0} \\ \beta_i^w &= \frac{\exp \theta_i^1}{1 - \exp \theta_i^1} \end{aligned}$$

  The logit transformations of β_i^b and β_i^w, θ_i^0, and θ_i^1 now take value on the real line. (Future versions may allow β_i^b and β_i^w to be functions of observed covariates.)

- The *priors* for θ_i^0 and θ_i^1 are given by

$$\begin{aligned} \theta_i^0 \mid \sigma_0^2 &\propto \frac{1}{\sigma_0^p} \exp\left(-\frac{1}{2\sigma_0^2}\theta_0' P \theta_0\right) \\ \theta_i^1 \mid \sigma_1^2 &\propto \frac{1}{\sigma_1^p} \exp\left(-\frac{1}{2\sigma_1^2}\theta_1' P \theta_1\right) \end{aligned}$$

where P is a $p \times p$ matrix whose off diagonal elements P_{ts} $(t \neq s)$ equal $-W_{ts}$ (the negative values of the corresponding elements of the weight matrix W), and diagonal elements $P_{tt} = \sum_{s \neq t} W_{ts}$. Scale parameters σ_0^2 and σ_1^2 have hyperprior distributions as given below.

- The *hyperpriors* for σ_0^2 and σ_1^2 are given by

$$\begin{aligned}\sigma_0^2 &\sim \text{Inverse Gamma}\left(\frac{a_0}{2}, \frac{b_0}{2}\right), \\ \sigma_1^2 &\sim \text{Inverse Gamma}\left(\frac{a_1}{2}, \frac{b_1}{2}\right),\end{aligned}$$

  where $a_0/2$ and $a_1/2$ are the shape parameters of the (independent) Gamma distributions while $b_0/2$ and $b_1/2$ are the scale parameters.

  The default hyperpriors for μ_0, μ_1, σ_0^2, and σ_1^2 are chosen such that the prior distributions for β^b and β^w are flat.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
> z.out <- (cbind(t0, t1) ~ x0 + x1, N = NULL,
          model = "ei.dynamic", data = mydata)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the quantities of interest by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the parameters.
  - `data`: the name of the input data frame.
  - `N`: the total counts when the inputs are fractions.
  - `seed`: the random seed used in the model.
- From `summary(z.out)`, you may extract:
  - `summary`: a matrix containing the summary information of the posterior estimation of β_i^b andβ_i^w for each unit and the parameters μ_0, μ_1, σ_1 and σ_2 based on the posterior distribution. The first p rows correspond to β_i^b, $i = 1, \ldots p$, the row

names are in the form of `p0table`i. The $(p+1)$-th to the $2p$-th rows correspond to β_i^w, $i = 1, \ldots, p$. The row names are in the form of `p1table`i. The last four rows contain information about μ_0, μ_1, σ_0^2 and σ_1^2, the prior means and variances of θ_0 and θ_1.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays indexed by simulation $\times$ column $\times$ row $\times$ observation, where column and row refer to the column dimension and the row dimension of the ecological table, respectively. In this model, only 2×2 contingency tables are analyzed, hence column$=2$ and row$=2$ in all cases. Available quantities are:
  - `qi$ev`: the simulated expected values of each internal cell given the observed marginals.
  - `qi$pr`: the simulated expected values of each internal cell given the observed marginals.

How to Cite

To cite the *ei.dynamic* Zelig model use:

> Ben Goodrich and Ying Lu. 2007. "ei.dynamic: Quinn's Dynamic Ecological Inference," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

ei.dynamic function is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005). Sample date are adapted from Martin and Quinn (2005).

12.9 ei.hier: Hierarchical Ecological Inference Model for 2×2 Tables

Given contingency tables with observed marginals, ecological inference (EI) models estimate each internal cell value for each table. The hierarchical EI model estimates a Bayesian model for 2×2 tables. The model is implemented using a Markov Chain Monte Carlo algorithm (via a combination of slice and Gibbs sampling). For a Bayesian implementation of EI that accounts for temporal dependence, see Quinn's dynamic EI model (Section 12.8). For contingency tables larger than 2 rows by 2 columns, see R×C EI (Section **??**).

Syntax

```
> z.out <- zelig(cbind(t0, t1) ~ x0 + x1, N = NULL,
                 model = "MCMCei.hier", data = mydata)
> x.out <- setx(z.out, fn = NULL, cond = TRUE)
> s.out <- sim(z.out, x = x.out)
```

Inputs

- `t0`, `t1`: numeric vectors (either counts or proportions) containing the column margins of the units to be analyzed.
- `x0`, `x1`: numeric vectors (either counts or proportions) containing the row margins of the units to be analyzed.
- `N`: total counts per contingency table (unit). If `t0`,`t1`, `x0` and `x1` are proportions, you must specify `N`.

Additional Inputs

In addition, `zelig()` accepts the following additional inputs for `ei.hier` to monitor the convergence of the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 5,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults to 50,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.
- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.
- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed of 12345.

The model also accepts the following additional arguments to specify prior parameters used in the model:

- `m0`: prior mean of μ_0 (defaults to 0).
- `M0`: prior variance of μ_0 (defaults to 2.287656).
- `m1`: prior mean of μ_1 (defaults to 0).
- `M1`: prior variance of μ_1 (defaults to 2.287656).
- `a0`: $a_0/2$ is the shape parameter for the Inverse Gamma prior on σ_0^2 (defaults to 0.825).
- `b0`: $b_0/2$ is the scale parameter for the Inverse Gamma prior on σ_0^2 (defaults to 0.0105).
- `a1`: $a_1/2$ is the shape parameter for the Inverse Gamma prior on σ_1^2 (defaults to 0.825).
- `b1`: $b_1/2$ is the scale parameter for the Inverse Gamma prior on σ_1^2 (defaults to 0.0105).

Users may wish to refer to `help(MCMChierEI)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.

- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.

- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic examples
   Attaching the example dataset:

```
> data(eidat)
> eidat
```

   Estimating the model using `ei.hier`:

```
> z.out <- zelig(cbind(t0, t1) ~ x0 + x1, model = "ei.hier", data = eidat,
+     mcmc = 40000, thin = 10, burnin = 10000, verbose = TRUE)
> summary(z.out)
```

   Setting values for in-sample simulations given marginal values of `x0`, `x1`, `t0`, and `t1`:

```
> x.out <- setx(z.out, fn = NULL, cond = TRUE)
```

   In-sample simulations from the posterior distribution:

```
> s.out <- sim(z.out, x = x.out)
```

   Summarizing in-sample simulations at aggregate level weighted by the count in each unit:

```
> summary(s.out)
```

   Summarizing in-sample simulations at unit level for the first 5 units:

```
> summary(s.out, subset = 1:5)
```

Model

Consider the following 2×2 contingency table for the racial voting example. For each geographical unit $i = 1, \ldots, p$, the marginals t_i^0, t_i^1, x_i^0, and x_i^1 are known, and we would like to estimate n_i^{00}, n_i^{01}, n_i^{10}, and n_i^{11}.

	No Vote	Vote	
Black	n_i^{00}	n_i^{01}	x_i^0
White	n_i^{10}	n_i^{11}	x_i^1
	t_i^0	t_i^1	N_i

The marginal values x_i^0, x_i^1, t_i^0, t_i^1 are observed as either counts or fractions. If fractions, the counts can be obtained by multiplying by the total counts per table $N_i = n_i^{00} + n_i^{01} + n_i^{10} + n_i^{11}$ and rounding to the nearest integer. Although there are four internal cells, only two unknowns are modeled since $n_i^{01} = x_i^0 - n_i^{00}$ and $n_i^{11} = s_i^1 - n_i^{10}$.

The hierarchical Bayesian model for ecological inference in 2×2 is illustrated as following:

- The *stochastic component* of the model assumes that

$$\begin{aligned} n_i^{00} \mid x_i^0, \beta_i^b &\sim \text{Binomial}\left(x_i^0, \beta_i^b\right), \\ n_i^{10} \mid x_i^1, \beta_i^w &\sim \text{Binomial}\left(x_i^1, \beta_i^w\right) \end{aligned}$$

  where β_i^b is the fraction of the black voters who vote and β_i^w is the fraction of the white voters who vote. β_i^b and β_i^w as well as their aggregate level summaries are the focus of inference.

- The *systematic component* is

$$\begin{aligned} \beta_i^b &= \frac{\exp \theta_i^0}{1 - \exp \theta_i^0} \\ \beta_i^w &= \frac{\exp \theta_i^1}{1 - \exp \theta_i^1} \end{aligned}$$

  The logit transformations of β_i^b and β_i^w, θ_i^0, and θ_i^1 now take value on the real line. (Future versions may allow β_i^b and β_i^w to be functions of observed covariates.)

- The *priors* for θ_i^0 and θ_i^1 are given by

$$\begin{aligned} \theta_i^0 \mid \mu_0, \sigma_0^2 &\sim \text{Normal}\left(\mu_0, \sigma_0^2\right), \\ \theta_i^1 \mid \mu_1, \sigma_1^2 &\sim \text{Normal}\left(\mu_1, \sigma_1^2\right) \end{aligned}$$

  where μ_0 and μ_1 are the means, and σ_0^2 and σ_1^2 are the variances of the two corresponding (independent) normal distributions.

- The *hyperpriors* for μ_0 and μ_1 are given by

$$\begin{aligned} \mu_0 &\sim \text{Normal}\,(m_0, M_0)\,, \\ \mu_1 &\sim \text{Normal}\,(m_1, M_1)\,, \end{aligned}$$

  where m_0 and m_1 are the means of the (independent) normal distributions while M_0 and M_1 are the variances.

- The *hyperpriors* for σ_0^2 and σ_1^2 are given by

$$\begin{aligned} \sigma_0^2 &\sim \text{Inverse Gamma}\left(\frac{a_0}{2}, \frac{b_0}{2}\right), \\ \sigma_1^2 &\sim \text{Inverse Gamma}\left(\frac{a_1}{2}, \frac{b_1}{2}\right), \end{aligned}$$

  where $a_0/2$ and $a_1/2$ are the shape parameters of the (independent) Gamma distributions while $b_0/2$ and $b_1/2$ are the scale parameters.

  The default hyperpriors for μ_0, μ_1, σ_0^2, and σ_1^2 are chosen such that the prior distributions of β^b and β^w are flat.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run

```
> z.out <- (cbind(t0, t1) ~ x0 + x1, N = NULL,
          model = "ei.hier", data = mydata)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the quantities of interest by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the parameters.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
  - `N`: the total counts when the inputs are fractions.
  - `seed`: the random seed used in the model.
- From `summary(z.out)`, you may extract:

- `summary`: a matrix containing the summary information of the posterior estimation of β_i^b andβ_i^w for each unit and the parameters μ_0, μ_1, σ_1 and σ_2 based on the posterior distribution. The first p rows correspond to β_i^b, $i = 1, \ldots p$, the row names are in the form of `p0table`i. The $(p+1)$-th to the $2p$-th rows correspond to β_i^w, $i = 1, \ldots, p$. The row names are in the form of `p1table`i. The last four rows contain information about μ_0, μ_1, σ_0^2 and σ_1^2, the prior means and variances of θ_0 and θ_1.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays indexed by simulation $\times$ column $\times$ row $\times$ observation, where column and row refer to the column dimension and the row dimension of the contingency table, respectively. In this model, only 2×2 contingency tables are analyzed, hence column$= 2$ and row$= 2$ in all cases. Available quantities are:

  - `qi$ev`: the simulated expected values of each internal cell given the observed marginals.
  - `qi$pr`: the simulated expected values of each internal cell given the observed marginals.

How to Cite

To cite the *ei.hier* Zelig model use:

Ben Goodrich and Ying Lu. 2007. "ei.hier: Hierarchical Ecological Inference Model for 2x2 tables" in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

ei.hier function is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005). Sample date are adapted from Martin and Quinn (2005).

12.10 `ei.RxC`: Hierarchical Multinomial-Dirichlet Ecological Inference Model for $R \times C$ Tables

Given n contingency tables, each with observed marginals (column and row totals), ecological inference (EI) estimates the internal cell values in each table. The hierarchical Multinomial-Dirichlet model estimates cell counts in $R \times C$ tables. The model is implemented using a nonlinear least squares approximation and, with bootstrapping for standard errors, had good frequentist properties.

Syntax

```
> z.out <- zelig(cbind(T0, T1, T2, T3) ~ X0 + X1,
                 covar = NULL,
                 model = "ei.RxC", data = mydata)
> x.out <- setx(z.out, fn = NULL)
> s.out <- sim(z.out)
```

Inputs

- `T0`, `T1`, `T2`,..., `TC`: numeric vectors (either counts, or proportions that sum to one for each row) containing the column margins of the units to be analyzed.
- `X0`, `X1`, `X2`,...,`XR`: numeric vectors (either counts, or proportions that sum to one for each row) containing the row margins of the units to be analyzed.
- `covar`: (optional) a covariate that varies across tables, specified as `covar = ~ Z1`, for example. (The model only accepts one covariate.)

Examples

1. Basic examples: No covariate
   Attaching the example dataset:

   ```
   > data(Weimar)
   ```

   Estimating the model:

   ```
   > z.out <- zelig(cbind(Nazi, Government, Communists, FarRight,
   +     Other) ~ shareunemployed + shareblue + sharewhite + shareself +
   +     sharedomestic, model = "ei.RxC", data = Weimar)
   > summary(z.out)
   ```

   Estimate fractions of different social groups that support political parties:

   ```
   > s.out <- sim(z.out, num = 10)
   ```

Summarizing fractions of different social groups that support political parties:

```
> summary(s.out)
```

2. Example of covariates being present in the model

Using the example dataset Weimar and estimating the model

```
> z.out <- zelig(cbind(Nazi, Government, Communists, FarRight,
+     Other) ~ shareunemployed + shareblue + sharewhite + shareself +
+     sharedomestic, covar = ~shareprotestants, model = "ei.RxC",
+     data = Weimar)
> summary(z.out)
```

Set the covariate to its default (mean/median) value

```
> x.out <- setx(z.out)
```

Estimate fractions of different social groups that support political parties:

```
> s.out <- sim(z.out, num = 100)
```

Summarizing fractions of different social groups that support political parties:

```
> s.out <- summary(s.out)
```

Model

Consider the following 5×5 contingency table for the voting patterns in Weimar Germany. For each geographical unit i ($i = 1, \ldots, p$), the marginals $T_{1i}, \ldots, T_{Ci}$, $X_{1i}, \ldots, X_{Ri}$ are known for each of the p electoral precincts, and we would like to estimate $(\beta_i^{rc}, r = 1, \ldots, R, c = 1, \ldots, C-1)$ which are the fractions of people in social class r who vote for party c, for all r and c.

	Nazi	Government	Communists	Far Right	Other	
Unemployed	β^i_{11}	β^i_{12}	β^i_{13}	β^i_{14}	$1-\sum_{c=1}^{4}\beta^i_{1c}$	X^i_1
Blue	β^i_{21}	β^i_{22}	β^i_{23}	β^i_{24}	$1-\sum_{c=1}^{4}\beta^i_{2c}$	X^i_2
White	β^i_{31}	β^i_{32}	β^i_{33}	β^i_{34}	$1-\sum_{c=1}^{4}\beta^i_{3c}$	X^i_3
Self	β^i_{41}	β^i_{42}	β^i_{43}	β^i_{44}	$1-\sum_{c=1}^{4}\beta^i_{4c}$	X^i_4
Domestic	β^i_{51}	β^i_{52}	β^i_{53}	β^i_{54}	$1-\sum_{c=1}^{4}\beta^i_{5c}$	X^i_5
	T_{1i}	T_{2i}	T_{3i}	T_{4i}	$1-\sum_{c=1}^{4}\beta_{ci}$	

The marginal values $X_{1i}, \ldots, X_{Ri}$, $T_{1i}, \ldots, T_{Ci}$ may be observed as counts or fractions.

Let $T_i^{'} = (T_{1i}^{'}, T_{2i}^{'}, \ldots, T_{Ci}^{'})$ be the number of voting age persons who turn out to vote for different parties. There are three levels of hierarchy in the Multinomial-Dirichlet EI model. At the first stage, we model the data as:

- The *stochastic component* is described $T_i^{'}$ which follows a multinomial distribution:

$$T_i^{'} \sim \text{Multinomial}(\Theta_{1i}, \ldots, \Theta_{Ci})$$

- The *systematic components* are

$$\Theta_{ci} = \sum_{r=1}^{R} \beta^i_{rc} X_{ri} \quad \text{for} \quad c = 1, \ldots, C$$

At the second stage, we use an optional covariate to model Θ_{ci}'s and β^i_{qrc}:

- The *stochastic component* is described by $\beta^i_r = (\beta_{r1}, \beta_{r2}, \ldots, \beta_{r,C-1})$ for $i = 1, \ldots, , p$ and $r = 1, \ldots, R$, which follows a Dirichlet distribution:

$$\beta^i_r \sim \text{Dirichlet}(\alpha^i_{r1}, \ldots, \alpha^i_{rc})$$

- The *systematic components* are

$$\alpha^i_{rc} = \frac{d_r exp(\gamma_{rc} + \delta_{rc} Z_i)}{d_r(1 + \sum_{j=1}^{C-1} exp(\gamma_{rj} + \delta_{rj} Z_i))} = \frac{exp(\gamma_{rc} + \delta_{rc} Z_i)}{1 + \sum_{j=1}^{C-1} exp(\gamma_{rj} + \delta_{rj} Z_i)}$$

  for $i = 1, \ldots, p$, $r = 1, \ldots, R$, and $c = 1, \ldots, C-1$.

  In the third stage, we assume that the regression parameters (the γ_{rc}'s and δ_{rc}'s) are *a priori* independent, and put a flat prior on these regression parameters. The parameters d_r for $r = 1, \ldots, R$ are assumed to follow exponential distributions with mean $\frac{1}{\lambda}$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run

```
> z.out <- zelig(cbind(T0, T1, T2) ~ X0 + X1 + X2,
          model = "ei.RxC", data = mydata)
```

then you may examine the available information in `z.out` by using `names(z.out)`. For example,

- From the `zelig()` output object `z.out$coefficients` are the estimates of γ_{ij} (and also δ_{ij}, if covariates are present). The parameters are returned as a single vector of length $R \times (C-1)$. If there is a covariate, δ is concatenated to it.
- From the `sim()` output object, you may extract the parameters β_{ij} corresponding to the estimated fractions of different social groups that support different political parties, by using `s.out$qi$ev`. For each precinct, that will be a matrix with dimensions: simulations $\times R \times C$.
  `summary(s.out)` will give you the nationwide aggregate parameters.

How to Cite

To cite the *ei.RxC* Zelig model use:

Jason Wittenberg, Ferdinand Alimadhi, Badri Narayan Bhaskar, and Olivia Lau. 2007. "ei.RxC:Hierarchical Multinomial-Dirichlet Ecological Inference Model," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

For more information please see Rosen et al. (2001)

12.11 exp: Exponential Regression for Duration Dependent Variables

Use the exponential duration regression model if you have a dependent variable representing a duration (time until an event). The model assumes a constant hazard rate for all events. The dependent variable may be censored (for observations have not yet been completed when data were collected).

Syntax

```
> z.out <- zelig(Surv(Y, C) ~ X, model = "exp", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Exponential models require that the dependent variable be in the form `Surv(Y, C)`, where `Y` and `C` are vectors of length n. For each observation i in 1, ..., n, the value y_i is the duration (lifetime, for example), and the associated c_i is a binary variable such that $c_i = 1$ if the duration is not censored (*e.g.*, the subject dies during the study) or $c_i = 0$ if the duration is censored (*e.g.*, the subject is still alive at the end of the study and is know to live at least as long as y_i). If c_i is omitted, all Y are assumed to be completed; that is, time defaults to 1 for all observations.

Input Values

In addition to the standard inputs, `zelig()` takes the following additional options for exponential regression:

- `robust`: defaults to `FALSE`. If `TRUE`, `zelig()` computes robust standard errors based on sandwich estimators (see Huber (1981) and White (1980)) and the options selected in `cluster`.

- `cluster`: if `robust = TRUE`, you may select a variable to define groups of correlated observations. Let `x3` be a variable that consists of either discrete numeric values, character strings, or factors that define strata. Then

  ```
  > z.out <- zelig(y ~ x1 + x2, robust = TRUE, cluster = "x3",
                   model = "exp", data = mydata)
  ```

  means that the observations can be correlated within the strata defined by the variable `x3`, and that robust standard errors should be calculated according to those clusters. If `robust = TRUE` but `cluster` is not specified, `zelig()` assumes that each observation falls into its own cluster.

Example

Attach the sample data:

```
> data(coalition)
```

Estimate the model:

```
> z.out <- zelig(Surv(duration, ciep12) ~ fract + numst2, model = "exp",
+     data = coalition)
```

View the regression output:

```
> summary(z.out)
```

Set the baseline values (with the ruling coalition in the minority) and the alternative values (with the ruling coalition in the majority) for X:

```
> x.low <- setx(z.out, numst2 = 0)
> x.high <- setx(z.out, numst2 = 1)
```

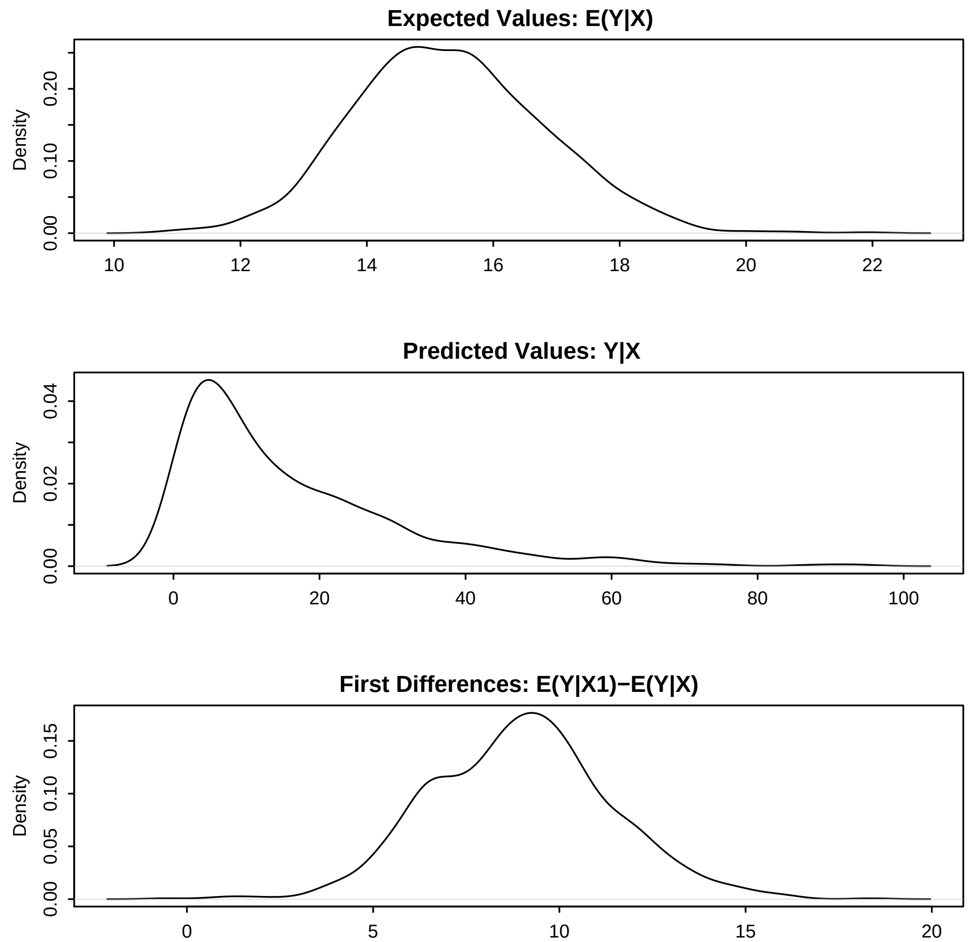
Simulate expected values (`qi$ev`) and first differences (`qi$fd`):

```
> s.out <- sim(z.out, x = x.low, x1 = x.high)
```

Summarize quantities of interest and produce some plots:

```
> summary(s.out)
```

```
> plot(s.out)
```



Model

Let Y_i^* be the survival time for observation i. This variable might be censored for some observations at a fixed time y_c such that the fully observed dependent variable, Y_i, is defined as

$$Y_i = \begin{cases} Y_i^* & \text{if } Y_i^* \leq y_c \\ y_c & \text{if } Y_i^* > y_c \end{cases}$$

- The *stochastic component* is described by the distribution of the partially observed variable Y^*. We assume Y_i^* follows the exponential distribution whose density function is given by

  $$f(y_i^* \mid \lambda_i) = \frac{1}{\lambda_i} \exp\left(-\frac{y_i^*}{\lambda_i}\right)$$

  for $y_i^* \geq 0$ and $\lambda_i > 0$. The mean of this distribution is λ_i.

  In addition, survival models like the exponential have three additional properties. The hazard function $h(t)$ measures the probability of not surviving past time t given survival up to t. In general, the hazard function is equal to $f(t)/S(t)$ where the survival function $S(t) = 1 - \int_0^t f(s)ds$ represents the fraction still surviving at time t. The cumulative hazard function $H(t)$ describes the probability of dying before time t. In general, $H(t) = \int_0^t h(s)ds = -\log S(t)$. In the case of the exponential model,

$$
\begin{aligned}
h(t) &= \frac{1}{\lambda_i} \\
S(t) &= \exp\left(-\frac{t}{\lambda_i}\right) \\
H(t) &= \frac{t}{\lambda_i}
\end{aligned}
$$

For the exponential model, the hazard function $h(t)$ is constant over time. The Weibull model and lognormal models allow the hazard function to vary as a function of elapsed time (see Section 12.68 and Section 12.31 respectively).

- The *systematic component* λ_i is modeled as

$$\lambda_i = \exp(x_i \beta),$$

where x_i is the vector of explanatory variables, and β is the vector of coefficients.

Quantities of Interest

- The expected values (`qi$ev`) for the exponential model are simulations of the expected duration given x_i and draws of β from its posterior,

$$E(Y) = \lambda_i = \exp(x_i \beta).$$

- The predicted values (`qi$pr`) are draws from the exponential distribution with rate equal to the expected value.

- The first difference (or difference in expected values, `qi$ev.diff`), is

$$\text{FD} = E(Y \mid x_1) - E(Y \mid x), \tag{12.1}$$

where x and x_1 are different vectors of values for the explanatory variables.

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. When $Y_i(t_i = 1)$ is censored rather than observed, we replace it with a simulation from the model given available knowledge of the censoring process. Variation in the simulations is due to two factors: uncertainty in the imputation process for censored y_i^* and uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. When $Y_i(t_i = 1)$ is censored rather than observed, we replace it with a simulation from the model given available knowledge of the censoring process. Variation in the simulations is due to two factors: uncertainty in the imputation process for censored y_i^* and uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(Surv(Y, C) ~ X, model = "exp", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `icoef`: parameter estimates for the intercept and scale parameter. While the scale parameter varies for the Weibull distribution, it is fixed to 1 for the exponential distribution (which is modeled as a special case of the Weibull).
  - `var`: the variance-covariance matrix for the estimates of β.
  - `loglik`: a vector containing the log-likelihood for the model and intercept only (respectively).
  - `linear.predictors`: the vector of $x_i\beta$.
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.

- Most of this may be conveniently summarized using `summary(z.out)`. From `summary(z.out)`, you may additionally extract:
  - `table`: the parameter estimates with their associated standard errors, p-values, and t-statistics. For example, `summary(z.out)$table`
- From the `sim()` output stored in `s.out`:
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from a distribution defined by the expected values.
  - `qi$fd`: the simulated first differences between the simulated expected values for `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *exp* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "exp: Exponential Regression for Duration Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The exponential function is part of the survival library by Terry Therneau, ported to R by Thomas Lumley. Advanced users may wish to refer to `help(survfit)` in the survival library and Venables and Ripley (2002).Sample data are from King et al. (1990a).

12.12 `factor.bayes`: Bayesian Factor Analysis

Given some unobserved explanatory variables and observed dependent variables, the Normal theory factor analysis model estimates the latent factors. The model is implemented using a Markov Chain Monte Carlo algorithm (Gibbs sampling with data augmentation). For factor analysis with ordinal dependent variables, see ordered factor analysis (Section 12.14), and for a mix of types of dependent variables, see the mixed factor analysis model (Section 12.13).

Syntax

```
> z.out <- zelig(cbind(Y1 ,Y2, Y3) ~ NULL, factors = 2,
                 model = "factor.bayes", data = mydata)
```

Inputs

`zelig()` takes the following functions for `factor.bayes`:

- `Y1`, `Y2`, and `Y3`: variables of interest in factor analysis (manifest variables), assumed to be normally distributed. The model requires a minimum of three manifest variables.
- `factors`: number of the factors to be fitted (defaults to 2).

Additional Inputs

In addition, `zelig()` accepts the following additional arguments for model specification:

- `lambda.constraints`: list containing the equality or inequality constraints on the factor loadings. Choose from one of the following forms:
  - `varname = list()`: by default, no constraints are imposed.
  - `varname = list(d, c)`: constrains the dth loading for the variable named `varname` to be equal to `c`.
  - `varname = list(d, "+")`: constrains the dth loading for the variable named `varname` to be positive;
  - `varname = list(d, "-")`: constrains the dth loading for the variable named `varname` to be negative.
- `std.var`: defaults to `FALSE` (manifest variables are rescaled to zero mean, but retain observed variance). If `TRUE`, the manifest variables are rescaled to be mean zero and unit variance.

In addition, `zelig()` accepts the following additional inputs for `bayes.factor`:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults to 20,000).

- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.

- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.

- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed 12345.

- `Lambda.start`: starting values of the factor loading matrix Λ, either a scalar (all unconstrained loadings are set to that value), or a matrix with compatible dimensions. The default is `NA`, where the start value are set to be 0 for unconstrained factor loadings, and 0.5 or -0.5 for constrained factor loadings (depending on the nature of the constraints).

- `Psi.start`: starting values for the uniquenesses, either a scalar (the starting values for all diagonal elements of Ψ are set to be this value), or a vector with length equal to the number of manifest variables. In the latter case, the starting values of the diagonal elements of Ψ take the values of `Psi.start`. The default value is `NA` where the starting values of the all the uniquenesses are set to be 0.5.

- `store.lambda`: defaults to `TRUE`, which stores the posterior draws of the factor loadings.

- `store.scores`: defaults to `FALSE`. If `TRUE`, stores the posterior draws of the factor scores. (Storing factor scores may take large amount of memory for a large number of draws or observations.)

The model also accepts the following additional arguments to specify prior parameters:

- `l0`: mean of the Normal prior for the factor loadings, either a scalar or a matrix with the same dimensions as Λ. If a scalar value, that value will be the prior mean for all the factor loadings. Defaults to 0.

- `L0`: precision parameter of the Normal prior for the factor loadings, either a scalar or a matrix with the same dimensions as Λ. If `L0` takes a scalar value, then the precision matrix will be a diagonal matrix with the diagonal elements set to that value. The default value is 0, which leads to an improper prior.

- `a0`: the shape parameter of the Inverse Gamma prior for the uniquenesses is `a0/2`. It can take a scalar value or a vector. The default value is 0.001.

- `b0`: the shape parameter of the Inverse Gamma prior for the uniquenesses is `b0/2`. It can take a scalar value or a vector. The default value is 0.001.

Zelig users may wish to refer to `help(MCMCfactanal)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example

   Attaching the sample dataset:

   ```
   > data(swiss)
   > names(swiss) <- c("Fert", "Agr", "Exam", "Educ", "Cath", "InfMort")
   ```

   Factor analysis:

   ```
   > z.out <- zelig(cbind(Agr, Exam, Educ, Cath, InfMort) ~ NULL,
   +      model = "factor.bayes", data = swiss, factors = 2, verbose = TRUE,
   +      a0 = 1, b0 = 0.15, burnin = 5000, mcmc = 50000)
   ```

   Checking for convergence before summarizing the estimates:

   ```
   > algor <- try(geweke.diag(z.out$coefficients), silent = T)
   > if (class(algor) == "try-error") print(algor)
   ```

   Since the algorithm did not converge, we now add some constraints on Λ.

2. Putting Constraints on Λ
   Put constraints on Lambda to optimize the algorithm:

```
> z.out <- zelig(cbind(Agr, Exam, Educ, Cath, InfMort) ~ NULL,
+     model = "factor.bayes", data = swiss, factors = 2, lambda.constraints = list
+         "+"), Exam = list(2, "-"), Educ = c(2, 0), InfMort = c(1,
+         0)), verbose = TRUE, a0 = 1, b0 = 0.15, burnin = 5000,
+     mcmc = 50000)
> geweke.diag(z.out$coefficients)
> heidel.diag(z.out$coefficients)
> raftery.diag(z.out$coefficients)
> summary(z.out)
```

Model

Suppose for observation i we observe K variables and hypothesize that there are d underlying factors such that:

$$Y_i = \Lambda\phi_i + \epsilon_i$$

where Y_i is the vector of K manifest variables for observation i. Λ is the $K \times d$ factor loading matrix and ϕ_i is the d-vector of latent factor scores. Both Λ and ϕ need to be estimated.

- The *stochastic component* is given by:

$$\epsilon_i \sim \text{Normal}(0, \Psi).$$

  where Ψ is a diagonal, positive definite matrix. The diagonal elements of Ψ are referred to as uniquenesses.

- The *systematic component* is given by

$$\mu_i = E(Y_i) = \Lambda\phi_i$$

- The independent conjugate *prior* for each Λ_{ij} is given by

$$\Lambda_{ij} \sim \text{Normal}(l_{0_{ij}}, L_{0_{ij}}^{-1}) \text{ for } i = 1, \ldots, k; \quad j = 1, \ldots, d.$$

- The independent conjugate *prior* for each Ψ_{ii} is given by

$$\Psi_{ii} \sim \text{InverseGamma}(\frac{a_0}{2}, \frac{b_0}{2}), \text{ for } i = 1, \ldots, k.$$

- The *prior* for ϕ_i is

$$\phi_i \quad \sim \quad \text{Normal}(0, I_d), \text{ for } i = 1, \ldots, n.$$

  where I_d is a $d \times d$ identity matrix.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(cbind(Y1, Y2, Y3), model = "factor.bayes", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated factor loadings and the uniquenesses. If `store.scores = TRUE`, the estimated factors scores are also contained in `coefficients`.
  - `data`: the name of the input data frame.
  - `seed`: the random seed used in the model.
- Since there are no explanatory variables, the `sim()` procedure is not applicable for factor analysis models.

How to Cite

To cite the *factor.bayes* Zelig model:

> Ben Goodrich and Ying Lu. 2007. "factor.bayes: Bayesian Factor Analysis," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

12.13 `factor.mix`: Mixed Data Factor Analysis

Mixed data factor analysis takes both continuous and ordinal dependent variables and estimates a model for a given number of latent factors. The model is estimated using a Markov Chain Monte Carlo algorithm (Gibbs sampler with data augmentation). Alternative models include Bayesian factor analysis for continuous variables (Section 12.12) and Bayesian factor analysis for ordinal variables (Section 12.14).

Syntax

```
> z.out <- zelig(cbind(Y1 ,Y2, Y3) ~ NULL, factors = 1,
                 model = "factor.mix", data = mydata)
```

Inputs

`zelig()` accepts the following arguments for `factor.mix`:

- `Y1`, `Y2`, `Y3`, `...`: The dependent variables of interest, which can be a mix of ordinal and continuous variables. You must have more dependent variables than factors.
- `factors`: The number of the factors to be fitted.

Additional Inputs

The model accepts the following additional arguments to monitor convergence:

- `lambda.constraints`: A list that contains the equality or inequality constraints on the factor loadings.
  - `varname = list()`: by default, no constraints are imposed.
  - `varname = list(d, c)`: constrains the dth loading for the variable named `varname` to be equal to `c`.
  - `varname = list(d, "+")`: constrains the dth loading for the variable named `varname` to be positive;
  - `varname = list(d, "-")`: constrains the dth loading for the variable named `varname` to be negative.

  Unlike Bayesian factor analysis for continuous variables (Section 12.12), the first column of Λ corresponds to negative item difficulty parameters and should not be constrained in general.

- `std.mean`: defaults to `TRUE`, which rescales the continuous manifest variables to have mean 0.

- `std.var`: defaults to `TRUE`. which rescales the continuous manifest variables to have unit variance.

`factor.mix` accepts the following additional arguments to monitor the sampling scheme for the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded. The default value is 1,000.

- `mcmc`: number of the MCMC iterations after burnin. The default value is 20,000.

- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.

- `tune`: tuning parameter, which can be either a scalar or a vector of length K. The value of the tuning parameter must be positive. The default value is `1.2`.

- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen. The default is `FALSE`.

- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed 12345.

- `lambda.start`: starting values of the factor loading matrix Λ for the Markov chain, either a scalar (starting values of the unconstrained loadings will be set to that value), or a matrix with compatible dimensions. The default is `NA`, where the start values for the first column of Λ are set based on the observed pattern, while for the rest of the columns of Λ, the start values are set to be 0 for unconstrained factor loadings, and 1 or -1 for constrained factor loadings (depending on the nature of the constraints).

- `psi.start`: starting values for the diagonals of the error variance (uniquenesses) matrix. Since the starting values for the ordinal variables are constrained to 1 (to identify the model), you may only specify the starting values for the continuous variables. For the continuous variables, you may specify `psi.start` as a scalar or a vector with length equal to the number of continuous variables. If a scalar, that starting value is recycled for all continuous variables. If a vector, the starting values should correspond to each of the continuous variables. The default value is `NA`, which means the starting values of all the continuous variable uniqueness are set to 0.5.

- `store.lambda`: defaults to `TRUE`, storing the posterior draws of the factor loadings.

- `store.scores`: defaults to `FALSE`. If `TRUE`, the posterior draws of the factor scores are stored. (Storing factor scores may take large amount of memory for a a large number of draws or observations.)

Use the following additional arguments to specify prior parameters used in the model:

- `l0`: mean of the Normal prior for the factor loadings, either a scalar or a matrix with the same dimensions as Λ. If a scalar value, then that value will be the prior mean for all the factor loadings. The default value is 0.

- `L0`: precision parameter of Normal prior for the factor loadings, either a scalar or a matrix with the same dimensions as Λ. If a scalar value, then the precision matrix will be a diagonal matrix with the diagonal elements set to that value. The default value is 0 which leads to an improper prior.

- `a0`: `a0/2` is the shape parameter of the Inverse Gamma priors for the uniquenesses. It can take a scalar value or a vector. The default value is 0.001.

- `b0`: `b0/2` is the shape parameter of the Inverse Gamma priors for the uniquenesses. It can take a scalar value or a vector. The default value is 0.001.

Zelig users may wish to refer to `help(MCMCmixfactanal)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.

- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.

- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example
   Attaching the sample dataset:

   ```
   > data(PErisk)
   ```

   Factor analysis for mixed data using `factor.mix`:

   ```
   > z.out <- zelig(cbind(courts, barb2, prsexp2, prscorr2, gdpw2) ~
   +     NULL, data = PErisk, model = "factor.mix", factors = 1, burnin = 5000,
   +     mcmc = 1e+05, thin = 50, verbose = TRUE, L0 = 0.25, tune = 1.2)
   ```

   Checking for convergence before summarizing the estimates:

   ```
   > geweke.diag(z.out$coefficients)
   ```

   ```
   > heidel.diag(z.out$coefficients)
   ```

   ```
   > summary(z.out)
   ```

Model

Let Y_i be a K-vector of observed variables for observation i, The kth variable can be either continuous or ordinal. When Y_{ik} is an ordinal variable, it takes value from 1 to J_k for $k = 1, \ldots, K$ and for $i = 1, \ldots, n$. The distribution of Y_{ik} is assumed to be governed by another K-vector of unobserved continuous variable Y^*_{ik}. There are d underlying factors. When Y_{ik} is continuous, we let $Y^*_{ik} = Y_{ik}$.

- The *stochastic component* is described in terms of Y_i^*:

  $$Y_i^* \sim \text{Normal}_K(\mu_i, I_K),$$

  where $Y_i^* = (Y^*_{i1}, \ldots, Y^*_{iK})$, and $\mu_i = (\mu_{i1}, \ldots, \mu_{iK})$.

  For ordinal Y_{ik},

  $$Y_{ik} = j \quad \text{if} \quad \gamma_{(j-1),k} \le Y^*_{ik} \le \gamma_{jk} \quad \text{for} \quad j = 1, \ldots, J_k; k = 1, \ldots, K.$$

  where $\gamma_{jk}, j = 0, \ldots, J$ are the threshold parameters for the kth variable with the following constraints, $\gamma_{lk} < \gamma_{mk}$ for $l < m$, and $\gamma_{0k} = -\infty, \gamma_{J_k k} = \infty$ for any $k = 1, \ldots, K$. It follows that the probability of observing Y_{ik} belonging to category j is,

  $$\Pr(Y_{ik} = j) = \Phi(\gamma_{jk} \mid \mu_{ik}) - \Phi(\gamma_{(j-1),k} \mid \mu_{ik}) \quad \text{for } j = 1, \ldots, J_k$$

  where $\Phi(\cdot \mid \mu_{ik})$ is the cumulative distribution function of the Normal distribution with mean μ_{ik} and variance 1.

- The *systematic component* is given by,

$$\mu_i \;=\; \Lambda\phi_i,$$

where Λ is a $K \times d$ matrix of factor loadings for each variable, ϕ_i is a d-vector of factor scores for observation i. Note both Λ and ϕ are estimated..

- The independent conjugate *prior* for each Λ_{ij} is given by

$$\Lambda_{ij} \;\sim\; \text{Normal}(l_{0_{ij}}, L_{0_{ij}}^{-1}) \text{ for } i = 1, \ldots, k; \quad j = 1, \ldots, d.$$

- The *prior* for ϕ_i is,

$$\phi_i \sim \text{Normal}(0, I_{d-1}), \quad \text{for} \quad i = 2, \ldots, n.$$

where I_{d-1} is a $(d-1) \times (d-1)$ identity matrix. Note the first element of ϕ_i is 1.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(cbind(Y1, Y2, Y3), model = "factor.mix", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated factor loadings, the estimated cut points γ for each variable. Note the first element of γ is normalized to be 0. If `store.scores = TRUE`, the estimated factors scores are also contained in `coefficients`.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
  - `seed`: the random seed used in the model.

- Since there are no explanatory variables, the `sim()` procedure is not applicable for factor analysis models.

How to Cite

To cite the *factor.mix* Zelig model:

> Ben Goodrich and Ying Lu. 2007. "factor.mix: Mixed Data Factor Analysis ," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

12.14 `factor.ord`: Ordinal Data Factor Analysis

Given some unobserved explanatory variables and observed ordinal dependent variables, this model estimates latent factors using a Gibbs sampler with data augmentation. For factor analysis for continuous data, see Section 12.12. For factor analysis for mixed data (including both continuous and ordinal variables), see Section 12.13.

Syntax

```
> z.out <- zelig(cbind(Y1 ,Y2, Y3) ~ NULL, factors = 1,
                 model = "factor.ord", data = mydata)
```

Inputs

`zelig()` accepts the following arguments for `factor.ord`: :

- `Y1, Y2`, and `Y3`: variables of interest in factor analysis (manifest variables), assumed to be ordinal variables. The number of manifest variables must be greater than the number of the factors.
- `factors`: number of the factors to be fitted (defaults to 1).

Additional Inputs

In addition, `zelig()` accepts the following arguments for model specification:

- `lambda.constraints`: list that contains the equality or inequality constraints on the factor loadings. A typical entry in the list has one of the following forms:
  - `varname = list()`: by default, no constraints are imposed.
  - `varname = list(d, c)`: constrains the dth loading for the variable named `varname` to be equal to `c`;
  - `varname = list(d, "+")`: constrains the dth loading for the variable named `varname` to be positive;
  - `varname = list(d, "-")`: constrains the dth loading for the variable named `varname` to be negative.

  The first column of Λ should not be constrained in general.
- `drop.constantvars`: defaults to `TRUE`, dropping the manifest variables that have no variation before fitting the model.

The model accepts the following arguments to monitor the convergence of the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).

- `mcmc`: number of the MCMC iterations after burnin (defaults to 20,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.
- `tune`: tuning parameter for Metropolis-Hasting sampling, either a scalar or a vector of length K. The value of the tuning parameter must be positive. The default value is 1.2.
- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.
- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed 12345.
- `Lambda.start`: starting values of the factor loading matrix Λ for the Markov chain, either a scalar (all unconstrained loadings are set to that value), or a matrix with compatible dimensions. The default is `NA`, such that the start values for the first column are set based on the observed pattern, while the remaining columns have start values set to 0 for unconstrained factor loadings, and -1 or 1 for constrained loadings (depending on the nature of the constraints).
- `store.lambda`: defaults to `TRUE`, which stores the posterior draws of the factor loadings.
- `store.scores`: defaults to `FALSE`. If `TRUE`, stores the posterior draws of the factor scores. (Storing factor scores may take large amount of memory for a a large number of draws or observations.)

Use the following parameters to specify the model's priors:

- `l0`: mean of the Normal prior for the factor loadings, either a scalar or a matrix with the same dimensions as Λ. If a scalar value, that value will be the prior mean for all the factor loadings. Defaults to 0.
- `L0`: precision parameter of the Normal prior for the factor loadings, either a scalar or a matrix with the same dimensions as Λ. If `L0` takes a scalar value, then the precision matrix will be a diagonal matrix with the diagonal elements set to that value. The default value is 0, which leads to an improper prior.

Zelig users may wish to refer to `help(MCMCordfactanal)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.

- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.

- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example
   Attaching the sample dataset:

   ```
   > data(newpainters)
   ```

   Factor analysis for ordinal data using `factor.ord`:

   ```
   > z.out <- zelig(cbind(Composition, Drawing, Colour, Expression) ~
   +      NULL, data = newpainters, model = "factor.ord", factors = 1,
   +      L0 = 0.5, burin = 5000, mcmc = 30000, thin = 5, tune = 1.2,
   +      verbose = TRUE)
   ```

   Checking for convergence before summarizing the estimates:

   ```
   > geweke.diag(z.out$coefficients)

   > heidel.diag(z.out$coefficients)

   > raftery.diag(z.out$coefficients)

   > summary(z.out)
   ```

Model

Let Y_i be a vector of K observed ordinal variables for observation i, each ordinal variable k for $k = 1, \ldots, K$ takes integer value $j = 1, \ldots, J_k$. The distribution of Y_i is assumed to be governed by another k-vector of unobserved continuous variable Y_i^*. There are d underlying factors.

- The *stochastic component* is described in terms of the latent variable Y_i^*:

$$Y_i^* \sim \text{Normal}_K(\mu_i, I_K),$$

  where $Y_i^* = (Y_{i1}^*, \ldots, Y_{iK}^*)$, and μ_i is the mean vector for Y_i^*, and $\mu_i = (\mu_{i1}, \ldots, \mu_{iK})$. Instead of Y_{ik}^*, we observe ordinal variable Y_{ik},

$$Y_{ik} = j \text{ if } \gamma_{(j-1),k} \leq Y_{ik}^* \leq \gamma_{jk} \text{ for } \quad j = 1, \ldots, J_k, k = 1, \ldots, K.$$

  where $\gamma_{jk}, j = 0, \ldots, J$ are the threshold parameters for the kth variable with the following constraints, $\gamma_{lk} < \gamma_{mk}$ for $l < m$, and $\gamma_{0k} = -\infty, \gamma_{J_k k} = \infty$ for any $k = 1, \ldots, K$. It follows that the probability of observing Y_{ik} belonging to category j is,

$$\Pr(Y_{ik} = j) = \Phi(\gamma_{jk} \mid \mu_{ik}) - \Phi(\gamma_{(j-1),k} \mid \mu_{ik}) \text{ for } j = 1, \ldots, J_k$$

  where $\Phi(\cdot \mid \mu_{ik})$ is the cumulative distribution function of the Normal distribution with mean μ_{ik} and variance 1.

- The *systematic component* is given by,

$$\mu_i = \Lambda \phi_i,$$

  where Λ is a $K \times d$ matrix of factor loadings for each variable, ϕ_i is a d-vector of factor scores for observation i. Note both Λ and ϕ need to be estimated.

- The independent conjugate *prior* for each element of Λ, Λ_{ij} is given by

$$\Lambda_{ij} \sim \text{Normal}(l_{0_{ij}}, L_{0_{ij}}^{-1}) \text{ for } i = 1, \ldots, k; \quad j = 1, \ldots, d.$$

- The *prior* for ϕ_i is,

$$\phi_{i(2:d)} \sim \text{Normal}(0, I_{d-1}), \text{ for } \quad i = 2, \ldots, n.$$

  where I_{d-1} is a $(d-1) \times (d-1)$ identity matrix. Note the first element of ϕ_i is 1.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(cbind(Y1, Y2, Y3), model = "factor.ord", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated factor loadings, the estimated cut points γ for each variable. Note the first element of γ is normalized to be 0. If `store.scores=TRUE`, the estimated factors scores are also contained in `coefficients`.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
  - `seed`: the random seed used in the model.
- Since there are no explanatory variables, the `sim()` procedure is not applicable for factor analysis models.

How to Cite

To cite the *factor.ord* Zelig model:

> Ben Goodrich and Ying Lu. 2007. "factor.ord: Ordinal Data Factor Analysis," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

12.15 gamma: Gamma Regression for Continuous, Positive Dependent Variables

Use the gamma regression model if you have a positive-valued dependent variable such as the number of years a parliamentary cabinet endures, or the seconds you can stay airborne while jumping. The gamma distribution assumes that all waiting times are complete by the end of the study (censoring is not allowed).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "gamma", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out, x1 = NULL)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for gamma regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see Zeileis (2004)). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    * `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    * `"kernHAC"`: HAC standard errors using the weights given in Andrews (1991).
    * `"weave"`: HAC standard errors using the weights given in Lumley and Heagerty (1999).
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and Zeileis (2004) for more options.

Example

Attach the sample data:

```
> data(coalition)
```

Estimate the model:

```
> z.out <- zelig(duration ~ fract + numst2, model = "gamma", data = coalition)
```

View the regression output:

```
> summary(z.out)
```

Set the baseline values (with the ruling coalition in the minority) and the alternative values (with the ruling coalition in the majority) for X:

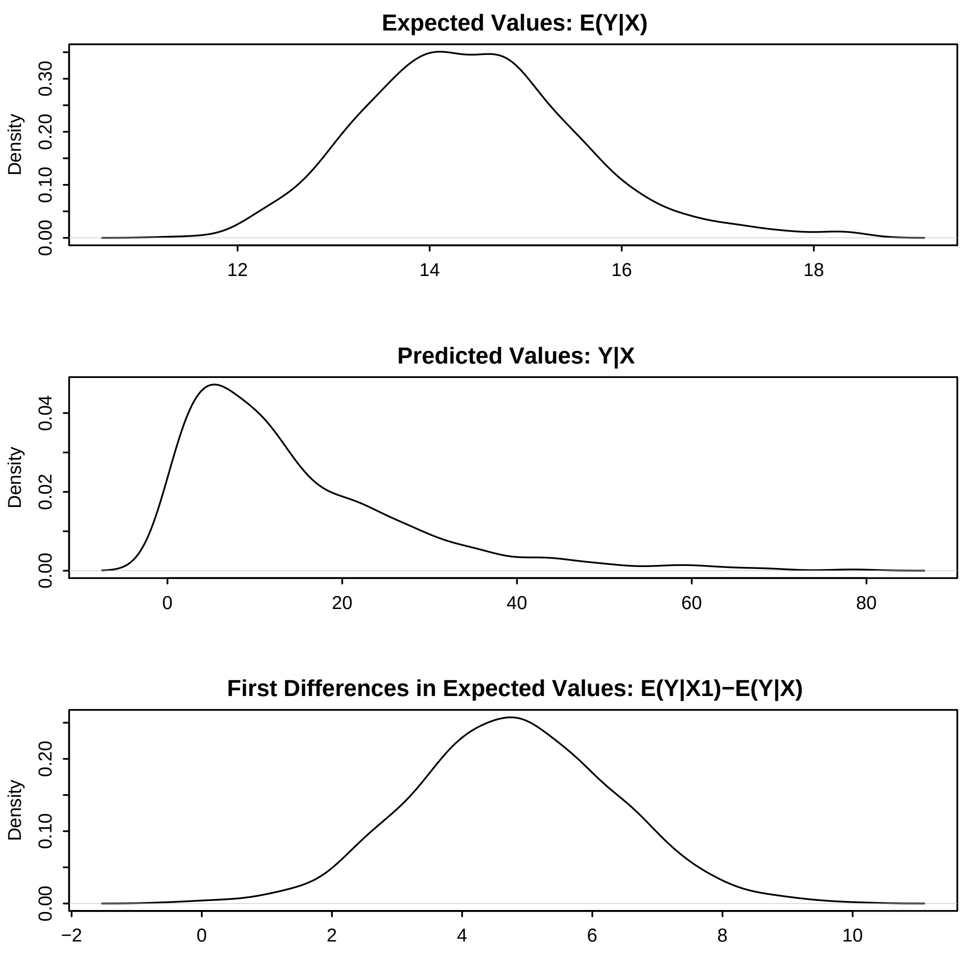
```
> x.low <- setx(z.out, numst2 = 0)
> x.high <- setx(z.out, numst2 = 1)
```

Simulate expected values (`qi$ev`) and first differences (`qi$fd`):

```
> s.out <- sim(z.out, x = x.low, x1 = x.high)
```

```
> summary(s.out)
```

```
> plot(s.out)
```



Model

- The Gamma distribution with scale parameter α has a *stochastic component*:

$$
\begin{aligned}
Y &\sim \text{Gamma}(y_i \mid \lambda_i, \alpha) \\
f(y) &= \frac{1}{\alpha^{\lambda_i}\,\Gamma\lambda_i}\, y_i^{\lambda_i - 1} \exp -\left\{\frac{y_i}{\alpha}\right\}
\end{aligned}
$$

  for $\alpha, \lambda_i, y_i > 0$.

- The *systematic component* is given by

$$
\lambda_i = \frac{1}{x_i\beta}
$$

Quantities of Interest

- The expected values (`qi$ev`) are simulations of the mean of the stochastic component given draws of α and β from their posteriors:

$$
E(Y) = \alpha\lambda_i.
$$

- The predicted values (`qi$pr`) are draws from the gamma distribution for each given set of parameters (α, λ_i).

- If `x1` is specified, `sim()` also returns the differences in the expected values (`qi$fd`),

$$
E(Y \mid x_1) - E(Y \mid x)
$$

  .

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$
\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},
$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "gamma", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: the vector of fitted values.
  - `linear.predictors`: the vector of $x_i \beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:

- `qi$ev`: the simulated expected values for the specified values of `x`.
- `qi$pr`: the simulated predicted values drawn from a distribution defined by (α, λ_i).
- `qi$fd`: the simulated first difference in the expected values for the specified values in `x` and `x1`.
- `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
- `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *gamma* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "gamma: Gamma Regression for Continuous, Positive Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The gamma model is part of the stats package by Venables and Ripley (2002). Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as McCullagh and Nelder (1989). Robust standard errors are implemented via the sandwich package by Zeileis (2004). Sample data are from King et al. (2000).

12.16 `gamma.gee`: Generalized Estimating Equation for Gamma Regression

The GEE gamma is similar to standard gamma regression (appropriate when you have an uncensored, positive-valued, continuous dependent variable such as the time until a parliamentary cabinet falls). Unlike in gamma regression, GEE gamma allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. GEE models make no distributional assumptions but require three specifications: a mean function, a variance function, and a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See Zorn (2001)).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "gamma.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:
  - Independence (`corstr = "independence"`): $\mathrm{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard gamma regression. The "working" correlation matrix is the identity matrix.

– Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

– Stationary m dependent (`corstr = "stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t - t'| \leq m \\ 0 & \text{if} \quad |t - t'| > m \end{cases}$$

If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t - t'|$ periods apart for $|t - t'| \leq m$.

Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

– Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t - t'| \leq m \\ 0 & \text{if} \quad |t - t'| > m \end{cases}$$

If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t - t'|$ periods apart.

Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

– Exchangeable (`corstr = "exchangeable"`): $\text{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

- Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\text{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

  Sample "working" correlation for Stationary AR-1 (`Mv`=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

- Unstructured (`corstr = "unstructured"`): $\text{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.

- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with Exchangeable Dependence

   Attaching the sample turnout dataset:

```
> data(coalition)
```

   Sorted variable identifying clusters

```
> coalition$cluster <- c(rep(c(1:62), 5), rep(c(63), 4))
> sorted.coalition <- coalition[order(coalition$cluster), ]
```

   Estimating model and presenting summary:

```
> z.out <- zelig(duration ~ fract + numst2, model = "gamma.gee",
+     id = "cluster", data = sorted.coalition, robust = TRUE, corstr = "exchangeab
> summary(z.out)
```

Setting the explanatory variables at their default values (mode for factor variables and mean for non-factor variables), with numst2 set to the vector 0 = no crisis, 1 = crisis.

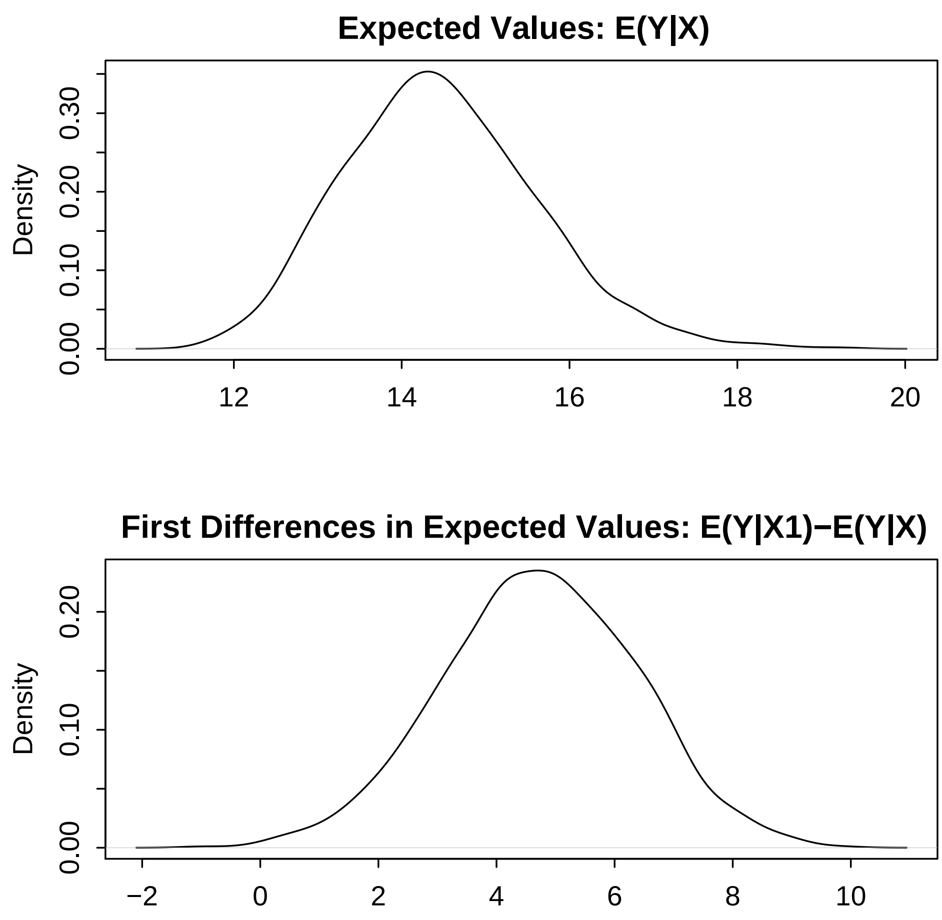
```
> x.low <- setx(z.out, numst2 = 0)
> x.high <- setx(z.out, numst2 = 1)
```

Simulate quantities of interest

```
> s.out <- sim(z.out, x = x.low, x1 = x.high)
> summary(s.out)
```

Generate a plot of quantities of interest:

```
> plot(s.out)
```



The Model

Suppose we have a panel dataset, with Y_{it} denoting the positive-valued, continuous dependent variable for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \lambda_i) \\ Y_{it} &\sim g(y_{it} \mid \lambda_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means λ_i and λ_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\lambda_{it} = \frac{1}{x_{it}\beta}$$

  where x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = \lambda_{it}^2 = \frac{1}{(x_{it}\beta)^2}$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi\, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = \lambda_{it}^2$ as the tth diagonal element, $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.

- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations. See **?** for more details.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.
- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.
- The expected values (`qi$ev`) for the GEE gamma model is the mean:

  $$E(Y) = \lambda_c = \frac{1}{x_c \beta},$$

  given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.
- The first difference (`qi$fd`) for the GEE gamma model is defined as

  $$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

  $$\frac{1}{\sum_{i=1}^{n} \sum_{t=1}^{T} tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it} = 1) - E[Y_{it}(tr_{it} = 0)] \right\},$$

  where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "gamma.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.
  - `fitted.values`: the vector of fitted values for the systemic component.
  - `linear.predictors`: the vector of $x_{it}\beta$
  - `max.id`: the size of the largest cluster.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How To Cite

To cite the *gamma.gee* Zelig model:

> Patrick Lam. 2007. "gamma.gee: Generalized Estimating Equation for Gamma Regression," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from King et al. (1990a).

12.17 `gamma.mixed`: Mixed effects gamma regression

Use generalized multi-level linear regression if you have covariates that are grouped according to one or more classification factors. Gamma regression models a continuous, positive dependent variable.

While generally called multi-level models in the social sciences, this class of models is often referred to as mixed-effects models in the statistics literature and as hierarchical models in a Bayesian setting. This general class of models consists of linear models that are expressed as a function of both *fixed effects*, parameters corresponding to an entire population or certain repeatable levels of experimental factors, and *random effects*, parameters corresponding to individual experimental units drawn at random from a population.

Syntax

```
z.out <- zelig(formula= y ~ x1 + x2 + tag(z1 + z2 | g),
               data=mydata, model="gamma.mixed")

z.out <- zelig(formula= list(mu=y ~ xl + x2 + tag(z1, delta | g),
               delta= ~ tag(w1 + w2 | g)), data=mydata, model="gamma.mixed")
```

Inputs

`zelig()` takes the following arguments for `mixed`:

- `formula`: a two-sided linear formula object describing the systematic component of the model, with the response on the left of a ~ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1 + ... + zn | g)` with `z1 + ... + zn` specifying the model for the random effects and `g` the grouping structure. Random intercept terms are included with the notation `tag(1 | g)`.
  Alternatively, `formula` may be a list where the first entry, `mu`, is a two-sided linear formula object describing the systematic component of the model, with the repsonse on the left of a ~ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1, delta | g)` with `z1` specifying the individual level model for the random effects, `g` the grouping structure and `delta` references the second equation in the list. The `delta` equation is one-sided linear formula object with the group level model for the random effects on the right side of a ~ operator. The model is specified with the notation `tag(w1 + ... + wn | g)` with `w1 + ... + wn` specifying the group level model and `g` the grouping structure.

Additional Inputs

In addition, `zelig()` accepts the following additional arguments for model specification:

- `data:` An optional data frame containing the variables named in `formula`. By default, the variables are taken from the environment from which `zelig()` is called.

- `method:` a character string. The criterion is always the log-likelihood but this criterion does not have a closed form expression and must be approximated. The default approximation is `"PQL"` or penalized quasi-likelihood. Alternatives are `"Laplace"` or `"AGQ"` indicating the Laplacian and adaptive Gaussian quadrature approximations respectively.

- `na.action:` A function that indicates what should happen when the data contain `NAs`. The default action (`na.fail`) causes `zelig()` to print an error message and terminate if there are any incomplete observations.

Additionally, users may with to refer to `lmer` in the package `lme4` for more information, including control parameters for the estimation algorithm and their defaults.

Examples

1. Basic Example with First Differences

   Attach sample data:

```
> data(coalition2)
```

   Estimate model using optional arguments to specify approximation method for the log-likelihood, and the log link function for the Gamma family:

```
> z.out1 <- zelig(duration ~ invest + fract + polar + numst2 +
+      crisis + tag(1 | country), data = coalition2, model = "gamma.mixed",
+      method = "PQL", family = Gamma(link = log))
```

   Summarize regression coefficients and estimated variance of random effects:

```
> summary(z.out1)
```

   Set the baseline values (with the ruling coalition in the minority) and the alternative values (with the ruling coalition in the majority) for X:

```
> x.high <- setx(z.out1, numst2 = 1)
> x.low <- setx(z.out1, numst2 = 0)
```

   Simulate expected values (`qi$ev`) and first differences(`qi$fd`):

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
> summary(s.out1)
```

Mixed effects gamma regression Model

Let Y_{ij} be the continuous, positive dependent variable, realized for observation j in group i as y_{ij}, for $i = 1, \ldots, M$, $j = 1, \ldots, n_i$.

- The *stochastic component* is described by a Gamma model with scale parameter α.

$$Y_{ij} \sim \text{Gamma}(y_{ij}|\lambda_{ij}, \alpha)$$

  where

$$Gamma(y_{ij}|\lambda_{ij}, \alpha) = \frac{1}{\alpha^{\lambda_{ij}} \Gamma \lambda_{ij}} y_{ij}^{\lambda_{ij}-1} \exp(-\{\frac{y_{ij}}{\alpha}\})$$

  for $\alpha,\ \lambda_{ij},\ y_{ij}\ > 0$.

- The q-dimensional vector of *random effects*, b_i, is restricted to be mean zero, and therefore is completely characterized by the variance covarance matrix Ψ, a $(q \times q)$ symmetric positive semi-definite matrix.

$$b_i \sim Normal(0, \Psi)$$

- The *systematic component* is

$$\lambda_{ij} \equiv \frac{1}{X_{ij}\beta + Z_{ij}b_i}$$

  where X_{ij} is the $(n_i \times p \times M)$ array of known fixed effects explanatory variables, β is the p-dimensional vector of fixed effects coefficients, Z_{ij} is the $(n_i \times q \times M)$ array of known random effects explanatory variables and b_i is the q-dimensional vector of random effects.

Quantities of Interest

- The predicted values (`qi$pr`) are draws from the gamma distribution for each given set of parameters (α, λ_{ij}), for

$$\lambda_{ij} = \frac{1}{X_{ij}\beta + Z_{ij}b_i}$$

  given X_{ij} and Z_{ij} and simulations of of β and b_i from their posterior distributions. The estimated variance covariance matrices are taken as correct and are themselves not simulated.

- The expected values (qi$ev) are simulations of the mean of the stochastic component given draws of α, β from their posteriors:

$$E(Y_{ij}|X_{ij}) = \alpha\lambda_{ij} = \frac{\alpha}{X_{ij}\beta}.$$

- The first difference (qi$fd) is given by the difference in expected values, conditional on X_{ij} and X'_{ij}, representing different values of the explanatory variables.

$$FD(Y_{ij}|X_{ij}, X'_{ij}) = E(Y_{ij}|X_{ij}) - E(Y_{ij}|X'_{ij})$$

- In conditional prediction models, the average predicted treatment effect (qi$att.pr) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij}=1) - Y_{ij}\widehat{(t_{ij}=0)}\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $Y_{ij}(t_{ij} = 0)$, the counterfactual predicted value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

- In conditional prediction models, the average expected treatment effect (qi$att.ev) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij}=1) - E[Y_{ij}(t_{ij}=0)]\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $E[Y_{ij}(t_{ij} = 0)]$, the counterfactual expected value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. You may examine the available information in `z.out` by using `slotNames(z.out)`, see the fixed effect coefficients by using `summary(z.out)@coefs`, and a default summary of information through `summary(z.out)`. Other elements available through the operator are listed below.

- From the `zelig()` output stored in `summary(z.out)`, you may extract:
  - `fixef`: numeric vector containing the conditional estimates of the fixed effects.

  - `ranef`: numeric vector containing the conditional modes of the random effects.
  - `frame`: the model frame for the model.

- From the `sim()` output stored in `s.out`, you may extract quantities of interest stored in a data frame:
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by the expected values.
  - `qi$ev`: the simulated expected values for the specified values of x.
  - `qi$fd`: the simulated first differences in the expected values for the values specified in x and x1.
  - `qi$ate.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.
  - `qi$ate.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *gamma.mixed* Zelig model:

> Delia Bailey, Ferdinand Alimadhi. 2007. "gamma.mixed: Mixed effects gamma regression" in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Mixed effects gamma regression is part of `lme4` package by Douglas M. Bates (Bates 2007). For a detailed discussion of mixed-effects models, please see **?**

12.18 gamma.net: Network Gamma Regression for Continuous, Positive Proximity Matrix Dependent Variables

Use the network gamma regression model if you have a positive-valued dependent variable that is a binary valued proximity matrix (a.k.a. sociomatricies, adjacency matrices, or matrix representations of directed graphs). The gamma distribution assumes that all waiting times are complete by the end of the study (censoring is not allowed).

Syntax

```
> z.out <- zelig(y ~ x1 + x2, model = "gamma.net", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for network gamma regression:

- `LF`: specifies the link function to be used for the network gamma regression. Default is `LF="inverse"`, but `LF` can also be set to `"identity"` or `"log"` by the user.

Examples

1. Basic Example

   Load the sample data (see `?friendship` for details on the structure of the network dataframe):

   ```
   > data(friendship)
   ```

   Estimate model:

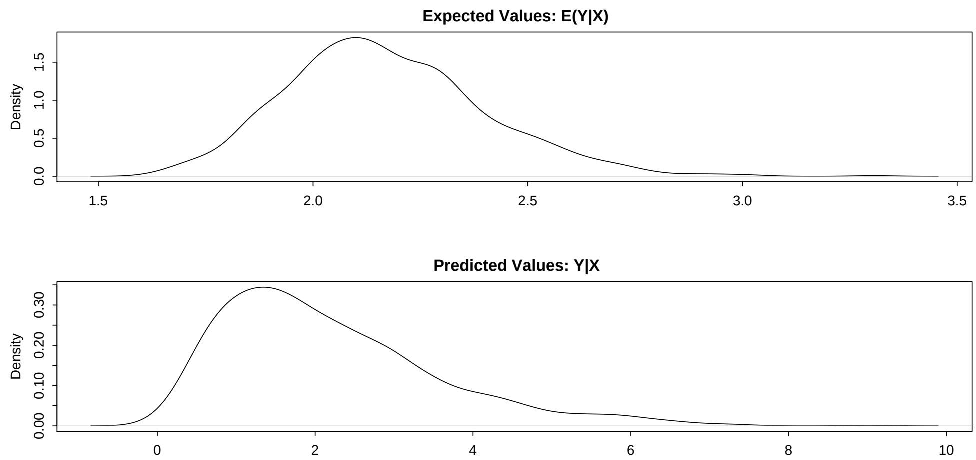
   ```
   > z.out <- zelig(per ~ perpower, LF = "inverse", model = "gamma.net",
   +     data = friendship)
   > summary(z.out)
   ```

   Setting values for the explanatory variables to their default values:

   ```
   > x.out <- setx(z.out)
   ```

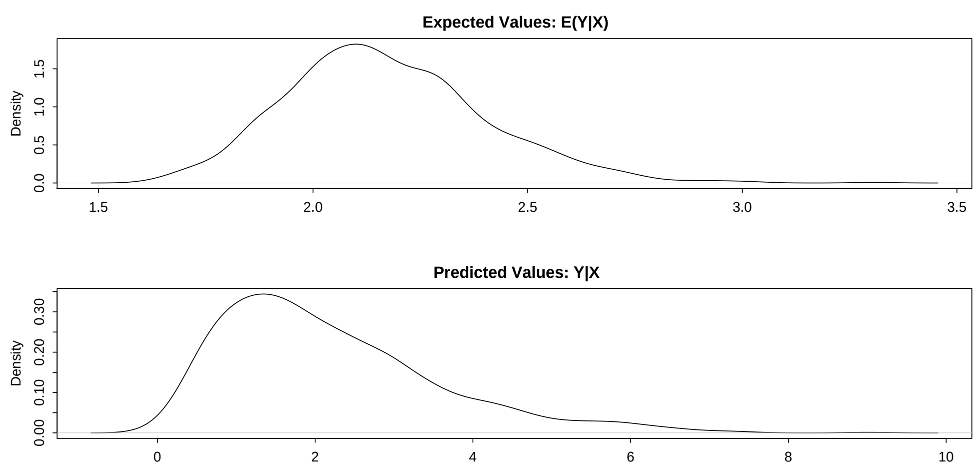
   Simulating quantities of interest from the posterior distribution.

```
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
> plot(s.out)
```



2. Simulating First Differences

```
> x.low <- setx(z.out, numst2 = 0)
> x.high <- setx(z.out, numst2 = 1)
> s.out2 <- sim(z.out, x = x.low, x1 = x.high)
> summary(s.out2)
> plot(s.out2)
```



Model

The `gamma.net` model performs a gamma regression of the proximity matrix $\mathbf{Y}$, a $m \times m$ matrix representing network ties, on a set of proximity matrices $\mathbf{X}$. This network regression model is directly analogous to standard gamma regression element-wise on the appropriately vectorized matrices. Proximity matrices are vectorized by creating Y, a $m^2 \times 1$ vector to represent the proximity matrix. The vectorization which produces the Y vector from the $\mathbf{Y}$ matrix is performed by simple row-concatenation of $\mathbf{Y}$. For example, if $\mathbf{Y}$ is a 15×15 matrix, the $\mathbf{Y}_{1,1}$ element is the first element of Y, and the $\mathbf{Y}_{2,1}$ element is the second element of Y and so on. Once the input matrices are vectorized, standard gamma regression is performed.

Let Y_i be the dependent variable, produced by vectorizing a binary proximity matrix, for observation i.

- The Gamma distribution with scale parameter α has a *stochastic component* given by

$$\begin{aligned} Y \quad & \sim \text{Gamma}(y_i|\lambda_i, \alpha) \\ f(y) \quad & = \tfrac{1}{\alpha^{\lambda_i}\Gamma\lambda_i} y_i^{\lambda_i - 1} \exp -\left[\tfrac{y_i}{\alpha}\right] \end{aligned}$$

  for $\alpha, \lambda_i, y_i > 0$.

- The *systematic component* is given by:

$$\lambda_i = \frac{1}{x_i\beta}.$$

Quantities of Interest

The quantities of interest for the network gamma regression are the same as those for the standard gamma regression.

- The expected values (`qi$ev`) are simulations of the mean of the stochastic component given draws of α and β from their posteriors:

$$E(Y) = \alpha_i\lambda.$$

- The predicted values (`qi$pr`) are draws from the gamma distribution for each set of parameters (α, λ_i).

- The first difference (`qi$fd`) for the network gamma model is defined as

$$FD = \Pr(Y|x_1) - \Pr(Y|x)$$

Output Values

The output of each Zelig command contains useful information which you may view. For example, you run `z.out <- zelig(y ~ x, model = "gamma.net", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `linear.predictors`: the vector of $x_i\beta$.
  - `aic`: Akaikeś Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `bic`: the Bayesian Information Criterion (minus twice the maximized log-likelihood plus the number of coefficients times $\log n$).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`
- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `mod.coefficients`: the parameter estimates with their associated standard errors, p-values, and t statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from a distribution defined by (α_i, λ).
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite

To cite the *gamma.net* Zelig model:

> Skyler J. Cranmer. 2007. "gamma.net: Network Gamma Regression for Continuous, Positive Proximity Matrix Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The network gamma regression is part of the `netglm` package by Skyler J. Cranmer and is built using some of the functionality of the `sna` package by Carter T. Butts (Butts and Carley 2001).In addition, advanced users may wish to refer to `help(gamma.net)`. Sample data are fictional.

12.19 `gamma.survey`: Survey-Weighted Gamma Regression for Continuous, Positive Dependent Variables

The survey-weighted Gamma regression model is appropriate for survey data obtained using complex sampling techniques, such as stratified random or cluster sampling (e.g., not simple random sampling). Like the conventional Gamma regression models (see Section 12.15), survey-weighted Gamma regression specifies a continuous, positive dependent variable as function of a set of explanatory variables. The survey-weighted Gamma model reports estimates of model parameters identical to conventional Gamma estimates, but uses information from the survey design to correct variance estimates.

The `gamma.survey` model accommodates three common types of complex survey data. Each method listed here requires selecting specific options which are detailed in the "Additional Inputs" section below.

1. **Survey weights**: Survey data are often published along with weights for each observation. For example, if a survey intentionally over-samples a particular type of case, weights can be used to correct for the over-representation of that type of case in the dataset. Survey weights come in two forms:
   (a) *Probability* weights report the probability that each case is drawn from the population. For each stratum or cluster, this is computed as the number of observations in the sample drawn from that group divided by the number of observations in the population in the group.
   (b) *Sampling* weights are the inverse of the probability weights.
2. **Strata/cluster identification**: A complex survey dataset may include variables that identify the strata or cluster from which observations are drawn. For stratified random sampling designs, observations may be nested in different strata. There are two ways to employ these identifiers:
   (a) Use *finite population corrections* to specify the total number of cases in the stratum or cluster from which each observation was drawn.
   (b) For stratified random sampling designs, use the raw strata ids to compute sampling weights from the data.
3. **Replication weights**: To preserve the anonymity of survey participants, some surveys exclude strata and cluster ids from the public data and instead release only precomputed replicate weights.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "gamma.survey", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard `zelig` inputs (see Section ??), survey-weighted Gamma models accept the following optional inputs:

1. Datasets that include survey weights:.
    - `probs`: An optional formula or numerical vector specifying each case's probability weight, the probability that the case was selected. Probability weights need not (and, in most cases, will not) sum to one. Cases with lower probability weights are weighted more heavily in the computation of model coefficients.
    - `weights`: An optional numerical vector specifying each case's sample weight, the inverse of the probability that the case was selected. Sampling weights need not (and, in most cases, will not) sum to one. Cases with higher sampling weights are weighted more heavily in the computation of model coefficients.
2. Datasets that include strata/cluster identifiers:
    - `ids`: An optional formula or numerical vector identifying the cluster from which each observation was drawn (ordered from largest level to smallest level). For survey designs that do not involve cluster sampling, `ids` defaults to `NULL`.
    - `fpc`: An optional numerical vector identifying each case's frequency weight, the total number of units in the population from which each observation was sampled.
    - `strata`: An optional formula or vector identifying the stratum from which each observation was sampled. Entries may be numerical, logical, or strings. For survey designs that do not involve cluster sampling, `strata` defaults to `NULL`.
    - `nest`: An optional logical value specifying whether primary sampling unites (PSUs) have non-unique ids across multiple strata. `nest=TRUE` is appropriate when PSUs reuse the same identifiers across strata. Otherwise, `nest` defaults to `FALSE`.
    - `check.strata`: An optional input specifying whether to check that clusters are nested in strata. If `check.strata` is left at its default, `!nest`, the check is not performed. If `check.strata` is specified as `TRUE`, the check is carried out.
3. Datasets that include replication weights:

- `repweights`: A formula or matrix specifying replication weights, numerical vectors of weights used in a process in which the sample is repeatedly re-weighted and parameters are re-estimated in order to compute the variance of the model parameters.
- `type`: A string specifying the type of replication weights being used. This input is required if replicate weights are specified. The following types of replication weights are recognized: `"BRR"`, `"Fay"`, `"JK1"`, `"JKn"`, `"bootstrap"`, or `"other"`.
- `weights`: An optional vector or formula specifying each case's sample weight, the inverse of the probability that the case was selected. If a survey includes both sampling weights and replicate weights separately for the same survey, both should be included in the model specification. In these cases, sampling weights are used to correct potential biases in in the computation of coefficients and replication weights are used to compute the variance of coefficient estimates.
- `combined.weights`: An optional logical value that should be specified as `TRUE` if the replicate weights include the sampling weights. Otherwise, `combined.weights` defaults to `FALSE`.
- `rho`: An optional numerical value specifying a shrinkage factor for replicate weights of type `"Fay"`.
- `bootstrap.average`: An optional numerical input specifying the number of iterations over which replicate weights of type `"bootstrap"` were averaged. This input should be left as `NULL` for `"bootstrap"` weights that were not were created by averaging.
- `scale`: When replicate weights are included, the variance is computed as the sum of squared deviations of the replicates from their mean. `scale` is an optional overall multiplier for the standard deviations.
- `rscale`: Like `scale`, `rscale` specifies an optional vector of replicate-specific multipliers for the squared deviations used in variance computation.
- `fpc`: For models in which `"JK1"`, `"JKn"`, or `"other"` replicates are specified, `fpc` is an optional numerical vector (with one entry for each replicate) designating the replicates' finite population corrections.
- `fpctype`: When a finite population correction is included as an `fpc` input, `fpctype` is a required input specifying whether the input to `fpc` is a sampling fraction (`fpctype="fraction"`) or a direct correction (`fpctype="correction"`).
- `return.replicates`: An optional logical value specifying whether the replicates should be returned as a component of the output. `return.replicates` defaults to `FALSE`.

Examples

1. A dataset that includes survey weights:

Attach the sample data:

```
> data(api, package = "survey")
```

Suppose that a dataset included a positive and continuous measure of public schools' performance (api00), a measure of the percentage of students at each school who receive subsidized meals (meals), an indicator for whether each school holds classes year round (year.rnd), and sampling weights computed by the survey house (pw). Estimate a model that regresses school performance on the meals and year.rnd variables:

```
> z.out1 <- zelig(api00 ~ meals + yr.rnd, model = "gamma.survey",
+     weights = ~pw, data = apistrat)
```

Summarize regression coefficients:

```
> summary(z.out1)
```

Set explanatory variables to their default (mean/mode) values, and set a high (80th percentile) and low (20th percentile) value for "meals":

```
> x.low <- setx(z.out1, meals = quantile(apistrat$meals, 0.2))
> x.high <- setx(z.out1, meals = quantile(apistrat$meals, 0.8))
```

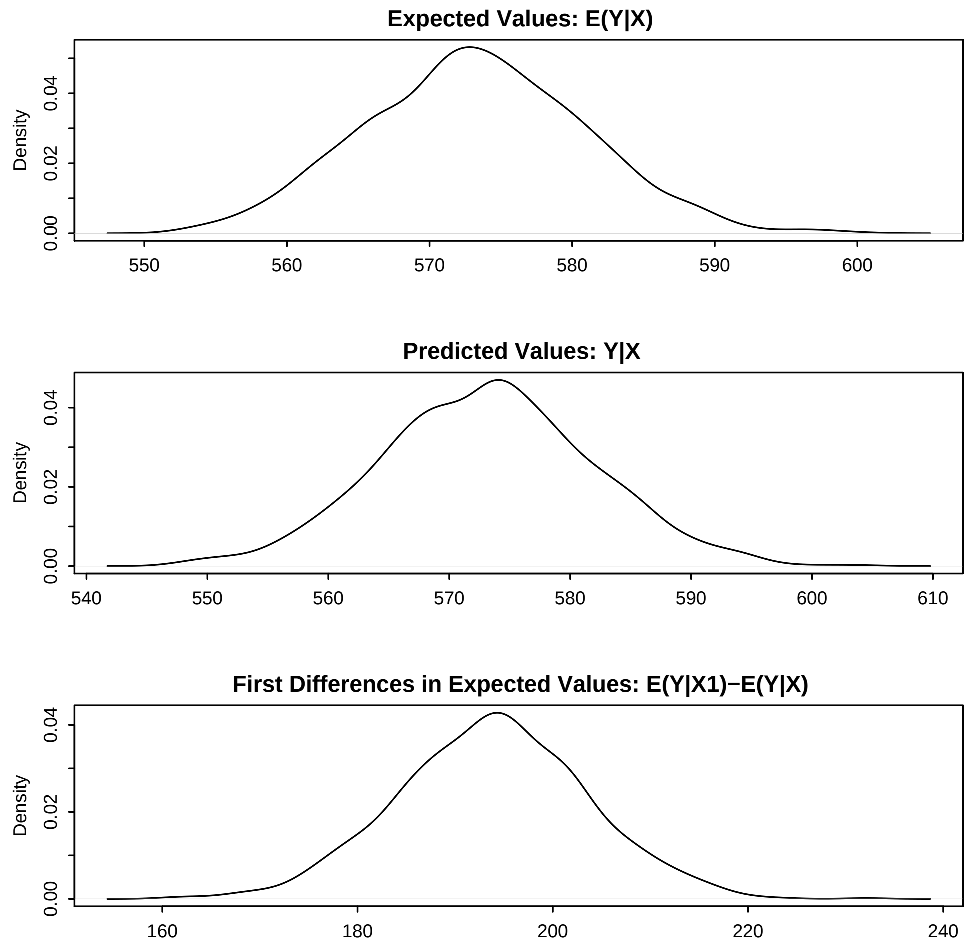
Generate first differences for the effect of high versus low concentrations of children receiving subsidized meals on the probability of holding school year-round:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
```

```
> summary(s.out1)
```

Generate a visual summary of the quantities of interest:

```
> plot(s.out1)
```



2. A dataset that includes strata/cluster identifiers:

   Suppose that the survey house that provided the dataset used in the previous example excluded sampling weights but made other details about the survey design available. A user can still estimate a model without sampling weights that instead uses inputs that identify the stratum and/or cluster to which each observation belongs and the size of the finite population from which each observation was drawn.

   Continuing the example above, suppose the survey house drew at random a fixed number of elementary schools, a fixed number of middle schools, and a fixed number of high schools. If the variable `stype` is a vector of characters (`"E"` for elementary schools, `"M"` for middle schools, and `"H"` for high schools) that identifies the type of school each case represents and `fpc` is a numerical vector that identifies for each case the total number of schools of the same type in the population, then the user could estimate the following model:

```
> z.out2 <- zelig(api00 ~ meals + yr.rnd, model = "gamma.survey",
+     strata = ~stype, fpc = ~fpc, data = apistrat)
```

Summarize the regression output:

```
> summary(z.out2)
```

The coefficient estimates for this example are identical to the point estimates in the first example, when pre-existing sampling weights were used. When sampling weights are omitted, they are estimated automatically for `"gamma.survey"` models based on the user-defined description of sampling designs.

Moreover, because the user provided information about the survey design, the standard error estimates are lower in this example than in the previous example, in which the user omitted variables pertaining to the details of the complex survey design.

3. A dataset that includes replication weights:

   Survey houses sometimes supply replicate weights (in lieu of details about the survey design). Suppose that the survey house that published these school data withheld strata/cluster identifiers and instead published replication weights. For the sake of illustrating how replicate weights can be used as inputs in `gamma.survey` models, create a set of jack-knife (JK1) replicate weights:

```
> jk1reps <- jk1weights(psu = apistrat$dnum)
```

   Again, estimate a model that regresses school performance on the `meals` and `year.rnd` variables, using the JK1 replicate weights in `jk1reps` to compute standard errors:

```
> z.out3 <- zelig(api00 ~ meals + yr.rnd, model = "gamma.survey",
+      data = apistrat, repweights = jk1reps$weights, type = "JK1")
```

   Summarize the regression coefficients:

```
> summary(z.out3)
```

   Set the explanatory variable `meals` at its 20th and 80th quantiles:

```
> x.low <- setx(z.out3, meals = quantile(apistrat$meals, 0.2))
> x.high <- setx(z.out3, meals = quantile(apistrat$meals, 0.8))
```

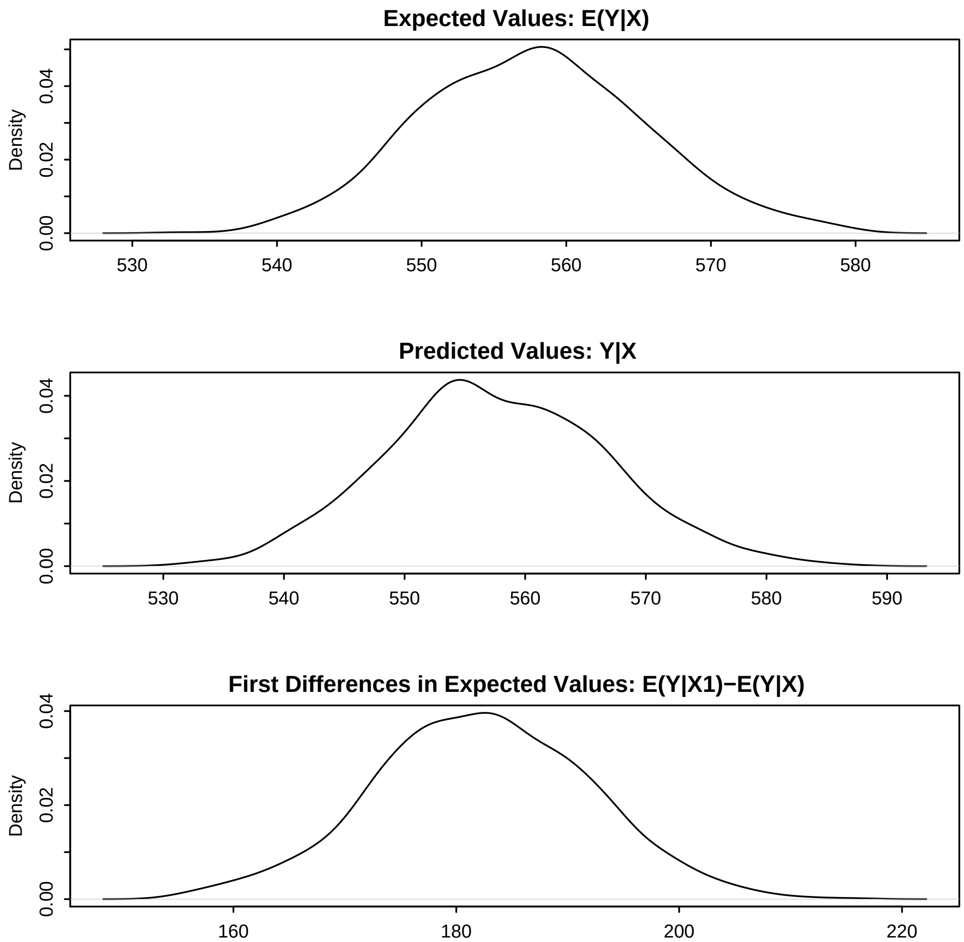
   Generate first differences for the effect of high versus low concentrations of poverty on school performance:

```
> s.out3 <- sim(z.out3, x = x.high, x1 = x.low)
```

```
> summary(s.out3)
```

   Generate a visual summary of quantities of interest:

```
> plot(s.out3)
```



Model

- The Gamma distribution with scale parameter α has a *stochastic component*:

$$\begin{aligned} Y &\sim \text{Gamma}(y_i \mid \lambda_i, \alpha) \\ f(y) &= \frac{1}{\alpha^{\lambda_i}\,\Gamma\lambda_i}\, y_i^{\lambda_i - 1} \exp - \left\{ \frac{y_i}{\alpha} \right\} \end{aligned}$$

for $\alpha, \lambda_i, y_i > 0$.

- The *systematic component* is given by

$$\lambda_i = \frac{1}{x_i \beta}$$

Variance

When replicate weights are not used, the variance of the coefficients is estimated as

$$\hat{\mathbf{\Sigma}}\left[\sum_{i=1}^{n}\frac{(1-\pi_i)}{\pi_i^2}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_i(Y_i-\mu_i))+2\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{(\pi_{ij}-\pi_i\pi_j)}{\pi_i\pi_j\pi_{ij}}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_j(Y_j-\mu_j))\right]\hat{\mathbf{\Sigma}}$$

where π_i is the probability of case i being sampled, $\mathbf{X}_i$ is a vector of the values of the explanatory variables for case i, Y_i is value of the dependent variable for case i, $\hat{\mu}_i$ is the predicted value of the dependent variable for case i based on the linear model estimates, and $\hat{\mathbf{\Sigma}}$ is the conventional variance-covariance matrix in a parametric glm. This statistic is derived from the method for estimating the variance of sums described in Binder (1983) and the Horvitz-Thompson estimator of the variance of a sum described in Horvitz and Thompson (1952).

When replicate weights are used, the model is re-estimated for each set of replicate weights, and the variance of each parameter is estimated by summing the squared deviations of the replicates from their mean.

Quantities of Interest

- The expected values (`qi$ev`) are simulations of the mean of the stochastic component given draws of α and β from their posteriors:

$$E(Y) = \alpha\lambda_i.$$

- The predicted values (`qi$pr`) are draws from the gamma distribution for each given set of parameters (α, λ_i).

- If `x1` is specified, `sim()` also returns the differences in the expected values (`qi$fd`),

$$E(Y \mid x_1) - E(Y \mid x)$$

.

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i}\sum_{i:t_i=1}^{n}\left\{Y_i(t_i=1)-E[Y_i(t_i=0)]\right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "gamma.survey", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: the vector of fitted values.
  - `linear.predictors`: the vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation × `x`-observation (for more than one `x`-observation). Available quantities are:

- `qi$ev`: the simulated expected values for the specified values of `x`.
- `qi$pr`: the simulated predicted values drawn from a distribution defined by (α, λ_i).
- `qi$fd`: the simulated first difference in the expected values for the specified values in `x` and `x1`.
- `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
- `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

When users estimate `gamma.survey` models with replicate weights in `Zelig`, an object called `.survey.prob.weights` is created in the global environment. `Zelig` will over-write any existing object with that name, and users are therefore advised to re-name any object called `.survey.prob.weights` before using `gamma.survey` models in `Zelig`.

How to Cite

To cite the *gamma.survey* Zelig model:

> Nicholas Carnes. 2008. "gamma.survey: Survey-Weighted Gamma Regression for Continuous, Positive Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Survey-weighted linear models and the sample data used in the examples above are a part of the `survey` package by Thomas Lumley. Users may wish to refer to the help files for the three functions that Zelig draws upon when estimating survey-weighted models, namely, `help(svyglm)`, `help(svydesign)`, and `help(svrepdesign)`. The Gamma model is part of the stats package by Venables and Ripley (2002). Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as McCullagh and Nelder (1989).

12.20 `irt1d`: One Dimensional Item Response Model

Given several observed dependent variables and an unobserved explanatory variable, item response theory estimates the latent variable (ideal points). The model is estimated using the Markov Chain Monte Carlo algorithm via a Gibbs sampler and data augmentation. Use this model if you believe that the ideal points lie in one dimension, and see the k-dimensional item response model (Section 12.21) for k hypothesized latent variables.

Syntax

```
> z.out <- zelig(cbind(Y1, Y2, Y3) ~ NULL, model = "irt1d", data = mydata)
```

Inputs

`irt1d` accepts the following argument:

- `Y1, Y2`, and `Y3`: `Y1` contains the items for subject "Y1", `Y2` contains the items for subject "Y2", and so on.

Additional arguments

`irt1d` accepts the following additional arguments for model specification:

- `theta.constraints`: a list specifying possible equality or inequality constraints on the ability parameters θ. A typical entry takes one of the following forms:
  - `varname = list()`: by default, no constraints are imposed.
  - `varname = c`: constrains the ability parameter for the subject named `varname` to be equal to `c`.
  - `varname = "+"`: constrains the ability parameter for the subject named `varname` to be positive.
  - `varname = "-"`: constrains the ability parameter for the subject named `varname` to be negative.

The model also accepts the following arguments to monitor the sampling scheme for the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults to 20,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.

- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.
- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed 12345.
- `theta.start`: starting values for the subject abilities (ideal points), either a scalar or a vector with length equal to the number of subjects. If a scalar, that value will be the starting value for all subjects. The default is `NA`, which sets the starting values based on an eigenvalue-eigenvector decomposition of the agreement score matrix formed from the model response matrix (`cbind(Y1, Y2, ...)`).
- `alpha.start`: starting values for the difficulty parameters α, either a scalar or a vector with length equal to the number of the items. If a scalar, the value will be the starting value for all α. The default is `NA`, which sets the starting values based on a series of probit regressions that condition on `theta.start`.
- `beta.start`: starting values for the β discrimination parameters, either a scalar or a vector with length equal to the number of the items. If a scalar, the value will be the starting value for all β. The default is `NA`, which sets the starting values based on a series of probit regressions conditioning on `theta.start`.
- `store.item`: defaults to `TRUE`, storing the posterior draws of the item parameters. (For a large number of draws or a large number observations, this may take a lot of memory.)
- `drop.constant.items`: defaults to `TRUE`, dropping items with no variation before fitting the model.

`irt1d` accepts the following additional arguments to specify prior parameters used in the model:

- `t0`: prior mean of the subject abilities (ideal points). The default is 0.
- `T0`: prior precision of the subject abilities (ideal points). The default is 0.
- `ab0`: prior mean of (α, β). It can be a scalar or a vector of length 2. If it takes a scalar value, then the prior means for both α and β will be set to that value. The default is 0.
- `AB0`: prior precision of (α, β). It can be a scalar or a 2×2 matrix. If it takes a scalar value, then the prior precision will be `diag(AB0,2)`. The prior precision is assumed to be same for all the items. The default is 0.25.

Zelig users may wish to refer to `help(MCMCirt1d)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example
   Attaching the sample dataset:

   ```
   > data(SupremeCourt)
   > names(SupremeCourt) <- c("Rehnquist", "Stevens", "OConnor", "Scalia",
   +      "Kennedy", "Souter", "Thomas", "Ginsburg", "Breyer")
   ```

   Fitting a one-dimensional item response theory model using `irt1d`:

   ```
   > z.out <- zelig(cbind(Rehnquist, Stevens, OConnor, Scalia, Kennedy,
   +      Souter, Thomas, Ginsburg, Breyer) ~ NULL, data = SupremeCourt,
   +      model = "irt1d", B0.alpha = 0.2, B0.beta = 0.2, burnin = 500,
   +      mcmc = 10000, thin = 20, verbose = TRUE)
   ```

   Checking for convergence before summarizing the estimates:

   ```
   > geweke.diag(z.out$coefficients)
   ```

```
> heidel.diag(z.out$coefficients)

> summary(z.out)
```

Model

Let Y_i be a vector of choices on J items made by subject i for $i = 1, \ldots, n$. The choice Y_{ij} is assumed to be determined by an unobserved utility Z_{ij}, which is a function of the subject i's abilities (ideal points) θ_i and item parameters α_j and β_j as follows:

$$Z_{ij} = -\alpha_j + \beta_j' \theta_i + \epsilon_{ij}.$$

- The *stochastic component* is given by

  $$\begin{aligned} Y_{ij} &\sim \text{Bernoulli}(\pi_{ij}) \\ &= \pi_{ij}^{Y_{ij}} (1 - \pi_{ij})^{1 - Y_{ij}}, \end{aligned}$$

  where $\pi_{ij} = \Pr(Y_{ij} = 1) = E(Z_{ij})$.

  The error term in the unobserved utility equation is independently and identically distributed with

  $$\epsilon_{ij} \sim \text{Normal}(0, 1).$$

- The *systematic component* is given by

  $$\pi_{ij} = \Phi(-\alpha_j + \beta_j' \theta_i),$$

  where $\Phi(\cdot)$ is the cumulative density function of the standard normal distribution with mean 0 and variance 1, θ_i is the subject ability (ideal point) parameter, and α_j and β_j are the item parameters. Both subject abilities and item parameters are estimated from the model, such that the model is identified by placing constraints on the subject ability parameters.

- The *prior* for θ_i is given by

  $$\theta_i \sim \text{Normal}\left(t_0, T_0^{-1}\right)$$

- The joint *prior* for α_j and β_j is given by

  $$(\alpha_j, \beta_j)' \sim \text{Normal}\left(ab_0, AB_0^{-1}\right)$$

  where ab_0 is a 2-vector of prior means and AB_0 is a 2×2 prior precision matrix.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(cbind(Y1, Y2, Y3) ~ NULL, model = "irt1d", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated subject abilities(ideal points). If `store.item = TRUE`, the estimated item parameters α and β are also contained in `coefficients`.
  - `data`: the name of the input data frame.
  - `seed`: the random seed used in the model.
- Since there are no explanatory variables, the `sim()` procedure is not applicable for item response models.

How to Cite

To cite the *irt1d* Zelig model use:

Ben Goodrich and Ying Lu. 2007. "irt1d: One Dimensional Item Response Mode," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The unidimensional item-response function is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005). Sample data are adapted from Martin and Quinn (2005).

12.21 `irtkd`: k-Dimensional Item Response Theory Model

Given several observed dependent variables and an unobserved explanatory variable, item response theory estimates the latent variable (ideal points). The model is estimated using the Markov Chain Monte Carlo algorithm, via a combination of Gibbs sampling and data augmentation. Use this model if you believe that the ideal points lie in k dimensions. See the unidimensional item response model (Section 12.20) for a single hypothesized latent variable.

Syntax

```
> z.out <- zelig(cbind(Y1, Y2, Y3) ~ NULL, dimensions = 1,
                 model = "irtkd", data = mydata)
```

Inputs

`irtkd` accepts the following arguments:

- `Y1`, `Y2`, and `Y3`: `Y1` contains the items for subject "Y1", `Y2` contains the items for subject "Y2", and so on.
- `dimensions`: The number of dimensions in the latent space. The default is 1.

Additional arguments

`irtkd` accepts the following additional arguments for model specification:

- `item.constraints`: a list of lists specifying possible simple equality or inequality constraints on the item parameters. A typical entry has one of the following forms:
  - `varname = list()`: by default, no constraints are imposed.
  - `varname = list(d, c)`: constrains the dth item parameter for the item named `varname` to be equal to `c`.
  - `varname = list(d, "+")`: constrains the dth item parameter for the item named `varname` to be positive;
  - `varname = list(d, "-")`: constrains the dth item parameter for the item named `varname` to be negative.

  In a k dimensional model, the first item parameter for item i is the difficulty parameter α_i, the second item parameter is the discrimination parameter on dimension 1, $(\beta_{i,1})$, the third item parameter is the discrimination parameter on dimension 2, $(\beta_{i,2}), \ldots$, and $(k+1)$th item parameter is the discrimination parameter on dimension k, $(\beta_{i,k})$. The item difficulty parameter(α) should not be constrained in general.

`irtkd` accepts the following additional arguments to monitor the sampling scheme for the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).

- `mcmc`: number of the MCMC iterations after burnin (defaults to 20,000).

- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.

- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen. The default is `FALSE`.

- `zelig.data`: the input data frame if `save.data = TRUE`.

- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed 12345.

- `alphabeta.start`: starting values for the item parameters α and β, either a scalar or a $(k+1) \times \mathit{items}$ matrix. If it is a scalar, then that value will be the starting value for all the elements of `alphabeta.start`. The default is `NA` which sets the starting values for the unconstrained elements based on a series of proportional odds logistic regressions. The starting values for the inequality constrained elements are set to be either 1.0 or -1.0 depending on the nature of the constraints.

- `store.item`: defaults to `FALSE`. If `TRUE` stores the posterior draws of the item parameters. (For a large number of draws or a large number observations, this may take a lot of memory.)

- `store.ability`: defaults to `TRUE`, storing the posterior draws of the subject abilities. (For a large number of draws or a large number observations, this may take a lot of memory.)

- `drop.constant.items`: defaults to `TRUE`, dropping items with no variation before fitting the model.

`irtkd` accepts the following additional arguments to specify prior parameters used in the model:

- `b0`: prior mean of (α, β), either as a scalar or a vector of compatible length. If a scalar value, then the prior means for both α and β will be set to that value. The default is 0.

- `B0`: prior precision for (α, β), either a scalar or a $(k+1) \times \mathit{items}$ matrix. If a scalar value, the prior precision will be a blocked diagonal matrix with elements `diag(B0,items)`. The prior precision is assumed to be same for all the items. The default is 0.25.

Zelig users may wish to refer to `help(MCMCirtKd)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example
   Attaching the sample dataset:

   ```
   > data(SupremeCourt)
   > names(SupremeCourt) <- c("Rehnquist", "Stevens", "OConnor", "Scalia",
   +      "Kennedy", "Souter", "Thomas", "Ginsburg", "Breyer")
   ```

   Fitting a one-dimensional item response theory model using `irtkd`:

   ```
   > z.out <- zelig(cbind(Rehnquist, Stevens, OConnor, Scalia, Kennedy,
   +      Souter, Thomas, Ginsburg, Breyer) ~ NULL, dimensions = 1,
   +      data = SupremeCourt, model = "irtkd", B0 = 0.25, burnin = 5000,
   +      mcmc = 50000, thin = 10, verbose = TRUE)
   ```

   Checking for convergence before summarizing the estimates:

   ```
   > geweke.diag(z.out$coefficients)
   ```

```
> heidel.diag(z.out$coefficients)

> raftery.diag(z.out$coefficients)

> summary(z.out)
```

Model

Let Y_i be a vector of choices on J items made by subject i for $i = 1, \ldots, n$. The choice Y_{ij} is assumed to be determined by unobserved utility Z_{ij}, which is a function of subject abilities (ideal points) θ_i and item parameters α_j and β_j,

$$Z_{ij} = -\alpha_j + \beta_j' \theta_i + \epsilon_{ij}.$$

In the k-dimensional item response theory model, each subject's ability is represented by a k-vector, θ_i. Each item has a difficulty parameter α_j and a k-dimensional discrimination parameter β_j. In one-dimensional item response theory model, $k = 1$.

- The *stochastic component* is given by

  $$\begin{aligned} Y_{ij} &\sim \text{Bernoulli}(\pi_{ij}) \\ &= \pi_{ij}^{Y_{ij}} (1 - \pi_{ij})^{1 - Y_{ij}}, \end{aligned}$$

  where $\pi_{ij} = \Pr(Y_{ij} = 1) = E(Z_{ij})$.

  The error term in the unobserved utility equation has a standard normal distribution,

  $$\epsilon_{ij} \sim \text{Normal}(0, 1).$$

- The *systematic component* is given by

  $$\pi_{ij} = \Phi(-\alpha_j + \beta_j \theta_i),$$

  where $\Phi(\cdot)$ is the cumulative density function of the standard normal distribution with mean 0 and variance 1, while θ_i contains the k-dimensional subject abilities(ideal points), and α_j and β_j are the item parameters. Both subject abilities and item parameters need to estimated from the model. The model is identified by placing constraints on the item parameters.

- The *prior* for θ_i is given by

  $$\theta_i \sim \text{Normal}_k(0, I_k)$$

- The joint *prior* for α_j and β_j is given by

  $$(\alpha_j, \beta_j)' \sim \text{Normal}_{k+1}\left(b_{0_j}, B_{0_j}^{-1}\right)$$

  where b_{0_j} is a $(k+1)$-vector of prior mean and B_{0_j} is a $(k+1) \times (k+1)$ prior precision matrix which is assumed to be diagonal.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(cbind(Y1, Y2, Y3) ~ NULL, model = "irtkd", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated subject abilities(ideal points). If `store.item = TRUE`, the estimated item parameters α and β are also contained in `coefficients`.
  - `data`: the name of the input data frame.
  - `seed`: the random seed used in the model.
- Since there are no explanatory variables, the `sim()` procedure is not applicable for item response models.

How to Cite

To cite the *irtkd* Zelig model use:

Ben Goodrich and Ying Lu. 2007. "irtkd: K-Dimensional Item Response Model," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The k dimensional item-response function is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005). Sample data are adapted from Martin and Quinn (2005).

12.22 `logit`: Logistic Regression for Dichotomous Dependent Variables

Logistic regression specifies a dichotomous dependent variable as a function of a set of explanatory variables. For a Bayesian implementation, see Section 12.23.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "logit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out, x1 = NULL)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for logistic regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see Zeileis (2004)). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    - `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    - `"kernHAC"`: HAC standard errors using the weights given in Andrews (1991).
    - `"weave"`: HAC standard errors using the weights given in Lumley and Heagerty (1999).
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame) The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and Zeileis (2004) for more options.

Examples

1. Basic Example

   Attaching the sample turnout dataset:

   ```
   > data(turnout)
   ```

Estimating parameter values for the logistic regression:

```
> z.out1 <- zelig(vote ~ age + race, model = "logit", data = turnout)
```

Setting values for the explanatory variables:

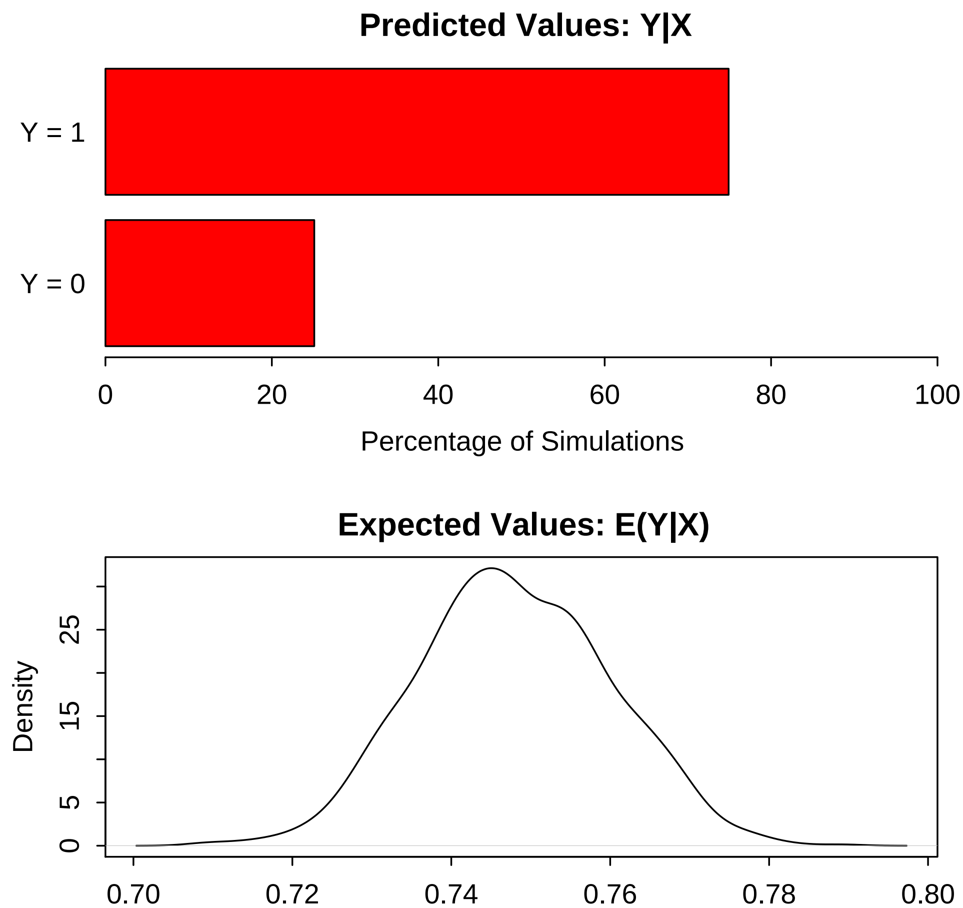
```
> x.out1 <- setx(z.out1, age = 36, race = "white")
```

Simulating quantities of interest from the posterior distribution.

```
> s.out1 <- sim(z.out1, x = x.out1)

> summary(s.out1)

> plot(s.out1)
```



2. Simulating First Differences

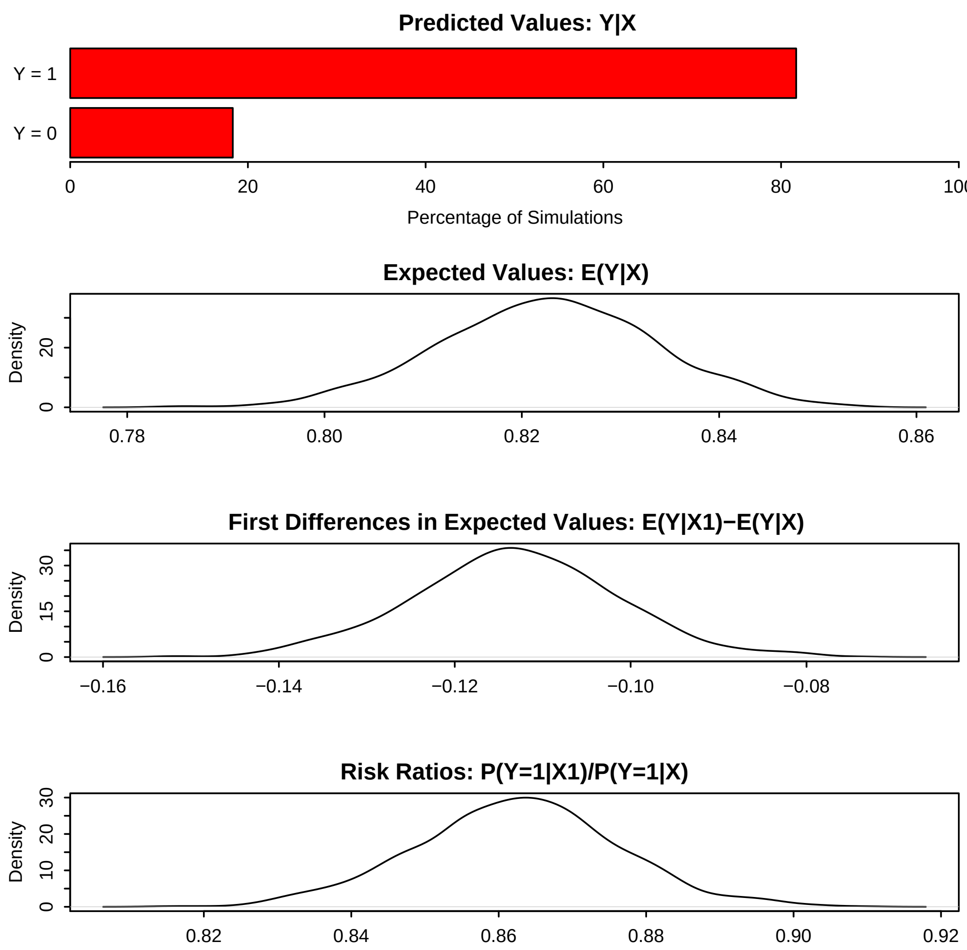
   Estimating the risk difference (and risk ratio) between low education (25th percentile) and high education (75th percentile) while all the other variables held at their default values.

```
> z.out2 <- zelig(vote ~ race + educate, model = "logit", data = turnout)
> x.high <- setx(z.out2, educate = quantile(turnout$educate, prob = 0.75))
> x.low <- setx(z.out2, educate = quantile(turnout$educate, prob = 0.25))

> s.out2 <- sim(z.out2, x = x.high, x1 = x.low)

> summary(s.out2)

> plot(s.out2)
```

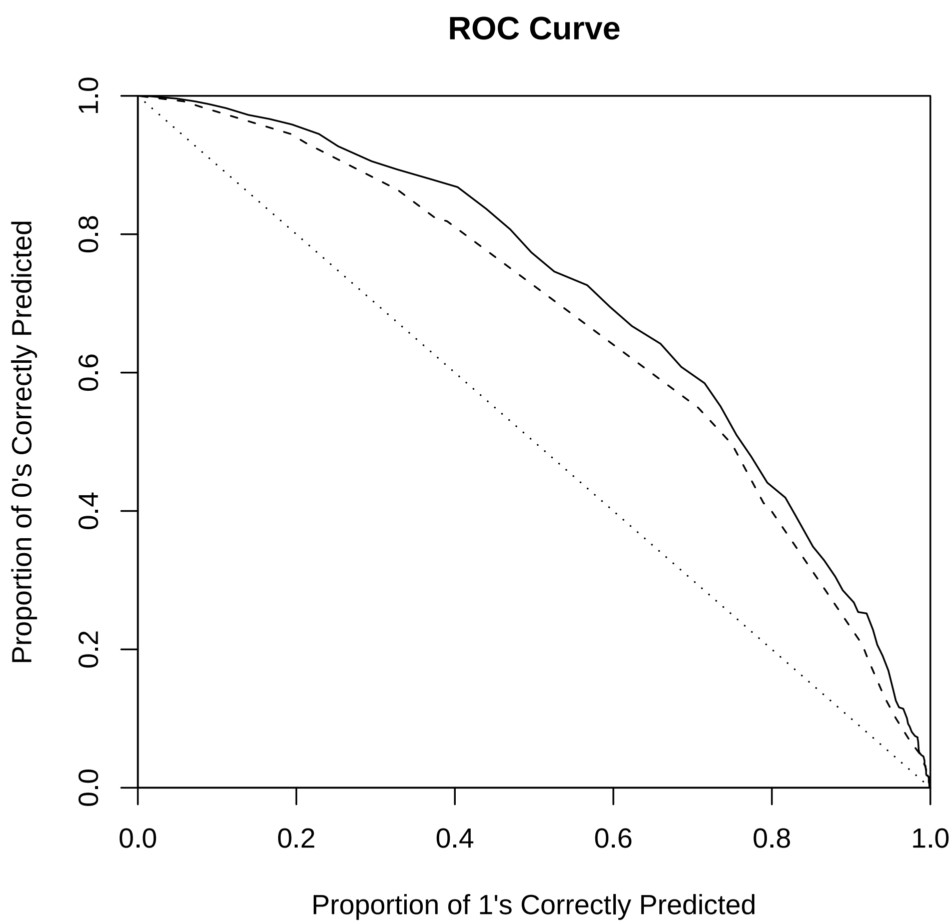


3. Presenting Results: An ROC Plot

   One can use an ROC plot to evaluate the fit of alternative model specifications. (Use `demo(roc)` to view this example, or see King and Zeng (2002).)

```
> z.out1 <- zelig(vote ~ race + educate + age, model = "logit",
+      data = turnout)
> z.out2 <- zelig(vote ~ race + educate, model = "logit", data = turnout)

> rocplot(z.out1$y, z.out2$y, fitted(z.out1), fitted(z.out2))
```



Model

Let Y_i be the binary dependent variable for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$\begin{aligned} Y_i &\sim \text{Bernoulli}(y_i \mid \pi_i) \\ &= \pi_i^{y_i}(1-\pi_i)^{1-y_i} \end{aligned}$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = \frac{1}{1+\exp(-x_i\beta)}.$$

  where x_i is the vector of k explanatory variables for observation i and β is the vector of coefficients.

Quantities of Interest

- The expected values (`qi$ev`) for the logit model are simulations of the predicted probability of a success:

$$E(Y) = \pi_i = \frac{1}{1+\exp(-x_i\beta)},$$

  given draws of β from its sampling distribution.

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the logit model is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

$$\text{RR} = \Pr(Y = 1 \mid x_1) \ / \ \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "logit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_i.
  - `linear.predictors`: the vector of $x_i\beta$
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `data`: the name of the input data frame.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:

– `qi$ev`: the simulated expected probabilities for the specified values of `x`.
– `qi$pr`: the simulated predicted values for the specified values of `x`.
– `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
– `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
– `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
– `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *logit* Zelig model:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "logit: Logistic Regression for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The logit model is part of the stats package by Venables and Ripley (2002). Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as McCullagh and Nelder (1989). Robust standard errors are implemented via the sandwich package by Zeileis (2004). Sample data are from King et al. (2000).

12.23 `logit.bayes`: Bayesian Logistic Regression

Logistic regression specifies a dichotomous dependent variable as a function of a set of explanatory variables using a random walk Metropolis algorithm. For a maximum likelihood implementation, see Section 12.22.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "logit.bayes", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

Use the following arguments to monitor the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults to 10,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.
- `tune`: Metropolis tuning parameter, either a positive scalar or a vector of length k, where k is the number of coefficients. The tuning parameter should be set such that the acceptance rate of the Metropolis algorithm is satisfactory (typically between 0.20 and 0.5) before using the posterior density for inference. The default value is 1.1.
- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.
- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed of 12345.
- `beta.start`: starting values for the Markov chain, either a scalar or vector with length equal to the number of estimated coefficients. The default is `NA`, such that the maximum likelihood estimates are used as the starting values.

Use the following parameters to specify the model's priors:

- `b0`: prior mean for the coefficients, either a numeric vector or a scalar. If a scalar value, that value will be the prior mean for all the coefficients. The default is 0.
- `B0`: prior precision parameter for the coefficients, either a square matrix (with the dimensions equal to the number of coefficients) or a scalar. If a scalar value, that value times an identity matrix will be the prior precision parameter. The default is 0, which leads to an improper prior.

Zelig users may wish to refer to `help(logit.bayes)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example
   Attaching the sample dataset:

   ```
   > data(turnout)
   ```

   Estimating the logistic regression using `logit.bayes`:

   ```
   > z.out <- zelig(vote ~ race + educate, model = "logit.bayes",
   +      data = turnout, verbose = TRUE)
   ```

   Convergence diagnostics before summarizing the estimates:

   ```
   > geweke.diag(z.out$coefficients)
   ```

   ```
   > heidel.diag(z.out$coefficients)
   ```

```
> raftery.diag(z.out$coefficients)
```

```
> summary(z.out)
```

Setting values for the explanatory variables to their sample averages:

```
> x.out <- setx(z.out)
```

Simulating quantities of interest from the posterior distribution given `x.out`.

```
> s.out1 <- sim(z.out, x = x.out)
```

```
> summary(s.out1)
```

2. Simulating First Differences
   Estimating the first difference (and risk ratio) in individual's probability of voting when education is set to be low (25th percentile) versus high (75th percentile) while all the other variables held at their default values.

```
> x.high <- setx(z.out, educate = quantile(turnout$educate, prob = 0.75))
```

```
> x.low <- setx(z.out, educate = quantile(turnout$educate, prob = 0.25))
```

```
> s.out2 <- sim(z.out, x = x.high, x1 = x.low)
```

```
> summary(s.out2)
```

Model

Let Y_i be the binary dependent variable for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$\begin{aligned} Y_i &\sim \text{Bernoulli}(\pi_\text{i}) \\ &= \pi_i^{Y_i}(1-\pi_i)^{1-Y_i}, \end{aligned}$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by

$$\pi_i = \frac{1}{1 + \exp(-x_i \beta)},$$

  where x_i is the vector of k explanatory variables for observation i and β is the vector of coefficients.

- The *prior* for β is given by

$$\beta \sim \text{Normal}_k \left(b_0, B_0^{-1} \right)$$

  where b_0 is the vector of means for the k explanatory variables and B_0 is the $k \times k$ precision matrix (the inverse of a variance-covariance matrix).

Quantities of Interest

- The expected values (`qi$ev`) for the logit model are simulations of the predicted probability of a success:

$$E(Y) = \pi_i = \frac{1}{1 + \exp(-x_i \beta)},$$

given the posterior draws of β from the MCMC iterations.

- The predicted values (`qi$pr`) are draws from the Bernoulli distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the logit model is defined as

$$\text{FD} = \Pr(Y = 1 \mid X_1) - \Pr(Y = 1 \mid X).$$

- The risk ratio (`qi$rr`)is defined as

$$\text{RR} = \Pr(Y = 1 \mid X_1) \ / \ \Pr(Y = 1 \mid X).$$

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is

$$\frac{1}{\sum t_i} \sum_{i:t_i=1} [Y_i(t_i = 1) - E[Y_i(t_i = 0)]],$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups.

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group is

$$\frac{1}{\sum t_i} \sum_{i:t_i=1} [Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}],$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run

```
z.out <- zelig(y ~ x, model = "logit.bayes", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated parameters.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
  - `seed`: the random seed used in the model.
- From the `sim()` output object `s.out`:
  - `qi$ev`: the simulated expected values(probabilities) for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected values for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected values simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *logit.bayes* Zelig model:

> Ben Goodrich and Ying Lu. 2007. "logit.bayes: Bayesian Logistic Regression for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Bayesian logistic regression is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005).

12.24 `logit.gam`: Generalized Additive Model for Dichotomous Dependent Variables

This function runs a nonparametric Generalized Additive Model (GAM) for dichotomous dependent variables.

Syntax

```
> z.out <- zelig(y ~ x1 + s(x2), model = "logit.gam", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Where `s()` indicates a variable to be estimated via nonparametric smooth. All variables for which `s()` is not specified, are estimated via standard parametric methods.

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for GAM models.

- `method`: Controls the fitting method to be used. Fitting methods are selected via a list environment within `method=gam.method()`. See `gam.method()` for details.
- `scale`: Generalized Cross Validation (GCV) is used if `scale = 0` (see the "Model" section for details) except for Logit models where a Un-Biased Risk Estimator (UBRE) (also see the "Model" section for details) is used with a scale parameter assumed to be 1. If `scale` is greater than 1, it is assumed to be the scale parameter/variance and UBRE is used. If `scale` is negative GCV is used.
- `knots`: An optional list of knot values to be used for the construction of basis functions.
- `H`: A user supplied fixed quadratic penalty on the parameters of the GAM can be supplied with this as its coefficient matrix. For example, ridge penalties can be added to the parameters of the GAM to aid in identification on the scale of the linear predictor.
- `sp`: A vector of smoothing parameters for each term.
- `...`: additional options passed to the `logit.gam` model. See the `mgcv` library for details.

Examples

1. Basic Example

   Create some count data:

```
> set.seed(0);  n <- 400; sig <- 2;
> x0 <- runif(n, 0, 1);  x1 <- runif(n, 0, 1)
> x2 <- runif(n, 0, 1);  x3 <- runif(n, 0, 1)
> g <- (f-5)/3
> g <- binomial()$linkinv(g)
> y <- rbinom(g,1,g)
> my.data <- as.data.frame(cbind(y, x0, x1, x2, x3))
```

Estimate the model, summarize the results, and plot nonlinearities:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), model = "logit.gam",
+      data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

Note that the `plot()` function can be used after model estimation and before simulation to view the nonlinear relationships in the independent variables:

Set values for the explanatory variables to their default (mean/mode) values, then simulate, summarize and plot quantities of interest:

```
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
> plot(s.out)
```

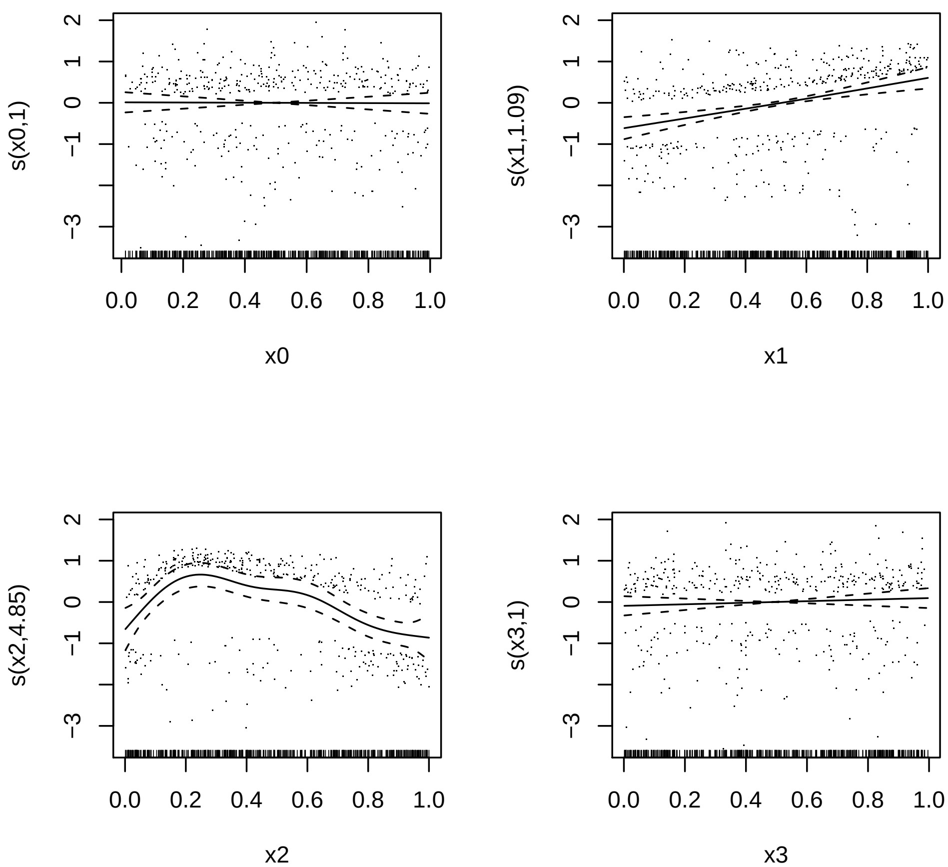
2. Simulating First Differences

   Estimating the risk difference (and risk ratio) between low values (20th percentile) and high values (80th percentile) of the explanatory variable `x3` while all the other variables are held at their default (mean/mode) values.

```
> x.high <- setx(z.out, x3 = quantile(my.data$x3, 0.8))
> x.low <- setx(z.out, x3 = quantile(my.data$x3, 0.2))
> s.out <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out)
> plot(s.out)
```

3. Variations in GAM model specification. Note that `setx` and `sim` work as shown in the above examples for any GAM model. As such, in the interest of parsimony, I will not re-specify the simulations of quantities of interest.

   An extra ridge penalty (useful with convergence problems):



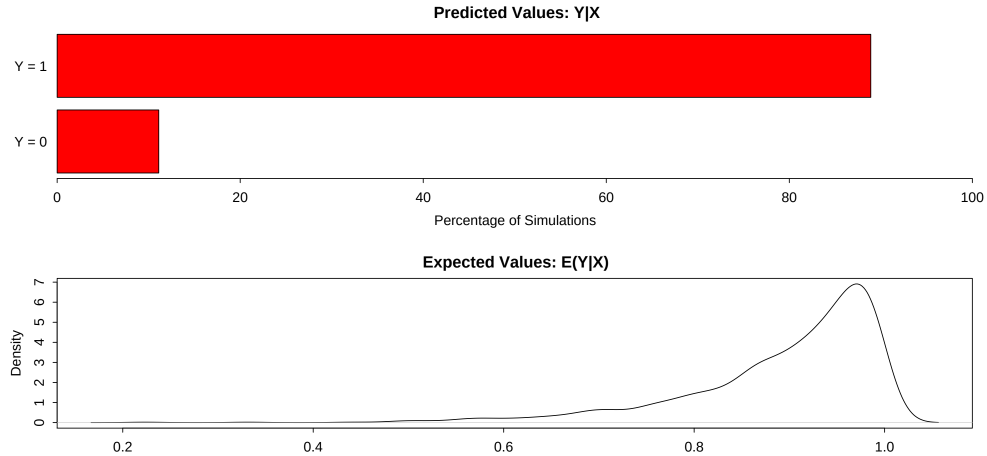

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), H = diag(0.5,
+     37), model = "logit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

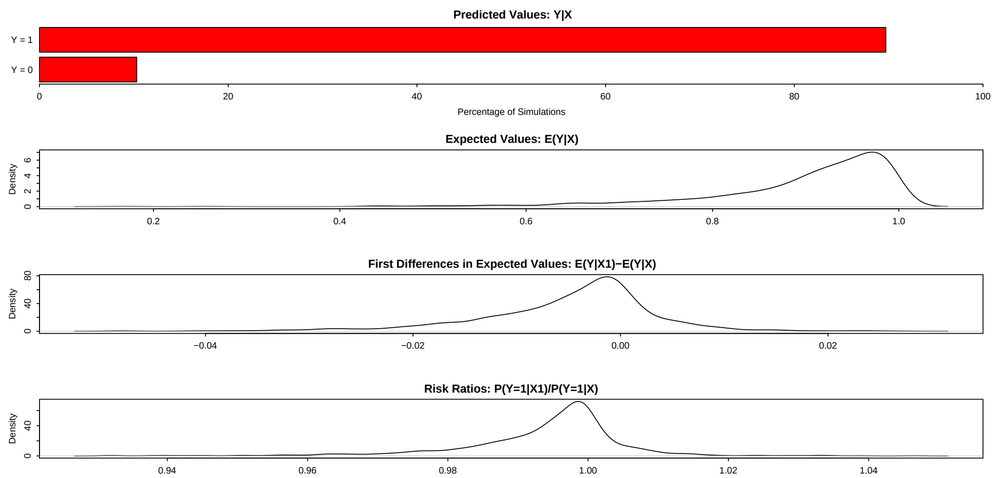
Set the smoothing parameter for the first term, estimate the rest:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), sp = c(0.01,
+     -1, -1, -1), model = "logit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Set lower bounds on smoothing parameters:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), min.sp = c(0.001,
+     0.01, 0, 10), model = "logit.gam", data = my.data)
```






```
> summary(z.out)
> plot(z.out, pages = 1)
```

A GAM with 3df regression spline term & 2 penalized terms:

```
> z.out <- zelig(y ~ s(x0, k = 4, fx = TRUE, bs = "tp") + s(x1,
+     k = 12) + s(x2, k = 15), model = "logit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Model

GAM models use families the same way GLM models do: they specify the distribution and link function to use in model fitting. In the case of `logit.gam` a logistic link function is used. Specifically, let Y_i be the binary dependent variable for observation i which takes the value of either 0 or 1.

- The logistic distribution has *stochastic component*

$$\begin{aligned} Y_i &\sim \text{Bernoulli}(y_i|\pi_i) \\ &= \pi_i^{y_i}(1-\pi_i)^{1-y_i} \end{aligned}$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = \frac{1}{1 + \exp\left(-x_i\beta + \sum_{j=1}^{J} f_j(Z_j)\right)},$$

  where x_i is the vector of covariates, β is the vector of coefficients and $f_j(Z_j)$ for $j = 1, \ldots J$ is the set of smooth terms..

Generalized additive models (GAMs) are similar in many respects to generalized linear models (GLMs). Specifically, GAMs are generally fit by penalized maximum likelihood estimation and GAMs have (or can have) a parametric component identical to that of a GLM. The difference is that GAMs also include in their linear predictors a specified sum of smooth functions.

In this GAM implementation, smooth functions are represented using penalized regression splines. Two techniques may be used to estimate smoothing parameters: Generalized Cross Validation (GCV),

$$n\frac{D}{(n - DF)^2}, \tag{12.2}$$

or an Un-Biased Risk Estimator (UBRE) (which is effectively just a rescaled AIC),

$$\frac{D}{n} + 2s\frac{DF}{n - s}, \tag{12.3}$$

where D is the deviance, n is the number of observations, s is the scale parameter, and DF is the effective degrees of freedom of the model. The use of GCV or UBRE can be set by the user with the `scale` command described in the "Additional Inputs" section and in either case, smoothing parameters are chosen to minimize the GCV or UBRE score for the model.

Estimation for GAM models proceeds as follows: first, basis functions and a set (one or more) of quadratic penalty coefficient matrices are constructed for each smooth term. Second, a model matrix is is obtained for the parametric component of the GAM. These matrices are combined to produce a complete model matrix and a set of penalty matrices for the smooth terms. Iteratively Reweighted Least Squares (IRLS) is then used to estimate the model; at each iteration of the IRLS, a penalized weighted least squares model is run and the smoothing parameters of that model are estimated by GCV or UBRE. This process is repeated until convergence is achieved.

Further details of the GAM fitting process are given in Wood (2000, 2004, 2006).

Quantities of Interest

The quantities of interest for the `logit.gam` model are the same as those for the standard logistic regression.

- The expected value (`qi$ev`) for the `logit.gam` model is the mean of simulations from the stochastic component,

$$\pi_i = \frac{1}{1 + \exp\left(-x_i\beta + \sum_{j=1}^{J} f_j(Z_j)\right)},$$

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the logit.gam model is defined as

$$FD = \Pr(Y|w_1) - \Pr(Y|w)$$

  for $w = \{X, Z\}$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "logit.gam", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `coefficients(z.out)`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IRLS fit.
  - `linear.predictors`: the vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `method`: the fitting method used.
  - `converged`: logical indicating weather the model converged or not.
  - `smooth`: information about the smoothed parameters.
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `data`: the input data frame.

  - `model`: the model matrix used.

- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `p.coeff`: the coefficients of the parametric components of the model.
  - `se`: the standard errors of the entire model.
  - `p.table`: the coefficients, standard errors, and associated t statistics for the parametric portion of the model.
  - `s.table`: the table of estimated degrees of freedom, estimated rank, F statistics, and p-values for the nonparametric portion of the model.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite

To cite the *logit.gam* Zelig model:

> Skyler J. Cranmer. 2007. "logit.gam: Generalized Additive Model for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The `logit.gam` model is adapted from the `mgcv` package by Simon N. Wood (Wood 2006). Advanced users may wish to refer to `help(gam)`, Wood (2004), Wood (2000), and other documentation accompanying the `mgcv` package. All examples are reproduced and extended from `mgcv`'s `gam()` help pages.

12.25 `logit.gee`: Generalized Estimating Equation for Logistic Regression

The GEE logit estimates the same model as the standard logistic regression (appropriate when you have a dichotomous dependent variable and a set of explanatory variables). Unlike in logistic regression, GEE logit allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. The user must first specify a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See Zorn (2001)).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "logit.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:
  - Independence (`corstr = "independence"`): $\mathrm{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard logistic regression. The "working" correlation matrix is the identity matrix.
  - Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

– Stationary m dependent (`corstr = "stat_M_dep"`):

$$\mathrm{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t-t'| \le m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t-t'|$ periods apart for $|t-t'| \le m$.

Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

– Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\mathrm{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t-t'| \le m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t-t'|$ periods apart.

Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

– Exchangeable (`corstr = "exchangeable"`): $\mathrm{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

- Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\text{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

  Sample "working" correlation for Stationary AR-1 (`Mv`=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

- Unstructured (`corstr = "unstructured"`): $\text{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.

- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with Stationary 3 Dependence

   Attaching the sample turnout dataset:

```
> data(turnout)
```

   Variable identifying clusters

```
> turnout$cluster <- rep(c(1:200), 10)
```

   Sorting by cluster

```
> sorted.turnout <- turnout[order(turnout$cluster), ]
```

   Estimating parameter values for the logistic regression:

```
> z.out1 <- zelig(vote ~ race + educate, model = "logit.gee", id = "cluster",
+      data = sorted.turnout, robust = TRUE, corstr = "stat_M_dep",
+      Mv = 3)
```

Setting values for the explanatory variables to their default values:

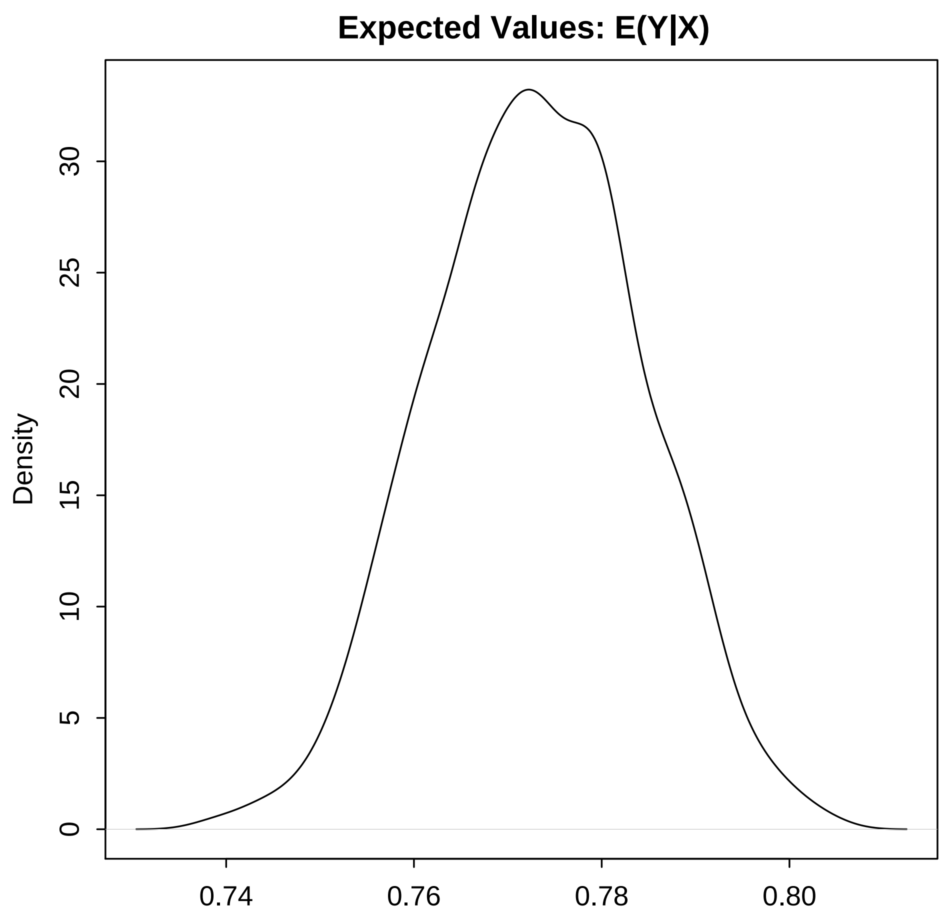
```
> x.out1 <- setx(z.out1)
```

Simulating quantities of interest:

```
> s.out1 <- sim(z.out1, x = x.out1)
```

```
> summary(s.out1)
```

```
> plot(s.out1)
```



2. Simulating First Differences

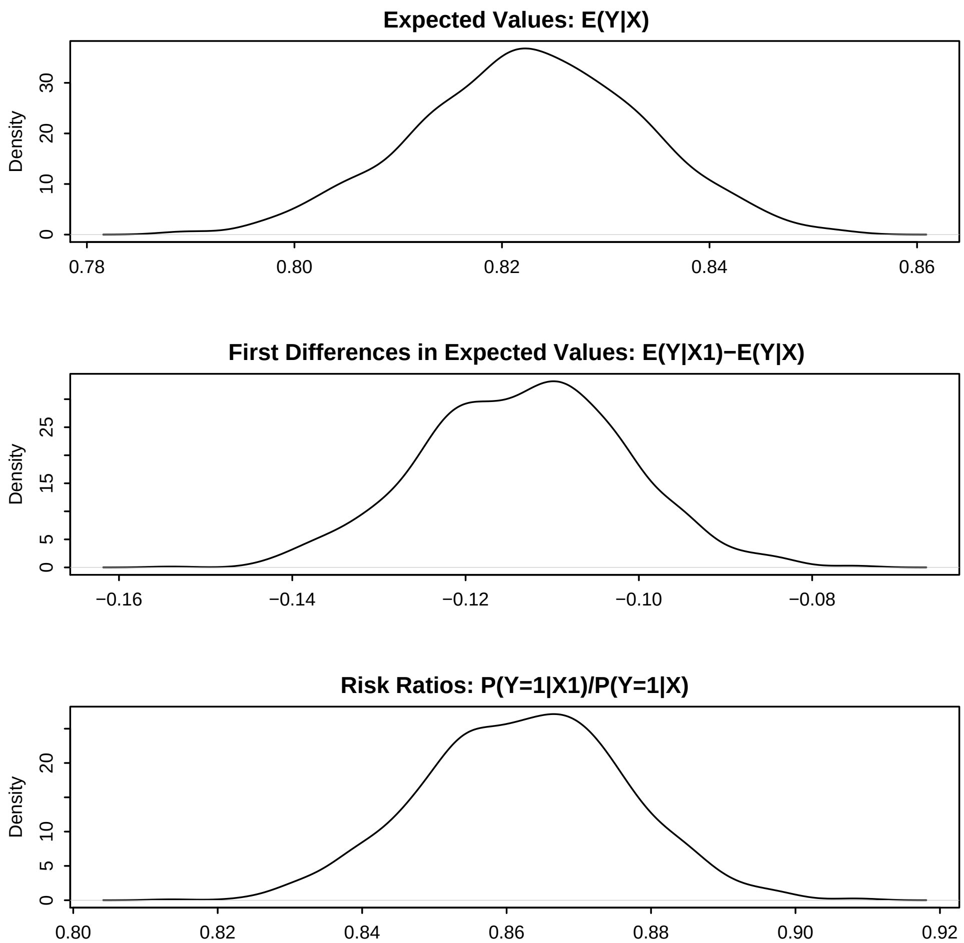
   Estimating the risk difference (and risk ratio) between low education (25th percentile) and high education (75th percentile) while all the other variables held at their default values.

```
> x.high <- setx(z.out1, educate = quantile(turnout$educate, prob = 0.75))
> x.low <- setx(z.out1, educate = quantile(turnout$educate, prob = 0.25))

> s.out2 <- sim(z.out1, x = x.high, x1 = x.low)

> summary(s.out2)

> plot(s.out2)
```



3. Example with Fixed Correlation Structure

   User-defined correlation structure

```
> corr.mat <- matrix(rep(0.5, 100), nrow = 10, ncol = 10)
> diag(corr.mat) <- 1
```

Generating empirical estimates:

```
> z.out2 <- zelig(vote ~ race + educate, model = "logit.gee", id = "cluster",
+     data = sorted.turnout, robust = TRUE, corstr = "fixed", R = corr.mat)
```

Viewing the regression output:

```
> summary(z.out2)
```

The Model

Suppose we have a panel dataset, with Y_{it} denoting the binary dependent variable for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \pi_i) \\ Y_{it} &\sim g(y_{it} \mid \pi_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means π_i and π_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\pi_{it} = \frac{1}{1 + \exp(-x_{it}\beta)}$$

  where x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = \pi_{it}(1 - \pi_{it})$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi \, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = \pi_{it}(1 - \pi_{it})$ as the tth diagonal element, $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.
- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations. See **?** for more details.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.
- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.
- The expected values (`qi$ev`) for the GEE logit model are simulations of the predicted probability of a success:

$$E(Y) = \pi_c = \frac{1}{1 + \exp(-x_c \beta)},$$

  given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.
- The first difference (`qi$fd`) for the GEE logit model is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

$$\text{RR} = \Pr(Y = 1 \mid x_1) \ / \ \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n \sum_{t=1}^T tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it} = 1) - E[Y_{it}(tr_{it} = 0)] \right\},$$

where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "logit.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_{it}.
  - `linear.predictors`: the vector of $x_{it}\beta$
  - `max.id`: the size of the largest cluster.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How To Cite

To cite the *logit.gee* Zelig model:

> Patrick Lam. 2007. "logit.gee: Generalized Estimating Equation for Logit Regression," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from King et al. (2000).

12.26 `logit.gee`: Generalized Estimating Equation for Logistic Regression

The GEE logit estimates the same model as the standard logistic regression (appropriate when you have a dichotomous dependent variable and a set of explanatory variables). Unlike in logistic regression, GEE logit allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. The user must first specify a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See Zorn (2001)).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "logit.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:
  - Independence (`corstr = "independence"`): $\text{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard logistic regression. The "working" correlation matrix is the identity matrix.
  - Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

– Stationary m dependent (`corstr = "stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t-t'|$ periods apart for $|t-t'| \leq m$.

Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

– Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t-t'|$ periods apart.

Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

– Exchangeable (`corstr = "exchangeable"`): $\text{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

- Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\text{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

  Sample "working" correlation for Stationary AR-1 (`Mv`=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

- Unstructured (`corstr = "unstructured"`): $\text{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.
- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with Stationary 3 Dependence

   Attaching the sample turnout dataset:

   ```
   > data(turnout)
   ```

   Variable identifying clusters

   ```
   > turnout$cluster <- rep(c(1:200), 10)
   ```

   Sorting by cluster

   ```
   > sorted.turnout <- turnout[order(turnout$cluster), ]
   ```

   Estimating parameter values for the logistic regression:

```
> z.out1 <- zelig(vote ~ race + educate, model = "logit.gee", id = "cluster",
+      data = sorted.turnout, robust = TRUE, corstr = "stat_M_dep",
+      Mv = 3)
```

Setting values for the explanatory variables to their default values:

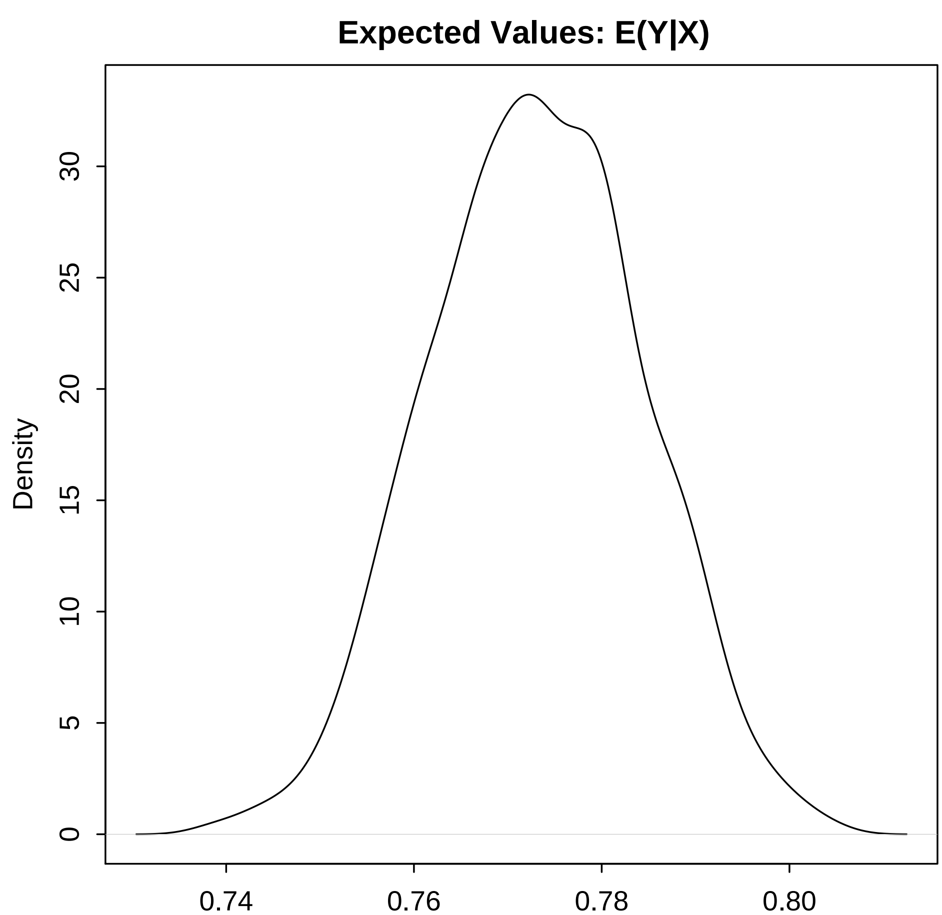
```
> x.out1 <- setx(z.out1)
```

Simulating quantities of interest:

```
> s.out1 <- sim(z.out1, x = x.out1)
```

```
> summary(s.out1)
```

```
> plot(s.out1)
```



2. Simulating First Differences

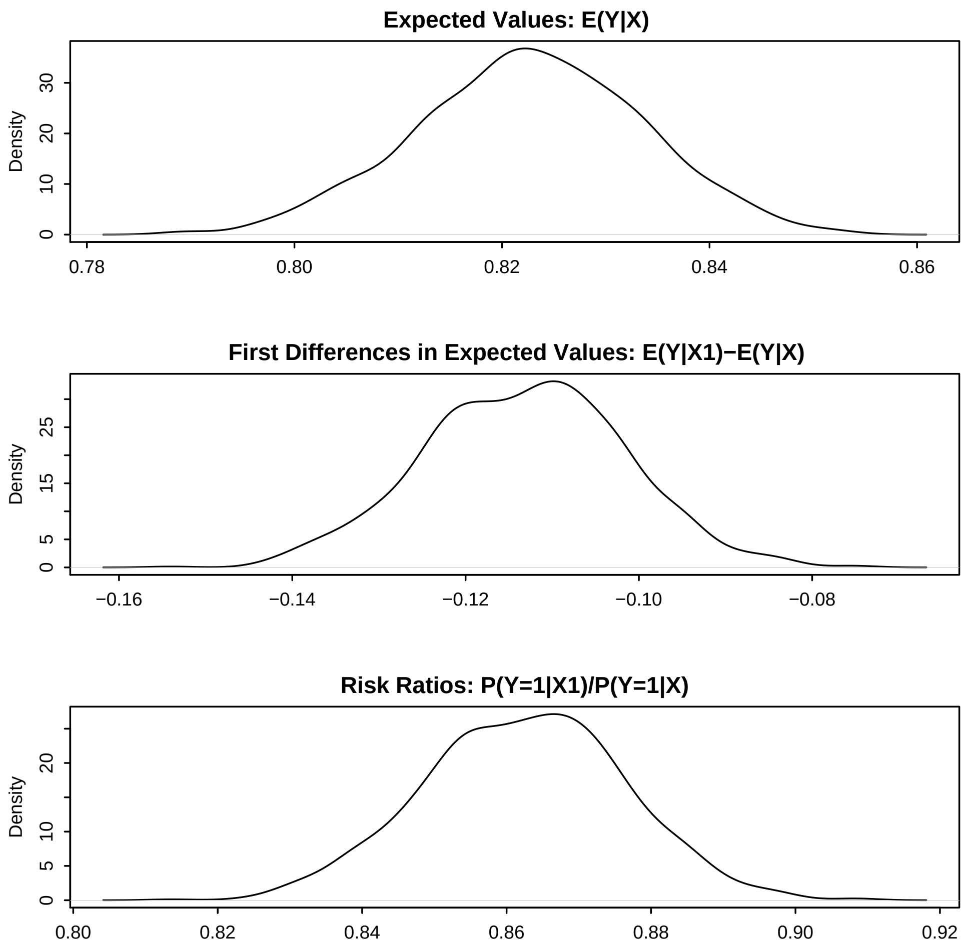
   Estimating the risk difference (and risk ratio) between low education (25th percentile) and high education (75th percentile) while all the other variables held at their default values.

```
> x.high <- setx(z.out1, educate = quantile(turnout$educate, prob = 0.75))
> x.low <- setx(z.out1, educate = quantile(turnout$educate, prob = 0.25))

> s.out2 <- sim(z.out1, x = x.high, x1 = x.low)

> summary(s.out2)

> plot(s.out2)
```



3. Example with Fixed Correlation Structure

   User-defined correlation structure

```
> corr.mat <- matrix(rep(0.5, 100), nrow = 10, ncol = 10)
> diag(corr.mat) <- 1
```

Generating empirical estimates:

```
> z.out2 <- zelig(vote ~ race + educate, model = "logit.gee", id = "cluster",
+     data = sorted.turnout, robust = TRUE, corstr = "fixed", R = corr.mat)
```

Viewing the regression output:

```
> summary(z.out2)
```

The Model

Suppose we have a panel dataset, with Y_{it} denoting the binary dependent variable for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \pi_i) \\ Y_{it} &\sim g(y_{it} \mid \pi_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means π_i and π_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\pi_{it} = \frac{1}{1 + \exp(-x_{it}\beta)}$$

  where x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = \pi_{it}(1 - \pi_{it})$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi \, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = \pi_{it}(1 - \pi_{it})$ as the tth diagonal element, $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.
- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations. See **?** for more details.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.
- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.
- The expected values (`qi$ev`) for the GEE logit model are simulations of the predicted probability of a success:

  $$E(Y) = \pi_c = \frac{1}{1 + \exp(-x_c\beta)},$$

  given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.
- The first difference (`qi$fd`) for the GEE logit model is defined as

  $$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

  $$\text{RR} = \Pr(Y = 1 \mid x_1) \; / \; \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

  $$\frac{1}{\sum_{i=1}^n \sum_{t=1}^T tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it} = 1) - E[Y_{it}(tr_{it} = 0)] \right\},$$

where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "logit.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_{it}.
  - `linear.predictors`: the vector of $x_{it}\beta$
  - `max.id`: the size of the largest cluster.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How To Cite

To cite the *logit.gee* Zelig model:

> Patrick Lam. 2007. "logit.gee: Generalized Estimating Equation for Logit Regression," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from King et al. (2000).

12.27 `logit.mixed`: Mixed effects logistic Regression

Use generalized multi-level linear regression if you have covariates that are grouped according to one or more classification factors. The logit model is appropriate when the dependent variable is dichotomous.

While generally called multi-level models in the social sciences, this class of models is often referred to as mixed-effects models in the statistics literature and as hierarchical models in a Bayesian setting. This general class of models consists of linear models that are expressed as a function of both *fixed effects*, parameters corresponding to an entire population or certain repeatable levels of experimental factors, and *random effects*, parameters corresponding to individual experimental units drawn at random from a population.

Syntax

```
z.out <- zelig(formula= y ~ x1 + x2 + tag(z1 + z2 | g),
               data=mydata, model="logit.mixed")

z.out <- zelig(formula= list(mu=y ~ xl + x2 + tag(z1, gamma | g),
               gamma= ~ tag(w1 + w2 | g)), data=mydata, model="logit.mixed")
```

Inputs

`zelig()` takes the following arguments for `mixed`:

- `formula`: a two-sided linear formula object describing the systematic component of the model, with the response on the left of a ~ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1 + ... + zn | g)` with `z1 + ... + zn` specifying the model for the random effects and `g` the grouping structure. Random intercept terms are included with the notation `tag(1 | g)`.
  Alternatively, `formula` may be a list where the first entry, `mu`, is a two-sided linear formula object describing the systematic component of the model, with the repsonse on the left of a ~ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1, gamma | g)` with `z1` specifying the individual level model for the random effects, `g` the grouping structure and `gamma` references the second equation in the list. The `gamma` equation is one-sided linear formula object with the group level model for the random effects on the right side of a ~ operator. The model is specified with the notation `tag(w1 + ... + wn | g)` with `w1 + ... + wn` specifying the group level model and `g` the grouping structure.

Additional Inputs

In addition, `zelig()` accepts the following additional arguments for model specification:

- `data`: An optional data frame containing the variables named in `formula`. By default, the variables are taken from the environment from which `zelig()` is called.

- `method`: a character string. The criterion is always the log-likelihood but this criterion does not have a closed form expression and must be approximated. The default approximation is `"PQL"` or penalized quasi-likelihood. Alternatives are `"Laplace"` or `"AGQ"` indicating the Laplacian and adaptive Gaussian quadrature approximations respectively.

- `na.action`: A function that indicates what should happen when the data contain `NA`s. The default action (`na.fail`) causes `zelig()` to print an error message and terminate if there are any incomplete observations.

Additionally, users may with to refer to `lmer` in the package `lme4` for more information, including control parameters for the estimation algorithm and their defaults.

Examples

1. Basic Example with First Differences

   Attach sample data:

   ```
   > data(voteincome)
   ```

   Estimate model:

   ```
   > z.out1 <- zelig(vote ~ education + age + female + tag(1 | state),
   +     data = voteincome, model = "logit.mixed")
   ```

   Summarize regression coefficients and estimated variance of random effects:

   ```
   > summary(z.out1)
   ```

   Set explanatory variables to their default values, with high (80th percentile) and low (20th percentile) values for education:

   ```
   > x.high <- setx(z.out1, education = quantile(voteincome$education,
   +     0.8))
   > x.low <- setx(z.out1, education = quantile(voteincome$education,
   +     0.2))
   ```

   Generate first differences for the effect of high versus low education on voting:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
> summary(s.out1)
```

Mixed effects Logistic Regression Model

Let Y_{ij} be the binary dependent variable, realized for observation j in group i as y_{ij} which takes the value of either 0 or 1, for $i = 1, \dots, M$, $j = 1, \dots, n_i$.

- The *stochastic component* is described by a Bernoulli distribution with mean vector π_{ij}.
$$Y_{ij} \sim \text{Bernoulli}(y_{ij}|\pi_{ij}) = \pi_{ij}^{y_{ij}}(1-\pi_{ij})^{1-y_{ij}}$$
where
$$\pi_{ij} = \Pr(Y_{ij} = 1)$$

- The q-dimensional vector of *random effects*, b_i, is restricted to be mean zero, and therefore is completely characterized by the variance covarance matrix Ψ, a $(q \times q)$ symmetric positive semi-definite matrix.
$$b_i \sim Normal(0, \Psi)$$

- The *systematic component* is
$$\pi_{ij} \equiv \frac{1}{1 + \exp(-(X_{ij}\beta + Z_{ij}b_i))}$$
where X_{ij} is the $(n_i \times p \times M)$ array of known fixed effects explanatory variables, β is the p-dimensional vector of fixed effects coefficients, Z_{ij} is the $(n_i \times q \times M)$ array of known random effects explanatory variables and b_i is the q-dimensional vector of random effects.

Quantities of Interest

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value, π_{ij} for
$$\pi_{ij} = \frac{1}{1 + \exp(-(X_{ij}\beta + Z_{ij}b_i))}$$
given X_{ij} and Z_{ij} and simulations of of β and b_i from their posterior distributions. The estimated variance covariance matrices are taken as correct and are themselves not simulated.

- The expected values (`qi$ev`) are simulations of the predicted probability of a success given draws of β from its posterior:
$$E(Y_{ij}|X_{ij}) = \pi_{ij} = \frac{1}{1 + exp(-X_{ij}\beta)}.$$

- The first difference (`qi$fd`) is given by the difference in predicted probabilities, conditional on X_{ij} and X'_{ij}, representing different values of the explanatory variables.

$$FD(Y_{ij}|X_{ij}, X'_{ij}) = Pr(Y_{ij} = 1|X_{ij}) - Pr(Y_{ij} = 1|X'_{ij})$$

- The risk ratio (`qi$rr`) is defined as

$$RR(Y_{ij}|X_{ij}, X'_{ij}) = \frac{Pr(Y_{ij} = 1|X_{ij})}{Pr(Y_{ij} = 1|X'_{ij})}$$

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij}=1) - Y_{ij}\widehat{(t_{ij}=0)}\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $Y_{ij}(t_{ij} = 0)$, the counterfactual predicted value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij}=1) - E[Y_{ij}(t_{ij}=0)]\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $E[Y_{ij}(t_{ij} = 0)]$, the counterfactual expected value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. You may examine the available information in `z.out` by using `slotNames(z.out)`, see the fixed effect coefficients by using `summary(z.out)@coefs`, and a default summary of information through `summary(z.out)`. Other elements available through the operator are listed below.

- From the `zelig()` output stored in `summary(z.out)`, you may extract:
  - `fixef`: numeric vector containing the conditional estimates of the fixed effects.

- `ranef`: numeric vector containing the conditional modes of the random effects.
- `frame`: the model frame for the model.

- From the `sim()` output stored in `s.out`, you may extract quantities of interest stored in a data frame:
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by the expected values.
  - `qi$ev`: the simulated expected values for the specified values of x.
  - `qi$fd`: the simulated first differences in the expected values for the values specified in x and x1.
  - `qi$ate.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.
  - `qi$ate.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *logit.mixed* Zelig model:

> Delia Bailey, Ferdinand Alimadhi. 2007. "logit.mixed: Mixed effects logistic regression" in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Mixed effects logistic regression is part of `lme4` package by Douglas M. Bates (Bates 2007). For a detailed discussion of mixed-effects models, please see **?**

12.28 `logit.mixed`: Mixed effects logistic Regression

Use generalized multi-level linear regression if you have covariates that are grouped according to one or more classification factors. The logit model is appropriate when the dependent variable is dichotomous.

While generally called multi-level models in the social sciences, this class of models is often referred to as mixed-effects models in the statistics literature and as hierarchical models in a Bayesian setting. This general class of models consists of linear models that are expressed as a function of both *fixed effects*, parameters corresponding to an entire population or certain repeatable levels of experimental factors, and *random effects*, parameters corresponding to individual experimental units drawn at random from a population.

Syntax

```
z.out <- zelig(formula= y ~ x1 + x2 + tag(z1 + z2 | g),
               data=mydata, model="logit.mixed")

z.out <- zelig(formula= list(mu=y ~ xl + x2 + tag(z1, gamma | g),
               gamma= ~ tag(w1 + w2 | g)), data=mydata, model="logit.mixed")
```

Inputs

`zelig()` takes the following arguments for `mixed`:

- `formula`: a two-sided linear formula object describing the systematic component of the model, with the response on the left of a ˜ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1 + ... + zn | g)` with `z1 + ... + zn` specifying the model for the random effects and `g` the grouping structure. Random intercept terms are included with the notation `tag(1 | g)`.
  Alternatively, `formula` may be a list where the first entry, `mu`, is a two-sided linear formula object describing the systematic component of the model, with the repsonse on the left of a ˜ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1, gamma | g)` with `z1` specifying the individual level model for the random effects, `g` the grouping structure and `gamma` references the second equation in the list. The `gamma` equation is one-sided linear formula object with the group level model for the random effects on the right side of a ˜ operator. The model is specified with the notation `tag(w1 + ... + wn | g)` with `w1 + ... + wn` specifying the group level model and `g` the grouping structure.

Additional Inputs

In addition, `zelig()` accepts the following additional arguments for model specification:

- `data`: An optional data frame containing the variables named in `formula`. By default, the variables are taken from the environment from which `zelig()` is called.
- `method`: a character string. The criterion is always the log-likelihood but this criterion does not have a closed form expression and must be approximated. The default approximation is `"PQL"` or penalized quasi-likelihood. Alternatives are `"Laplace"` or `"AGQ"` indicating the Laplacian and adaptive Gaussian quadrature approximations respectively.
- `na.action`: A function that indicates what should happen when the data contain `NA`s. The default action (`na.fail`) causes `zelig()` to print an error message and terminate if there are any incomplete observations.

Additionally, users may with to refer to `lmer` in the package `lme4` for more information, including control parameters for the estimation algorithm and their defaults.

Examples

1. Basic Example with First Differences

   Attach sample data:

   ```
   > data(voteincome)
   ```

   Estimate model:

   ```
   > z.out1 <- zelig(vote ~ education + age + female + tag(1 | state),
   +     data = voteincome, model = "logit.mixed")
   ```

   Summarize regression coefficients and estimated variance of random effects:

   ```
   > summary(z.out1)
   ```

   Set explanatory variables to their default values, with high (80th percentile) and low (20th percentile) values for education:

   ```
   > x.high <- setx(z.out1, education = quantile(voteincome$education,
   +     0.8))
   > x.low <- setx(z.out1, education = quantile(voteincome$education,
   +     0.2))
   ```

   Generate first differences for the effect of high versus low education on voting:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
> summary(s.out1)
```

Mixed effects Logistic Regression Model

Let Y_{ij} be the binary dependent variable, realized for observation j in group i as y_{ij} which takes the value of either 0 or 1, for $i = 1, \ldots, M$, $j = 1, \ldots, n_i$.

- The *stochastic component* is described by a Bernoulli distribution with mean vector π_{ij}.
$$Y_{ij} \sim \text{Bernoulli}(y_{ij}|\pi_{ij}) = \pi_{ij}^{y_{ij}}(1-\pi_{ij})^{1-y_{ij}}$$
where
$$\pi_{ij} = \Pr(Y_{ij} = 1)$$

- The q-dimensional vector of *random effects*, b_i, is restricted to be mean zero, and therefore is completely characterized by the variance covarance matrix Ψ, a $(q \times q)$ symmetric positive semi-definite matrix.
$$b_i \sim Normal(0, \Psi)$$

- The *systematic component* is
$$\pi_{ij} \equiv \frac{1}{1 + \exp(-(X_{ij}\beta + Z_{ij}b_i))}$$
where X_{ij} is the $(n_i \times p \times M)$ array of known fixed effects explanatory variables, β is the p-dimensional vector of fixed effects coefficients, Z_{ij} is the $(n_i \times q \times M)$ array of known random effects explanatory variables and b_i is the q-dimensional vector of random effects.

Quantities of Interest

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value, π_{ij} for
$$\pi_{ij} = \frac{1}{1 + \exp(-(X_{ij}\beta + Z_{ij}b_i))}$$
given X_{ij} and Z_{ij} and simulations of of β and b_i from their posterior distributions. The estimated variance covariance matrices are taken as correct and are themselves not simulated.

- The expected values (`qi$ev`) are simulations of the predicted probability of a success given draws of β from its posterior:
$$E(Y_{ij}|X_{ij}) = \pi_{ij} = \frac{1}{1 + exp(-X_{ij}\beta)}.$$

- The first difference (`qi$fd`) is given by the difference in predicted probabilities, conditional on X_{ij} and X'_{ij}, representing different values of the explanatory variables.

$$FD(Y_{ij}|X_{ij}, X'_{ij}) = Pr(Y_{ij} = 1|X_{ij}) - Pr(Y_{ij} = 1|X'_{ij})$$

- The risk ratio (`qi$rr`) is defined as

$$RR(Y_{ij}|X_{ij}, X'_{ij}) = \frac{Pr(Y_{ij} = 1|X_{ij})}{Pr(Y_{ij} = 1|X'_{ij})}$$

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M} \sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij} = 1) - Y_{ij}\widehat{(t_{ij} = 0)}\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $Y_{ij}(t_{ij} = 0)$, the counterfactual predicted value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M} \sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij} = 1) - E[Y_{ij}(t_{ij} = 0)]\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $E[Y_{ij}(t_{ij} = 0)]$, the counterfactual expected value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. You may examine the available information in `z.out` by using `slotNames(z.out)`, see the fixed effect coefficients by using `summary(z.out)@coefs`, and a default summary of information through `summary(z.out)`. Other elements available through the operator are listed below.

- From the `zelig()` output stored in `summary(z.out)`, you may extract:
  - `fixef`: numeric vector containing the conditional estimates of the fixed effects.

  - `ranef`: numeric vector containing the conditional modes of the random effects.
  - `frame`: the model frame for the model.

- From the `sim()` output stored in `s.out`, you may extract quantities of interest stored in a data frame:
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by the expected values.
  - `qi$ev`: the simulated expected values for the specified values of x.
  - `qi$fd`: the simulated first differences in the expected values for the values specified in x and x1.
  - `qi$ate.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.
  - `qi$ate.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *logit.mixed* Zelig model:

> Delia Bailey, Ferdinand Alimadhi. 2007. "logit.mixed: Mixed effects logistic regression" in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Mixed effects logistic regression is part of `lme4` package by Douglas M. Bates (Bates 2007). For a detailed discussion of mixed-effects models, please see **?**

12.29 `logit.net`: Network Logistic Regression for Dichotomous Proximity Matrix Dependent Variables

Use network logistic regression analysis for a dependent variable that is a binary valued proximity matrix (a.k.a. sociomatricies, adjacency matrices, or matrix representations of directed graphs).

Syntax

```
> z.out <- zelig(y ~ x1 + x2, model = "logit.net", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Examples

1. Basic Example

   Load the sample data (see `?friendship` for details on the structure of the network dataframe):

   ```
   > data(friendship)
   ```

   Estimate model:

   ```
   > z.out <- zelig(friends ~ advice + prestige + perpower, model = "logit.net",
   +     data = friendship)
   > summary(z.out)
   ```

   Setting values for the explanatory variables to their default values:

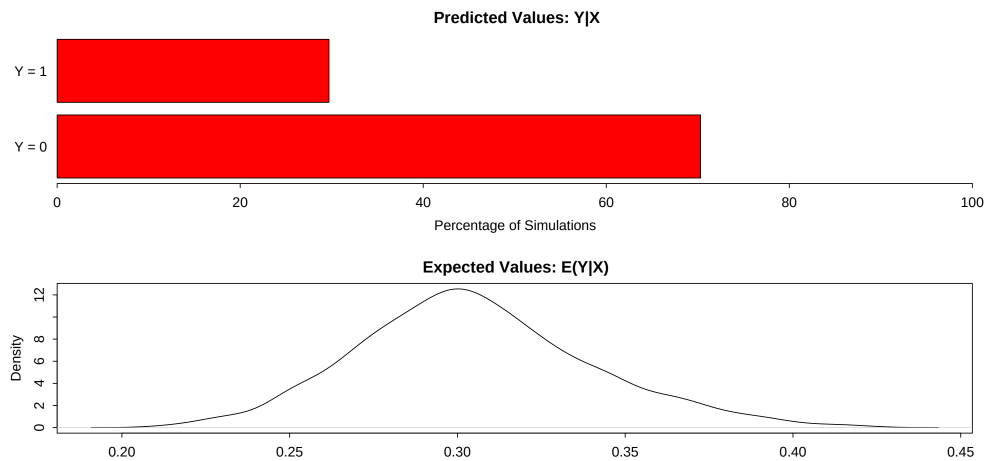
   ```
   > x.out <- setx(z.out)
   ```

   Simulating quantities of interest from the posterior distribution.

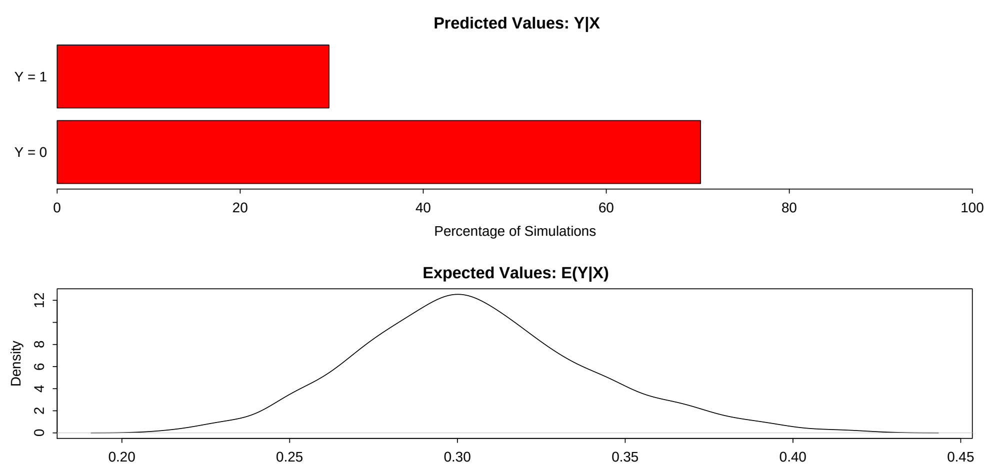
   ```
   > s.out <- sim(z.out, x = x.out)
   > summary(s.out)
   > plot(s.out)
   ```

2. Simulating First Differences

   Estimating the risk difference (and risk ratio) between low personal power (25th percentile) and high personal power (75th percentile) while all the other variables are held at their default values.




```
> x.high <- setx(z.out, perpower = quantile(friendship$perpower,
+     prob = 0.75))
> x.low <- setx(z.out, perpower = quantile(friendship$perpower,
+     prob = 0.25))
> s.out2 <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out2)
> plot(s.out2)
```



Model

The `logit.net` model performs a logistic regression of the proximity matrix $\mathbf{Y}$, a $m \times m$ matrix representing network ties, on a set of proximity matrices $\mathbf{X}$. This network regression model is directly analogous to standard logistic regression element-wise on the appropriately vectorized matrices. Proximity matrices are vectorized by creating Y, a $m^2 \times 1$ vector to represent the proximity matrix. The vectorization which produces the Y vector from the $\mathbf{Y}$ matrix is performed by simple row-concatenation of $\mathbf{Y}$. For example, if $\mathbf{Y}$ is a 15×15 matrix, the $\mathbf{Y}_{1,1}$ element is the first element of Y, and the $\mathbf{Y}_{2,1}$ element is the second element of Y and so on. Once the input matrices are vectorized, standard logistic regression is performed.

Let Y_i be the binary dependent variable, produced by vectorizing a binary proximity matrix, for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$\begin{aligned} Y_i &\sim \text{Bernoulli}(y_i|\pi_i) \\ &= \pi_i^{y_i}(1-\pi_i)^{1-y_i} \end{aligned}$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = \frac{1}{1+\exp(-x_i\beta)}.$$

  where x_i is the vector of k covariates for observation i and β is the vector of coefficients.

Quantities of Interest

The quantities of interest for the network logistic regression are the same as those for the standard logistic regression.

- The expected values (`qi$ev`) for the `logit.net` model are simulations of the predicted probability of a success:

$$E(Y) = \pi_i = \frac{1}{1+\exp(-x_i\beta)},$$

  given draws of β from its sampling distribution.

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the network logit model is defined as

$$FD = \Pr(Y = 1|x_1) - \Pr(Y = 1|x)$$

Output Values

The output of each Zelig command contains useful information which you may view. For example, you run `z.out <- zelig(y ~ x, model = "logit.net", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `linear.predictors`: the vector of $x_i\beta$.
  - `aic`: Akaikés Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `bic`: the Bayesian Information Criterion (minus twice the maximized log-likelihood plus the number of coefficients times $\log n$).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`
- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `mod.coefficients`: the parameter estimates with their associated standard errors, p-values, and t statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite

To cite the *logit.net* Zelig model:

Skyler J. Cranmer. 2007. "logit.net: Network Logistic Regression for Dichotomous Proximity Matrix Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The network logistic regression is part of the `netglm` package by Skyler J. Cranmer and is built using some of the functionality of the `sna` package by Carter T. Butts (Butts and Carley 2001).In addition, advanced users may wish to refer to `help(netgamma)`. Sample data are fictional.

12.30 `logit.survey`: Survey-Weighted Logistic Regression for Dichotomous Dependent Variables

The survey-weighted logistic regression model is appropriate for survey data obtained using complex sampling techniques, such as stratified random or cluster sampling (e.g., not simple random sampling). Like the conventional logistic regression models (see Section 12.22), survey-weighted logistic regression specifies a dichotomous dependent variable as function of a set of explanatory variables. The survey-weighted logit model reports estimates of model parameters identical to conventional logit estimates, but uses information from the survey design to correct variance estimates.

The `logit.survey` model accommodates three common types of complex survey data. Each method listed here requires selecting specific options which are detailed in the "Additional Inputs" section below.

1. **Survey weights**: Survey data are often published along with weights for each observation. For example, if a survey intentionally over-samples a particular type of case, weights can be used to correct for the over-representation of that type of case in the dataset. Survey weights come in two forms:
   (a) *Probability* weights report the probability that each case is drawn from the population. For each stratum or cluster, this is computed as the number of observations in the sample drawn from that group divided by the number of observations in the population in the group.
   (b) *Sampling* weights are the inverse of the probability weights.
2. **Strata/cluster identification**: A complex survey dataset may include variables that identify the strata or cluster from which observations are drawn. For stratified random sampling designs, observations may be nested in different strata. There are two ways to employ these identifiers:
   (a) Use *finite population corrections* to specify the total number of cases in the stratum or cluster from which each observation was drawn.
   (b) For stratified random sampling designs, use the raw strata ids to compute sampling weights from the data.
3. **Replication weights**: To preserve the anonymity of survey participants, some surveys exclude strata and cluster ids from the public data and instead release only pre-computed replicate weights.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "logit.survey", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard `zelig` inputs (see Section ??), survey-weighted logistic models accept the following optional inputs:

1. Datasets that include survey weights:.

    - `probs`: An optional formula or numerical vector specifying each case's probability weight, the probability that the case was selected. Probability weights need not (and, in most cases, will not) sum to one. Cases with lower probability weights are weighted more heavily in the computation of model coefficients.
    - `weights`: An optional numerical vector specifying each case's sample weight, the inverse of the probability that the case was selected. Sampling weights need not (and, in most cases, will not) sum to one. Cases with higher sampling weights are weighted more heavily in the computation of model coefficients.

2. Datasets that include strata/cluster identifiers:

    - `ids`: An optional formula or numerical vector identifying the cluster from which each observation was drawn (ordered from largest level to smallest level). For survey designs that do not involve cluster sampling, `ids` defaults to `NULL`.
    - `fpc`: An optional numerical vector identifying each case's frequency weight, the total number of units in the population from which each observation was sampled.
    - `strata`: An optional formula or vector identifying the stratum from which each observation was sampled. Entries may be numerical, logical, or strings. For survey designs that do not involve cluster sampling, `strata` defaults to `NULL`.
    - `nest`: An optional logical value specifying whether primary sampling unites (PSUs) have non-unique ids across multiple strata. `nest=TRUE` is appropriate when PSUs reuse the same identifiers across strata. Otherwise, `nest` defaults to `FALSE`.
    - `check.strata`: An optional input specifying whether to check that clusters are nested in strata. If `check.strata` is left at its default, `!nest`, the check is not performed. If `check.strata` is specified as `TRUE`, the check is carried out.

3. Datasets that include replication weights:

    - `repweights`: A formula or matrix specifying replication weights, numerical vectors of weights used in a process in which the sample is repeatedly re-weighted and parameters are re-estimated in order to compute the variance of the model parameters.
    - `type`: A string specifying the type of replication weights being used. This input is required if replicate weights are specified. The following types of replication weights are recognized: `"BRR"`, `"Fay"`, `"JK1"`, `"JKn"`, `"bootstrap"`, or `"other"`.

- `weights`: An optional vector or formula specifying each case's sample weight, the inverse of the probability that the case was selected. If a survey includes both sampling weights and replicate weights separately for the same survey, both should be included in the model specification. In these cases, sampling weights are used to correct potential biases in in the computation of coefficients and replication weights are used to compute the variance of coefficient estimates.
- `combined.weights`: An optional logical value that should be specified as `TRUE` if the replicate weights include the sampling weights. Otherwise, `combined.weights` defaults to `FALSE`.
- `rho`: An optional numerical value specifying a shrinkage factor for replicate weights of type `"Fay"`.
- `bootstrap.average`: An optional numerical input specifying the number of iterations over which replicate weights of type `"bootstrap"` were averaged. This input should be left as `NULL` for `"bootstrap"` weights that were not were created by averaging.
- `scale`: When replicate weights are included, the variance is computed as the sum of squared deviations of the replicates from their mean. `scale` is an optional overall multiplier for the standard deviations.
- `rscale`: Like `scale`, `rscale` specifies an optional vector of replicate-specific multipliers for the squared deviations used in variance computation.
- `fpc`: For models in which `"JK1"`, `"JKn"`, or `"other"` replicates are specified, `fpc` is an optional numerical vector (with one entry for each replicate) designating the replicates' finite population corrections.
- `fpctype`: When a finite population correction is included as an `fpc` input, `fpctype` is a required input specifying whether the input to `fpc` is a sampling fraction (`fpctype="fraction"`) or a direct correction (`fpctype="correction"`).
- `return.replicates`: An optional logical value specifying whether the replicates should be returned as a component of the output. `return.replicates` defaults to `FALSE`.

Examples

1. A dataset that includes survey weights:

   Attach the sample data:

   ```
   > data(api, package = "survey")
   ```

   Suppose that a dataset included a dichotomous indicator for whether each public school attends classes year round (`yr.rnd`), a measure of the percentage of students at each school who receive subsidized meals (`meals`), a measure of the percentage of students at each school who are new to to the school (`mobility`), and sampling weights computed by the survey house (`pw`). Estimate a model that regresses the year-round schooling indicator on the `meals` and `mobility` variables:

```
> z.out1 <- zelig(yr.rnd ~ meals + mobility, model = "logit.survey",
+      weights = ~pw, data = apistrat)
```

Summarize regression coefficients:

```
> summary(z.out1)
```

Set explanatory variables to their default (mean/mode) values, and set a high (80th percentile) and low (20th percentile) value for "meals":

```
> x.low <- setx(z.out1, meals = quantile(apistrat$meals, 0.2))
> x.high <- setx(z.out1, meals = quantile(apistrat$meals, 0.8))
```

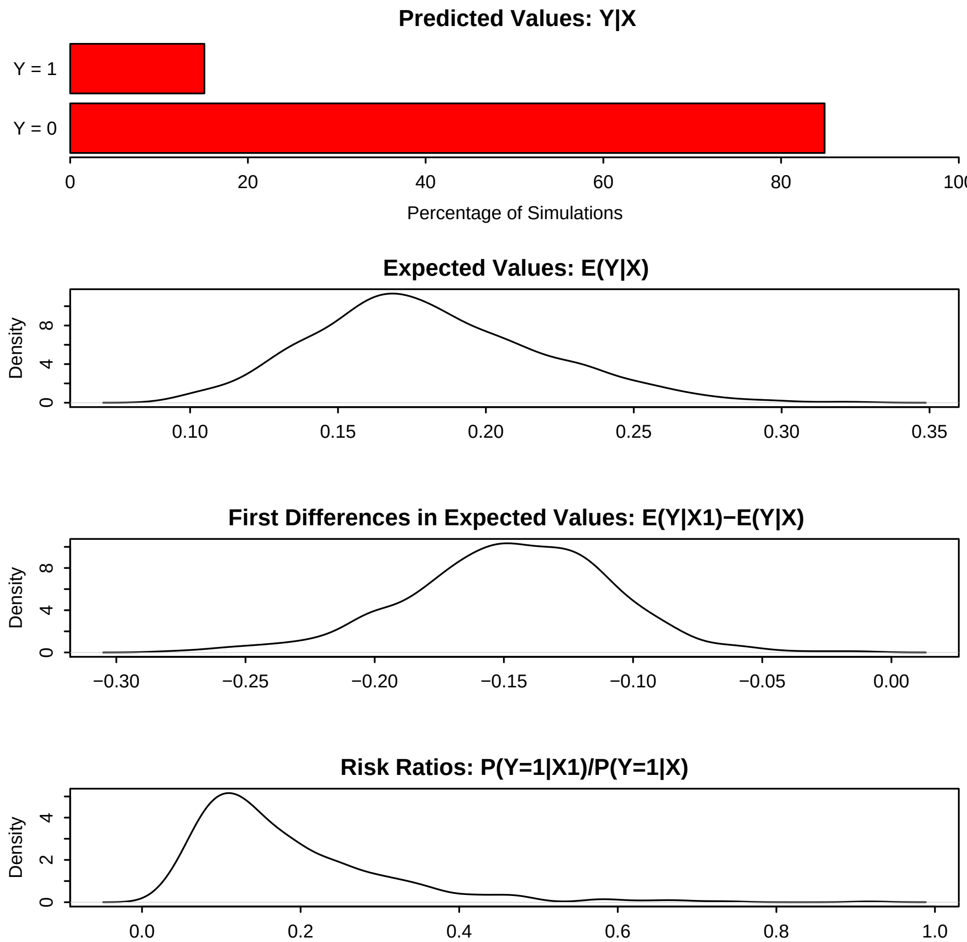
Generate first differences for the effect of high versus low concentrations of children receiving subsidized meals on the probability of holding school year-round:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)

> summary(s.out1)
```

Generate a visual summary of the quantities of interest:

```
> plot(s.out1)
```



2. A dataset that includes strata/cluster identifiers:

   Suppose that the survey house that provided the dataset used in the previous example excluded sampling weights but made other details about the survey design available. A user can still estimate a model without sampling weights that instead uses inputs that identify the stratum and/or cluster to which each observation belongs and the size of the finite population from which each observation was drawn.

   Continuing the example above, suppose the survey house drew at random a fixed number of elementary schools, a fixed number of middle schools, and a fixed number of high schools. If the variable `stype` is a vector of characters (`"E"` for elementary schools, `"M"` for middle schools, and `"H"` for high schools) that identifies the type of school each case represents and `fpc` is a numerical vector that identifies for each case the total number of schools of the same type in the population, then the user could estimate the following model:

```
> z.out2 <- zelig(yr.rnd ~ meals + mobility, model = "logit.survey",
+     strata = ~stype, fpc = ~fpc, data = apistrat)
```

Summarize the regression output:

```
> summary(z.out2)
```

The coefficient estimates for this example are identical to the point estimates in the first example, when pre-existing sampling weights were used. When sampling weights are omitted, they are estimated automatically for `"logit.survey"` models based on the user-defined description of sampling designs.

Moreover, because the user provided information about the survey design, the standard error estimates are lower in this example than in the previous example, in which the user omitted variables pertaining to the details of the complex survey design.

3. A dataset that includes replication weights:

   Consider a dataset that includes information for a sample of hospitals about the number of out-of-hospital cardiac arrests that each hospital treats and the number of patients who arrive alive at each hospital:

   ```
   > data(scd, package = "survey")
   ```

   Survey houses sometimes supply replicate weights (in lieu of details about the survey design). For the sake of illustrating how replicate weights can be used as inputs in `logit.survey` models, create a set of balanced repeated replicate (BRR) weights and an (artificial) dependent variable to simulate an indicator for whether each hospital was sued:

   ```
   > BRRrep <- 2 * cbind(c(1, 0, 1, 0, 1, 0), c(1, 0, 0, 1, 0, 1),
   +     c(0, 1, 1, 0, 0, 1), c(0, 1, 0, 1, 1, 0))
   > scd$sued <- as.vector(c(0, 0, 0, 1, 1, 1))
   ```

   Estimate a model that regresses the indicator for hospitals that were sued on the number of patients who arrive alive in each hospital and the number of cardiac arrests that each hospital treats, using the BRR replicate weights in `BRRrep` to compute standard errors.

   ```
   > z.out3 <- zelig(formula = sued ~ arrests + alive, model = "logit.survey",
   +     repweights = BRRrep, type = "BRR", data = scd)
   ```

   Summarize the regression coefficients:

   ```
   > summary(z.out3)
   ```

   Set `alive` at its mean and set `arrests` at its 20th and 80th quantiles:

   ```
   > x.low <- setx(z.out3, arrests = quantile(scd$arrests, 0.2))
   > x.high <- setx(z.out3, arrests = quantile(scd$arrests, 0.8))
   ```

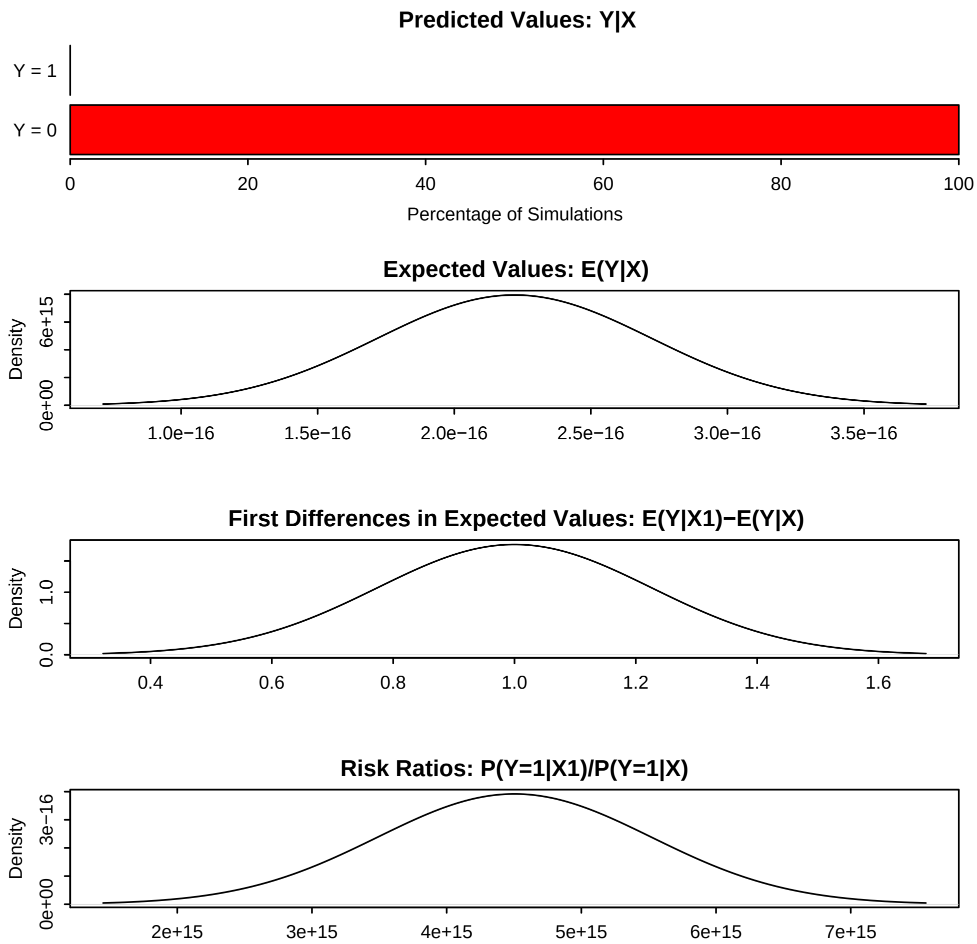
Generate first differences for the effect of high versus low cardiac arrests on the probability that a hospital will be sued:

```
> s.out3 <- sim(z.out3, x = x.high, x1 = x.low)
```

```
> summary(s.out3)
```

Generate a visual summary of quantities of interest:

```
> plot(s.out3)
```



Model

Let Y_i be the binary dependent variable for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$\begin{aligned} Y_i &\sim \text{Bernoulli}(y_i \mid \pi_i) \\ &= \pi_i^{y_i}(1-\pi_i)^{1-y_i} \end{aligned}$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = \frac{1}{1+\exp(-x_i\beta)}.$$

  where x_i is the vector of k explanatory variables for observation i and β is the vector of coefficients.

Variance

When replicate weights are not used, the variance of the coefficients is estimated as

$$\hat{\boldsymbol{\Sigma}}\left[\sum_{i=1}^{n}\frac{(1-\pi_i)}{\pi_i^2}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_i(Y_i-\mu_i)) + 2\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{(\pi_{ij}-\pi_i\pi_j)}{\pi_i\pi_j\pi_{ij}}(\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_j(Y_j-\mu_j))\right]\hat{\boldsymbol{\Sigma}}$$

where π_i is the probability of case i being sampled, $\mathbf{X}_i$ is a vector of the values of the explanatory variables for case i, Y_i is value of the dependent variable for case i, $\hat{\mu}_i$ is the predicted value of the dependent variable for case i based on the linear model estimates, and $\hat{\boldsymbol{\Sigma}}$ is the conventional variance-covariance matrix in a parametric glm. This statistic is derived from the method for estimating the variance of sums described in Binder (1983) and the Horvitz-Thompson estimator of the variance of a sum described in Horvitz and Thompson (1952).

When replicate weights are used, the model is re-estimated for each set of replicate weights, and the variance of each parameter is estimated by summing the squared deviations of the replicates from their mean.

Quantities of Interest

- The expected values (`qi$ev`) for the survey-weighted logit model are simulations of the predicted probability of a success:

$$E(Y) = \pi_i = \frac{1}{1+\exp(-x_i\beta)},$$

  given draws of β from its sampling distribution.

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the survey-weighted logit model is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

$$\text{RR} = \Pr(Y = 1 \mid x_1) \ / \ \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \{Y_i(t_i = 1) - E[Y_i(t_i = 0)]\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}\right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "logit.survey", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.

- `fitted.values`: the vector of fitted values for the systemic component, π_i.
- `linear.predictors`: the vector of $x_i\beta$
- `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
- `df.residual`: the residual degrees of freedom.
- `df.null`: the residual degrees of freedom for the null model.
- `data`: the name of the input data frame.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

When users estimate `logit.survey` models with replicate weights in `Zelig`, an object called `.survey.prob.weights` is created in the global environment. `Zelig` will over-write any existing object with that name, and users are therefore advised to re-name any object called `.survey.prob.weights` before using `logit.survey` models in `Zelig`.

How to Cite

To cite the *logit.survey* Zelig model:

Nicholas Carnes. 2008. "logit.survey: Survey-weighted Logistic Regression for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Survey-weighted linear models and the sample data used in the examples above are a part of the `survey` package by Thomas Lumley. Users may wish to refer to the help files for the three functions that Zelig draws upon when estimating survey-weighted models, namely, `help(svyglm)`, `help(svydesign)`, and `help(svrepdesign)`. The logit model is part of the stats package by Venables and Ripley (2002). Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as McCullagh and Nelder (1989).

12.31 `lognorm`: Log-Normal Regression for Duration Dependent Variables

The log-normal model describes an event's duration, the dependent variable, as a function of a set of explanatory variables. The log-normal model may take time censored dependent variables, and allows the hazard rate to increase and decrease.

Syntax

```
> z.out <- zelig(Surv(Y, C) ~ X, model = "lognorm", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Log-normal models require that the dependent variable be in the form `Surv(Y, C)`, where `Y` and `C` are vectors of length n. For each observation i in 1, ..., n, the value y_i is the duration (lifetime, for example) of each subject, and the associated c_i is a binary variable such that $c_i = 1$ if the duration is not censored (*e.g.*, the subject dies during the study) or $c_i = 0$ if the duration is censored (*e.g.*, the subject is still alive at the end of the study). If c_i is omitted, all Y are assumed to be completed; that is, time defaults to 1 for all observations.

Input Values

In addition to the standard inputs, `zelig()` takes the following additional options for log-normal regression:

- `robust`: defaults to `FALSE`. If `TRUE`, `zelig()` computes robust standard errors based on sandwich estimators (see Huber (1981) and White (1980)) based on the options in `cluster`.
- `cluster`: if `robust = TRUE`, you may select a variable to define groups of correlated observations. Let `x3` be a variable that consists of either discrete numeric values, character strings, or factors that define strata. Then

  ```
  > z.out <- zelig(y ~ x1 + x2, robust = TRUE, cluster = "x3",
                   model = "exp", data = mydata)
  ```

  means that the observations can be correlated within the strata defined by the variable `x3`, and that robust standard errors should be calculated according to those clusters. If `robust = TRUE` but `cluster` is not specified, `zelig()` assumes that each observation falls into its own cluster.

Example

Attach the sample data:

```
> data(coalition)
```

Estimate the model:

```
> z.out <- zelig(Surv(duration, ciep12) ~ fract + numst2, model = "lognorm",
+     data = coalition)
```

View the regression output:

```
> summary(z.out)
```

Set the baseline values (with the ruling coalition in the minority) and the alternative values (with the ruling coalition in the majority) for X:

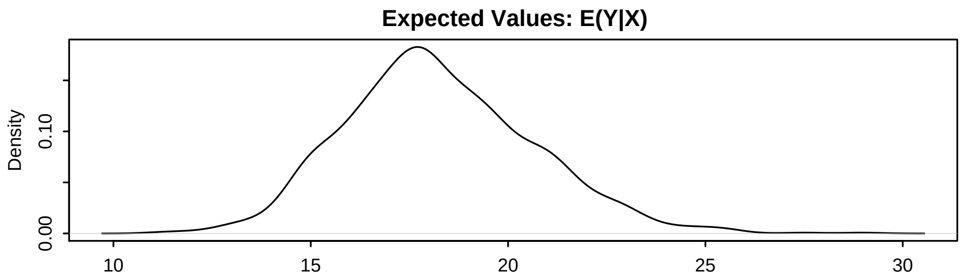
```
> x.low <- setx(z.out, numst2 = 0)
> x.high <- setx(z.out, numst2 = 1)
```

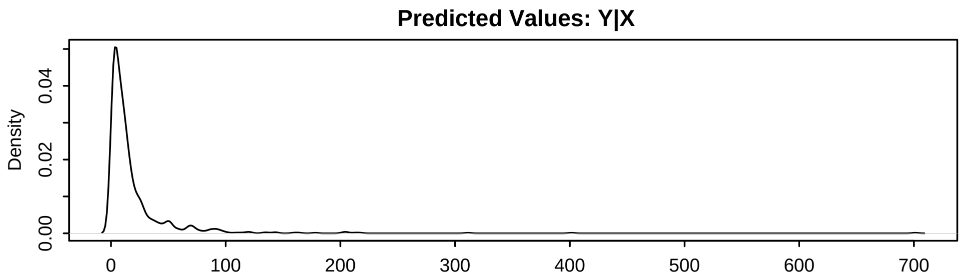
Simulate expected values (`qi$ev`) and first differences (`qi$fd`):

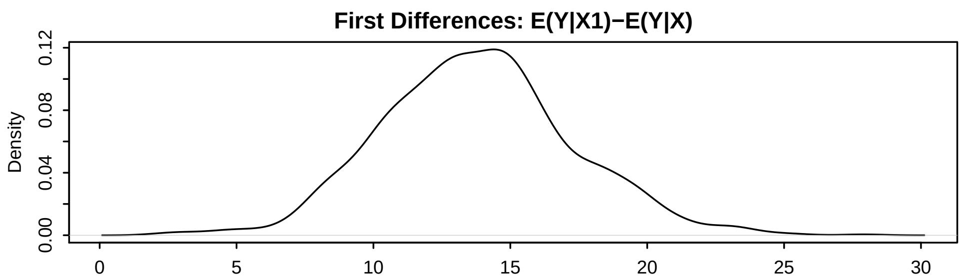
```
> s.out <- sim(z.out, x = x.low, x1 = x.high)
```

```
> summary(s.out)
```

```
> plot(s.out)
```







Model

Let Y_i^* be the survival time for observation i with the density function $f(y)$ and the corresponding distribution function $F(t) = \int_0^t f(y)dy$. This variable might be censored for some observations at a fixed time y_c such that the fully observed dependent variable, Y_i, is defined as

$$Y_i = \begin{cases} Y_i^* & \text{if } Y_i^* \leq y_c \\ y_c & \text{if } Y_i^* > y_c \end{cases}$$

- The *stochastic component* is described by the distribution of the partially observed variable, Y^*. For the lognormal model, there are two equivalent representations:

$$Y_i^* \sim \text{LogNormal}(\mu_i, \sigma^2) \quad \text{or} \quad \log(Y_i^*) \sim \text{Normal}(\mu_i, \sigma^2)$$

  where the parameters μ_i and σ^2 are the mean and variance of the Normal distribution. (Note that the output from `zelig()` parameterizes `scale`$= \sigma$.)

In addition, survival models like the lognormal have three additional properties. The hazard function $h(t)$ measures the probability of not surviving past time t given survival up to t. In general, the hazard function is equal to $f(t)/S(t)$ where the survival function $S(t) = 1 - \int_0^t f(s)ds$ represents the fraction still surviving at time t. The cumulative hazard function $H(t)$ describes the probability of dying before time t. In general, $H(t) = \int_0^t h(s)ds = -\log S(t)$. In the case of the lognormal model,

$$\begin{aligned} h(t) &= \frac{1}{\sqrt{2\pi}\,\sigma t\, S(t)} \exp\left\{-\frac{1}{2\sigma^2}(\log \lambda t)^2\right\} \\ S(t) &= 1 - \Phi\left(\frac{1}{\sigma}\log \lambda t\right) \\ H(t) &= -\log\left\{1 - \Phi\left(\frac{1}{\sigma}\log \lambda t\right)\right\} \end{aligned}$$

where $\Phi(\cdot)$ is the cumulative density function for the Normal distribution.

- The *systematic component* is described as:

$$\mu_i = x_i \beta.$$

Quantities of Interest

- The expected values (`qi$ev`) for the lognormal model are simulations of the expected duration:

$$E(Y) = \exp\left(\mu_i + \frac{1}{2}\sigma^2\right),$$

given draws of β and σ from their sampling distributions.

- The predicted value is a draw from the log-normal distribution given simulations of the parameters (λ_i, σ).

- The first difference (`qi$fd`) is

$$\text{FD} = E(Y \mid x_1) - E(Y \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \{Y_i(t_i = 1) - E[Y_i(t_i = 0)]\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. When $Y_i(t_i = 1)$ is censored rather than observed, we replace it with a simulation from the model given available knowledge of the censoring process. Variation in the simulations is due to two factors: uncertainty in the imputation process for censored y_i^* and uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \{Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. When $Y_i(t_i = 1)$ is censored rather than observed, we replace it with a simulation from the model given available knowledge of the censoring process. Variation in the simulations are due to two factors: uncertainty in the imputation process for censored y_i^* and uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(Surv(Y, C) ~ X, model = "lognorm", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `icoef`: parameter estimates for the intercept and σ.
  - `var`: Variance-covariance matrix.
  - `loglik`: Vector containing the log-likelihood for the model and intercept only (respectively).
  - `linear.predictors`: the vector of $x_i\beta$.
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- Most of this may be conveniently summarized using `summary(z.out)`. From `summary(z.out)`, you may additionally extract:
  - `table`: the parameter estimates with their associated standard errors, p-values, and t-statistics.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation × `x`-observation (for more than one `x`-observation). Available quantities are:

  – `qi$ev`: the simulated expected values for the specified values of `x`.

  – `qi$pr`: the simulated predicted values drawn from the distribution defined by (λ_i, σ).

  – `qi$fd`: the simulated first differences between the simulated expected values for `x` and `x1`.

  – `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

  – `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *lognorm* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "lognorm: Log-Normal Regression for Duration Dependent Variable," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The exponential function is part of the survival library by by Terry Therneau, ported to R by Thomas Lumley. Advanced users may wish to refer to `help(survfit)` in the survival library, and Venables and Ripley (2002).Sample data are from King et al. (1990a).

12.32 `ls`: Least Squares Regression for Continuous Dependent Variables

Use least squares regression analysis to estimate the best linear predictor for the specified dependent variables.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "ls", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for least squares regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors based on sandwich estimators (see Zeileis (2004), Huber (1981), and White (1980)). The default type of robust standard error is heteroskedastic consistent (HC), *not* heteroskedastic and autocorrelation consistent (HAC).

  In addition, `robust` may be a list with the following options:

  - `method`: choose from
    - `"vcovHC"`: (the default if `robust = TRUE`), HC standard errors.
    - `"vcovHAC"`: HAC standard errors without weights.
    - `"kernHAC"`: HAC standard errors using the weights given in Andrews (1991).
    - `"weave"`: HAC standard errors using the weights given in Lumley and Heagerty (1999).
  - `order.by`: only applies to the HAC methods above. Defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a time index (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and Zeileis (2004) for more options.

Examples

1. Basic Example with First Differences

   Attach sample data:

```
> data(macro)
```

Estimate model:

```
> z.out1 <- zelig(unem ~ gdp + capmob + trade, model = "ls", data = macro)
```

Summarize regression coefficients:

```
> summary(z.out1)
```

Set explanatory variables to their default (mean/mode) values, with high (80th percentile) and low (20th percentile) values for the trade variable:

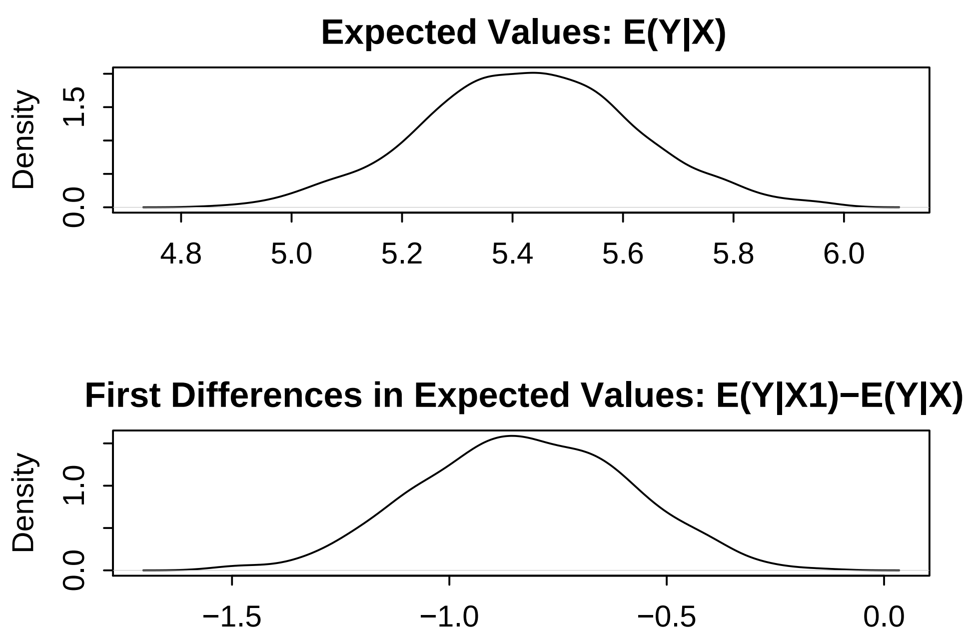
```
> x.high <- setx(z.out1, trade = quantile(macro$trade, 0.8))
> x.low <- setx(z.out1, trade = quantile(macro$trade, 0.2))
```

Generate first differences for the effect of high versus low trade on GDP:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
```

```
> summary(s.out1)
```

```
> plot(s.out1)
```



2. Using Dummy Variables

   Estimate a model with fixed effects for each country (see Section 2 for help with dummy variables). Note that you do not need to create dummy variables, as the program will automatically parse the unique values in the selected variable into discrete levels.

```
> z.out2 <- zelig(unem ~ gdp + trade + capmob + as.factor(country),
+      model = "ls", data = macro)
```

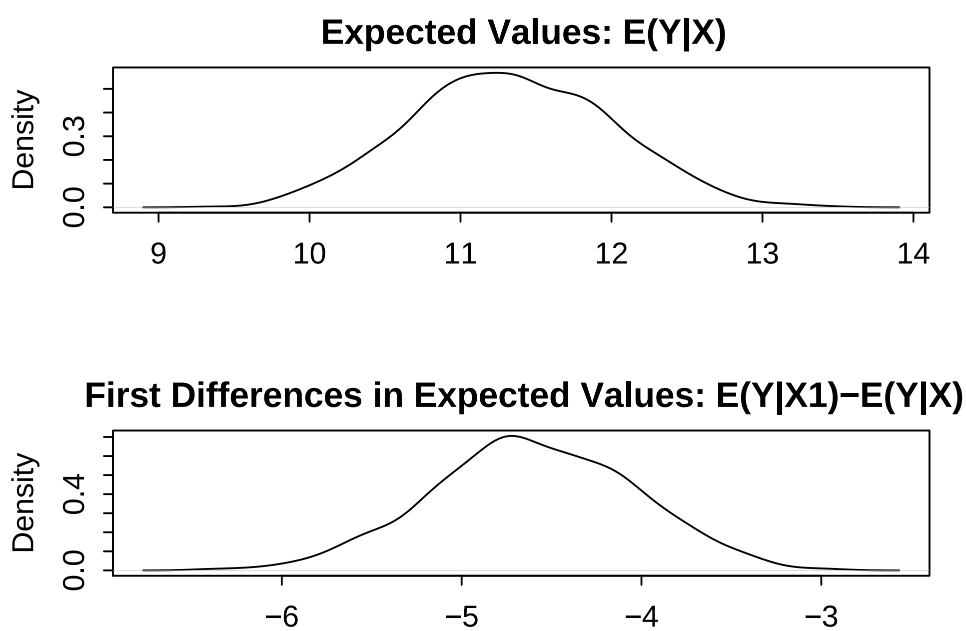
   Set values for the explanatory variables, using the default mean/mode values, with country set to the United States and Japan, respectively:

```
> x.US <- setx(z.out2, country = "United States")
> x.Japan <- setx(z.out2, country = "Japan")
```

   Simulate quantities of interest:

```
> s.out2 <- sim(z.out2, x = x.US, x1 = x.Japan)
```

```
> plot(s.out2)
```



3. Multiple responses (least squares regression will be fitted separately to each dependent variable)

   Two responses for data set macro:

```
> z.out3 <- zelig(cbind(unem, gdp) ~ capmob + trade, model = "ls",
+     data = macro)
```

```
> summary(z.out3)
```

Set values for the explanatory variables, using the default mean/mode values, with country set to the United States and Japan, respectively:

```
> x.US <- setx(z.out3, country = "United States")
> x.Japan <- setx(z.out3, country = "Japan")
```

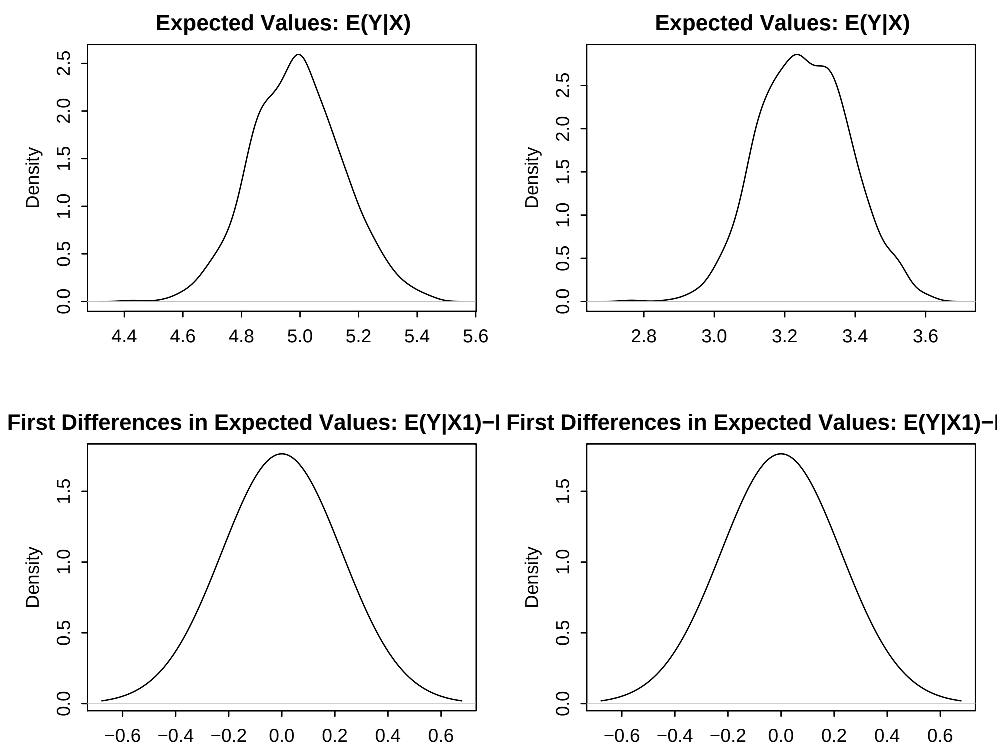
Simulate quantities of interest:

```
> s.out3 <- sim(z.out3, x = x.US, x1 = x.Japan)
```

Summary

```
> summary(s.out3)
```

```
> plot(s.out3)
```



Model

- The *stochastic component* is described by a density with mean μ_i and the common variance σ^2
$$Y_i \sim f(y_i \mid \mu_i, \sigma^2).$$
- The *systematic component* models the conditional mean as
$$\mu_i = x_i\beta$$
where x_i is the vector of covariates, and β is the vector of coefficients.

  The least squares estimator is the best linear predictor of a dependent variable given x_i, and minimizes the sum of squared residuals, $\sum_{i=1}^n (Y_i - x_i\beta)^2$.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the stochastic component,
$$E(Y) = x_i\beta,$$
given a draw of β from its sampling distribution.
- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is
$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$
where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "ls", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.

- `residuals`: the working residuals in the final iteration of the IWLS fit.
- `fitted.values`: fitted values.
- `df.residual`: the residual degrees of freedom.
- `zelig.data`: the input data frame if `save.data = TRUE`.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.

$$\hat{\beta} = \left(\sum_{i=1}^{n} x_i' x_i\right)^{-1} \sum x_i y_i$$

  - `sigma`: the square root of the estimate variance of the random error e:

$$\hat{\sigma} = \frac{\sum (Y_i - x_i \hat{\beta})^2}{n - k}$$

  - `r.squared`: the fraction of the variance explained by the model.

$$R^2 = 1 - \frac{\sum (Y_i - x_i \hat{\beta})^2}{\sum (y_i - \bar{y})^2}$$

  - `adj.r.squared`: the above R^2 statistic, penalizing for an increased number of explanatory variables.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$fd`: the simulated first differences (or differences in expected values) for the specified values of `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *ls* Zelig model:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "ls: Least Squares Regression for Continuous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The least squares regression is part of the stats package by William N. Venables and Brian D. Ripley (Venables and Ripley 2002).In addition, advanced users may wish to refer to `help(lm)` and `help(lm.fit)`.Robust standard errors are implemented via the sandwich package by Achim Zeileis (Zeileis 2004).Sample data are from King et al. (2000).

12.33 `ls.mixed`: Mixed effects Linear Regression

Use multi-level linear regression if you have covariates that are grouped according to one or more classification factors and a continuous dependent variable.

While generally called multi-level models in the social sciences, this class of models is often referred to as mixed-effects models in the statistics literature and as hierarchical models in a Bayesian setting. This general class of models consists of linear models that are expressed as a function of both *fixed effects*, parameters corresponding to an entire population or certain repeatable levels of experimental factors, and *random effects*, parameters corresponding to individual experimental units drawn at random from a population.

Syntax

```
z.out <- zelig(formula= y ~ x1 + x2 + tag(z1 + z2 | g),
               data=mydata, model="lm.multi")

z.out <- zelig(formula= list(mu=y ~ xl + x2 + tag(z1, gamma | g),
               gamma= ~ tag(w1 + w2 | g)), data=mydata, model="lm.multi")
```

Inputs

`zelig()` takes the following arguments for `multi`:

- `formula`: a two-sided linear formula object describing the systematic component of the model, with the response on the left of a ˜ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1 + ... + zn | g)` with `z1 + ... + zn` specifying the model for the random effects and `g` the grouping structure. Random intercept terms are included with the notation `tag(1 | g)`.
  Alternatively, `formula` may be a list where the first entry, `mu`, is a two-sided linear formula object describing the systematic component of the model, with the repsonse on the left of a ˜ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1, gamma | g)` with `z1` specifying the individual level model for the random effects, `g` the grouping structure and `gamma` references the second equation in the list. The `gamma` equation is one-sided linear formula object with the group level model for the random effects on the right side of a ˜ operator. The model is specified with the notation `tag(w1 + ... + wn | g)` with `w1 + ... + wn` specifying the group level model and `g` the grouping structure.

Additional Inputs

In addition, `zelig()` accepts the following additional arguments for model specification:

- `data`: An optional data frame containing the variables named in `formula`. By default, the variables are taken from the environment from which `zelig()` is called.
- `family`: A GLM family, see `glm` and `family` in the `stats` package. If `family` is missing then a linear mixed model is fit; otherwise a generalized linear mixed model is fit. In the later case only `gaussian` family with `"log"` link is supported at the moment.
- `method`: A character string. For a linear mixed model the default is `"REML"` indicating that the model should be fit by maximizing the restricted log-likelihood. The alternative is `"ML"` indicating that the log-likelihood should be maximized.
- `na.action`: A function that indicates what should happen when the data contain `NA`s. The default action (`na.fail`) causes `zelig()` to print an error message and terminate if there are any incomplete observations.

Additionally, users may wish to refer to `lmer` in the package `lme4` for more information, including control parameters for the estimation algorithm and their defaults.

Examples

1. Basic Example with First Differences

   Attach sample data:

   ```
   > data(voteincome)
   ```

   Estimate model:

   ```
   > z.out1 <- zelig(income ~ education + age + female + tag(1 | state),
   +     data = voteincome, model = "ls.mixed")
   ```

   Summarize regression coefficients and estimated variance of random effects:

   ```
   > summary(z.out1)
   ```

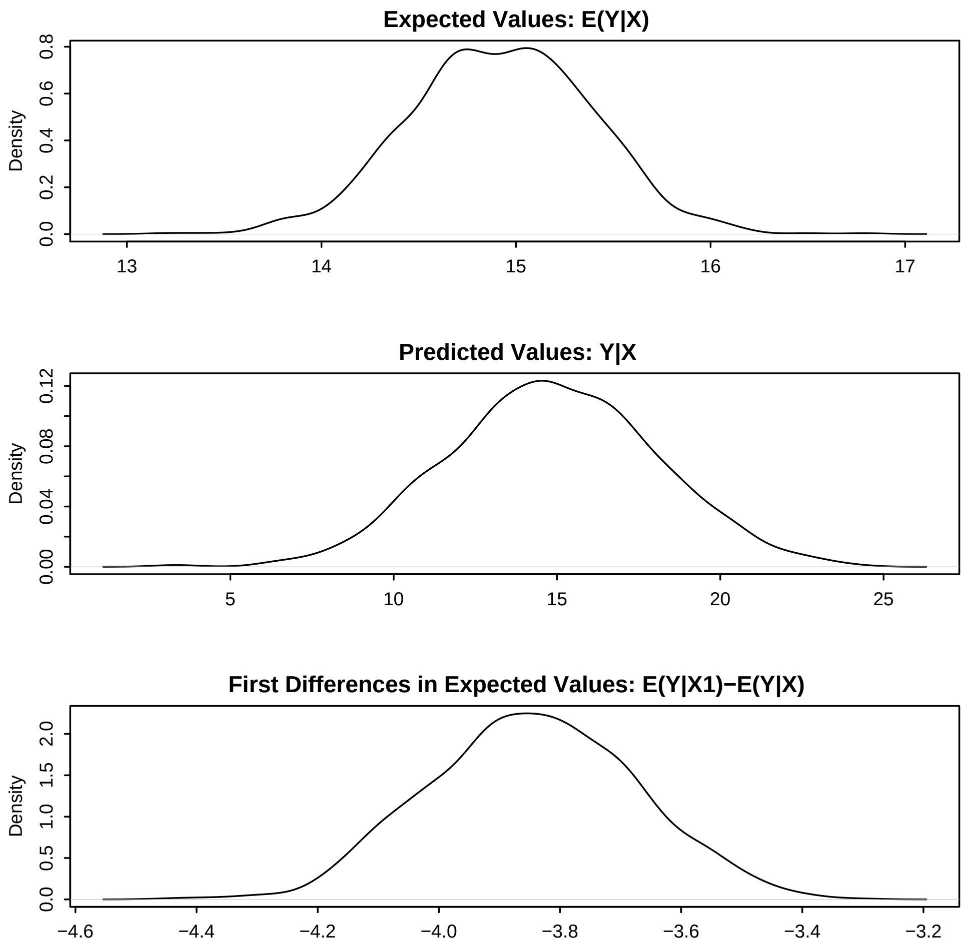
   Set explanatory variables to their default values, with high (80th percentile) and low (20th percentile) values for education:

   ```
   > x.high <- setx(z.out1, education = quantile(voteincome$education,
   +     0.8))
   > x.low <- setx(z.out1, education = quantile(voteincome$education,
   +     0.2))
   ```

Generate first differences for the effect of high versus low education on income:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
> summary(s.out1)

> plot(s.out1)
```



Mixed effects linear regression model

Let Y_{ij} be the continuous dependent variable, realized for observation j in group i as y_{ij}, for $i = 1, \ldots, M$, $j = 1, \ldots, n_i$.

- The *stochastic component* is described by a univariate normal model with a vector of means μ_{ij} and scalar variance σ^2.

$$Y_{ij} \sim \text{Normal}(y_{ij}|\mu_{ij}, \sigma^2)$$

- The q-dimensional vector of *random effects*, b_i, is restricted to be mean zero, and therefore is completely characterized by the variance covarance matrix Ψ, a $(q \times q)$ symmetric positive semi-definite matrix.

$$b_i \sim Normal(0, \Psi)$$

- The *systematic component* is

$$\mu_{ij} \equiv X_{ij}\beta + Z_{ij}b_i$$

where X_{ij} is the $(n_i \times p \times M)$ array of known fixed effects explanatory variables, β is the p-dimensional vector of fixed effects coefficients, Z_{ij} is the $(n_i \times q \times M)$ array of known random effects explanatory variables and b_i is the q-dimensional vector of random effects.

Quantities of Interest

- The predicted values (`qi$pr`) are draws from the normal distribution defined by mean μ_{ij} and variance σ^2,

$$\mu_{ij} = X_{ij}\beta + Z_{ij}b_i$$

given X_{ij} and Z_{ij} and simulations of β and b_i from their posterior distributions. The estimated variance covariance matrices are taken as correct and are themselves not simulated.

- The expected values (`qi$ev`) are averaged over the stochastic components and are given by

$$E(Y_{ij}|X_{ij}) = X_{ij}\beta.$$

- The first difference (`qi$fd`) is given by the difference in expected values, conditional on X_{ij} and X'_{ij}, representing different values of the explanatory variables.

$$FD(Y_{ij}|X_{ij}, X'_{ij}) = E(Y_{ij}|X_{ij}) - E(Y_{ij}|X'_{ij})$$

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij}=1) - Y_{ij}\widehat{(t_{ij}=0)}\},$$

where t_{ij} is a binary explanatory variable defining the treatment $(t_{ij} = 1)$ and control $(t_{ij} = 0)$ groups. Variation in the simulations is due to uncertainty in simulating $Y_{ij}(t_{ij} = 0)$, the counterfactual predicted value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij}=1) - E[Y_{ij}(t_{ij}=0)]\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $E[Y_{ij}(t_{ij} = 0)]$, the counterfactual expected value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

- If `"log"` link is used, expected values are computed as above and then exponentiated, while predicted values are draws from the log-normal distribution whose logarithm has mean and variance equal to μ_{ij} and σ^2, respectively.

Output Values

The output of each Zelig command contains useful information which you may view. You may examine the available information in `z.out` by using `slotNames(z.out)`, see the fixed effect coefficients by using `summary(z.out)@coefs`, and a default summary of information through `summary(z.out)`. Other elements available through the operator are listed below.

- From the `zelig()` output stored in `summary(z.out)`, you may extract:
  - `fixef`: numeric vector containing the conditional estimates of the fixed effects.
  - `ranef`: numeric vector containing the conditional modes of the random effects.
  - `frame`: the model frame for the model.
- From the `sim()` output stored in `s.out`, you may extract quantities of interest stored in a data frame:
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by the expected values.
  - `qi$ev`: the simulated expected values for the specified values of x.
  - `qi$fd`: the simulated first differences in the expected values for the values specified in x and x1.
  - `qi$ate.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.
  - `qi$ate.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *ls.mixed* Zelig model:

> Delia Bailey, Ferdinand Alimadhi. 2007. "ls.mixed: Mixed effects linear regression" in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Mixed effects linear regression is part of `lme4` package by Douglas M. Bates (Bates 2007). For a detailed discussion of mixed-effects models, please see **?**

12.34 `ls.net`: Network Least Squares Regression for Continuous Proximity Matrix Dependent Variables

Use network least squares regression analysis to estimate the best linear predictor when the dependent variable is a continuously-valued proximity matrix (a.k.a. sociomatrices, adjacency matrices, or matrix representations of directed graphs).

Syntax

```
> z.out <- zelig(y ~  x1 + x2, model = "ls.net", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Examples

1. Basic Example with First Differences

   Load sample data and format it for social networkx analysis:

   ```
   > data(sna.ex)
   ```

   Estimate model:

   ```
   > z.out <- zelig(Var1 ~ Var2 + Var3 + Var4, model = "ls.net", data = sna.ex)
   ```

   Summarize regression results:

   ```
   > summary(z.out)
   ```

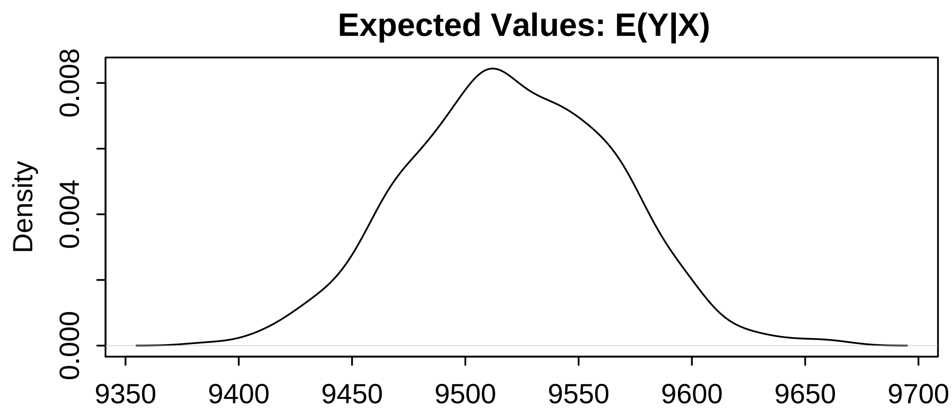
   Set explanatory variables to their default (mean/mode) values, with high (80th percentile) and low (20th percentile) for the second explanatory variable (Var3).

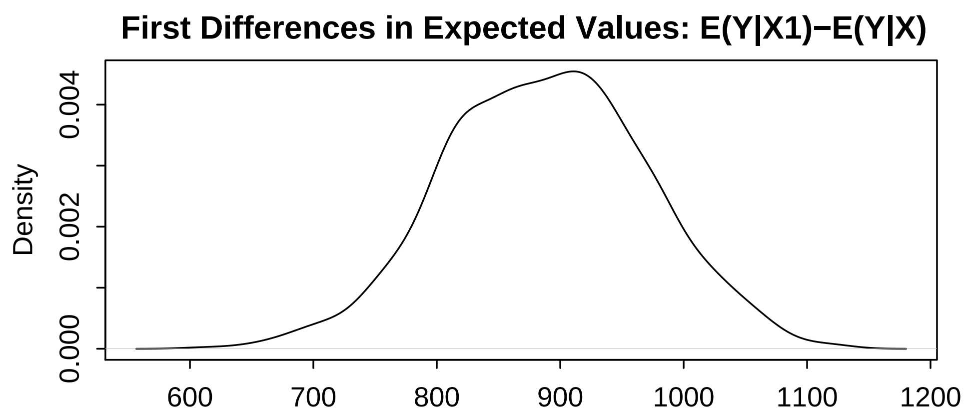
   ```
   > x.high <- setx(z.out, Var3 = quantile(sna.ex$Var3, 0.8))
   > x.low <- setx(z.out, Var3 = quantile(sna.ex$Var3, 0.2))
   ```

   Generate first differences for the effect of high versus low values of Var3 on the outcome variable.

   ```
   > try(s.out <- sim(z.out, x = x.high, x1 = x.low))
   > try(summary(s.out))
   ```

   ```
   > plot(s.out)
   ```





Model

The `ls.net` model performs a least squares regression of the sociomatrix $\mathbf{Y}$, a $m \times m$ matrix representing network ties, on a set of sociomatrices $\mathbf{X}$. This network regression model is a directly analogue to standard least squares regression element-wise on the appropriately vectorized matrices. Sociomatrices are vectorized by creating Y, an $m^2 \times 1$ vector to represent the sociomatrix. The vectorization which produces the Y vector from the $\mathbf{Y}$ matrix is preformed by simple row-concatenation of $\mathbf{Y}$. For example if $\mathbf{Y}$ is a 15×15 matrix, the $\mathbf{Y}_{1,1}$ element is the first element of Y, and the $\mathbf{Y}_{21}$ element is the second element of Y and so on. Once the input matrices are vectorized, standard least squares regression is performed. As such:

- The *stochastic component* is described by a density with mean μ_i and the common variance σ^2

$$Y_i \sim f(y_i|\mu_i, \sigma^2).$$

- The *systematic component* models the conditional mean as

$$\mu_i = x_i \beta$$

where x_i is the vector of covariates, and β is the vector of coefficients.

The least squares estimator is the best linear predictor of a dependent variable given x_i, and minimizes the sum of squared errors $\sum_{i=1}^{n}(Y_i - x_i\beta)^2$.

Quantities of Interest

The quantities of interest for the network least squares regression are the same as those for the standard least squares regression.

- The expected value (`qi$ev`) is the mean of simulations from the stochastic component,

$$E(Y) = x_i \beta,$$

given a draw of β from its sampling distribution.

- The first difference (`qi$fd`) is:

$$FD = E(Y|x_1) - E(Y|x)$$

Output Values

The output of each Zelig command contains useful information which you may view. For example, you run `z.out <- zelig(y   x, model="ls.net", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `df.residual`: the residual degrees of freedom.
  - `zelig.data`: the input data frame if `save.data = TRUE`
- From `summary(z.out)`, you may extract:
  - `mod.coefficients`: the parameter estimates with their associated standard errors, p-values, and t statistics.

$$\hat{\beta} = \left(\sum_{i=1}^{n} x_i' x_i\right)^{-1} \sum x_i y_i$$

– `sigma`: the square root of the estimate variance of the random error ε:

$$\hat{\sigma} = \frac{\sum(Y_i - x_i\hat{\beta})^2}{n-k}$$

– `r.squared`: the fraction of the variance explained by the model.

$$R^2 = 1 - \frac{\sum(Y_i - x_i\hat{\beta})^2}{\sum(y_i - \bar{y})^2}$$

– `adj.r.squared`: the above R^2 statistic, penalizing for an increased number of explanatory variables.

– `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:

  – `qi$ev`: the simulated expected values for the specified values of `x`.

  – `qi$fd`: the simulated first differences (or differences in expected values) for the specified values of `x` and `x1`.

How to Cite

To cite the *ls.net* Zelig model:

> Skyler J. Cranmer. 2007. "ls.net: Network Least Squares Regression for Continuous Proximity Matrix Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The network least squares regression is part of the sna package by Carter T. Butts (Butts and Carley 2001).In addition, advanced users may wish to refer to `help(netlm)`.

12.35 `mlogit`: Multinomial Logistic Regression for Dependent Variables with Unordered Categorical Values

Use the multinomial logit distribution to model unordered categorical variables. The dependent variable may be in the format of either character strings or integer values. See for a Bayesian version of this model.

Syntax

```
> z.out <- zelig(as.factor(Y) ~ X1 + X2, model = "mlogit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Input Values

If the user wishes to use the same formula across all levels, then `formula <- as.factor(Y) ~ X1 + X2` may be used. If the user wants to use different formula for each level then the following syntax should be used:

```
formulae <- list(list(id(Y, "apples")~ X1,
                      id(Y, "bananas")~ X1 + X2)
```

where Y above is supposed to be a factor variable with levels apples,bananas,oranges. By default, oranges is the last level and omitted. (You cannot specify a different base level at this time.) For J equations, there must be $J + 1$ levels.

Examples

1. The same formula for each level

   Load the sample data:

   ```
   > data(mexico)
   ```

   Estimate the empirical model:

   ```
   > z.out1 <- zelig(as.factor(vote88) ~ pristr + othcok + othsocok,
   +     model = "mlogit", data = mexico)
   ```

   Set the explanatory variables to their default values, with `pristr` (for the strength of the PRI) equal to 1 (weak) in the baseline values, and equal to 3 (strong) in the alternative values:

   ```
   > x.weak <- setx(z.out1, pristr = 1)
   > x.strong <- setx(z.out1, pristr = 3)
   ```

Generate simulated predicted probabilities `qi$ev` and differences in the predicted probabilities `qi$fd`:

```
> s.out1 <- sim(z.out1, x = x.strong, x1 = x.weak)
```

```
> summary(s.out1)
```

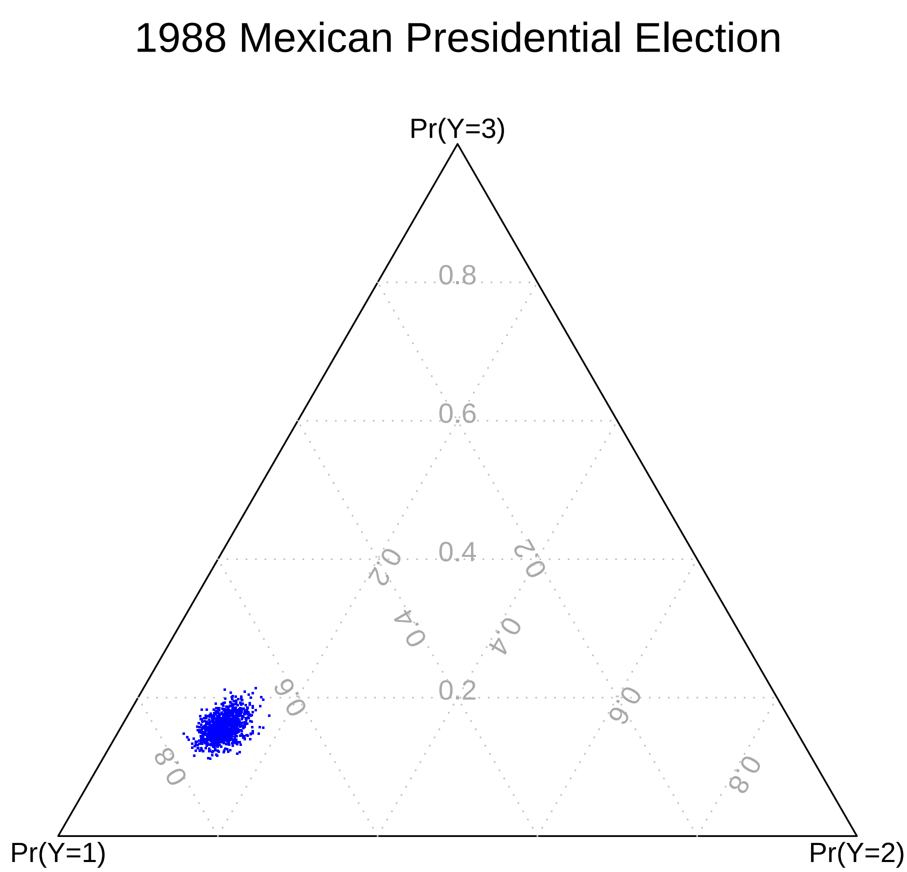
Generate simulated predicted probabilities `qi$ev` for the alternative values:

```
> ev.weak <- s.out1$qi$ev + s.out1$qi$fd
```

Plot the differences in the predicted probabilities.

```
> library(vcd)
```

```
> ternaryplot(x = s.out1$qi$ev, pch = ".", col = "blue", main = "1988 Mexican Pres
```



2. Different formula for each level

   Estimate the empirical model:

```
> z.out2 <- zelig(list(id(vote88, "1") ~ pristr + othcok, id(vote88,
+     "2") ~ othsocok), model = "mlogit", data = mexico)
```

   Set the explanatory variables to their default values, with `pristr` (for the strength of the PRI) equal to 1 (weak) in the baseline values, and equal to 3 (strong) in the alternative values:

```
> x.weak <- setx(z.out2, pristr = 1)
> x.strong <- setx(z.out2, pristr = 3)
```

   Generate simulated predicted probabilities `qi$ev` and differences in the predicted probabilities `qi$fd`:

```
> s.out1 <- sim(z.out2, x = x.strong, x1 = x.weak)
```

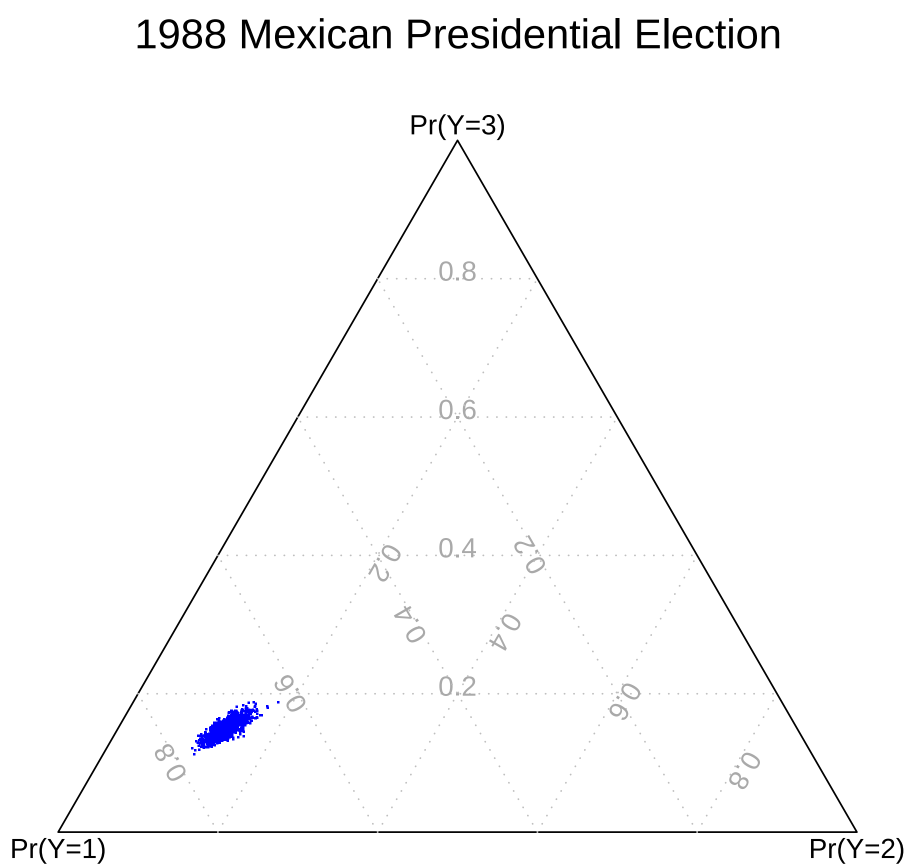
```
> summary(s.out1)
```

   Generate simulated predicted probabilities `qi$ev` for the alternative values:

```
> ev.weak <- s.out1$qi$ev + s.out1$qi$fd
```

   Using the vcd package, plot the differences in the predicted probabilities.

```
> ternaryplot(s.out1$qi$ev, pch = ".", col = "blue", main = "1988 Mexican Presiden
```



Model

Let Y_i be the unordered categorical dependent variable that takes one of the values from 1 to J, where J is the total number of categories.

- The stochastic component is given by

$$Y_i \sim \text{Multinomial}(y_i \mid \pi_{ij}),$$

  where $\pi_{ij} = \Pr(Y_i = j)$ for $j = 1, \ldots, J$.

- The systemic component is given by:

$$\pi_{ij} = \frac{\exp(x_i \beta_j)}{\sum_{k=1}^{J} \exp(x_i \beta_k)},$$

  where x_i is the vector of explanatory variables for observation i, and β_j is the vector of coefficients for category j.

Quantities of Interest

- The expected value (`qi$ev`) is the predicted probability for each category:

$$E(Y) \; = \; \pi_{ij} \; = \; \frac{\exp(x_i \beta_j)}{\sum_{k=1}^{J} \exp(x_i \beta_k)}.$$

- The predicted value (`qi$pr`) is a draw from the multinomial distribution defined by the predicted probabilities.

- The first difference in predicted probabilities (`qi$fd`), for each category is given by:

$$\text{FD}_j = \Pr(Y = j \mid x_1) - \Pr(Y = j \mid x) \quad \text{for} \quad j = 1, \ldots, J.$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "mlogit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: the named vector of coefficients.
  - `fitted.values`: an $n \times J$ matrix of the in-sample fitted values.

- `predictors`: an $n \times (J-1)$ matrix of the linear predictors $x_i \beta_j$.
- `residuals`: an $n \times (J-1)$ matrix of the residuals.
- `df.residual`: the residual degrees of freedom.
- `df.total`: the total degrees of freedom.
- `rss`: the residual sum of squares.
- `y`: an $n \times J$ matrix of the dependent variables.
- `zelig.data`: the input data frame if `save.data = TRUE`.

- From `summary(z.out)`, you may extract:
  - `coef3`: a table of the coefficients with their associated standard errors and t-statistics.
  - `cov.unscaled`: the variance-covariance matrix.
  - `pearson.resid`: an $n \times (m-1)$ matrix of the Pearson residuals.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays. Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`, indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation).
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by the expected probabilities, indexed by simulation $\times$ `x`-observation.
  - `qi$fd`: the simulated first difference in the predicted probabilities for the values specified in `x` and `x1`, indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation).
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models,
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *mlogit* Zelig model use:

> Kosuke Imai,Gary King, and Olivia Lau. 2007. "mlogit: Multinomial Logistic Regression for Dependent Variables with Unordered Categorical Values," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The multinomial logit function is part of the VGAM package by Thomas Yee (Yee and Hastie 2003). In addition, advanced users may wish to refer to `help(vglm)` in the VGAM library. Additional documentation is available at http://www.stat.auckland.ac.nz/˜ yee.Sample data are from King et al. (2000).

12.36 `mlogit.bayes`: Bayesian Multinomial Logistic Regression

Use Bayesian multinomial logistic regression to model unordered categorical variables. The dependent variable may be in the format of either character strings or integer values. The model is estimated via a random walk Metropolis algorithm or a slice sampler. See Section 12.35 for the maximum-likelihood estimation of this model.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "mlogit.bayes", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

`zelig()` accepts the following arguments for `mlogit.bayes`:

- `baseline`: either a character string or numeric value (equal to one of the observed values in the dependent variable) specifying a baseline category. The default value is `NA` which sets the baseline to the first alphabetical or numerical unique value of the dependent variable.

The model accepts the following additional arguments to monitor the Markov chains:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults to 10,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.
- `mcmc.method`: either `"MH"` or `"slice"`, specifying whether to use Metropolis Algorithm or slice sampler. The default value is `"MH"`.
- `tune`: tuning parameter for the Metropolis-Hasting step, either a scalar or a numeric vector (for k coefficients, enter a k vector). The tuning parameter should be set such that the acceptance rate is satisfactory (between 0.2 and 0.5). The default value is 1.1.
- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.
- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed of 12345.

- `beta.start`: starting values for the Markov chain, either a scalar or a vector (for k coefficients, enter a k vector). The default is `NA` where the maximum likelihood estimates are used as the starting values.

Use the following arguments to specify the priors for the model:

- `b0`: prior mean for the coefficients, either a scalar or vector. If a scalar, that value will be the prior mean for all the coefficients. The default is 0.
- `B0`: prior precision parameter for the coefficients, either a square matrix with the dimensions equal to the number of coefficients or a scalar. If a scalar, that value times an identity matrix will be the prior precision parameter. The default is 0 which leads to an improper prior.

Zelig users may wish to refer to `help(MCMCmnl)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example
   Attaching the sample dataset:

```
> data(mexico)
```

   Estimating multinomial logistics regression using `mlogit.bayes`:

```
> z.out <- zelig(vote88 ~ pristr + othcok + othsocok, model = "mlogit.bayes",
+      data = mexico)
```

   Checking for convergence before summarizing the estimates:

```
> heidel.diag(z.out$coefficients)

> raftery.diag(z.out$coefficients)

> summary(z.out)
```

   Setting values for the explanatory variables to their sample averages:

```
> x.out <- setx(z.out)
```

   Simulating quantities of interest from the posterior distribution given `x.out`.

```
> s.out1 <- sim(z.out, x = x.out)

> summary(s.out1)
```

2. Simulating First Differences
   Estimating the first difference (and risk ratio) in the probabilities of voting different candidates when `pristr` (the strength of the PRI) is set to be weak (equal to 1) versus strong (equal to 3) while all the other variables held at their default values.

```
> x.weak <- setx(z.out, pristr = 1)
> x.strong <- setx(z.out, pristr = 3)

> s.out2 <- sim(z.out, x = x.strong, x1 = x.weak)

> summary(s.out2)
```

Model

Let Y_i be the (unordered) categorical dependent variable for observation i which takes an integer values $j = 1, \ldots, J$.

- The *stochastic component* is given by:

$$Y_i \sim \text{Multinomial}(Y_i \mid \pi_{ij}).$$

  where $\pi_{ij} = \Pr(Y_i = j)$ for $j = 1, \ldots, J$.

- The *systematic component* is given by

$$\pi_{ij} = \frac{\exp(x_i \beta_j)}{\sum_{k=1}^{J} \exp(x_i \beta_k)}, \text{ for } j = 1, \ldots, J-1,$$

  where x_i is the vector of k explanatory variables for observation i and β_j is the vector of coefficient for category j. Category J is assumed to be the baseline category.

- The *prior* for β is given by

$$\beta_j \sim \text{Normal}_k \left(b_0, B_0^{-1}\right) \text{ for } j = 1, \ldots, J-1,$$

  where b_0 is the vector of means for the k explanatory variables and B_0 is the $k \times k$ precision matrix (the inverse of a variance-covariance matrix).

Quantities of Interest

- The expected values (`qi$ev`) for the multinomial logistics regression model are the predicted probability of belonging to each category:

$$\Pr(Y_i = j) = \pi_{ij} = \frac{\exp(x_i \beta_j)}{\sum_{k=1}^{J} \exp(x_J \beta_k)}, \quad \text{for } j = 1, \ldots, J-1,$$

  and

$$\Pr(Y_i = J) = 1 - \sum_{j=1}^{J-1} \Pr(Y_i = j)$$

  given the posterior draws of β_j for all categories from the MCMC iterations.

- The predicted values (`qi$pr`) are the draws of Y_i from a multinomial distribution whose parameters are the expected values(`qi$ev`) computed based on the posterior draws of β from the MCMC iterations.

- The first difference (`qi$fd`) in category j for the multinomial logistic model is defined as

$$\text{FD}_j = \Pr(Y_i = j \mid X_1) - \Pr(Y_i = j \mid X).$$

- The risk ratio (`qi$rr`) in category j is defined as

$$\text{RR}_j = \Pr(Y_i = j \mid X_1) \ / \ \Pr(Y_i = j \mid X).$$

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group in category j is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} [Y_i(t_i = 1) - E[Y_i(t_i = 0)]],$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group in category j is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} [Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}],$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(y ~ x, model = "mlogit.bayes", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated coefficients β for each category except the baseline category.
  - `zelig.data`: the input data frame if `save.data = TRUE`.

- `seed`: the random seed used in the model.

- From the `sim()` output object `s.out`:
  - `qi$ev`: the simulated expected values(probabilities) of each of the J categories given the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the multinomial distribution defined by the expected values(`qi$ev`) given the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected values of each of the J categories for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected values of each of the J categories simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *mlogit.bayes* Zelig model:

> Ben Goodrich and Ying Lu. 2007. "mlogit.bayes: Bayesian Multinomial Logistic Regression for Dependent Variables with Unordered Categorical Values ," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Bayesian logistic regression is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005).

12.37 `mloglm`: Multinomial Log-Linear Regression for Contingency Table Models

Log-linear models are for modeling contingency tables, the cross-tabulation of discrete individual-level variables. Contingency table models take as the "unit of analysis" for the purpose of the statistical procedure, the cell of a contingency table. The "dependent variable" is then the count within each cell, and the explanatory variables indicate what categories the cells fall into. These models are highly efficient computationally since there are so few "observations," but they are asymptotically equivalent to logistic regression models run on the unpacked individual level data.

Syntax

```
> estimate <- zelig(Y ~ X1 + X2, model = "mloglm", data = mydata)
> Xval <- setx(estimate)
> results <- sim(estimate, x = Xval)
```

Examples

Model

Quantities of Interest

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `estimate <- zelig(y ~ x, model = "mloglm", data)`, then you may examine the available information in `estimate` by using `names(estimate)`, see the `coefficients` by using `estimate$coefficients`, and a default summary of information through `summary(estimate)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `estimate`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `deviance`: the residual deviance.
  - `fitted.values`: the $n \times m$ matrix of in-sample fitted values.
  - `df.residual`: the residual degrees of freedom.
  - `edf`: the effective degrees of freedom.
  - `AIC`: Akaike's An Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `Hessian`: the Hessian matrix.

- From `summary(estimate)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics. covariances.
- From the `sim()` output stored in `results`:
  - `qi$ev`: the simulated expected (or fitted values) for the specified values of `x`.
  - `qi$rd`: the difference in the expected values (or first difference) for the values specified in `x` and `x1`.

How to Cite

To cite the *mloglm* Zelig model use:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "mloglm: Multinomial Log-Linear Regression for Contingency Table Models," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The multinomial logit model is part of the nnet library by Brian D. Ripley. (**?**).Advanced users may wish to refer to the R-help for `help(multinom)` and **?**

12.38 `negbin`: Negative Binomial Regression for Event Count Dependent Variables

Use the negative binomial regression if you have a count of events for each observation of your dependent variable. The negative binomial model is frequently used to estimate over-dispersed event count models.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "negbin", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for negative binomial regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see Zeileis (2004)). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    - `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    - `"kernHAC"`: HAC standard errors using the weights given in Andrews (1991).
    - `"weave"`: HAC standard errors using the weights given in Lumley and Heagerty (1999).
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and Zeileis (2004) for more options.

Example

Load sample data:

```
> data(sanction)
```

Estimate the model:

```
> z.out <- zelig(num ~ target + coop, model = "negbin", data = sanction)
```

```
> summary(z.out)
```

Set values for the explanatory variables to their default mean values:

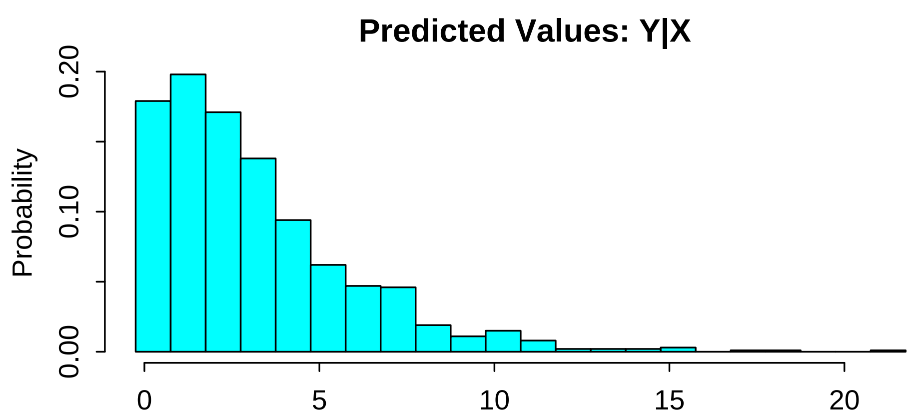
```
> x.out <- setx(z.out)
```

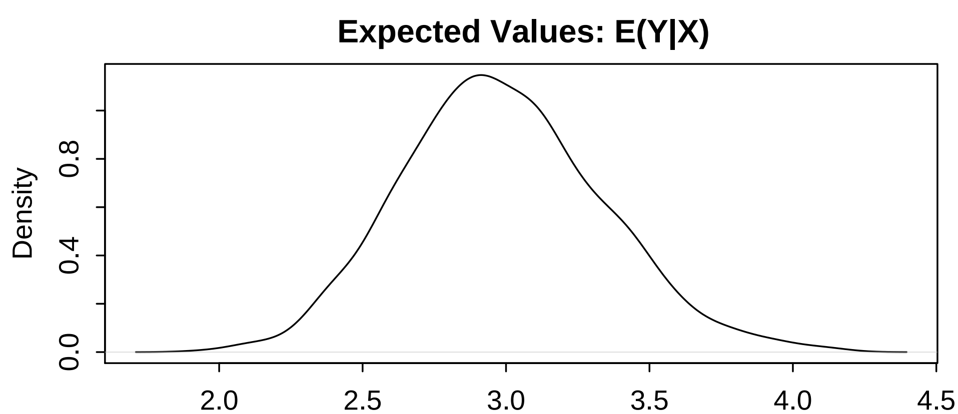
Simulate fitted values:

```
> s.out <- sim(z.out, x = x.out)
```

```
> summary(s.out)
```

```
> plot(s.out)
```





Model

Let Y_i be the number of independent events that occur during a fixed time period. This variable can take any non-negative integer value.

- The negative binomial distribution is derived by letting the mean of the Poisson distribution vary according to a fixed parameter ζ given by the Gamma distribution. The *stochastic component* is given by

$$\begin{aligned} Y_i \mid \zeta_i &\sim \text{Poisson}(\zeta_i \mu_i), \\ \zeta_i &\sim \frac{1}{\theta}\text{Gamma}(\theta). \end{aligned}$$

  The marginal distribution of Y_i is then the negative binomial with mean μ_i and variance $\mu_i + \mu_i^2/\theta$:

$$\begin{aligned} Y_i &\sim \text{NegBin}(\mu_i, \theta), \\ &= \frac{\Gamma(\theta + y_i)}{y!\,\Gamma(\theta)} \frac{\mu_i^{y_i}\,\theta^{\theta}}{(\mu_i + \theta)^{\theta + y_i}}, \end{aligned}$$

  where θ is the systematic parameter of the Gamma distribution modeling ζ_i.

- The *systematic component* is given by

$$\mu_i = \exp(x_i \beta)$$

  where x_i is the vector of k explanatory variables and β is the vector of coefficients.

Quantities of Interest

- The expected values (`qi$ev`) are simulations of the mean of the stochastic component. Thus,

$$E(Y) = \mu_i = \exp(x_i \beta),$$

  given simulations of β.

- The predicted value (`qi$pr`) drawn from the distribution defined by the set of parameters (μ_i, θ).

- The first difference (`qi$fd`) is

$$\text{FD} \;=\; E(Y|x_1) - E(Y \mid x)$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "negbin", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `theta`: the maximum likelihood estimate for the stochastic parameter θ.
  - `SE.theta`: the standard error for `theta`.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: a vector of the fitted values for the systemic component λ.
  - `linear.predictors`: a vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:

- `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
- `cov.scaled`: a $k \times k$ matrix of scaled covariances.
- `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:

  - `qi$ev`: the simulated expected values given the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by (μ_i, θ).
  - `qi$fd`: the simulated first differences in the simulated expected values given the specified values of `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *negbin* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "negbin: Negative Binomial Regression for Event Count Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The negative binomial model is part of the MASS package by William N. Venable and Brian D. Ripley (Venables and Ripley 2002). Advanced users may wish to refer to `help(glm.nb)` as well as McCullagh and Nelder (1989). Robust standard errors are implemented via sandwich package by Achim Zeileis (Zeileis 2004).Sample data are from Martin (1992).

12.39 `normal`: Normal Regression for Continuous Dependent Variables

The Normal regression model is a close variant of the more standard least squares regression model (see Section 12.32). Both models specify a continuous dependent variable as a linear function of a set of explanatory variables. The Normal model reports maximum likelihood (rather than least squares) estimates. The two models differ only in their estimate for the stochastic parameter σ.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "normal", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for normal regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see Zeileis (2004)). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    - `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    - `"kernHAC"`: HAC standard errors using the weights given in Andrews (1991).
    - `"weave"`: HAC standard errors using the weights given in Lumley and Heagerty (1999).
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and Zeileis (2004) for more options.

Examples

1. Basic Example with First Differences

   Attach sample data:

```
> data(macro)
```

Estimate model:

```
> z.out1 <- zelig(unem ~ gdp + capmob + trade, model = "normal",
+     data = macro)
```

Summarize of regression coefficients:

```
> summary(z.out1)
```

Set explanatory variables to their default (mean/mode) values, with high (80th percentile) and low (20th percentile) values for trade:

```
> x.high <- setx(z.out1, trade = quantile(macro$trade, 0.8))
> x.low <- setx(z.out1, trade = quantile(macro$trade, 0.2))
```

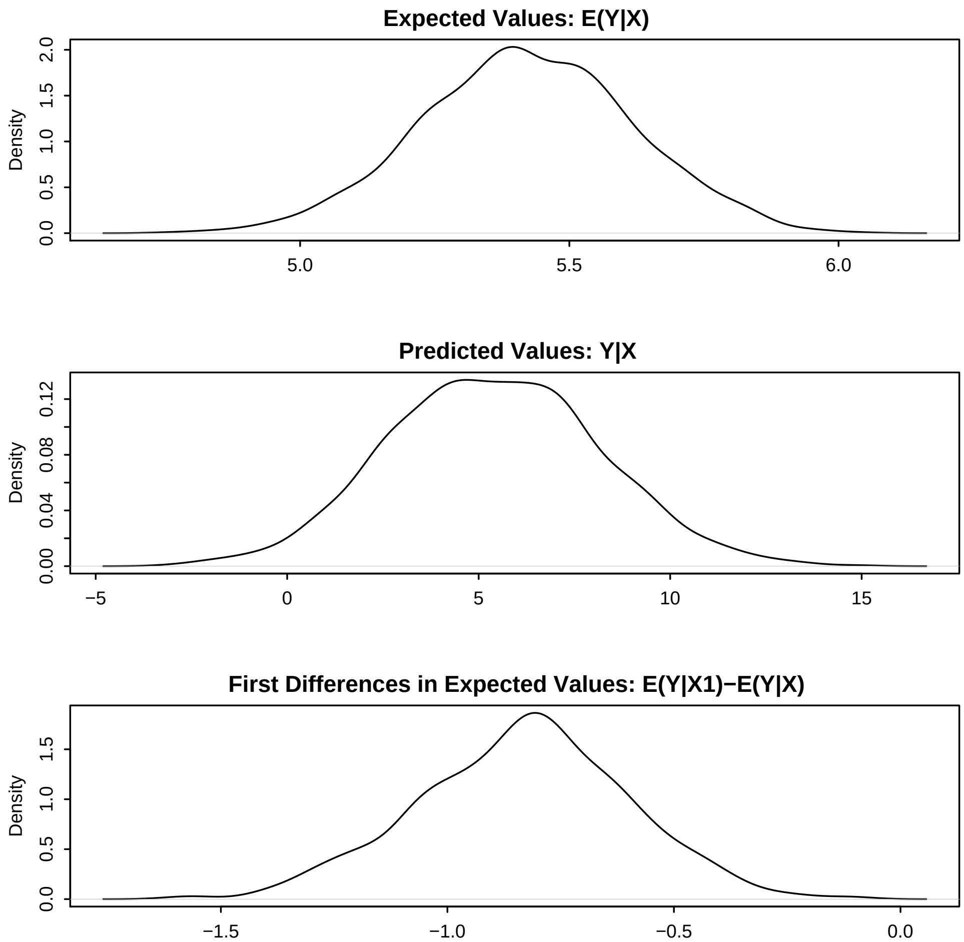
Generate first differences for the effect of high versus low trade on GDP:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
```

```
> summary(s.out1)
```

A visual summary of quantities of interest:

```
> plot(s.out1)
```



2. Using Dummy Variables

   Estimate a model with a dummy variable for each year and country (see 2 for help with dummy variables). Note that you do not need to create dummy variables, as the program will automatically parse the unique values in the selected variables into dummy variables.

```
> z.out2 <- zelig(unem ~ gdp + trade + capmob + as.factor(year) +
+      as.factor(country), model = "normal", data = macro)
```

   Set values for the explanatory variables, using the default mean/mode variables, with country set to the United States and Japan, respectively: Simulate quantities of interest:

Model

Let Y_i be the continuous dependent variable for observation i.

- The *stochastic component* is described by a univariate normal model with a vector of means μ_i and scalar variance σ^2:

$$Y_i \sim \text{Normal}(\mu_i, \sigma^2).$$

- The *systematic component* is

$$\mu_i = x_i \beta,$$

  where x_i is the vector of k explanatory variables and β is the vector of coefficients.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the the stochastic component,

$$E(Y) = \mu_i = x_i \beta,$$

  given a draw of β from its posterior.

- The predicted value (`qi$pr`) is drawn from the distribution defined by the set of parameters (μ_i, σ).

- The first difference (`qi$fd`) is:

$$\text{FD} = E(Y \mid x_1) - E(Y \mid x)$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "normal", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: fitted values. For the normal model, these are identical to the `linear predictors`.
  - `linear.predictors`: fitted values. For the normal model, these are identical to `fitted.values`.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by (μ_i, σ).

– `qi$fd`: the simulated first difference in the simulated expected values for the values specified in `x` and `x1`.

– `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

– `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *normal* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "normal: Normal Regression for Continuous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The normal model is part of the stats package by Venables and Ripley (2002). Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as McCullagh and Nelder (1989). Robust standard errors are implemented via the sandwich package by Zeileis (2004). Sample data are from King et al. (2000).

12.40 `normal.bayes`: Bayesian Normal Linear Regression

Use Bayesian regression to specify a continuous dependent variable as a linear function of specified explanatory variables. The model is implemented using a Gibbs sampler. See Section 12.39 for the maximum-likelihood implementation or Section 12.32 for the ordinary least squares variation.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "normal.bayes", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

Use the following arguments to monitor the convergence of the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults to 10,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.
- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.
- `seed`: seed for the random number generator. The default is `NA`, which corresponds to a random seed of 12345.
- `beta.start`: starting values for the Markov chain, either a scalar or vector with length equal to the number of estimated coefficients. The default is `NA`, which uses the least squares estimates as the starting values.

Use the following arguments to specify the model's priors:

- `b0`: prior mean for the coefficients, either a numeric vector or a scalar. If a scalar, that value will be the prior mean for all the coefficients. The default is 0.
- `B0`: prior precision parameter for the coefficients, either a square matrix (with the dimensions equal to the number of the coefficients) or a scalar. If a scalar, that value times an identity matrix will be the prior precision parameter. The default is 0, which leads to an improper prior.

- `c0`: `c0/2` is the shape parameter for the Inverse Gamma prior on the variance of the disturbance terms.
- `d0`: `d0/2` is the scale parameter for the Inverse Gamma prior on the variance of the disturbance terms.

Zelig users may wish to refer to `help(MCMCregress)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example
   Attaching the sample dataset:

   ```
   > data(macro)
   ```

   Estimating linear regression using `normal.bayes`:

   ```
   > z.out <- zelig(unem ~ gdp + capmob + trade, model = "normal.bayes",
   +      data = macro, verbose = TRUE)
   ```

Checking for convergence before summarizing the estimates:

```
> geweke.diag(z.out$coefficients)
```

```
> heidel.diag(z.out$coefficients)
```

```
> raftery.diag(z.out$coefficients)
```

```
> summary(z.out)
```

Setting values for the explanatory variables to their sample averages:

```
> x.out <- setx(z.out)
```

Simulating quantities of interest from the posterior distribution given `x.out`:

```
> s.out1 <- sim(z.out, x = x.out)
```

```
> summary(s.out1)
```

2. Simulating First Differences
   Set explanatory variables to their default(mean/mode) values, with high (80th percentile) and low (20th percentile) trade on GDP:

```
> x.high <- setx(z.out, trade = quantile(macro$trade, prob = 0.8))
> x.low <- setx(z.out, trade = quantile(macro$trade, prob = 0.2))
```

   Estimating the first difference for the effect of high versus low trade on unemployment rate:

```
> s.out2 <- sim(z.out, x = x.high, x1 = x.low)
```

```
> summary(s.out2)
```

Model

- The *stochastic component* is given by

$$\epsilon_i \sim \text{Normal}(0, \sigma^2)$$

  where $\epsilon_i = Y_i - \mu_i$.

- The *systematic component* is given by

$$\mu_i = x_i \beta,$$

  where x_i is the vector of k explanatory variables for observation i and β is the vector of coefficients.

- The *semi-conjugate priors* for β and σ^2 are given by

$$\begin{aligned} \beta &\sim \text{Normal}_k\left(b_0, B_0^{-1}\right) \\ \sigma^2 &\sim \text{InverseGamma}\left(\frac{c_0}{2}, \frac{d_0}{2}\right) \end{aligned}$$

  where b_0 is the vector of means for the k explanatory variables, B_0 is the $k \times k$ precision matrix (the inverse of a variance-covariance matrix), and $c_0/2$ and $d_0/2$ are the shape and scale parameters for σ^2. Note that β and σ^2 are assumed to be *a priori* independent.

Quantities of Interest

- The expected values (`qi$ev`) for the linear regression model are calculated as following:

$$E(Y) = x_i \beta,$$

  given posterior draws of β based on the MCMC iterations.

- The first difference (`qi$fd`) for the linear regression model is defined as

$$\text{FD} = E(Y \mid X_1) - E(Y \mid X).$$

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1} \{Y_i(t_i = 1) - E[Y_i(t_i = 0)]\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}\right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(y ~ x, model = "normal.bayes", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated parameters. The first k columns contain the posterior draws of the coefficients β, and the last column contains the posterior draws of the variance σ^2.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
  - `seed`: the random seed used in the model.
- From the `sim()` output object `s.out`:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected values for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *normal.bayes* Zelig model:

> Ben Goodrich and Ying Lu. 2007. "normal.bayes: Bayesian Normal Linear Regression ," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Bayesian normal regression is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005).

12.41 normal.gam: Generalized Additive Model for Continuous Dependent Variables

This function runs a nonparametric Generalized Additive Model (GAM) for continuous dependent variables.

Syntax

```
> z.out <- zelig(y ~ x1 + s(x2), model = "normal.gam", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Where `s()` indicates a variable to be estimated via nonparametric smooth. All variables for which `s()` is not specified, are estimated via standard parametric methods.

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for GAM models.

- `method`: Controls the fitting method to be used. Fitting methods are selected via a list environment within `method=gam.method()`. See `gam.method()` for details.
- `scale`: Generalized Cross Validation (GCV) is used if `scale = 0` (see the "Model" section for details) except for Normal models where a Un-Biased Risk Estimator (UBRE) (also see the "Model" section for details) is used with a scale parameter assumed to be 1. If `scale` is greater than 1, it is assumed to be the scale parameter/variance and UBRE is used. If `scale` is negative GCV is used.
- `knots`: An optional list of knot values to be used for the construction of basis functions.
- `H`: A user supplied fixed quadratic penalty on the parameters of the GAM can be supplied with this as its coefficient matrix. For example, ridge penalties can be added to the parameters of the GAM to aid in identification on the scale of the linear predictor.
- `sp`: A vector of smoothing parameters for each term.
- `...`: additional options passed to the `normal.gam` model. See the `mgcv` library for details.

Examples

1. Basic Example:

   Create some data:

```
> set.seed(0);  n <- 400; sig <- 2;
> x0 <- runif(n, 0, 1);  x1 <- runif(n, 0, 1)
> x2 <- runif(n, 0, 1);  x3 <- runif(n, 0, 1)
> f0 <- function(x) 2 * sin(pi * x)
> f1 <- function(x) exp(2 * x)
> f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 * (10 *
+     x)^3 * (1 - x)^10
> f3 <- function(x) 0 * x
> f <- f0(x0) + f1(x1) + f2(x2)
> e <- rnorm(n, 0, sig); y <- f + e
> my.data <- as.data.frame(cbind(y, x0, x1, x2, x3))
```

Estimate the model, summarize the results, and plot nonlinearities:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), model = "normal.gam",
+     data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

Note that the `plot()` function can be used after model estimation and before simulation to view the nonlinear relationships in the independent variables:

Set values for the explanatory variables to their default (mean/mode) values, then simulate, summarize and plot quantities of interest:

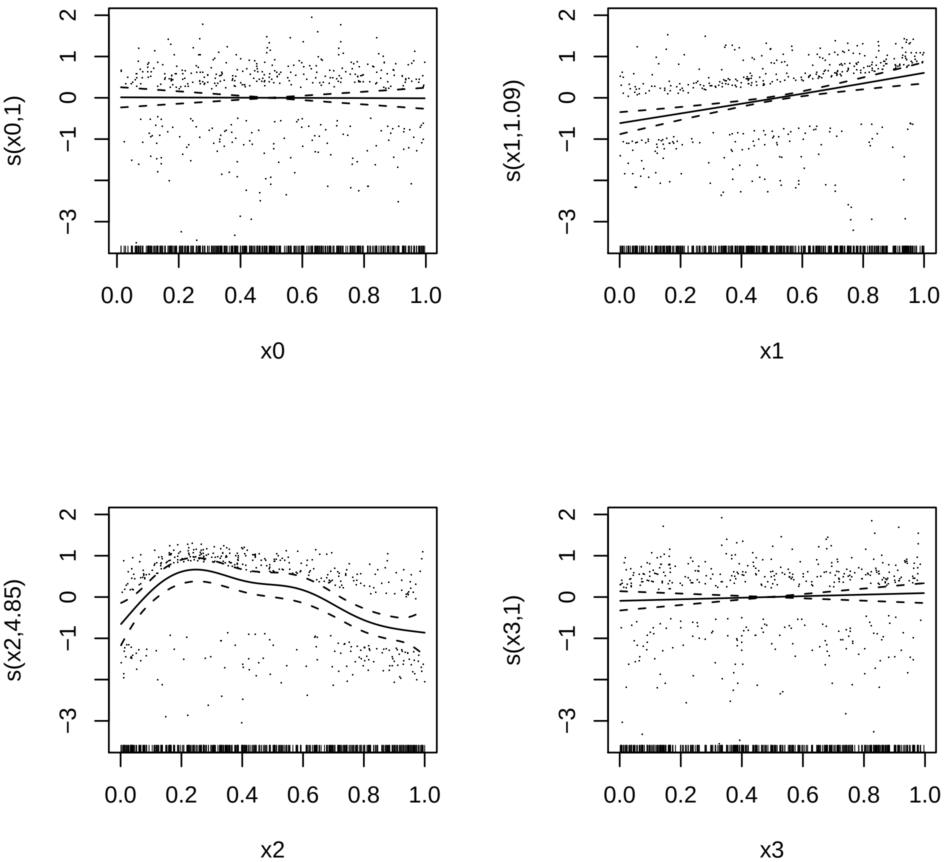
```
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
> plot(s.out)
```

2. Simulating First Differences

   Estimating the risk difference (and risk ratio) between low values (20th percentile) and high values (80th percentile) of the explanatory variable `x3` while all the other variables are held at their default (mean/mode) values.

```
> x.high <- setx(z.out, x3 = quantile(my.data$x3, 0.8))
> x.low <- setx(z.out, x3 = quantile(my.data$x3, 0.2))
> s.out <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out)
> plot(s.out)
```

3. Variations in GAM model specification. Note that `setx` and `sim` work as shown in the above examples for any GAM model. As such, in the interest of parsimony, I will not re-specify the simulations of quantities of interest.



An extra ridge penalty (useful with convergence problems):

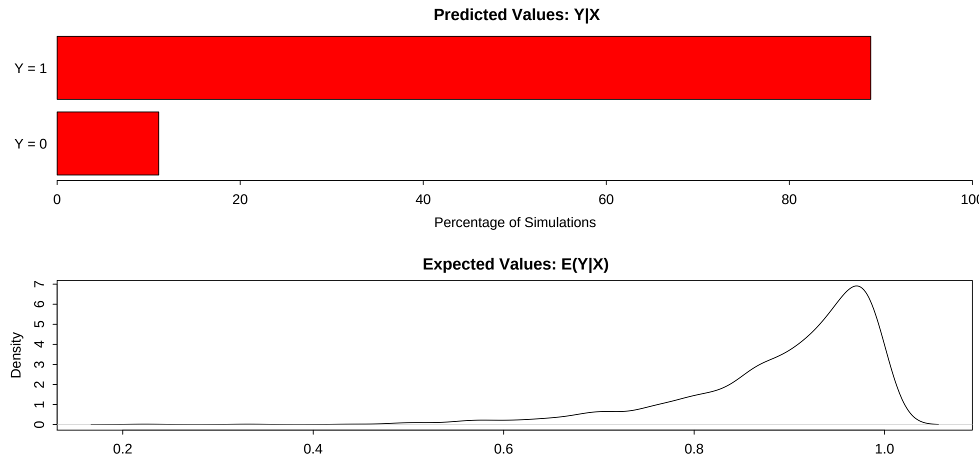
```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), H = diag(0.5,
+     37), model = "normal.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

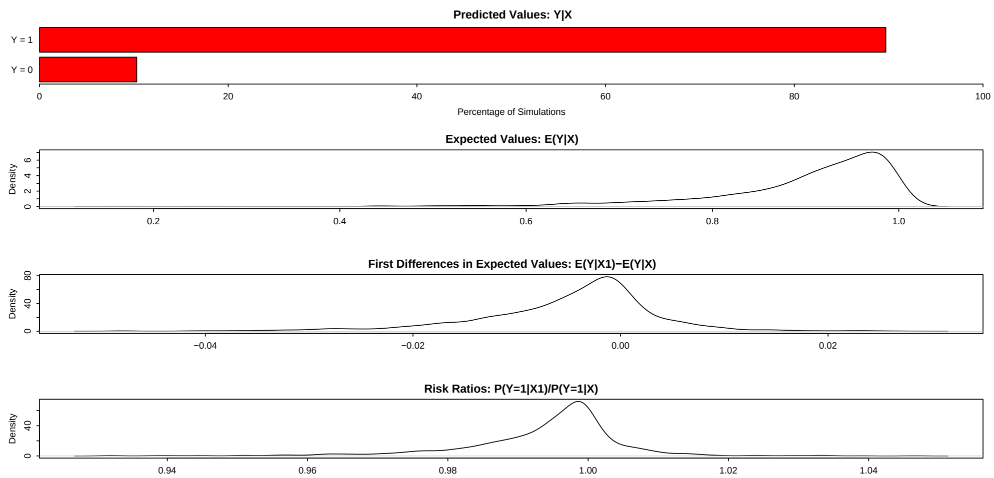
Set the smoothing parameter for the first term, estimate the rest:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), sp = c(0.01,
+     -1, -1, -1), model = "normal.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Set lower bounds on smoothing parameters:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), min.sp = c(0.001,
```






```
+     0.01, 0, 10), model = "normal.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

A GAM with 3df regression spline term & 2 penalized terms:

```
> z.out <- zelig(y ~ s(x0, k = 4, fx = TRUE, bs = "tp") + s(x1,
+     k = 12) + s(x2, k = 15), model = "normal.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Model

GAM models use families the same way GLM models do: they specify the distribution and link function to use in model fitting. In the case of `normal.gam` a normal link function is used. Specifically, let Y_i be the continuous dependent variable for observation i.

- The *stochastic component* is described by a univariate normal model with a vector of means μ_i and scalar variance σ^2:

$$Y_i \sim \text{Normal}(\mu_i, \sigma^2).$$

- The *systematic component* is given by:

$$\mu_i = x_i\beta + \sum_{j=1}^{J} f_j(Z_j).$$

  where x_i is the vector of k explanatory variables, β is the vector of coefficients and $f_j(Z_j)$ for $j = 1, \ldots J$ is the set of smooth terms.

Generalized additive models (GAMs) are similar in many respects to generalized linear models (GLMs). Specifically, GAMs are generally fit by penalized maximum likelihood estimation and GAMs have (or can have) a parametric component identical to that of a GLM. The difference is that GAMs also include in their linear predictors a specified sum of smooth functions.

In this GAM implementation, smooth functions are represented using penalized regression splines. Two techniques may be used to estimate smoothing parameters: Generalized Cross Validation (GCV),

$$n\frac{D}{(n - DF)^2}, \tag{12.4}$$

or an Un-Biased Risk Estimator (UBRE) (which is effectively just a rescaled AIC),

$$\frac{D}{n} + 2s\frac{DF}{n - s}, \tag{12.5}$$

where D is the deviance, n is the number of observations, s is the scale parameter, and DF is the effective degrees of freedom of the model. The use of GCV or UBRE can be set by the user with the `scale` command described in the "Additional Inputs" section and in either case, smoothing parameters are chosen to minimize the GCV or UBRE score for the model.

Estimation for GAM models proceeds as follows: first, basis functions and a set (one or more) of quadratic penalty coefficient matrices are constructed for each smooth term. Second, a model matrix is is obtained for the parametric component of the GAM. These matrices are combined to produce a complete model matrix and a set of penalty matrices for the smooth terms. Iteratively Reweighted Least Squares (IRLS) is then used to estimate the model; at each iteration of the IRLS, a penalized weighted least squares model is run and the smoothing parameters of that model are estimated by GCV or UBRE. This process is repeated until convergence is achieved.

Further details of the GAM fitting process are given in Wood (2000, 2004, 2006).

Quantities of Interest

The quantities of interest for the `normal.gam` model are the same as those for the standard Normal regression.

- The expected value (`qi$ev`) for the `normal.gam` model is the mean of simulations from the stochastic component,

$$E(Y) = \mu_i = x_i\beta + \sum_{j=1}^{J} f_j(Z_j).$$

- The predicted value (`qi$pr`) is a draw from the Normal distribution defined by the set of parameters (μ_i, σ^2).

- The first difference (`qi$fd`) for the normal.gam model is defined as

$$FD = \Pr(Y|w_1) - \Pr(Y|w)$$

  for $w = \{X, Z\}$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "normal.gam", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `coefficients(z.out)`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IRLS fit.
  - `linear.predictors`: the vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `method`: the fitting method used.
  - `converged`: logical indicating weather the model converged or not.
  - `smooth`: information about the smoothed parameters.
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `data`: the input data frame.

  - `model`: the model matrix used.

- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `p.coeff`: the coefficients of the parametric components of the model.
  - `se`: the standard errors of the entire model.
  - `p.table`: the coefficients, standard errors, and associated t statistics for the parametric portion of the model.
  - `s.table`: the table of estimated degrees of freedom, estimated rank, F statistics, and p-values for the nonparametric portion of the model.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite

To cite the *normal.gam* Zelig model:

> Skyler J. Cranmer. 2007. "normal.gam: Generalized Additive Model for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The `gam.logit` model is adapted from the `mgcv` package by Simon N. Wood (Wood 2006). Advanced users may wish to refer to `help(gam)`, Wood (2004), Wood (2000), and other documentation accompanying the `mgcv` package. All examples are reproduced and extended from `mgcv`'s `gam()` help pages.

12.42 `normal.gee`: Generalized Estimating Equation for Normal Regression

The GEE normal estimates the same model as the standard normal regression. Unlike in normal regression, GEE normal allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. The user must first specify a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See Zorn (2001)).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "normal.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:
  - Independence (`corstr = "independence"`): $\mathrm{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard normal regression. The "working" correlation matrix is the identity matrix.
  - Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

– Stationary m dependent (`corstr = "stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t-t'|$ periods apart for $|t-t'| \leq m$.

Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

– Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t-t'|$ periods apart.

Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

– Exchangeable (`corstr = "exchangeable"`): $\text{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

– Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\mathrm{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

Sample "working" correlation for Stationary AR-1 (`Mv`=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

– Unstructured (`corstr = "unstructured"`): $\mathrm{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.

- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with AR-1 Dependence

   Attaching the sample turnout dataset:

   ```
   > data(macro)
   ```

   Estimating model and presenting summary:

   ```
   > z.out <- zelig(unem ~ gdp + capmob + trade, model = "normal.gee",
   +      id = "country", data = macro, robust = TRUE, corstr = "AR-M",
   +      Mv = 1)
   > summary(z.out)
   ```

   Set explanatory variables to their default (mean/mode) values, with high (80th percentile) and low (20th percentile) values:

```
> x.high <- setx(z.out, trade = quantile(macro$trade, 0.8))
> x.low <- setx(z.out, trade = quantile(macro$trade, 0.2))
```

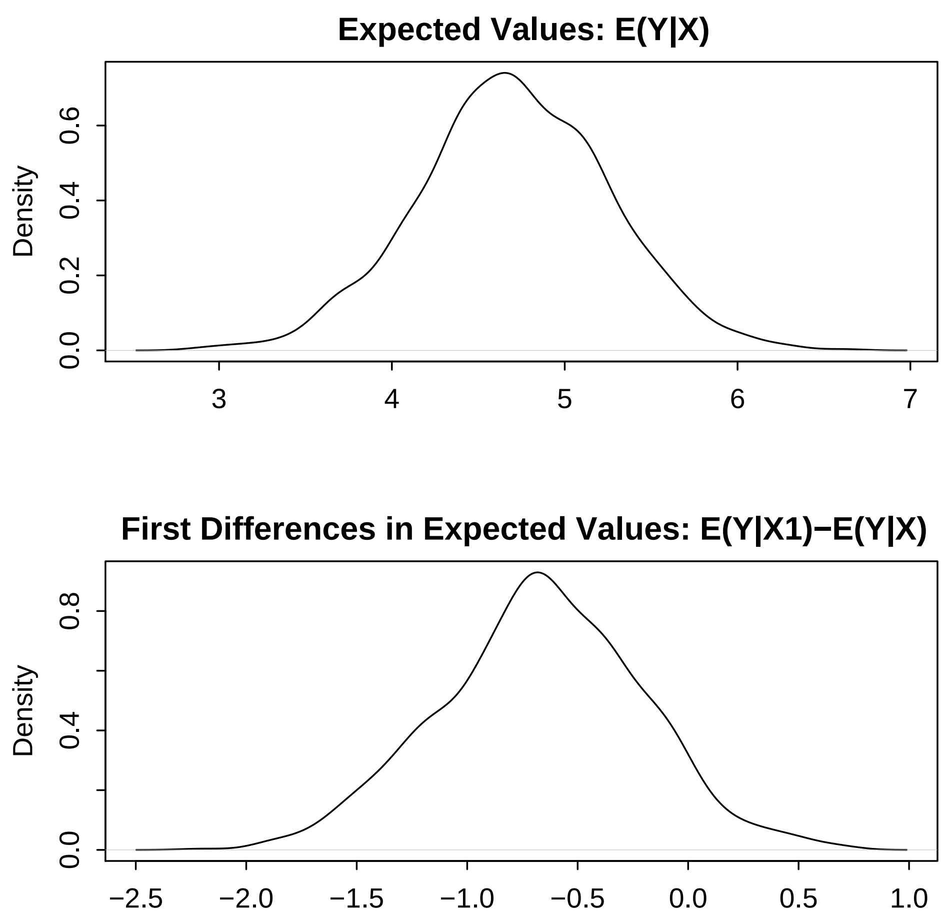
Generate first differences for the effect of high versus low trade on GDP:

```
> s.out <- sim(z.out, x = x.high, x1 = x.low)
```

```
> summary(s.out)
```

Generate a plot of quantities of interest:

```
> plot(s.out)
```



The Model

Suppose we have a panel dataset, with Y_{it} denoting the continuous dependent variable for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \mu_i) \\ Y_{it} &\sim g(y_{it} \mid \mu_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means μ_i and μ_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\mu_{it} = x_{it}\beta$$

  where x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = 1$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi\, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = 1$ as the tth diagonal element (in the case of GEE normal, A_i is the identity matrix), $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.

- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations. See **?** for more details.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.
- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.
- The expected values (`qi$ev`) for the GEE normal model is the mean of simulations from the stochastic component:

  $$E(Y) = \mu_c = x_c \beta,$$

  given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.
- The first difference (`qi$fd`) for the GEE normal model is defined as

  $$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

  $$\frac{1}{\sum_{i=1}^{n} \sum_{t=1}^{T} tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it} = 1) - E[Y_{it}(tr_{it} = 0)] \right\},$$

  where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "normal.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.

- `fitted.values`: the vector of fitted values for the systemic component, μ_{it}.
- `linear.predictors`: the vector of $x_{it}\beta$
- `max.id`: the size of the largest cluster.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation × `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How To Cite

To cite the *normal.gee* Zelig model:

> Patrick Lam. 2007. "normal.gee: Generalized Estimating Equation for Normal Regression," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from King et al. (2000).

12.43 `normal.net`: Network Normal Regression for Continuous Proximity Matrix Dependent Variables

The Network Normal regression model is a close variant of the more standard least squares regression model (see `netlm`). Both models specify a continuous proximity matrix (a.k.a. sociomatricies, adjacency matrices, or matrix representations of directed graphs) dependent variable as a linear function of a set of explanatory variables. The network Normal model reports maximum likelihood (rather than least squares) estimates. The two models differ only in their estimate for the stochastic parameter σ.

Syntax

```
> z.out <- zelig(y ~ x1 + x2, model = "normal.net", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for network normal regression:

- `LF`: specifies the link function to be used for the network normal regression. Default is `LF="identity"`, but `LF` can also be set to `"log"` or `"inverse"` by the user.

Examples

1. Basic Example

   Load the sample data (see `?friendship` for details on the structure of the network dataframe):

   ```
   > data(friendship)
   ```

   Estimate model:

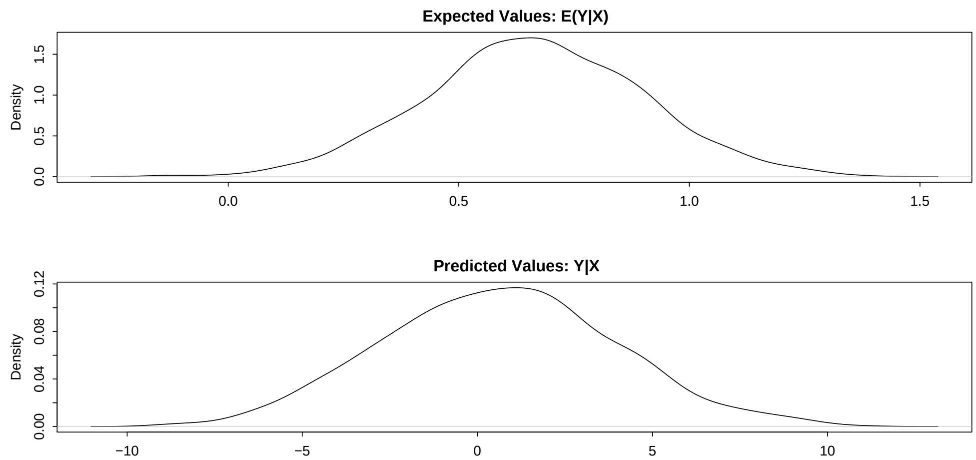
   ```
   > z.out <- zelig(perpower ~ friends + advice + prestige, model = "normal.net",
   +      data = friendship)
   > summary(z.out)
   ```

   Setting values for the explanatory variables to their default values:

   ```
   > x.out <- setx(z.out)
   ```

   Simulate fitted values.

```
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
> plot(s.out)
```



Model

The `normal.net` model performs a Normal regression of the proximity matrix $\mathbf{Y}$, a $m \times m$ matrix representing network ties, on a set of proximity matrices $\mathbf{X}$. This network regression model is directly analogous to standard Normal regression element-wise on the appropriately vectorized matrices. Proximity matrices are vectorized by creating Y, a $m^2 \times 1$ vector to represent the proximity matrix. The vectorization which produces the Y vector from the $\mathbf{Y}$ matrix is performed by simple row-concatenation of $\mathbf{Y}$. For example, if $\mathbf{Y}$ is a 15×15 matrix, the $\mathbf{Y}_{1,1}$ element is the first element of Y, and the $\mathbf{Y}_{2,1}$ element is the second element of Y and so on. Once the input matrices are vectorized, standard Normal regression is performed.

Let Y_i be the continuous dependent variable, produced by vectorizing a continuous proximity matrix, for observation i.

- The *stochastic component* is described by a univariate normal model with a vector of means μ_i and scalar variance σ^2:

$$Y_i \sim \text{Normal}(\mu_i, \sigma^2).$$

- The *systematic component* is given by:

$$\mu_i = x_i \beta.$$

  where x_i is the vector of k explanatory variables and β is the vector of coefficients.

Quantities of Interest

The quantities of interest for the network Normal regression are the same as those for the standard Normal regression.

- The expected value (`qi$ev`) for the `normal.net` model is the mean of simulations from the stochastic component,
  $$E(Y) = \mu_i = x_i\beta,$$
  given a draw of β from its posterior.
- The predicted value (`qi$pr`) is a draw from the distribution defined by the set of parameters (μ_i, σ^2).
- The first difference (`qi$fd`) for the network Normal model is defined as
  $$FD = \Pr(Y|x_1) - \Pr(Y|x)$$

Output Values

The output of each Zelig command contains useful information which you may view. For example, you run `z.out <- zelig(y ~ x, model = "normal.net", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the systemic component λ.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `linear.predictors`: fitted values. For the normal model, these are identical to fitted values.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `bic`: the Bayesian Information Criterion (minus twice the maximized log-likelihood plus the number of coefficients times $\log n$).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`
- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `mod.coefficients`: the parameter estimates with their associated standard errors, p-values, and t statistics.

- `cov.scaled`: a $k \times k$ matrix of scaled covariances.
- `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by (μ_i, σ^2).
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite

To cite the *normal.net* Zelig model:

> Skyler J. Cranmer. 2007. "normal.net: Network Normal Regression for Continuous Proximity Matrix Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The network normal regression is part of the `netglm` package by Skyler J. Cranmer and is built using some of the functionality of the `sna` package by Carter T. Butts (Butts and Carley 2001).In addition, advanced users may wish to refer to `help(normal.net)`. Sample data are fictional.

12.44 `normal.survey`: Survey-Weighted Normal Regression for Continuous Dependent Variables

The survey-weighted Normal regression model is appropriate for survey data obtained using complex sampling techniques, such as stratified random or cluster sampling (e.g., not simple random sampling). Like the least squares and Normal regression models (see Section 12.32 and Section 12.39), survey-weighted Normal regression specifies a continuous dependent variable as a linear function of a set of explanatory variables. The survey-weighted normal model reports estimates of model parameters identical to least squares or Normal regression estimates, but uses information from the survey design to correct variance estimates.

The `normal.survey` model accommodates three common types of complex survey data. Each method listed here requires selecting specific options which are detailed in the "Additional Inputs" section below.

1. **Survey weights**: Survey data are often published along with weights for each observation. For example, if a survey intentionally over-samples a particular type of case, weights can be used to correct for the over-representation of that type of case in the dataset. Survey weights come in two forms:

   (a) *Probability* weights report the probability that each case is drawn from the population. For each stratum or cluster, this is computed as the number of observations in the sample drawn from that group divided by the number of observations in the population in the group.

   (b) *Sampling* weights are the inverse of the probability weights.

2. **Strata/cluster identification**: A complex survey dataset may include variables that identify the strata or cluster from which observations are drawn. For stratified random sampling designs, observations may be nested in different strata. There are two ways to employ these identifiers:

   (a) Use *finite population corrections* to specify the total number of cases in the stratum or cluster from which each observation was drawn.

   (b) For stratified random sampling designs, use the raw strata ids to compute sampling weights from the data.

3. **Replication weights**: To preserve the anonymity of survey participants, some surveys exclude strata and cluster ids from the public data and instead release only pre-computed replicate weights.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "normal.survey", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard `zelig` inputs (see Section **??**), survey-weighted Normal models accept the following optional inputs:

1. Datasets that include survey weights:

   - `probs`: An optional formula or numerical vector specifying each case's probability weight, the probability that the case was selected. Probability weights need not (and, in most cases, will not) sum to one. Cases with lower probability weights are weighted more heavily in the computation of model coefficients.
   - `weights`: An optional numerical vector specifying each case's sample weight, the inverse of the probability that the case was selected. Sampling weights need not (and, in most cases, will not) sum to one. Cases with higher sampling weights are weighted more heavily in the computation of model coefficients.

2. Datasets that include strata/cluster identifiers:

   - `ids`: An optional formula or numerical vector identifying the cluster from which each observation was drawn (ordered from largest level to smallest level). For survey designs that do not involve cluster sampling, `ids` defaults to `NULL`.
   - `fpc`: An optional numerical vector identifying each case's frequency weight, the total number of units in the population from which each observation was sampled.
   - `strata`: An optional formula or vector identifying the stratum from which each observation was sampled. Entries may be numerical, logical, or strings. For survey designs that do not involve cluster sampling, `strata` defaults to `NULL`.
   - `nest`: An optional logical value specifying whether primary sampling unites (PSUs) have non-unique ids across multiple strata. `nest=TRUE` is appropriate when PSUs reuse the same identifiers across strata. Otherwise, `nest` defaults to `FALSE`.
   - `check.strata`: An optional input specifying whether to check that clusters are nested in strata. If `check.strata` is left at its default, `!nest`, the check is not performed. If `check.strata` is specified as `TRUE`, the check is carried out.

3. Datasets that include replication weights:

   - `repweights`: A formula or matrix specifying replication weights, numerical vectors of weights used in a process in which the sample is repeatedly re-weighted and parameters are re-estimated in order to compute the variance of the model parameters.
   - `type`: A string specifying the type of replication weights being used. This input is required if replicate weights are specified. The following types of replication weights are recognized: `"BRR"`, `"Fay"`, `"JK1"`, `"JKn"`, `"bootstrap"`, or `"other"`.

- `weights`: An optional vector or formula specifying each case's sample weight, the inverse of the probability that the case was selected. If a survey includes both sampling weights and replicate weights separately for the same survey, both should be included in the model specification. In these cases, sampling weights are used to correct potential biases in in the computation of coefficients and replication weights are used to compute the variance of coefficient estimates.
- `combined.weights`: An optional logical value that should be specified as `TRUE` if the replicate weights include the sampling weights. Otherwise, `combined.weights` defaults to `FALSE`.
- `rho`: An optional numerical value specifying a shrinkage factor for replicate weights of type `"Fay"`.
- `bootstrap.average`: An optional numerical input specifying the number of iterations over which replicate weights of type `"bootstrap"` were averaged. This input should be left as `NULL` for `"bootstrap"` weights that were not were created by averaging.
- `scale`: When replicate weights are included, the variance is computed as the sum of squared deviations of the replicates from their mean. `scale` is an optional overall multiplier for the standard deviations.
- `rscale`: Like `scale`, `rscale` specifies an optional vector of replicate-specific multipliers for the squared deviations used in variance computation.
- `fpc`: For models in which `"JK1"`, `"JKn"`, or `"other"` replicates are specified, `fpc` is an optional numerical vector (with one entry for each replicate) designating the replicates' finite population corrections.
- `fpctype`: When a finite population correction is included as an `fpc` input, `fpctype` is a required input specifying whether the input to `fpc` is a sampling fraction (`fpctype="fraction"`) or a direct correction (`fpctype="correction"`).
- `return.replicates`: An optional logical value specifying whether the replicates should be returned as a component of the output. `return.replicates` defaults to `FALSE`.

Examples

1. A dataset that includes survey weights:

   Attach the sample data:

   ```
   > data(api, package = "survey")
   ```

   Suppose that a dataset included a continuous measure of public schools' performance (`api00`), a measure of the percentage of students at each school who receive subsidized meals (`meals`), an indicator for whether each school holds classes year round (`year.rnd`), and sampling weights computed by the survey house (`pw`). Estimate a model that regresses school performance on the `meals` and `year.rnd` variables:

```
> z.out1 <- zelig(api00 ~ meals + yr.rnd, model = "normal.survey",
+     weights = ~pw, data = apistrat)
```

Summarize regression coefficients:

```
> summary(z.out1)
```

Set explanatory variables to their default (mean/mode) values, and set a high (80th percentile) and low (20th percentile) value for “meals”:

```
> x.low <- setx(z.out1, meals = quantile(apistrat$meals, 0.2))
> x.high <- setx(z.out1, meals = quantile(apistrat$meals, 0.8))
```

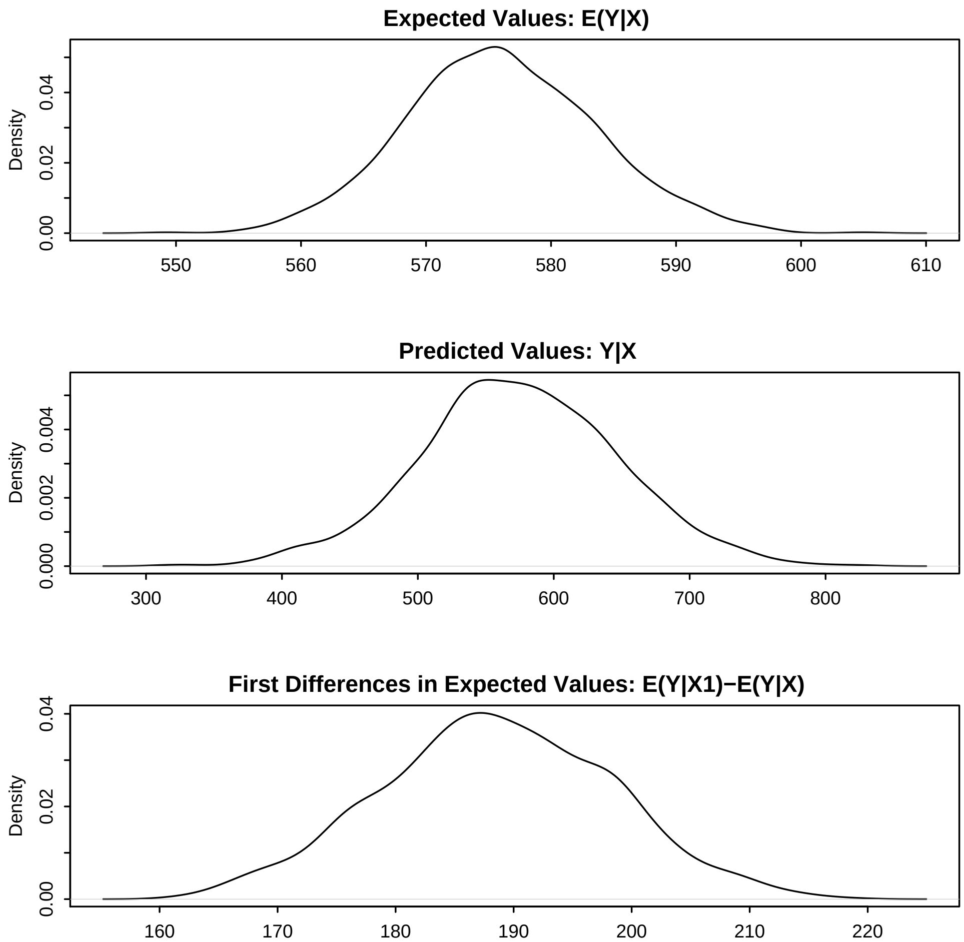
Generate first differences for the effect of high versus low concentrations of children receiving subsidized meals on academic performance:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
```

```
> summary(s.out1)
```

Generate a visual summary of the quantities of interest:

```
> plot(s.out1)
```



2. A dataset that includes strata/cluster identifiers:

   Suppose that the survey house that provided the dataset used in the previous example excluded sampling weights but made other details about the survey design available. A user can still estimate a model without sampling weights that instead uses inputs that identify the stratum and/or cluster to which each observation belongs and the size of the finite population from which each observation was drawn.

   Continuing the example above, suppose the survey house drew at random a fixed number of elementary schools, a fixed number of middle schools, and a fixed number of high schools. If the variable `stype` is a vector of characters (`"E"` for elementary schools, `"M"` for middle schools, and `"H"` for high schools) that identifies the type of school each case represents and `fpc` is a numerical vector that identifies for each case the total number of schools of the same type in the population, then the user could estimate the following model:

```
> z.out2 <- zelig(api00 ~ meals + yr.rnd, model = "normal.survey",
+     strata = ~stype, fpc = ~fpc, data = apistrat)
```

Summarize the regression output:

```
> summary(z.out2)
```

The coefficient estimates for this example are identical to the point estimates in the first example, when pre-existing sampling weights were used. When sampling weights are omitted, they are estimated automatically for "normal.survey" models based on the user-defined description of sampling designs.

Moreover, because the user provided information about the survey design, the standard error estimates are lower in this example than in the previous example, in which the user omitted variables pertaining to the details of the complex survey design.

3. A dataset that includes replication weights:

   Consider a dataset that includes information for a sample of hospitals that includes counts of the number of out-of-hospital cardiac arrests that each hospital treats and the number of patients who arrive alive at each hospital:

```
> data(scd, package = "survey")
```

   Survey houses sometimes supply replicate weights (in lieu of details about the survey design). For the sake of illustrating how replicate weights can be used as inputs in normal.survey models, create a set of balanced repeated replicate (BRR) weights:

```
> BRRrep <- 2 * cbind(c(1, 0, 1, 0, 1, 0), c(1, 0, 0, 1, 0, 1),
+     c(0, 1, 1, 0, 0, 1), c(0, 1, 0, 1, 1, 0))
```

   Estimate a model that regresses counts of patients who arrive alive in each hospital on the number of cardiac arrests that each hospital treats, using the BRR replicate weights in BRRrep to compute standard errors.

```
> z.out3 <- zelig(alive ~ arrests, model = "poisson.survey", repweights = BRRrep,
+     type = "BRR", data = scd)
```

   Summarize the regression coefficients:

```
> summary(z.out3)
```

   Set arrests at its 20th and 80th quantiles:

```
> x.low <- setx(z.out3, arrests = quantile(scd$arrests, 0.2))
> x.high <- setx(z.out3, arrests = quantile(scd$arrests, 0.8))
```

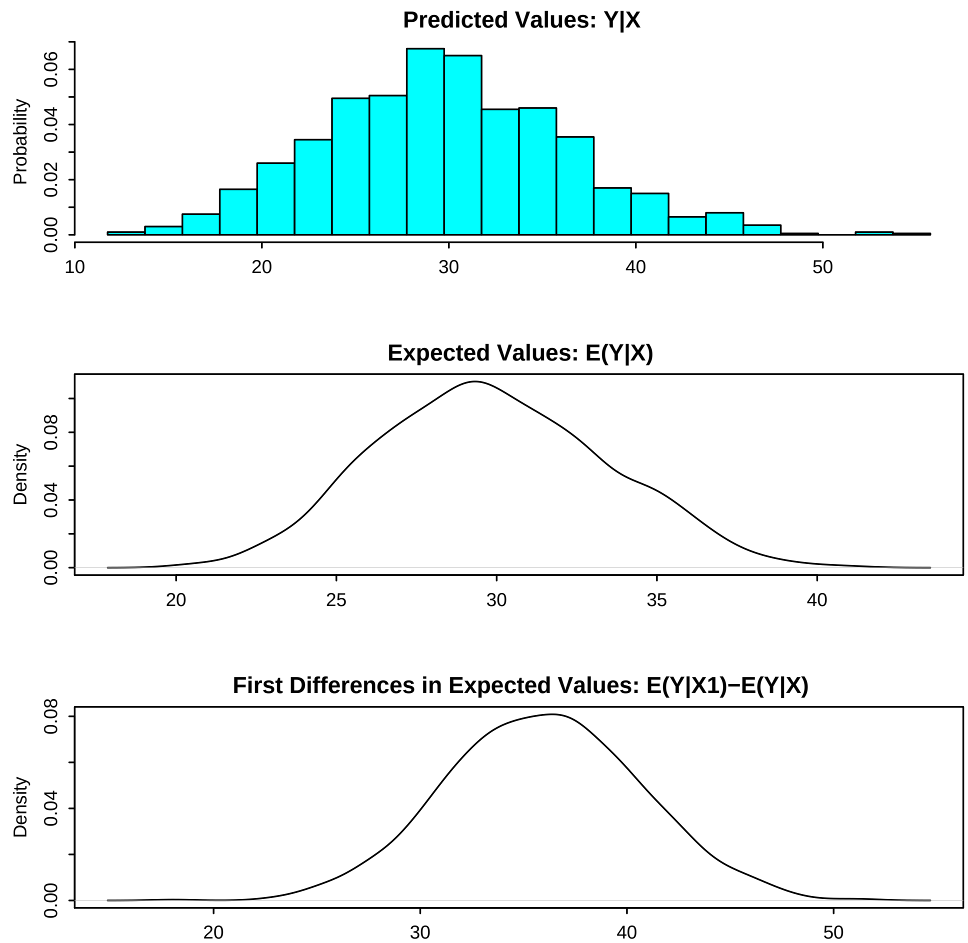
   Generate first differences for the effect of minimal versus maximal cardiac arrests on numbers of patients who arrive alive:

```
> s.out3 <- sim(z.out3, x = x.low, x1 = x.high)
```

```
> summary(s.out3)
```

Generate a visual summary of quantities of interest:

```
> plot(s.out3)
```



Model

Let Y_i be the continuous dependent variable for observation i.

- The *stochastic component* is described by a univariate normal model with a vector of means μ_i and scalar variance σ^2:

$$Y_i \sim \text{Normal}(\mu_i, \sigma^2).$$

- The *systematic component* is

$$\mu_i = x_i \beta,$$

where x_i is the vector of k explanatory variables and β is the vector of coefficients.

Variance

When replicate weights are not used, the variance of the coefficients is estimated as

$$\hat{\boldsymbol{\Sigma}} \left[\sum_{i=1}^{n} \frac{(1-\pi_i)}{\pi_i^2} (\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_i(Y_i-\mu_i)) + 2\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\pi_{ij}-\pi_i\pi_j)}{\pi_i\pi_j\pi_{ij}} (\mathbf{X}_i(Y_i-\mu_i))'(\mathbf{X}_j(Y_j-\mu_j)) \right] \hat{\boldsymbol{\Sigma}}$$

where π_i is the probability of case i being sampled, $\mathbf{X}_i$ is a vector of the values of the explanatory variables for case i, Y_i is value of the dependent variable for case i, $\hat{\mu}_i$ is the predicted value of the dependent variable for case i based on the linear model estimates, and $\hat{\boldsymbol{\Sigma}}$ is the conventional variance-covariance matrix in a parametric glm. This statistic is derived from the method for estimating the variance of sums described in Binder (1983) and the Horvitz-Thompson estimator of the variance of a sum described in Horvitz and Thompson (1952).

When replicate weights are used, the model is re-estimated for each set of replicate weights, and the variance of each parameter is estimated by summing the squared deviations of the replicates from their mean.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the the stochastic component,
$$E(Y) = \mu_i = x_i\beta,$$
given a draw of β from its posterior.

- The predicted value (`qi$pr`) is drawn from the distribution defined by the set of parameters (μ_i, σ).

- The first difference (`qi$fd`) is:
$$\text{FD} \; = \; E(Y \mid x_1) - E(Y \mid x)$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is
$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i=1) - E[Y_i(t_i=0)] \right\},$$
where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "normal.survey", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: fitted values. For the survey-weighted normal model, these are identical to the `linear predictors`.
  - `linear.predictors`: fitted values. For the survey-weighted normal model, these are identical to `fitted.values`.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:

  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by (μ_i, σ).
  - `qi$fd`: the simulated first difference in the simulated expected values for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

When users estimate `normal.survey` models with replicate weights in `Zelig`, an object called `.survey.prob.weights` is created in the global environment. `Zelig` will over-write any existing object with that name, and users are therefore advised to re-name any object called `.survey.prob.weights` before using `normal.survey` models in `Zelig`.

How to Cite

To cite the *normal.survey* Zelig model:

> Nicholas Carnes. 2008. "normal.survey: Survey-weighted Normal Regression for Continuous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Survey-weighted linear models and the sample data used in the examples above are a part of the `survey` package by Thomas Lumley. Users may wish to refer to the help files for the three functions that Zelig draws upon when estimating survey-weighted models, namely, `help(svyglm)`, `help(svydesign)`, and `help(svrepdesign)`. The normal model is part of the stats package by Venables and Ripley (2002). Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as McCullagh and Nelder (1989).

12.45 `ologit`: Ordinal Logistic Regression for Ordered Categorical Dependent Variables

Use the ordinal logit regression model if your dependent variable is ordered and categorical, either in the form of integer values or character strings.

Syntax

```
> z.out <- zelig(as.factor(Y) ~ X1 + X2, model = "ologit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

If `Y` takes discrete integer values, the `as.factor()` command will order automatically order the values. If `Y` takes on values composed of character strings, such as "strongly agree", "agree", and "disagree", `as.factor()` will order the values in the order in which they appear in `Y`. You will need to replace your dependent variable with a factored variable prior to estimating the model through `zelig()`. See Section 2 for more information on creating ordered factors and Example 1 below.

Example

1. Creating An Ordered Dependent Variable

   Load the sample data:

   ```
   > data(sanction)
   ```

   Create an ordered dependent variable:

   ```
   > sanction$ncost <- factor(sanction$ncost, ordered = TRUE, levels = c("net gain",
   +     "little effect", "modest loss", "major loss"))
   ```

   Estimate the model:

   ```
   > z.out <- zelig(ncost ~ mil + coop, model = "ologit", data = sanction)
   ```

   Set the explanatory variables to their observed values:

   ```
   > x.out <- setx(z.out, fn = NULL)
   ```

   Simulate fitted values given `x.out` and view the results:

   ```
   > s.out <- sim(z.out, x = x.out)
   ```

   ```
   > summary(s.out)
   ```

2. First Differences

   Using the sample data `sanction`, estimate the empirical model and returning the coefficients:

```
> z.out <- zelig(as.factor(cost) ~ mil + coop, model = "ologit",
+      data = sanction)
```

```
> summary(z.out)
```

   Set the explanatory variables to their means, with `mil` set to 0 (no military action in addition to sanctions) in the baseline case and set to 1 (military action in addition to sanctions) in the alternative case:

```
> x.low <- setx(z.out, mil = 0)
> x.high <- setx(z.out, mil = 1)
```

   Generate simulated fitted values and first differences, and view the results:

```
> s.out <- sim(z.out, x = x.low, x1 = x.high)
> summary(s.out)
```

Model

Let Y_i be the ordered categorical dependent variable for observation i that takes one of the integer values from 1 to J where J is the total number of categories.

- The *stochastic component* begins with an unobserved continuous variable, Y_i^*, which follows the standard logistic distribution with a parameter μ_i,

$$Y_i^* \sim \text{Logit}(y_i^* \mid \mu_i),$$

  to which we add an observation mechanism

$$Y_i = j \quad \text{if} \quad \tau_{j-1} \leq Y_i^* \leq \tau_j \quad \text{for} \quad j = 1, \dots, J.$$

  where τ_l (for $l = 0, \dots, J$) are the threshold parameters with $\tau_l < \tau_m$ for all $l < m$ and $\tau_0 = -\infty$ and $\tau_J = \infty$.

- The *systematic component* has the following form, given the parameters τ_j and β, and the explanatory variables x_i:

$$\Pr(Y \leq j) = \Pr(Y^* \leq \tau_j) = \frac{\exp(\tau_j - x_i\beta)}{1 + \exp(\tau_j - x_i\beta)},$$

  which implies:

$$\pi_j = \frac{\exp(\tau_j - x_i\beta)}{1 + \exp(\tau_j - x_i\beta)} - \frac{\exp(\tau_{j-1} - x_i\beta)}{1 + \exp(\tau_{j-1} - x_i\beta)}.$$

Quantities of Interest

- The expected values (qi$ev) for the ordinal logit model are simulations of the predicted probabilities for each category:

$$E(Y = j) \;=\; \pi_j \;=\; \frac{\exp(\tau_j - x_i\beta)}{1 + \exp(\tau_j - x_i\beta)} - \frac{\exp(\tau_{j-1} - x_i\beta)}{1 + \exp(\tau_{j-1} - x_i\beta)},$$

  given a draw of β from its sampling distribution.

- The predicted value (qi$pr) is drawn from the logit distribution described by μ_i, and observed as one of J discrete outcomes.

- The difference in each of the predicted probabilities (qi$fd) is given by

$$\Pr(Y = j \mid x_1) \; - \; \Pr(Y = j \mid x) \quad \text{for} \quad j = 1, \ldots, J.$$

- In conditional prediction models, the average expected treatment effect (att.ev) for the treatment group is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

- In conditional prediction models, the average predicted treatment effect (att.pr) for the treatment group is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "ologit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.

- `zeta`: a vector containing the estimated class boundaries τ_j.
- `deviance`: the residual deviance.
- `fitted.values`: the $n \times J$ matrix of in-sample fitted values.
- `df.residual`: the residual degrees of freedom.
- `edf`: the effective degrees of freedom.
- `Hessian`: the Hessian matrix.
- `zelig.data`: the input data frame if `save.data = TRUE`.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, and t-statistics.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays. Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`, indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation).
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by the expected probabilities, indexed by simulation $\times$ `x`-observation.
  - `qi$fd`: the simulated first difference in the predicted probabilities for the values specified in `x` and `x1`, indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation).
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *ologit* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "ologit: Ordinal Logistic Regression for Ordered Categorical Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The ordinal logit model is part of the MASS package by William N. Venable and Brian D. Ripley (Venables and Ripley 2002). Advanced users may wish to refer to `help(polr)` as well as McCullagh and Nelder (1989). Sample data are from Martin (1992).

12.46 `oprobit`: Ordinal Probit Regression for Ordered Categorical Dependent Variables

Use the ordinal probit regression model if your dependent variables are ordered and categorical. They may take on either integer values or character strings. For a Bayesian implementation of this model, see Section 12.47.

Syntax

```
> z.out <- zelig(as.factor(Y) ~ X1 + X2, model = "oprobit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

If `Y` takes discrete integer values, the `as.factor()` command will order it automatically. If `Y` takes on values composed of character strings, such as "strongly agree", "agree", and "disagree", `as.factor()` will order the values in the order in which they appear in `Y`. You will need to replace your dependent variable with a factored variable prior to estimating the model through `zelig()`. See Section 2 for more information on creating ordered factors and Example 1 below.

Example

1. Creating An Ordered Dependent Variable

   Load the sample data:

   ```
   > data(sanction)
   ```

   Create an ordered dependent variable:

   ```
   > sanction$ncost <- factor(sanction$ncost, ordered = TRUE, levels = c("net gain",
   +     "little effect", "modest loss", "major loss"))
   ```

   Estimate the model:

   ```
   > z.out <- zelig(ncost ~ mil + coop, model = "oprobit", data = sanction)
   > summary(z.out)
   ```

   Set the explanatory variables to their observed values:

   ```
   > x.out <- setx(z.out, fn = NULL)
   ```

   Simulate fitted values given `x.out` and view the results:

   ```
   > s.out <- sim(z.out, x = x.out)
   > summary(s.out)
   ```

2. First Differences

   Using the sample data `sanction`, let us estimate the empirical model and return the coefficients:

```
> z.out <- zelig(as.factor(cost) ~ mil + coop, model = "oprobit",
+      data = sanction)
```

```
> summary(z.out)
```

   Set the explanatory variables to their means, with `mil` set to 0 (no military action in addition to sanctions) in the baseline case and set to 1 (military action in addition to sanctions) in the alternative case:

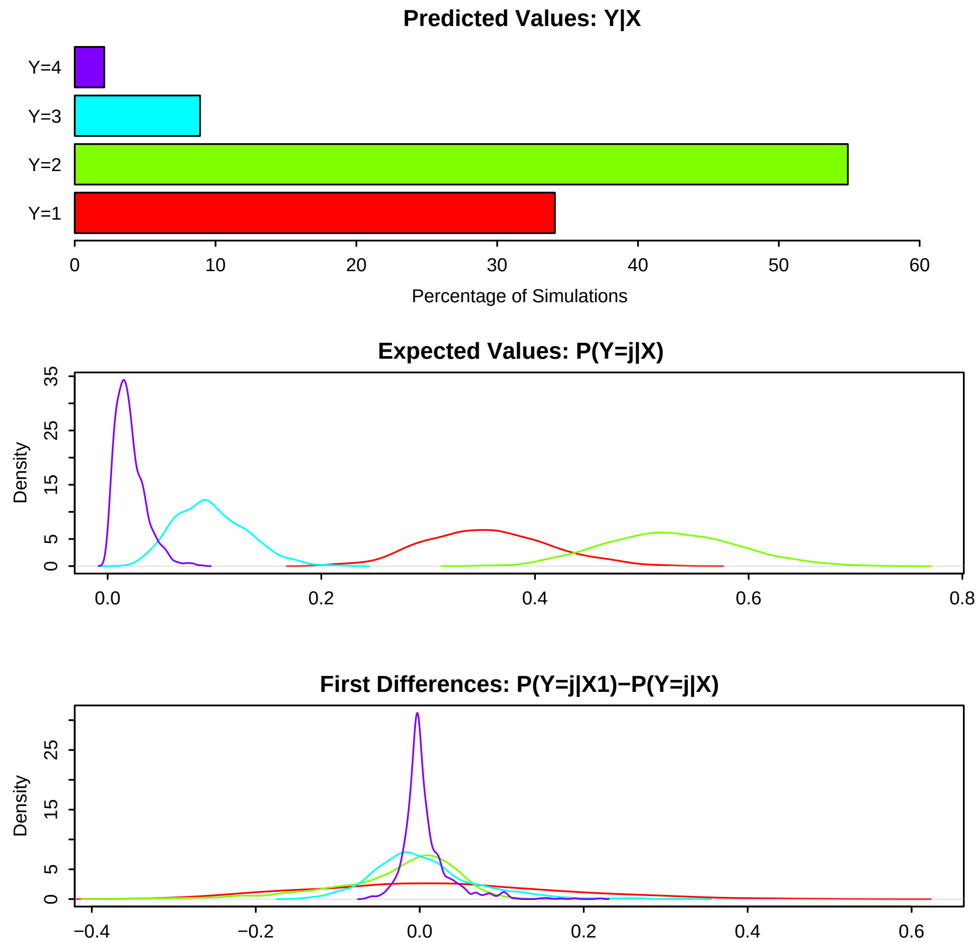
```
> x.low <- setx(z.out, mil = 0)
> x.high <- setx(z.out, mil = 1)
```

   Generate simulated fitted values and first differences, and view the results:

```
> s.out <- sim(z.out, x = x.low, x1 = x.high)
```

```
> summary(s.out)
```

```
> plot(s.out)
```



Model

Let Y_i be the ordered categorical dependent variable for observation i that takes one of the integer values from 1 to J where J is the total number of categories.

- The *stochastic component* is described by an unobserved continuous variable, Y_i^*, which follows the normal distribution with mean μ_i and unit variance

$$Y_i^* \sim N(\mu_i, 1).$$

The observation mechanism is

$$Y_i = j \quad \text{if} \quad \tau_{j-1} \leq Y_i^* \leq \tau_j \quad \text{for} \quad j = 1, \dots, J.$$

where τ_k for $k = 0, \dots, J$ is the threshold parameter with the following constraints; $\tau_l < \tau_m$ for all $l < m$ and $\tau_0 = -\infty$ and $\tau_J = \infty$.

Given this observation mechanism, the probability for each category, is given by

$$\Pr(Y_i = j) \;=\; \Phi(\tau_j \mid \mu_i) - \Phi(\tau_{j-1} \mid \mu_i) \quad \text{for} \quad j = 1, \ldots, J$$

where $\Phi(\mu_i)$ is the cumulative distribution function for the Normal distribution with mean μ_i and unit variance.

- The *systematic component* is given by

$$\mu_i \;=\; x_i \beta$$

where x_i is the vector of explanatory variables and β is the vector of coefficients.

Quantities of Interest

- The expected values (`qi$ev`) for the ordinal probit model are simulations of the predicted probabilities for each category:

$$E(Y_i = j) \;=\; \Pr(Y_i = j) \;=\; \Phi(\tau_j \mid \mu_i) - \Phi(\tau_{j-1} \mid \mu_i) \quad \text{for} \quad j = 1, \ldots, J,$$

given draws of β from its posterior.

- The predicted value (`qi$pr`) is the observed value of Y_i given the underlying standard normal distribution described by μ_i.

- The difference in each of the predicted probabilities (`qi$fd`) is given by

$$\Pr(Y = j \mid x_1) \;-\; \Pr(Y = j \mid x) \quad \text{for} \quad j = 1, \ldots, J.$$

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group in category j is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} [Y_i(t_i = 1) - E[Y_i(t_i = 0)]],$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group in category j is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} [Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}],$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of treated observations in category j.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "oprobit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: the named vector of coefficients.
  - `fitted.values`: an $n \times J$ matrix of the in-sample fitted values.
  - `predictors`: an $n \times (J-1)$ matrix of the linear predictors $x_i \beta_j$.
  - `residuals`: an $n \times (J-1)$ matrix of the residuals.
  - `df.residual`: the residual degrees of freedom.
  - `df.total`: the total degrees of freedom.
  - `rss`: the residual sum of squares.
  - `y`: an $n \times J$ matrix of the dependent variables.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coef3`: a table of the coefficients with their associated standard errors and t-statistics.
  - `cov.unscaled`: the variance-covariance matrix.
  - `pearson.resid`: an $n \times (m-1)$ matrix of the Pearson residuals.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as arrays. Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`, indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation).
  - `qi$pr`: the simulated predicted values drawn from the distribution defined by the expected probabilities, indexed by simulation $\times$ `x`-observation.
  - `qi$fd`: the simulated first difference in the predicted probabilities for the values specified in `x` and `x1`, indexed by simulation $\times$ quantity $\times$ `x`-observation (for more than one `x`-observation).
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *oprobit* Zelig model use:

> Kosuke Imai,Gary King, and Olivia Lau. 2007. "oprobit: Ordinal Probit Regression for Ordered Categorical Dependent Variables," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The ordinal probit function is part of the VGAM package by Thomas Yee (Yee and Hastie 2003). In addition, advanced users may wish to refer to `help(vglm)` in the VGAM library. Additional documentation is available at http://www.stat.auckland.ac.nz/˜ yee.Sample data are from Martin (1992)

12.47 `oprobit.bayes`: Bayesian Ordered Probit Regression

Use the ordinal probit regression model if your dependent variables are ordered and categorical. They may take either integer values or character strings. The model is estimated using a Gibbs sampler with data augmentation. For a maximum-likelihood implementation of this models, see Section 12.46.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "oprobit.bayes", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

`zelig()` accepts the following arguments to monitor the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults 10,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.
- `tune`: tuning parameter for the Metropolis-Hasting step. The default value is `NA` which corresponds to 0.05 divided by the number of categories in the response variable.
- `verbose`: defaults to `FALSE` If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.
- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed 12345.
- `beta.start`: starting values for the Markov chain, either a scalar or vector with length equal to the number of estimated coefficients. The default is `NA`, which uses the maximum likelihood estimates as the starting values.

Use the following parameters to specify the model's priors:

- `b0`: prior mean for the coefficients, either a numeric vector or a scalar. If a scalar value, that value will be the prior mean for all the coefficients. The default is 0.

- `B0`: prior precision parameter for the coefficients, either a square matrix (with dimensions equal to the number of coefficients) or a scalar. If a scalar value, that value times an identity matrix will be the prior precision parameter. The default is 0 which leads to an improper prior.

Zelig users may wish to refer to `help(MCMCoprobit)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example

   Attaching the sample dataset:

   ```
   > data(sanction)
   ```

   Estimating ordered probit regression using `oprobit.bayes`:

   ```
   > z.out <- zelig(ncost ~ mil + coop, model = "oprobit.bayes", data = sanction,
   +      verbose = TRUE)
   ```

   Creating an ordered dependent variable:

```
> sanction$ncost <- factor(sanction$ncost, ordered = TRUE, levels = c("net gain",
+     "little effect", "modest loss", "major loss"))
```

Checking for convergence before summarizing the estimates:

```
> heidel.diag(z.out$coefficients)
```

```
> raftery.diag(z.out$coefficients)
```

```
> summary(z.out)
```

Setting values for the explanatory variables to their sample averages:

```
> x.out <- setx(z.out)
```

Simulating quantities of interest from the posterior distribution given: `x.out`.

```
> s.out1 <- sim(z.out, x = x.out)
> summary(s.out1)
```

2. Simulating First Differences
   Estimating the first difference (and risk ratio) in the probabilities of incurring different level of cost when there is no military action versus military action while all the other variables held at their default values.

```
> x.high <- setx(z.out, mil = 0)
> x.low <- setx(z.out, mil = 1)
```

```
> s.out2 <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out2)
```

Model

Let Y_i be the ordered categorical dependent variable for observation i which takes an integer value $j = 1, \ldots, J$.

- The *stochastic component* is described by an unobserved continuous variable, Y_i^*,

$$Y_i^* \sim \text{Normal}(\mu_i, 1).$$

  Instead of Y_i^*, we observe categorical variable Y_i,

$$Y_i = j \quad \text{if } \tau_{j-1} \leq Y_i^* \leq \tau_j \text{ for } j = 1, \ldots, J.$$

  where τ_j for $j = 0, \ldots, J$ are the threshold parameters with the following constraints, $\tau_l < \tau_m$ for $l < m$, and $\tau_0 = -\infty, \tau_J = \infty$.

The probability of observing Y_i equal to category j is,

$$\Pr(Y_i = j) \;\; = \;\; \Phi(\tau_j \mid \mu_i) - \Phi(\tau_{j-1} \mid \mu_i) \text{ for } j = 1, \ldots, J$$

where $\Phi(\cdot \mid \mu_i)$ is the cumulative distribution function of the Normal distribution with mean μ_i and variance 1.

- The *systematic component* is given by

$$\mu_i = x_i \beta,$$

where x_i is the vector of k explanatory variables for observation i and β is the vector of coefficients.

- The *prior* for β is given by

$$\beta \sim \text{Normal}_k \left(b_0, B_0^{-1} \right)$$

where b_0 is the vector of means for the k explanatory variables and B_0 is the $k \times k$ precision matrix (the inverse of a variance-covariance matrix).

Quantities of Interest

- The expected values (`qi$ev`) for the ordered probit model are the predicted probability of belonging to each category:

$$\Pr(Y_i = j) = \Phi(\tau_j \mid x_i \beta) - \Phi(\tau_{j-1} \mid x_i \beta),$$

given the posterior draws of β and threshold parameters τ from the MCMC iterations.

- The predicted values (`qi$pr`) are the observed values of Y_i given the observation scheme and the posterior draws of β and cut points τ from the MCMC iterations.

- The first difference (`qi$fd`) in category j for the ordered probit model is defined as

$$\text{FD}_j = \Pr(Y_i = j \mid X_1) - \Pr(Y_i = j \mid X).$$

- The risk ratio (`qi$rr`) in category j is defined as

$$\text{RR}_j = \Pr(Y_i = j \mid X_1) \; / \; \Pr(Y_i = j \mid X).$$

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group in category j is

$$\frac{1}{n_j} \sum_{i:t_i=1}^{n_j} \{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of observations in the treatment group that belong to category j.

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group in category j is

$$\frac{1}{n_j}\sum_{i:t_i=1}^{n_j}[Y_i(t_i=1) - \widehat{Y_i(t_i=0)}],$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups, and n_j is the number of observations in the treatment group that belong to category j.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(y ~ x, model = "oprobit.bayes", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated coefficients β and threshold parameters τ. Note, element τ_1 is normalized to 0 and is not returned in the `coefficients` object.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
  - `seed`: the random seed used in the model.
- From the `sim()` output object `s.out`:
  - `qi$ev`: the simulated expected values (probabilities) of each of the J categories for the specified values of `x`.
  - `qi$pr`: the simulated predicted values (observed values) for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected values of each of the J categories for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected values of each of the J categories simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *oprobit.bayes* Zelig model use:

> Ben Goodrich and Ying Lu. 2007. "oprobit.bayes: Bayesian Ordered Probit Regression," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Bayesian ordinal probit regression is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005).

12.48 `poisson`: Poisson Regression for Event Count Dependent Variables

Use the Poisson regression model if the observations of your dependent variable represents the number of independent events that occur during a fixed period of time (see the negative binomial model, Section 12.38, for over-dispersed event counts.) For a Bayesian implementation of this model, see Section 12.49.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "poisson", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for poisson regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see Zeileis (2004)). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  – `method`: Choose from
    * `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    * `"kernHAC"`: HAC standard errors using the weights given in Andrews (1991).
    * `"weave"`: HAC standard errors using the weights given in Lumley and Heagerty (1999).

  – `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.

  – `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and Zeileis (2004) for more options.

Example

Load sample data:

```
> data(sanction)
```

Estimate Poisson model:

```
> z.out <- zelig(num ~ target + coop, model = "poisson", data = sanction)
```

```
> summary(z.out)
```

Set values for the explanatory variables to their default mean values:

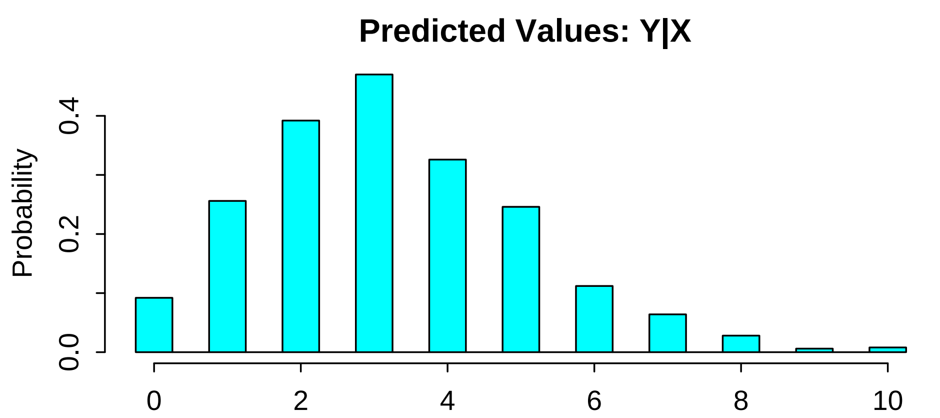
```
> x.out <- setx(z.out)
```

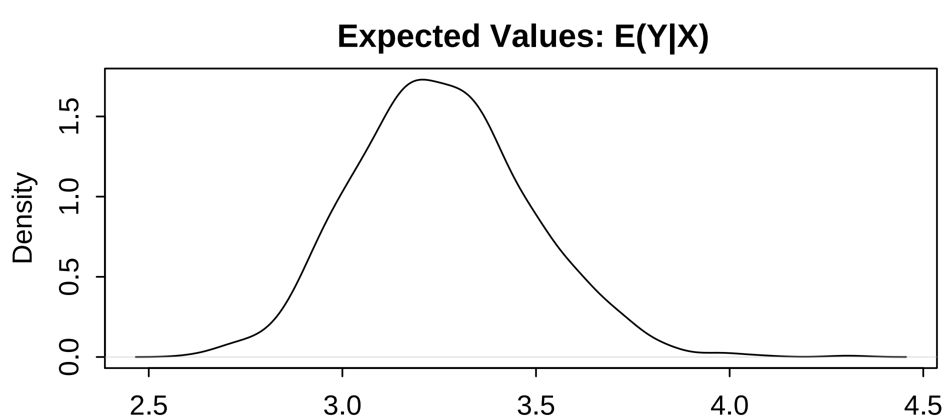
Simulate fitted values:

```
> s.out <- sim(z.out, x = x.out)
```

```
> summary(s.out)
```

```
> plot(s.out)
```





Model

Let Y_i be the number of independent events that occur during a fixed time period. This variable can take any non-negative integer.

- The Poisson distribution has *stochastic component*

$$Y_i \sim \text{Poisson}(\lambda_i),$$

  where λ_i is the mean and variance parameter.

- The *systematic component* is

$$\lambda_i = \exp(x_i \beta),$$

  where x_i is the vector of explanatory variables, and β is the vector of coefficients.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the stochastic component,

$$E(Y) = \lambda_i = \exp(x_i \beta),$$

  given draws of β from its sampling distribution.

- The predicted value (`qi$pr`) is a random draw from the poisson distribution defined by mean λ_i.

- The first difference in the expected values (`qi$fd`) is given by:

$$\text{FD} = E(Y|x_1) - E(Y \mid x)$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "poisson", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: a vector of the fitted values for the systemic component λ.
  - `linear.predictors`: a vector of $x_i \beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values given the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by λ_i.
  - `qi$fd`: the simulated first differences in the expected values given the specified values of `x` and `x1`.

– `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
– `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *poisson* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "poisson: Poisson Regression for Event Count Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The poisson model is part of the stats package by Venables and Ripley (2002). Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as McCullagh and Nelder (1989). Robust standard errors are implemented via the sandwich package by Zeileis (2004). Sample data are from Martin (1992).

12.49 `poisson.bayes`: Bayesian Poisson Regression

Use the Poisson regression model if the observations of your dependent variable represents the number of independent events that occur during a fixed period of time. The model is fit using a random walk Metropolis algorithm. For a maximum-likelihood estimation of this model see Section 12.48.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "poisson.bayes", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

Use the following argument to monitor the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults to 10,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.
- `tune`: Metropolis tuning parameter, either a positive scalar or a vector of length k, where k is the number of coefficients. The tuning parameter should be set such that the acceptance rate of the Metropolis algorithm is satisfactory (typically between 0.20 and 0.5). The default value is 1.1.
- `verbose`: default to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.
- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed of 12345.
- `beta.start`: starting values for the Markov chain, either a scalar or vector with length equal to the number of estimated coefficients. The default is `NA`, such that the maximum likelihood estimates are used as the starting values.

Use the following parameters to specify the model's priors:

- `b0`: prior mean for the coefficients, either a numeric vector or a scalar. If a scalar, that value will be the prior mean for all the coefficients. The default is 0.

- `B0`: prior precision parameter for the coefficients, either a square matrix (with the dimensions equal to the number of the coefficients) or a scalar. If a scalar, that value times an identity matrix will be the prior precision parameter. The default is 0, which leads to an improper prior.

Zelig users may wish to refer to `help(MCMCpoisson)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example
   Attaching the sample dataset:

   ```
   > data(sanction)
   ```

   Estimating the Poisson regression using `poisson.bayes`:

   ```
   > z.out <- zelig(num ~ target + coop, model = "poisson.bayes",
   +      data = sanction, verbose = TRUE)
   ```

   Checking convergence diagnostics before summarizing the estimates:

```
> geweke.diag(z.out$coefficients)
```

```
> heidel.diag(z.out$coefficients)
```

```
> raftery.diag(z.out$coefficients)
```

```
> summary(z.out)
```

Setting values for the explanatory variables to their sample averages:

```
> x.out <- setx(z.out)
```

Simulating quantities of interest from the posterior distribution given `x.out`.

```
> s.out1 <- sim(z.out, x = x.out)
```

```
> summary(s.out1)
```

2. Simulating First Differences
   Estimating the first difference in the number of countries imposing sanctions when the number of targets is set to be its maximum versus its minimum :

```
> x.max <- setx(z.out, target = max(sanction$target))
> x.min <- setx(z.out, target = min(sanction$target))
```

```
> s.out2 <- sim(z.out, x = x.max, x1 = x.min)
> summary(s.out2)
```

Model

Let Y_i be the number of independent events that occur during a fixed time period.

- The *stochastic component* is given by

$$Y_i \sim \text{Poisson}(\lambda_i)$$

  where λ_i is the mean and variance parameter.

- The *systematic component* is given by

$$\lambda_i = \exp(x_i \beta)$$

  where x_i is the vector of k explanatory variables for observation i and β is the vector of coefficients.

- The *prior* for β is given by

$$\beta \sim \text{Normal}_k \left(b_0, B_0^{-1} \right)$$

  where b_0 is the vector of means for the k explanatory variables and B_0 is the $k \times k$ precision matrix (the inverse of a variance-covariance matrix).

Quantities of Interest

- The expected values (`qi$ev`) for the Poisson model are calculated as following:

$$E(Y \mid X) = \lambda_i = \exp(x_i \beta),$$

  given the posterior draws of β based on the MCMC iterations.

- The predicted values (`qi$pr`) are draws from the Poisson distribution with parameter λ_i.

- The first difference (`qi$fd`) for the Poisson model is defined as

$$\text{FD} = E(Y \mid X_1) - E(Y \mid X).$$

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1} \{Y_i(t_i = 1) - E[Y_i(t_i = 0)]\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups.

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1} [Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}],$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(y ~ x, model = "poisson.bayes", data)
```

you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:

- `coefficients`: draws from the posterior distributions of the estimated parameters.
- `zelig.data`: the input data frame if `save.data = TRUE`.
- `seed`: the random seed used in the model.

- From the `sim()` output object `s.out`:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected values for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *poisson.bayes* Zelig model:

> Ben Goodrich and Ying Lu. 2007. "poisson.bayes: Bayesian Poisson Regression," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Bayesian poisson regression is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005).

12.50 `poisson.gam`: Generalized Additive Model for Count Dependent Variables

This function runs a nonparametric Generalized Additive Model (GAM) for count dependent variables.

Syntax

```
> z.out <- zelig(y ~ x1 + s(x2), model = "poisson.gam", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Where `s()` indicates a variable to be estimated via nonparametric smooth. All variables for which `s()` is not specified, are estimated via standard parametric methods.

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for GAM models.

- `method`: Controls the fitting method to be used. Fitting methods are selected via a list environment within `method=gam.method()`. See `gam.method()` for details.
- `scale`: Generalized Cross Validation (GCV) is used if `scale = 0` (see the "Model" section for details) except for Poisson models where a Un-Biased Risk Estimator (UBRE) (also see the "Model" section for details) is used with a scale parameter assumed to be 1. If `scale` is greater than 1, it is assumed to be the scale parameter/variance and UBRE is used. If `scale` is negative GCV is used.
- `knots`: An optional list of knot values to be used for the construction of basis functions.
- `H`: A user supplied fixed quadratic penalty on the parameters of the GAM can be supplied with this as its coefficient matrix. For example, ridge penalties can be added to the parameters of the GAM to aid in identification on the scale of the linear predictor.
- `sp`: A vector of smoothing parameters for each term.
- `...`: additional options passed to the `poisson.gam` model. See the `mgcv` library for details.

Examples

1. Basic Example

   Create some count data:

```
> set.seed(0);  n <- 400; sig <- 2;
> x0 <- runif(n, 0, 1);  x1 <- runif(n, 0, 1)
> x2 <- runif(n, 0, 1);  x3 <- runif(n, 0, 1)
> f0 <- function(x) 2 * sin(pi * x)
> f1 <- function(x) exp(2 * x)
> f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 * (10 *
+     x)^3 * (1 - x)^10
> f3 <- function(x) 0 * x
> f <- f0(x0) + f1(x1) + f2(x2)
> g <- exp(f/4);  y <- rpois(rep(1, n), g)
> my.data <- as.data.frame(cbind(y, x0, x1, x2, x3))
```

Estimate the model, summarize the results, and plot nonlinearities:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), model = "poisson.gam",
+     data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

Note that the `plot()` function can be used after model estimation and before simulation to view the nonlinear relationships in the independent variables:

Set values for the explanatory variables to their default (mean/mode) values, then simulate, summarize and plot quantities of interest:

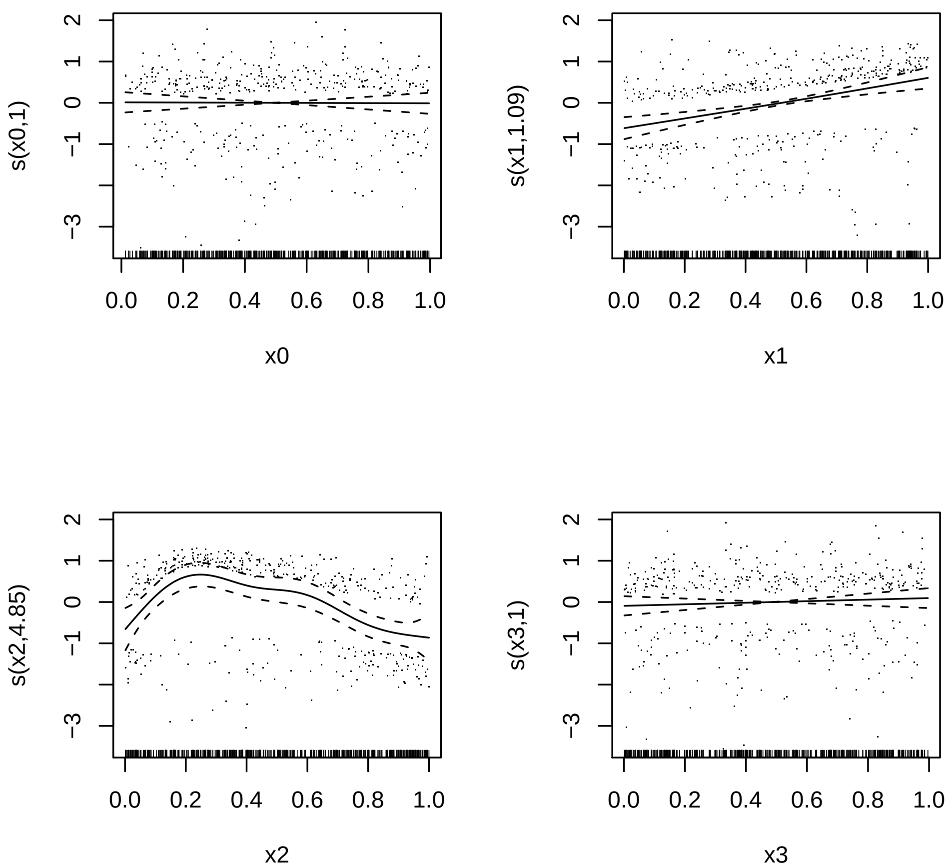
```
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
> plot(s.out)
```

2. Simulating First Differences

   Estimating the risk difference (and risk ratio) between low values (20th percentile) and high values (80th percentile) of the explanatory variable `x3` while all the other variables are held at their default (mean/mode) values.

```
> x.high <- setx(z.out, x3 = quantile(my.data$x3, 0.8))
> x.low <- setx(z.out, x3 = quantile(my.data$x3, 0.2))
> s.out <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out)
> plot(s.out)
```

3. Variations in GAM model specification. Note that `setx` and `sim` work as shown in the above examples for any GAM model. As such, in the interest of parsimony, I will not re-specify the simulations of quantities of interest.



An extra ridge penalty (useful with convergence problems):

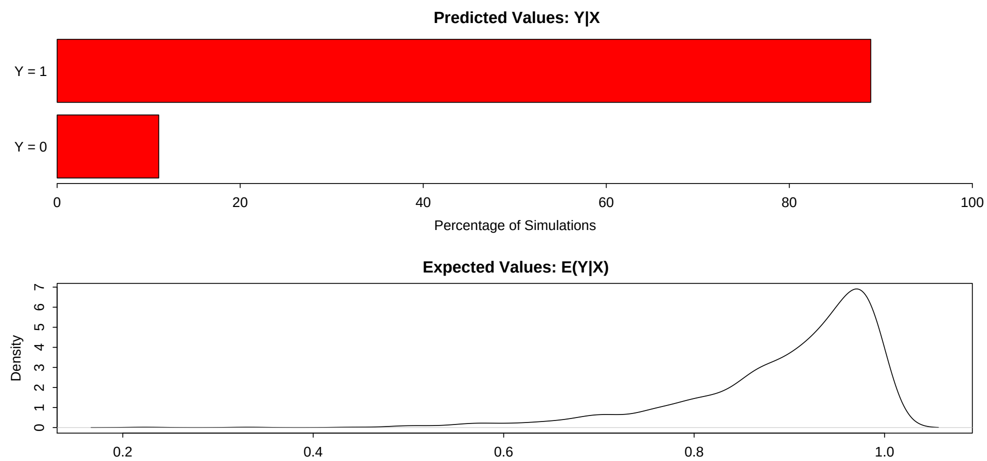
```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), H = diag(0.5,
+      37), model = "poisson.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

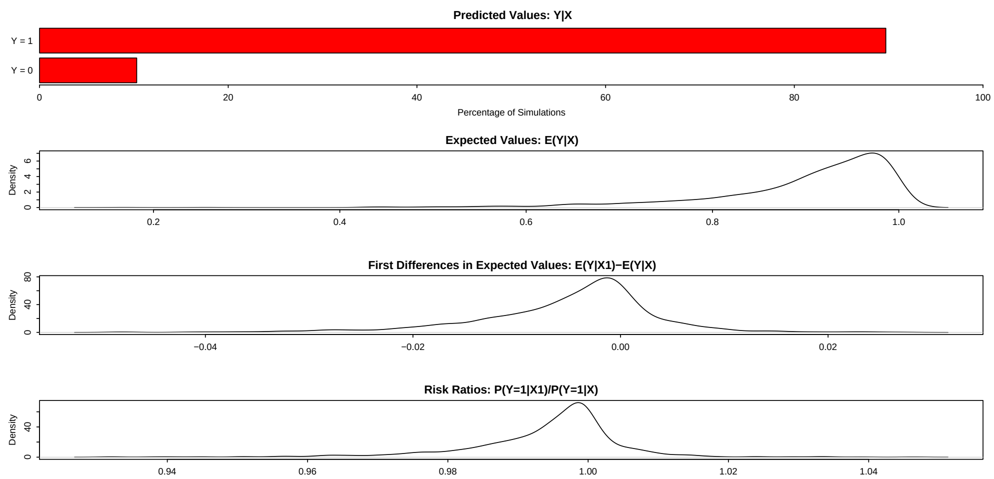
Set the smoothing parameter for the first term, estimate the rest:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), sp = c(0.01,
+      -1, -1, -1), model = "poisson.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Set lower bounds on smoothing parameters:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), min.sp = c(0.001,
```






```
+      0.01, 0, 10), model = "poisson.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

A GAM with 3df regression spline term & 2 penalized terms:

```
> z.out <- zelig(y ~ s(x0, k = 4, fx = TRUE, bs = "tp") + s(x1,
+      k = 12) + s(x2, k = 15), model = "poisson.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Model

GAM models use families the same way GLM models do: they specify the distribution and link function to use in model fitting. In the case of `poisson.gam` a Poisson link function is used. Specifically, let Y_i be the dependent variable for observation i. Y_i is thus the number of independent events that occur during a fixed time period. This variable can take any non-negative integer.

- The Poisson distribution has *stochastic component*

$$Y_i \sim \text{Poisson}(\lambda_i),$$

  where λ_i is the mean and variance parameter.

- The *systematic component* is given by:

$$\lambda_i = \exp\left(x_i\beta + \sum_{j=1}^{J} f_j(Z_j)\right).$$

  where x_i is the vector of explanatory variables, β is the vector of coefficients and $f_j(Z_j)$ for $j = 1, \ldots J$ is the set of smooth terms.

Generalized additive models (GAMs) are similar in many respects to generalized linear models (GLMs). Specifically, GAMs are generally fit by penalized maximum likelihood estimation and GAMs have (or can have) a parametric component identical to that of a GLM. The difference is that GAMs also include in their linear predictors a specified sum of smooth functions.

In this GAM implementation, smooth functions are represented using penalized regression splines. Two techniques may be used to estimate smoothing parameters: Generalized Cross Validation (GCV),

$$n\frac{D}{(n - DF)^2}, \tag{12.6}$$

or an Un-Biased Risk Estimator (UBRE) (which is effectively just a rescaled AIC),

$$\frac{D}{n} + 2s\frac{DF}{n - s}, \tag{12.7}$$

where D is the deviance, n is the number of observations, s is the scale parameter, and DF is the effective degrees of freedom of the model. The use of GCV or UBRE can be set by the user with the `scale` command described in the "Additional Inputs" section and in either case, smoothing parameters are chosen to minimize the GCV or UBRE score for the model.

Estimation for GAM models proceeds as follows: first, basis functions and a set (one or more) of quadratic penalty coefficient matrices are constructed for each smooth term. Second, a model matrix is is obtained for the parametric component of the GAM. These matrices are combined to produce a complete model matrix and a set of penalty matrices for the smooth terms. Iteratively Reweighted Least Squares (IRLS) is then used to estimate the model; at each iteration of the IRLS, a penalized weighted least squares model is run and the smoothing parameters of that model are estimated by GCV or UBRE. This process is repeated until convergence is achieved.

Further details of the GAM fitting process are given in Wood (2000, 2004, 2006).

Quantities of Interest

The quantities of interest for the `poisson.gam` model are the same as those for the standard Poisson regression.

- The expected value (`qi$ev`) for the `poisson.gam` model is the mean of simulations from the stochastic component,

$$E(Y) = \lambda_i = \exp\left(x_i\beta\sum_{j=1}^{J} f_j(Z_j)\right).$$

- The predicted value (`qi$pr`) is a random draw from the Poisson distribution defined by mean λ_i.

- The first difference (`qi$fd`) for the poisson.gam model is defined as

$$FD = \Pr(Y|w_1) - \Pr(Y|w)$$

  for $w = \{X, Z\}$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "poisson.gam", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `coefficients(z.out)`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IRLS fit.
  - `linear.predictors`: the vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `method`: the fitting method used.
  - `converged`: logical indicating weather the model converged or not.
  - `smooth`: information about the smoothed parameters.
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `data`: the input data frame.

  - `model`: the model matrix used.

- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `p.coeff`: the coefficients of the parametric components of the model.
  - `se`: the standard errors of the entire model.
  - `p.table`: the coefficients, standard errors, and associated t statistics for the parametric portion of the model.
  - `s.table`: the table of estimated degrees of freedom, estimated rank, F statistics, and p-values for the nonparametric portion of the model.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite

To cite the *poisson.gam* Zelig model:

> Skyler J. Cranmer. 2007. "poisson.gam: Generalized Additive Model for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The `gam.logit` model is adapted from the `mgcv` package by Simon N. Wood (Wood 2006). Advanced users may wish to refer to `help(gam)`, Wood (2004), Wood (2000), and other documentation accompanying the `mgcv` package. All examples are reproduced and extended from `mgcv`'s `gam()` help pages.

12.51 `poisson.gee`: Generalized Estimating Equation for Poisson Regression

The GEE poisson estimates the same model as the standard poisson regression (appropriate when your dependent variable represents the number of independent events that occur during a fixed period of time). Unlike in poisson regression, GEE poisson allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. The user must first specify a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See Zorn (2001)).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "poisson.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:
  - Independence (`corstr = "independence"`): $\mathrm{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard poisson regression. The "working" correlation matrix is the identity matrix.
  - Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

– Stationary m dependent (`corstr = "stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t-t'|$ periods apart for $|t-t'| \leq m$.

Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

– Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t-t'|$ periods apart.

Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

– Exchangeable (`corstr = "exchangeable"`): $\text{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

– Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\text{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

Sample "working" correlation for Stationary AR-1 (`Mv`=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

– Unstructured (`corstr = "unstructured"`): $\text{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.

- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with Exchangeable Dependence

   Attaching the sample turnout dataset:

   ```
   > data(sanction)
   ```

   Variable identifying clusters

   ```
   > sanction$cluster <- c(rep(c(1:15), 5), rep(c(16), 3))
   ```

   Sorting by cluster

   ```
   > sorted.sanction <- sanction[order(sanction$cluster), ]
   ```

   Estimating model and presenting summary:

```
> z.out <- zelig(num ~ target + coop, model = "poisson.gee", id = "cluster",
+      data = sorted.sanction, robust = TRUE, corstr = "exchangeable")
> summary(z.out)
```

Set explanatory variables to their default values:

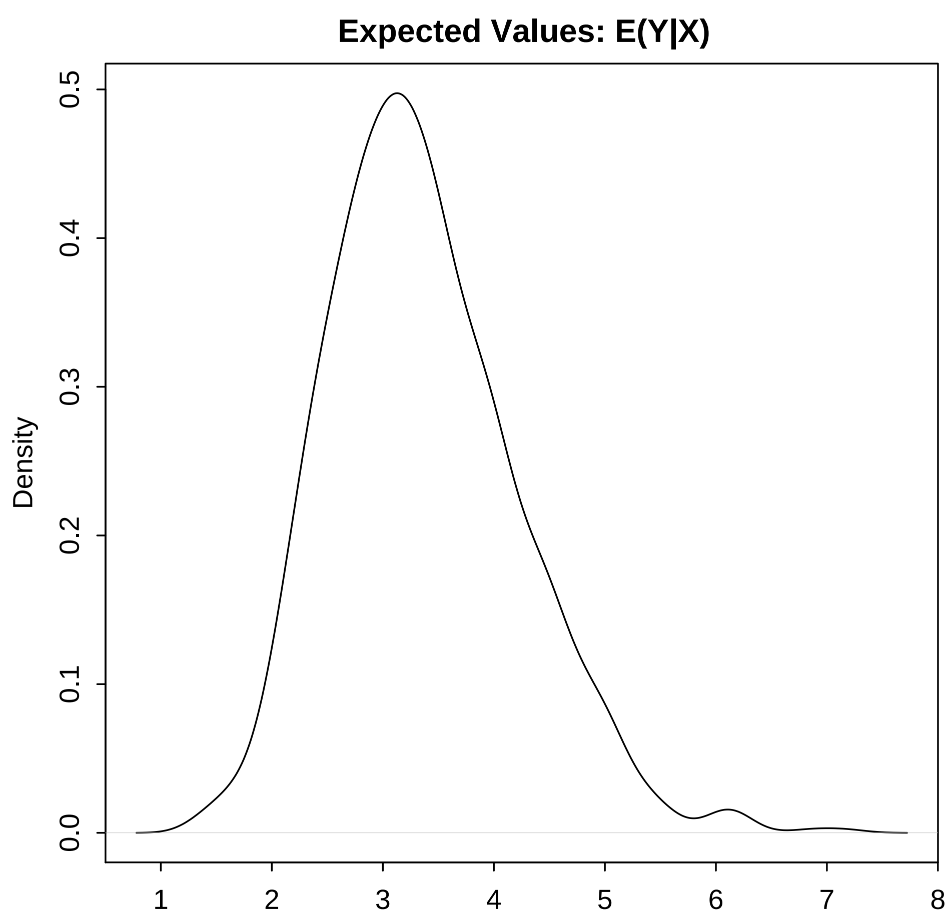
```
> x.out <- setx(z.out)
```

Simulate quantities of interest

```
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
```

Generate a plot of quantities of interest:

```
> plot(s.out)
```



The Model

Suppose we have a panel dataset, with Y_{it} denoting the dependent variable of the number of independent events for a fixed period of time for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \lambda_i) \\ Y_{it} &\sim g(y_{it} \mid \lambda_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means λ_i and λ_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\lambda_{it} = \exp(x_{it}\beta)$$

  where x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = \lambda_{it}$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi\, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = \lambda_{it}$ as the tth diagonal element, $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.

- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations. See **?** for more details.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.
- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.
- The expected values (`qi$ev`) for the GEE poisson model is the mean of simulations from the stochastic component:

  $$E(Y) = \lambda_c = \exp(x_c \beta),$$

  given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.
- The first difference (`qi$fd`) for the GEE poisson model is defined as

  $$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

  $$\frac{1}{\sum_{i=1}^{n} \sum_{t=1}^{T} tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it} = 1) - E[Y_{it}(tr_{it} = 0)] \right\},$$

  where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "poisson.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.
  - `fitted.values`: the vector of fitted values for the systemic component, λ_{it}.
  - `linear.predictors`: the vector of $x_{it}\beta$
  - `max.id`: the size of the largest cluster.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How To Cite

To cite the *poisson.gee* Zelig model:

> Patrick Lam. 2007. "poisson.gee: Generalized Estimating Equation for Poisson Regression," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from Martin (1992). Please inquire with Lisa Martin before publishing results from these data, as this dataset includes errors that have since been corrected.

12.52 `poisson.mixed`: Mixed effects poisson Regression

Use generalized multi-level linear regression if you have covariates that are grouped according to one or more classification factors. Poisson regression applies to dependent variables that represent the number of independent events that occur during a fixed period of time.

While generally called multi-level models in the social sciences, this class of models is often referred to as mixed-effects models in the statistics literature and as hierarchical models in a Bayesian setting. This general class of models consists of linear models that are expressed as a function of both *fixed effects*, parameters corresponding to an entire population or certain repeatable levels of experimental factors, and *random effects*, parameters corresponding to individual experimental units drawn at random from a population.

Syntax

```
z.out <- zelig(formula= y ~ x1 + x2 + tag(z1 + z2 | g),
               data=mydata, model="poisson.mixed")

z.out <- zelig(formula= list(mu=y ~ xl + x2 + tag(z1, gamma | g),
               gamma= ~ tag(w1 + w2 | g)), data=mydata, model="poisson.mixed")
```

Inputs

`zelig()` takes the following arguments for `mixed`:

- `formula`: a two-sided linear formula object describing the systematic component of the model, with the response on the left of a ˜ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1 + ... + zn | g)` with `z1 + ... + zn` specifying the model for the random effects and `g` the grouping structure. Random intercept terms are included with the notation `tag(1 | g)`.
  Alternatively, `formula` may be a list where the first entry, `mu`, is a two-sided linear formula object describing the systematic component of the model, with the repsonse on the left of a ˜ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1, gamma | g)` with `z1` specifying the individual level model for the random effects, `g` the grouping structure and `gamma` references the second equation in the list. The `gamma` equation is one-sided linear formula object with the group level model for the random effects on the right side of a ˜ operator. The model is specified with the notation `tag(w1 + ... + wn | g)` with `w1 + ... + wn` specifying the group level model and `g` the grouping structure.

Additional Inputs

In addition, `zelig()` accepts the following additional arguments for model specification:

- `data`: An optional data frame containing the variables named in `formula`. By default, the variables are taken from the environment from which `zelig()` is called.

- `method`: a character string. The criterion is always the log-likelihood but this criterion does not have a closed form expression and must be approximated. The default approximation is `"PQL"` or penalized quasi-likelihood. Alternatives are `"Laplace"` or `"AGQ"` indicating the Laplacian and adaptive Gaussian quadrature approximations respectively.

- `na.action`: A function that indicates what should happen when the data contain `NAs`. The default action (`na.fail`) causes `zelig()` to print an error message and terminate if there are any incomplete observations.

Additionally, users may with to refer to `lmer` in the package `lme4` for more information, including control parameters for the estimation algorithm and their defaults.

Examples

1. Basic Example

   Attach sample data:

   ```
   > data(homerun)
   ```

   Estimate model:

   ```
   > z.out1 <- zelig(homeruns ~ player + tag(player - 1 | month),
   +      data = homerun, model = "poisson.mixed")
   ```

   Summarize regression coefficients and estimated variance of random effects:

   ```
   > summary(z.out1)
   ```

   Set explanatory variables to their default values:

   ```
   > x.out <- setx(z.out1)
   ```

   Simulate draws using the default bootstrap method and view simulated quantities of interest:

   ```
   > s.out1 <- sim(z.out1, x = x.out)
   > summary(s.out1)
   ```

Mixed effects Poisson Regression Model

Let Y_{ij} be the number of independent events that occur during a fixed time period, realized for observation j in group i as y_{ij}, which takes any non-negative integer as its value, for $i = 1, \ldots, M$, $j = 1, \ldots, n_i$.

- The *stochastic component* is described by a Poisson distribution with mean and variance parameter λ_{ij}.
$$Y_{ij} \sim \text{Poisson}(y_{ij}|\lambda_{ij}) = \frac{\exp(-\lambda_{ij})\lambda_{ij}^{y_{ij}}}{y_{ij}!}$$
where
$$y_{ij} = 0, 1, \ldots$$
- The q-dimensional vector of *random effects*, b_i, is restricted to be mean zero, and therefore is completely characterized by the variance covarance matrix Ψ, a $(q \times q)$ symmetric positive semi-definite matrix.
$$b_i \sim Normal(0, \Psi)$$
- The *systematic component* is
$$\lambda_{ij} \equiv \exp(X_{ij}\beta + Z_{ij}b_i)$$
where X_{ij} is the $(n_i \times p \times M)$ array of known fixed effects explanatory variables, β is the p-dimensional vector of fixed effects coefficients, Z_{ij} is the $(n_i \times q \times M)$ array of known random effects explanatory variables and b_i is the q-dimensional vector of random effects.

Quantities of Interest

- The predicted values (`qi$pr`) are draws from the poisson distribution defined by mean λ_{ij}, for
$$\lambda_{ij} = \exp(X_{ij}\beta + Z_{ij}b_i)$$
given X_{ij} and Z_{ij} and simulations of of β and b_i from their posterior distributions. The estimated variance covariance matrices are taken as correct and are themselves not simulated.
- The expected values (`qi$ev`) is the mean of simulations of the stochastic component given draws of β from its posterior:
$$E(Y_{ij}|X_{ij}) = \lambda_{ij} = \exp(X_{ij}\beta).$$
- The first difference (`qi$fd`) is given by the difference in expected values, conditional on X_{ij} and X'_{ij}, representing different values of the explanatory variables.
$$FD(Y_{ij}|X_{ij}, X'_{ij}) = E(Y_{ij}|X_{ij}) - E(Y_{ij}|X'_{ij})$$

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij}=1) - \widehat{Y_{ij}(t_{ij}=0)}\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $Y_{ij}(t_{ij} = 0)$, the counterfactual predicted value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij}=1) - E[Y_{ij}(t_{ij}=0)]\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $E[Y_{ij}(t_{ij} = 0)]$, the counterfactual expected value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. You may examine the available information in `z.out` by using `slotNames(z.out)`, see the fixed effect coefficients by using `summary(z.out)@coefs`, and a default summary of information through `summary(z.out)`. Other elements available through the operator are listed below.

- From the `zelig()` output stored in `summary(z.out)`, you may extract:
  - `fixef`: numeric vector containing the conditional estimates of the fixed effects.
  - `ranef`: numeric vector containing the conditional modes of the random effects.
  - `frame`: the model frame for the model.
- From the `sim()` output stored in `s.out`, you may extract quantities of interest stored in a data frame:
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by the expected values.
  - `qi$ev`: the simulated expected values for the specified values of x.

- `qi$fd`: the simulated first differences in the expected values for the values specified in x and x1.
- `qi$ate.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.
- `qi$ate.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *poisson.mixed* Zelig model:

Delia Bailey, Ferdinand Alimadhi. 2007. "poisson.mixed: Mixed effects poisson regression" in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Mixed effects poisson regression is part of `lme4` package by Douglas M. Bates (Bates 2007). For a detailed discussion of mixed-effects models, please see **?**

12.53 `poisson.net`: Network Poisson Regression for Event Count Proximity Matrix Dependent Variables

Use network Poisson regression analysis for a dependent variable that represents the number of events that occur during a fixed period of time as a proximity matrix (a.k.a. sociomatricies, adjacency matrices, or matrix representations of directed graphs).

Syntax

```
> z.out <- zelig(y ~ x1 + x2, model = "poisson.net", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for network poisson regression:

- `LF`: specifies the link function to be used for the network poisson regression. Default is `LF="log"`, but `LF` can also be set to `"sqrt"` by the user.

Examples

1. Basic Example

   Load the sample data (see `?friendship` for details on the structure of the network dataframe):

   ```
   > data(friendship)
   ```

   Estimate model:

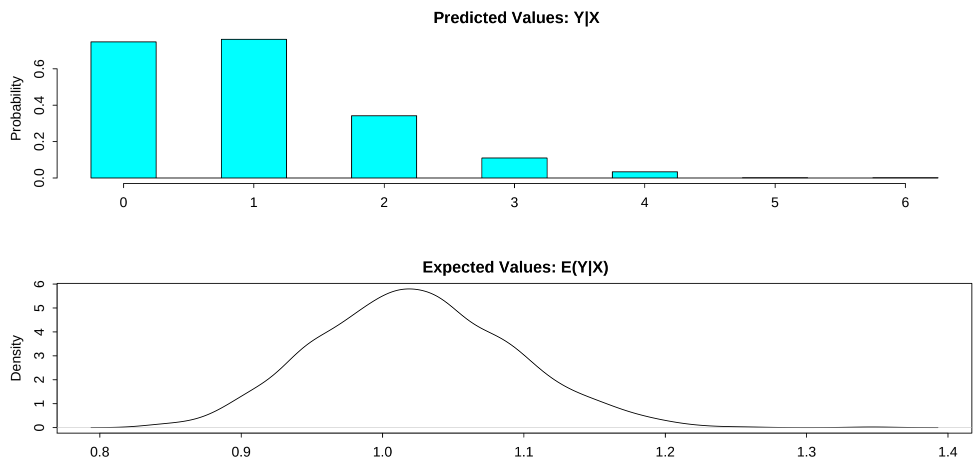
   ```
   > z.out <- zelig(count ~ advice + prestige + perpower, model = "poisson.net",
   +     data = friendship)
   > summary(z.out)
   ```

   Setting values for the explanatory variables to their default values:

   ```
   > x.out <- setx(z.out)
   ```

   Simulate fitted values.

   ```
   > s.out <- sim(z.out, x = x.out)
   > summary(s.out)
   > plot(s.out)
   ```



Model

The `poisson.net` model performs a Poisson regression of the proximity matrix $\mathbf{Y}$, a $m \times m$ matrix representing network ties, on a set of proximity matrices $\mathbf{X}$. This network regression model is directly analogous to standard Poisson regression element-wise on the appropriately vectorized matrices. Proximity matrices are vectorized by creating Y, a $m^2 \times 1$ vector to represent the proximity matrix. The vectorization which produces the Y vector from the $\mathbf{Y}$ matrix is performed by simple row-concatenation of $\mathbf{Y}$. For example, if $\mathbf{Y}$ is a 15×15 matrix, the $\mathbf{Y}_{1,1}$ element is the first element of Y, and the $\mathbf{Y}_{2,1}$ element is the second element of Y and so on. Once the input matrices are vectorized, standard Poisson regression is performed.

Let Y_i be the dependent variable, produced by vectorizing an event count proximity matrix, for observation i. Y_i is thus the number of independent events that occur during a fixed time period. This variable can take any non-negative integer.

- The Poisson distribution has *stochastic component*

$$Y_i \sim \text{Poisson}(\lambda_i),$$

  where λ_i is the mean and variance parameter.

- The *systematic component* is given by:

$$\lambda_i = \exp(x_i \beta).$$

  where x_i is the vector of explanatory variables and β is the vector of coefficients.

Quantities of Interest

The quantities of interest for the network Poisson regression are the same as those for the standard Poisson regression.

- The expected value (qi$ev) for the `poisson.net` model is the mean of simulations from the stochastic component,

$$E(Y) = \lambda_i = \exp(x_i \beta),$$

given draws of β from its sampling distribution.

- The predicted value (qi$pr) is a random draw from the Poisson distribution defined by mean λ_i.

- The first difference (qi$fd) for the network Poisson model is defined as

$$FD = \Pr(Y|x_1) - \Pr(Y|x)$$

Output Values

The output of each Zelig command contains useful information which you may view. For example, you run `z.out <- zelig(y ~ x, model = "poisson.net", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the systemic component λ.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `linear.predictors`: the vector of $x_i \beta$.
  - `aic`: Akaikeś Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `bic`: the Bayesian Information Criterion (minus twice the maximized log-likelihood plus the number of coefficients times log n).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`
- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `mod.coefficients`: the parameter estimates with their associated standard errors, p-values, and t statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite

To cite the *poisson.net* Zelig model:

> Skyler J. Cranmer. 2007. "poisson.net: Network Poisson Regression for Event Count Proximity Matrix Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The network normal regression is part of the `netglm` package by Skyler J. Cranmer and is built using some of the functionality of the `sna` package by Carter T. Butts (Butts and Carley 2001).In addition, advanced users may wish to refer to `help(poisson.net)`. Sample data are fictional.

12.54 `poisson.survey`: Survey-Weighted Poisson Regression for Event Count Dependent Variables

The survey-weighted poisson regression model is appropriate for survey data obtained using complex sampling techniques, such as stratified random or cluster sampling (e.g., not simple random sampling). Like the conventional poisson regression model (see Section 12.48), survey-weighted poisson regression specifies a dependent variable representing the number of independent events that occur during a fixed period of time as function of a set of explanatory variables. The survey-weighted poisson model reports estimates of model parameters identical to conventional poisson estimates, but uses information from the survey design to correct variance estimates.

The `poisson.survey` model accommodates three common types of complex survey data. Each method listed here requires selecting specific options which are detailed in the "Additional Inputs" section below.

1. **Survey weights**: Survey data are often published along with weights for each observation. For example, if a survey intentionally over-samples a particular type of case, weights can be used to correct for the over-representation of that type of case in the dataset. Survey weights come in two forms:
   (a) *Probability* weights report the probability that each case is drawn from the population. For each stratum or cluster, this is computed as the number of observations in the sample drawn from that group divided by the number of observations in the population in the group.
   (b) *Sampling* weights are the inverse of the probability weights.
2. **Strata/cluster identification**: A complex survey dataset may include variables that identify the strata or cluster from which observations are drawn. For stratified random sampling designs, observations may be nested in different strata. There are two ways to employ these identifiers:
   (a) Use *finite population corrections* to specify the total number of cases in the stratum or cluster from which each observation was drawn.
   (b) For stratified random sampling designs, use the raw strata ids to compute sampling weights from the data.
3. **Replication weights**: To preserve the anonymity of survey participants, some surveys exclude strata and cluster ids from the public data and instead release only pre-computed replicate weights.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "poisson.survey", data = mydata)
```

```
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard `zelig` inputs (see Section ??), survey-weighted poisson models accept the following optional inputs:

1. Datasets that include survey weights:.
   - `probs`: An optional formula or numerical vector specifying each case's probability weight, the probability that the case was selected. Probability weights need not (and, in most cases, will not) sum to one. Cases with lower probability weights are weighted more heavily in the computation of model coefficients.
   - `weights`: An optional numerical vector specifying each case's sample weight, the inverse of the probability that the case was selected. Sampling weights need not (and, in most cases, will not) sum to one. Cases with higher sampling weights are weighted more heavily in the computation of model coefficients.
2. Datasets that include strata/cluster identifiers:
   - `ids`: An optional formula or numerical vector identifying the cluster from which each observation was drawn (ordered from largest level to smallest level). For survey designs that do not involve cluster sampling, `ids` defaults to `NULL`.
   - `fpc`: An optional numerical vector identifying each case's frequency weight, the total number of units in the population from which each observation was sampled.
   - `strata`: An optional formula or vector identifying the stratum from which each observation was sampled. Entries may be numerical, logical, or strings. For survey designs that do not involve cluster sampling, `strata` defaults to `NULL`.
   - `nest`: An optional logical value specifying whether primary sampling unites (PSUs) have non-unique ids across multiple strata. `nest=TRUE` is appropriate when PSUs reuse the same identifiers across strata. Otherwise, `nest` defaults to `FALSE`.
   - `check.strata`: An optional input specifying whether to check that clusters are nested in strata. If `check.strata` is left at its default, `!nest`, the check is not performed. If `check.strata` is specified as `TRUE`, the check is carried out.
3. Datasets that include replication weights:
   - `repweights`: A formula or matrix specifying replication weights, numerical vectors of weights used in a process in which the sample is repeatedly re-weighted and parameters are re-estimated in order to compute the variance of the model parameters.

- `type`: A string specifying the type of replication weights being used. This input is required if replicate weights are specified. The following types of replication weights are recognized: `"BRR"`, `"Fay"`, `"JK1"`, `"JKn"`, `"bootstrap"`, or `"other"`.
- `weights`: An optional vector or formula specifying each case's sample weight, the inverse of the probability that the case was selected. If a survey includes both sampling weights and replicate weights separately for the same survey, both should be included in the model specification. In these cases, sampling weights are used to correct potential biases in in the computation of coefficients and replication weights are used to compute the variance of coefficient estimates.
- `combined.weights`: An optional logical value that should be specified as `TRUE` if the replicate weights include the sampling weights. Otherwise, `combined.weights` defaults to `FALSE`.
- `rho`: An optional numerical value specifying a shrinkage factor for replicate weights of type `"Fay"`.
- `bootstrap.average`: An optional numerical input specifying the number of iterations over which replicate weights of type `"bootstrap"` were averaged. This input should be left as `NULL` for `"bootstrap"` weights that were not were created by averaging.
- `scale`: When replicate weights are included, the variance is computed as the sum of squared deviations of the replicates from their mean. `scale` is an optional overall multiplier for the standard deviations.
- `rscale`: Like `scale`, `rscale` specifies an optional vector of replicate-specific multipliers for the squared deviations used in variance computation.
- `fpc`: For models in which `"JK1"`, `"JKn"`, or `"other"` replicates are specified, `fpc` is an optional numerical vector (with one entry for each replicate) designating the replicates' finite population corrections.
- `fpctype`: When a finite population correction is included as an `fpc` input, `fpctype` is a required input specifying whether the input to `fpc` is a sampling fraction (`fpctype="fraction"`) or a direct correction (`fpctype="correction"`).
- `return.replicates`: An optional logical value specifying whether the replicates should be returned as a component of the output. `return.replicates` defaults to `FALSE`.

Examples

1. A dataset that includes survey weights:

   Attach the sample data:

```
> data(api, package = "survey")
```

Suppose that a dataset included a variable reporting the number of times a new student enrolled during the previous school year (`enroll`), a measure of each school's academic performance (`api99`), an indicator for whether each school holds classes year round (`year.rnd`), and sampling weights computed by the survey house (`pw`). Estimate a model that regresses `enroll` on `api99` and `year.rnd`:

```
> z.out1 <- zelig(enroll ~ api99 + yr.rnd, model = "poisson.survey",
+     weights = ~pw, data = apistrat)
```

Summarize regression coefficients:

```
> summary(z.out1)
```

Set explanatory variables to their default (mean/mode) values, and set a high (80th percentile) and low (20th percentile) value for the measure of academic performance:

```
> x.low <- setx(z.out1, api99 = quantile(apistrat$api99, 0.2))
> x.high <- setx(z.out1, api99 = quantile(apistrat$api99, 0.8))
```

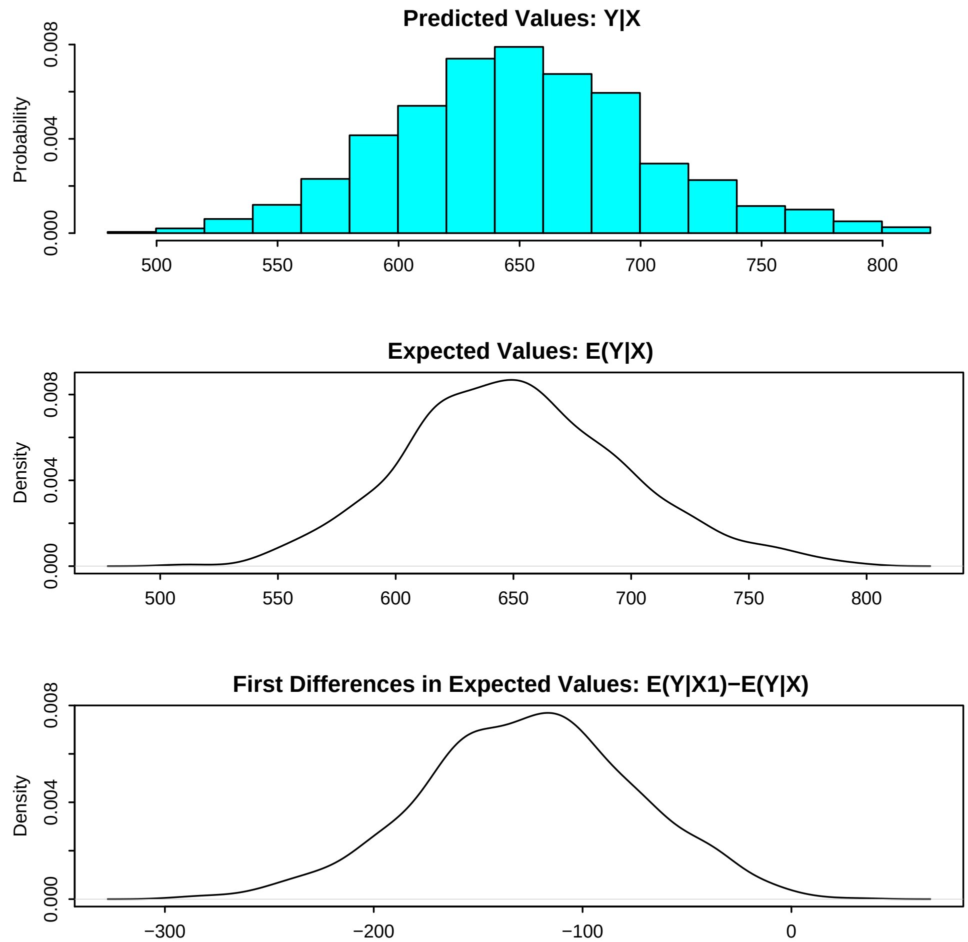
Generate first differences for the effect of high versus low concentrations of children receiving subsidized meals on the probability of holding school year-round:

```
> s.out1 <- sim(z.out1, x = x.low, x1 = x.high)
```

```
> summary(s.out1)
```

Generate a visual summary of the quantities of interest:

```
> plot(s.out1)
```



2. A dataset that includes strata/cluster identifiers:

   Suppose that the survey house that provided the dataset used in the previous example excluded sampling weights but made other details about the survey design available. A user can still estimate a model without sampling weights that instead uses inputs that identify the stratum and/or cluster to which each observation belongs and the size of the finite population from which each observation was drawn.

   Continuing the example above, suppose the survey house drew at random a fixed number of elementary schools, a fixed number of middle schools, and a fixed number of high schools. If the variable `stype` is a vector of characters (`"E"` for elementary schools, `"M"` for middle schools, and `"H"` for high schools) that identifies the type of school each case represents and `fpc` is a numerical vector that identifies for each case the total number of schools of the same type in the population, then the user could estimate the following model:

```
> z.out2 <- zelig(enroll ~ api99 + yr.rnd, model = "poisson.survey",
+      data = apistrat, strata = ~stype, fpc = ~fpc)
```

Summarize the regression output:

```
> summary(z.out2)
```

The coefficient estimates for this example are identical to the point estimates in the first example, when pre-existing sampling weights were used. When sampling weights are omitted, they are estimated automatically for "poisson.survey" models based on the user-defined description of sampling designs.

Moreover, because the user provided information about the survey design, the standard error estimates are lower in this example than in the previous example, in which the user omitted variables pertaining to the details of the complex survey design.

3. A dataset that includes replication weights:

   Consider a dataset that includes information for a sample of hospitals about the number of out-of-hospital cardiac arrests that each hospital treats and the number of patients who arrive alive at each hospital:

```
> data(scd, package = "survey")
```

   Survey houses sometimes supply replicate weights (in lieu of details about the survey design). For the sake of illustrating how replicate weights can be used as inputs in normal.survey models, create a set of balanced repeated replicate (BRR) weights:

```
> BRRrep <- 2 * cbind(c(1, 0, 1, 0, 1, 0), c(1, 0, 0, 1, 0, 1),
+     c(0, 1, 1, 0, 0, 1), c(0, 1, 0, 1, 1, 0))
```

   Estimate a model that regresses the count of patients who arrived alive at the hospital last year on the number of patients treated for cardiac arrests, using the BRR replicate weights in BRRrep to compute standard errors:

```
> z.out3 <- zelig(alive ~ arrests, model = "poisson.survey", repweights = BRRrep,
+     type = "BRR", data = scd)
> summary(z.out3)
```

   Summarize the regression coefficients:

```
> summary(z.out3)
```

   Set the explanatory variable arrests at its 20th and 80th quantiles:

```
> x.low <- setx(z.out3, arrests = quantile(scd$arrests, 0.2))
> x.high <- setx(z.out3, arrests = quantile(scd$arrests, 0.8))
```

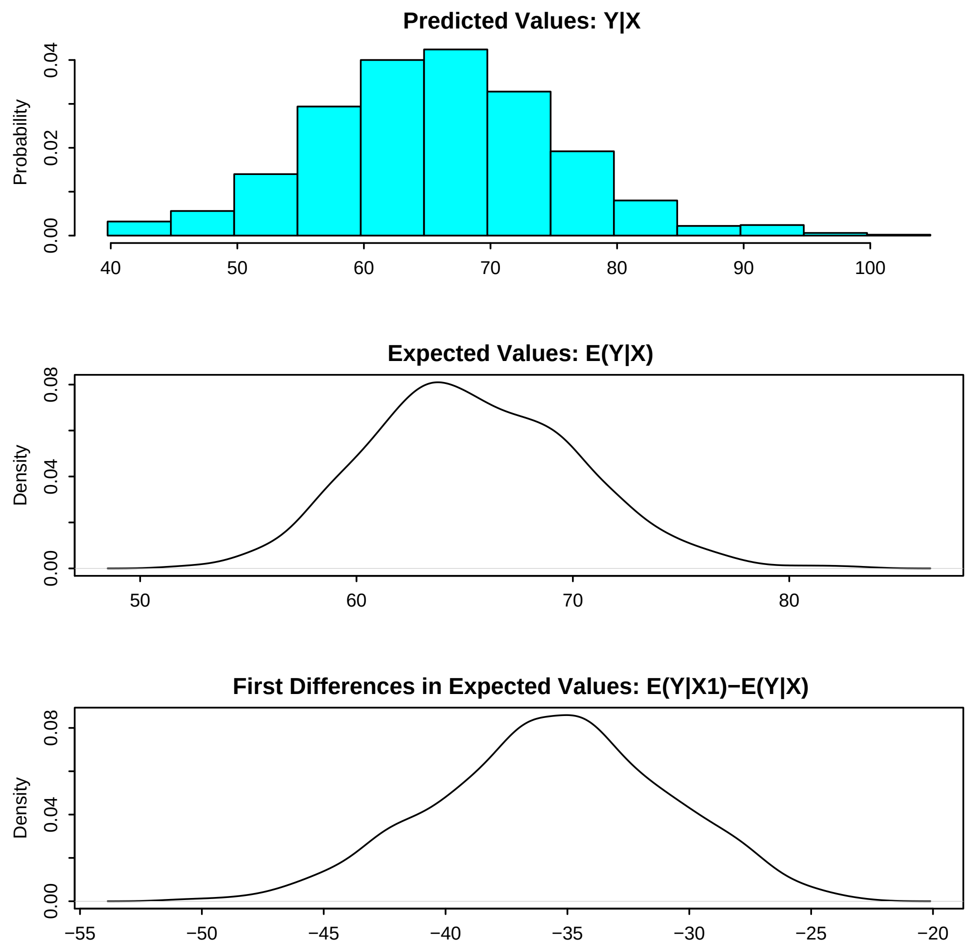
   Generate first differences for the effect of high versus low cardiac arrests on the count of patients who arrive alive:

```
> s.out3 <- sim(z.out3, x = x.high, x1 = x.low)
```

```
> summary(s.out3)
```

Generate a visual summary of quantities of interest:

```
> plot(s.out3)
```



Model

Let Y_i be the number of independent events that occur during a fixed time period. This variable can take any non-negative integer.

- The Poisson distribution has *stochastic component*

$$Y_i \sim \text{Poisson}(\lambda_i),$$

where λ_i is the mean and variance parameter.

- The *systematic component* is

$$\lambda_i = \exp(x_i \beta),$$

where x_i is the vector of explanatory variables, and β is the vector of coefficients.

Variance

When replicate weights are not used, the variance of the coefficients is estimated as

$$\hat{\mathbf{\Sigma}} \left[\sum_{i=1}^{n} \frac{(1-\pi_i)}{\pi_i^2} (\mathbf{X}_i(Y_i - \mu_i))'(\mathbf{X}_i(Y_i - \mu_i)) + 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\pi_{ij} - \pi_i \pi_j)}{\pi_i \pi_j \pi_{ij}} (\mathbf{X}_i(Y_i - \mu_i))'(\mathbf{X}_j(Y_j - \mu_j)) \right] \hat{\mathbf{\Sigma}}$$

where π_i is the probability of case i being sampled, $\mathbf{X}_i$ is a vector of the values of the explanatory variables for case i, Y_i is value of the dependent variable for case i, $\hat{\mu}_i$ is the predicted value of the dependent variable for case i based on the linear model estimates, and $\hat{\mathbf{\Sigma}}$ is the conventional variance-covariance matrix in a parametric glm. This statistic is derived from the method for estimating the variance of sums described in Binder (1983) and the Horvitz-Thompson estimator of the variance of a sum described in Horvitz and Thompson (1952).

When replicate weights are used, the model is re-estimated for each set of replicate weights, and the variance of each parameter is estimated by summing the squared deviations of the replicates from their mean.

Quantities of Interest

- The expected value (`qi$ev`) is the mean of simulations from the stochastic component,

$$E(Y) = \lambda_i = \exp(x_i \beta),$$

given draws of β from its sampling distribution.

- The predicted value (`qi$pr`) is a random draw from the poisson distribution defined by mean λ_i.

- The first difference in the expected values (`qi$fd`) is given by:

$$\text{FD} = E(Y|x_1) - E(Y \mid x)$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating

$E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "poisson.survey", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: a vector of the fitted values for the systemic component λ.
  - `linear.predictors`: a vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.

- `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation × `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values given the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by λ_i.
  - `qi$fd`: the simulated first differences in the expected values given the specified values of `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

When users estimate `poisson.survey` models with replicate weights in `Zelig`, an object called `.survey.prob.weights` is created in the global environment. `Zelig` will over-write any existing object with that name, and users are therefore advised to re-name any object called `.survey.prob.weights` before using `poisson.survey` models in `Zelig`.

How to Cite

To cite the *poisson.survey* Zelig model:

Nicholas Carnes. 2008. "poisson.survey: Survey-Weighted Poisson Regression for Event Count Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Survey-weighted linear models and the sample data used in the examples above are a part of the `survey` package by Thomas Lumley. Users may wish to refer to the help files for the three functions that Zelig draws upon when estimating survey-weighted models, namely, `help(svyglm)`, `help(svydesign)`, and `help(svrepdesign)`. The poisson model is part of the stats package by Venables and Ripley (2002). Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as McCullagh and Nelder (1989).

12.55 `probit`: Probit Regression for Dichotomous Dependent Variables

Use probit regression to model binary dependent variables specified as a function of a set of explanatory variables. For a Bayesian implementation of this model, see Section 12.56.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "probit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out, x1 = NULL)
```

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for probit regression:

- `robust`: defaults to `FALSE`. If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see Zeileis (2004)). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    - `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.
    - `"kernHAC"`: HAC standard errors using the weights given in Andrews (1991).
    - `"weave"`: HAC standard errors using the weights given in Lumley and Heagerty (1999).
  - `order.by`: defaults to `NULL` (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as `order.by = z`, where `z` exists outside the data frame; or as `order.by = ~z`, where `z` is a variable in the data frame). The observations are chronologically ordered by the size of `z`.
  - `...`: additional options passed to the functions specified in `method`. See the `sandwich` library and Zeileis (2004) for more options.

Examples

Attach the sample turnout dataset:

```
> data(turnout)
```

Estimate parameter values for the probit regression:

```
> z.out <- zelig(vote ~ race + educate, model = "probit", data = turnout)

> summary(z.out)
```

Set values for the explanatory variables to their default values.

```
> x.out <- setx(z.out)
```

Simulate quantities of interest from the posterior distribution.

```
> s.out <- sim(z.out, x = x.out)

> summary(s.out)
```

Model

Let Y_i be the observed binary dependent variable for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$Y_i \sim \text{Bernoulli}(\pi_i),$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is

$$\pi_i = \Phi(x_i \beta)$$

  where $\Phi(\mu)$ is the cumulative distribution function of the Normal distribution with mean 0 and unit variance.

Quantities of Interest

- The expected value (`qi$ev`) is a simulation of predicted probability of success

$$E(Y) = \pi_i = \Phi(x_i \beta),$$

  given a draw of β from its sampling distribution.

- The predicted value (`qi$pr`) is a draw from a Bernoulli distribution with mean π_i.

- The first difference (`qi$fd`) in expected values is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

$$\text{RR} = \Pr(Y = 1 \mid x_1) / \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \{Y_i(t_i = 1) - E[Y_i(t_i = 0)]\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}\right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "probit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: a vector of the in-sample fitted values.
  - `linear.predictors`: a vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.

- `data`: the name of the input data frame.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values, or predicted probabilities, for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by the predicted probabilities.
  - `qi$fd`: the simulated first differences in the predicted probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the predicted probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *probit* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "probit: Probit Regression for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The probit model is part of the stats package by Venables and Ripley (2002). Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as McCullagh and Nelder (1989). Robust standard errors are implemented via the sandwich package by Zeileis (2004). Sample data are from King et al. (2000).

12.56 `probit.bayes`: Bayesian Probit Regression

Use the probit regression model for model binary dependent variables specified as a function of a set of explanatory variables. The model is estimated using a Gibbs sampler. For other models suitable for binary response variables, see Bayesian logistic regression(Section 12.23), maximum likelihood logit regression (Section 12.22), and maximum likelihood probit regression (Section 12.55).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "probit.bayes", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

Using the following arguments to monitor the Markov chains:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults to 10,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.
- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.
- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed of 12345.
- `beta.start`: starting values for the Markov chain, either a scalar or vector with length equal to the number of estimated coefficients. The default is `NA`, such that the maximum likelihood estimates are used as the starting values.

Use the following parameters to specify the model's priors:

- `b0`: prior mean for the coefficients, either a numeric vector or a scalar. If a scalar value, that value will be the prior mean for all the coefficients. The default is 0.
- `B0`: prior precision parameter for the coefficients, either a square matrix (with the dimensions equal to the number of the coefficients) or a scalar. If a scalar value, that value times an identity matrix will be the prior precision parameter. The default is 0, which leads to an improper prior.

Use the following arguments to specify optional output for the model:

- `bayes.resid`: defaults to `FALSE`. If `TRUE`, the latent Bayesian residuals for all observations are returned. Alternatively, users can specify a vector of observations for which the latent residuals should be returned.

Zelig users may wish to refer to `help(MCMCprobit)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example
   Attaching the sample dataset:

   ```
   > data(turnout)
   ```

   Estimating the probit regression using `probit.bayes`:

   ```
   > z.out <- zelig(vote ~ race + educate, model = "probit.bayes",
   +      data = turnout, verbose = TRUE)
   ```

   Checking for convergence before summarizing the estimates:

```
> geweke.diag(z.out$coefficients)
```

```
> heidel.diag(z.out$coefficients)
```

```
> raftery.diag(z.out$coefficients)
```

```
> summary(z.out)
```

Setting values for the explanatory variables to their sample averages:

```
> x.out <- setx(z.out)
```

Simulating quantities of interest from the posterior distribution given: `x.out`

```
> s.out1 <- sim(z.out, x = x.out)
```

```
> summary(s.out1)
```

2. Simulating First Differences
   Estimating the first difference (and risk ratio) in individual's probability of voting when education is set to be low (25th percentile) versus high (75th percentile) while all the other variables are held at their default values:

```
> x.high <- setx(z.out, educate = quantile(turnout$educate, prob = 0.75))
> x.low <- setx(z.out, educate = quantile(turnout$educate, prob = 0.25))
```

```
> s.out2 <- sim(z.out, x = x.high, x1 = x.low)
```

```
> summary(s.out2)
```

Model

Let Y_i be the binary dependent variable for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$\begin{aligned} Y_i &\sim \text{Bernoulli}(\pi_i) \\ &= \pi_i^{Y_i}(1-\pi_i)^{1-Y_i}, \end{aligned}$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by

$$\pi_i = \Phi(x_i \beta),$$

  where $\Phi(\cdot)$ is the cumulative density function of the standard Normal distribution with mean 0 and variance 1, x_i is the vector of k explanatory variables for observation i, and β is the vector of coefficients.

- The *prior* for β is given by

$$\beta \sim \text{Normal}_k \left(b_0, B_0^{-1}\right)$$

where b_0 is the vector of means for the k explanatory variables and B_0 is the $k \times k$ precision matrix (the inverse of a variance-covariance matrix).

Quantities of Interest

- The expected values (`qi$ev`) for the probit model are the predicted probability of a success:

$$E(Y \mid X) = \pi_i = \Phi(x_i \beta),$$

given the posterior draws of β from the MCMC iterations.

- The predicted values (`qi$pr`) are draws from the Bernoulli distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the probit model is defined as

$$\text{FD} = \Pr(Y = 1 \mid X_1) - \Pr(Y = 1 \mid X).$$

- The risk ratio (`qi$rr`)is defined as

$$\text{RR} = \Pr(Y = 1 \mid X_1) \ / \ \Pr(Y = 1 \mid X).$$

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is

$$\frac{1}{\sum t_i} \sum_{i:t_i=1} [Y_i(t_i = 1) - E[Y_i(t_i = 0)]],$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups.

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group is

$$\frac{1}{\sum t_i} \sum_{i:t_i=1} [Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}],$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(y ~ x, model = "probit.bayes", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated parameters.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
  - `bayes.residuals`: When `bayes.residual` is `TRUE` or a set of observation numbers is given, this object contains the posterior draws of the latent Bayesian residuals of all the observations or the observations specified by the user.
  - `seed`: the random seed used in the model.
- From the `sim()` output object `s.out`:
  - `qi$ev`: the simulated expected values (probabilities) for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected values for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected values simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *probit.bayes* Zelig model:

> Ben Goodrich and Ying Lu. 2007. "probit.bayes: Bayesian Probit Regression for Dichotomous Dependent Variable," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.
>
> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Bayesian probit regression is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005).

12.57 `probit.gam`: Generalized Additive Model for Dichotomous Dependent Variables

This function runs a nonparametric Generalized Additive Model (GAM) for dichotomous dependent variables.

Syntax

```
> z.out <- zelig(y ~ x1 + s(x2), model = "probit.gam", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Where `s()` indicates a variable to be estimated via nonparametric smooth. All variables for which `s()` is not specified, are estimated via standard parametric methods.

Additional Inputs

In addition to the standard inputs, `zelig()` takes the following additional options for GAM models.

- `method`: Controls the fitting method to be used. Fitting methods are selected via a list environment within `method=gam.method()`. See `gam.method()` for details.
- `scale`: Generalized Cross Validation (GCV) is used if `scale = 0` (see the "Model" section for details) except for Logit models where a Un-Biased Risk Estimator (UBRE) (also see the "Model" section for details) is used with a scale parameter assumed to be 1. If `scale` is greater than 1, it is assumed to be the scale parameter/variance and UBRE is used. If `scale` is negative GCV is used.
- `knots`: An optional list of knot values to be used for the construction of basis functions.
- `H`: A user supplied fixed quadratic penalty on the parameters of the GAM can be supplied with this as its coefficient matrix. For example, ridge penalties can be added to the parameters of the GAM to aid in identification on the scale of the linear predictor.
- `sp`: A vector of smoothing parameters for each term.
- `...`: additional options passed to the `probit.gam` model. See the `mgcv` library for details.

Examples

1. Basic Example

   Create some count data:

```
> set.seed(0);  n <- 400; sig <- 2;
> x0 <- runif(n, 0, 1);  x1 <- runif(n, 0, 1)
> x2 <- runif(n, 0, 1);  x3 <- runif(n, 0, 1)
> g <- (f-5)/3
> g <- binomial()$linkinv(g)
> y <- rbinom(g,1,g)
> my.data <- as.data.frame(cbind(y, x0, x1, x2, x3))
```

Estimate the model, summarize the results, and plot nonlinearities:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), model = "probit.gam",
+     data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

Note that the `plot()` function can be used after model estimation and before simulation to view the nonlinear relationships in the independent variables:

Set values for the explanatory variables to their default (mean/mode) values, then simulate, summarize and plot quantities of interest:

```
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
> plot(s.out)
```

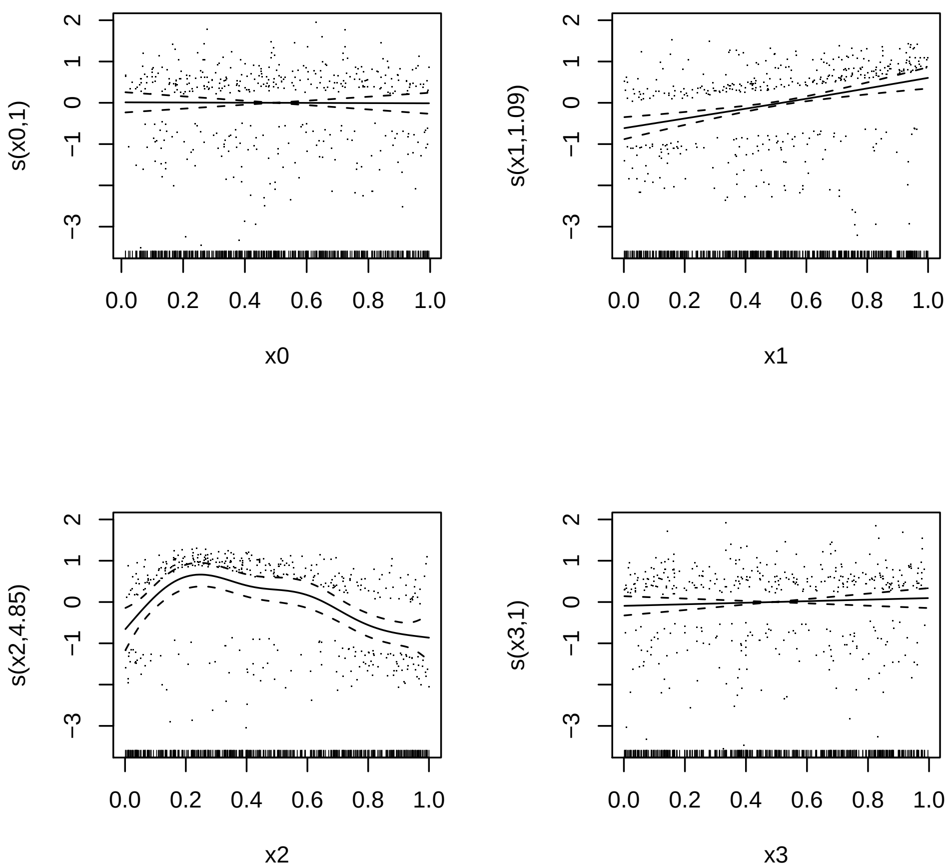
2. Simulating First Differences

   Estimating the risk difference (and risk ratio) between low values (20th percentile) and high values (80th percentile) of the explanatory variable `x3` while all the other variables are held at their default (mean/mode) values.

```
> x.high <- setx(z.out, x3 = quantile(my.data$x3, 0.8))
> x.low <- setx(z.out, x3 = quantile(my.data$x3, 0.2))
> s.out <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out)
> plot(s.out)
```

3. Variations in GAM model specification. Note that `setx` and `sim` work as shown in the above examples for any GAM model. As such, in the interest of parsimony, I will not re-specify the simulations of quantities of interest.

   An extra ridge penalty (useful with convergence problems):



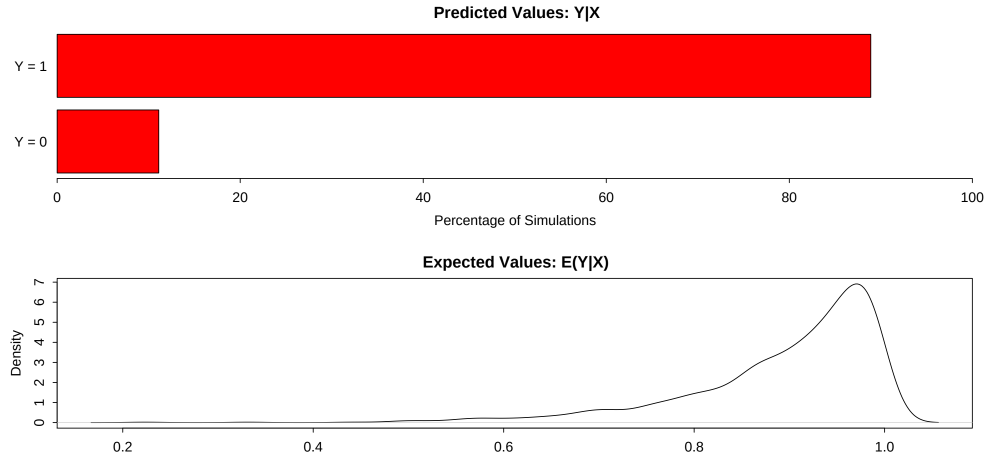

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), H = diag(0.5,
+     37), model = "probit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1, residuals = TRUE)
```

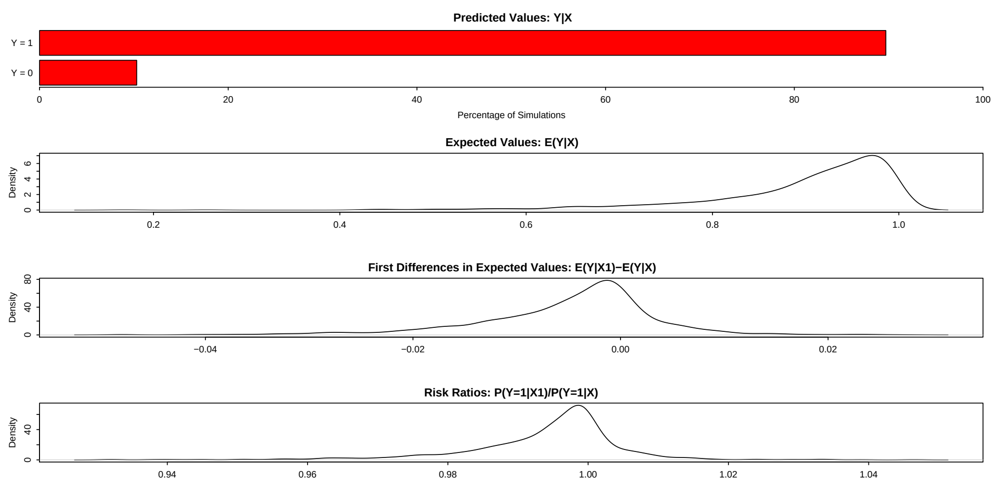
Set the smoothing parameter for the first term, estimate the rest:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), sp = c(0.01,
+     -1, -1, -1), model = "probit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Set lower bounds on smoothing parameters:

```
> z.out <- zelig(y ~ s(x0) + s(x1) + s(x2) + s(x3), min.sp = c(0.001,
+     0.01, 0, 10), model = "probit.gam", data = my.data)
```






```
> summary(z.out)
> plot(z.out, pages = 1)
```

A GAM with 3df regression spline term & 2 penalized terms:

```
> z.out <- zelig(y ~ s(x0, k = 4, fx = TRUE, bs = "tp") + s(x1,
+     k = 12) + s(x2, k = 15), model = "probit.gam", data = my.data)
> summary(z.out)
> plot(z.out, pages = 1)
```

Model

GAM models use families the same way GLM models do: they specify the distribution and link function to use in model fitting. In the case of `probit.gam` a normal link function is used. Specifically, let Y_i be the binary dependent variable for observation i which takes the value of either 0 or 1.

- The normal distribution has *stochastic component*

$$Y_i \sim \text{Bernoulli}(\pi_i)$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = \mathbf{\Phi}\left(x_i\beta + \sum_{j=1}^{J} f_j(Z_j)\right),$$

  where $\mathbf{\Phi}(\mu)$ is the cumulative distribution function of the Normal distribution with mean 0 and unit variance and $f_j(Z_j)$ for $j = 1, \ldots J$ is the set of smooth terms.

Generalized additive models (GAMs) are similar in many respects to generalized linear models (GLMs). Specifically, GAMs are generally fit by penalized maximum likelihood estimation and GAMs have (or can have) a parametric component identical to that of a GLM. The difference is that GAMs also include in their linear predictors a specified sum of smooth functions.

In this GAM implementation, smooth functions are represented using penalized regression splines. Two techniques may be used to estimate smoothing parameters: Generalized Cross Validation (GCV),

$$n\frac{D}{(n - DF)^2}, \tag{12.8}$$

or an Un-Biased Risk Estimator (UBRE) (which is effectively just a rescaled AIC),

$$\frac{D}{n} + 2s\frac{DF}{n - s}, \tag{12.9}$$

where D is the deviance, n is the number of observations, s is the scale parameter, and DF is the effective degrees of freedom of the model. The use of GCV or UBRE can be set by the user with the `scale` command described in the "Additional Inputs" section and in either case, smoothing parameters are chosen to minimize the GCV or UBRE score for the model.

Estimation for GAM models proceeds as follows: first, basis functions and a set (one or more) of quadratic penalty coefficient matrices are constructed for each smooth term. Second, a model matrix is is obtained for the parametric component of the GAM. These matrices are combined to produce a complete model matrix and a set of penalty matrices for the smooth terms. Iteratively Reweighted Least Squares (IRLS) is then used to estimate the model; at each iteration of the IRLS, a penalized weighted least squares model is run and the smoothing parameters of that model are estimated by GCV or UBRE. This process is repeated until convergence is achieved.

Further details of the GAM fitting process are given in Wood (2000, 2004, 2006).

Quantities of Interest

The quantities of interest for the `probit.gam` model are the same as those for the standard normal regression.

- The expected value (`qi$ev`) for the `probit.gam` model is the mean of simulations from the stochastic component,

$$\pi_i = \mathbf{\Phi}\left(x_i\beta + \sum_{j=1}^{J} f_j(Z_j)\right).$$

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the probit.gam model is defined as

$$FD = \Pr(Y|w_1) - \Pr(Y|w)$$

  for $w = \{X, Z\}$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "probit.gam", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `coefficients(z.out)`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IRLS fit.
  - `linear.predictors`: the vector of $x_i\beta$.
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `method`: the fitting method used.
  - `converged`: logical indicating weather the model converged or not.
  - `smooth`: information about the smoothed parameters.
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `data`: the input data frame.

– `model`: the model matrix used.

- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  – `p.coeff`: the coefficients of the parametric components of the model.
  – `se`: the standard errors of the entire model.
  – `p.table`: the coefficients, standard errors, and associated t statistics for the parametric portion of the model.
  – `s.table`: the table of estimated degrees of freedom, estimated rank, F statistics, and p-values for the nonparametric portion of the model.
  – `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  – `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.
- From the `sim()` output stored in `s.out`, you may extract:
  – `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  – `qi$pr`: the simulated predicted values for the specified values of `x`.
  – `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite

To cite the *probit.gam* Zelig model:

> Skyler J. Cranmer. 2007. "probit.gam: Generalized Additive Model for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The `gam.logit` model is adapted from the `mgcv` package by Simon N. Wood (Wood 2006). Advanced users may wish to refer to `help(gam)`, Wood (2004), Wood (2000), and other documentation accompanying the `mgcv` package. All examples are reproduced and extended from `mgcv`'s `gam()` help pages.

12.58 `probit.gee`: Generalized Estimating Equation for Probit Regression

The GEE probit estimates the same model as the standard probit regression (appropriate when you have a dichotomous dependent variable and a set of explanatory variables). Unlike in probit regression, GEE probit allows for dependence within clusters, such as in longitudinal data, although its use is not limited to just panel data. The user must first specify a "working" correlation matrix for the clusters, which models the dependence of each observation with other observations in the same cluster. The "working" correlation matrix is a $T \times T$ matrix of correlations, where T is the size of the largest cluster and the elements of the matrix are correlations between within-cluster observations. The appeal of GEE models is that it gives consistent estimates of the parameters and consistent estimates of the standard errors can be obtained using a robust "sandwich" estimator even if the "working" correlation matrix is incorrectly specified. If the "working" correlation matrix is correctly specified, GEE models will give more efficient estimates of the parameters. GEE models measure population-averaged effects as opposed to cluster-specific effects (See Zorn (2001)).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "probit.gee",
                 id = "X3", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

where `id` is a variable which identifies the clusters. The data should be sorted by `id` and should be ordered within each cluster when appropriate.

Additional Inputs

- `robust`: defaults to `TRUE`. If `TRUE`, consistent standard errors are estimated using a "sandwich" estimator.

Use the following arguments to specify the structure of the "working" correlations within clusters:

- `corstr`: defaults to `"independence"`. It can take on the following arguments:
  - Independence (`corstr = "independence"`): $\mathrm{cor}(y_{it}, y_{it'}) = 0$, $\forall t, t'$ with $t \neq t'$. It assumes that there is no correlation within the clusters and the model becomes equivalent to standard probit regression. The "working" correlation matrix is the identity matrix.
  - Fixed (`corstr = "fixed"`): If selected, the user must define the "working" correlation matrix with the `R` argument rather than estimating it from the model.

- Stationary m dependent (`corstr = "stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{|t-t'|} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

  If (`corstr = "stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. Choose this option when the correlations are assumed to be the same for observations of the same $|t-t'|$ periods apart for $|t-t'| \leq m$.

  Sample "working" correlation for Stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_1 & 1 & \alpha_1 & \alpha_2 & 0 \\ \alpha_2 & \alpha_1 & 1 & \alpha_1 & \alpha_2 \\ 0 & \alpha_2 & \alpha_1 & 1 & \alpha_1 \\ 0 & 0 & \alpha_2 & \alpha_1 & 1 \end{pmatrix}$$

- Non-stationary m dependent (`corstr = "non_stat_M_dep"`):

$$\text{cor}(y_{it}, y_{it'}) = \begin{cases} \alpha_{tt'} & \text{if} \quad |t-t'| \leq m \\ 0 & \text{if} \quad |t-t'| > m \end{cases}$$

  If (`corstr = "non_stat_M_dep"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. This option relaxes the assumption that the correlations are the same for all observations of the same $|t-t'|$ periods apart.

  Sample "working" correlation for Non-stationary 2 dependence (`Mv`=2)

$$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & 0 & 0 \\ \alpha_{12} & 1 & \alpha_{23} & \alpha_{24} & 0 \\ \alpha_{13} & \alpha_{23} & 1 & \alpha_{34} & \alpha_{35} \\ 0 & \alpha_{24} & \alpha_{34} & 1 & \alpha_{45} \\ 0 & 0 & \alpha_{35} & \alpha_{45} & 1 \end{pmatrix}$$

- Exchangeable (`corstr = "exchangeable"`): $\text{cor}(y_{it}, y_{it'}) = \alpha$, $\forall t, t'$ with $t \neq t'$. Choose this option if the correlations are assumed to be the same for all observations within the cluster.

  Sample "working" correlation for Exchangeable

$$\begin{pmatrix} 1 & \alpha & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & \alpha & 1 \end{pmatrix}$$

– Stationary mth order autoregressive (`corstr = "AR-M"`): If (`corstr = "AR-M"`), you must also specify `Mv` $= m$, where m is the number of periods t of dependence. For example, the first order autoregressive model (AR-1) implies $\text{cor}(y_{it}, y_{it'}) = \alpha^{|t-t'|}, \forall t, t'$ with $t \neq t'$. In AR-1, observation 1 and observation 2 have a correlation of α. Observation 2 and observation 3 also have a correlation of α. Observation 1 and observation 3 have a correlation of α^2, which is a function of how 1 and 2 are correlated (α) multiplied by how 2 and 3 are correlated (α). Observation 1 and 4 have a correlation that is a function of the correlation between 1 and 2, 2 and 3, and 3 and 4, and so forth.

Sample "working" correlation for Stationary AR-1 (`Mv`=1)

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha & \alpha^2 & \alpha^3 \\ \alpha^2 & \alpha & 1 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^2 & \alpha & 1 & \alpha \\ \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \end{pmatrix}$$

– Unstructured (`corstr = "unstructured"`): $\text{cor}(y_{it}, y_{it'}) = \alpha_{tt'}$, $\forall t, t'$ with $t \neq t'$. No constraints are placed on the correlations, which are then estimated from the data.

- `Mv`: defaults to 1. It specifies the number of periods of correlation and only needs to be specified when `corstr` is `"stat_M_dep"`, `"non_stat_M_dep"`, or `"AR-M"`.

- `R`: defaults to `NULL`. It specifies a user-defined correlation matrix rather than estimating it from the data. The argument is used only when `corstr` is `"fixed"`. The input is a $T \times T$ matrix of correlations, where T is the size of the largest cluster.

Examples

1. Example with Stationary 3 Dependence

   Attaching the sample turnout dataset:

```
> data(turnout)
```

   Variable identifying clusters

```
> turnout$cluster <- rep(c(1:200), 10)
```

   Sorting by cluster

```
> sorted.turnout <- turnout[order(turnout$cluster), ]
```

   Estimating parameter values:

```
> z.out1 <- zelig(vote ~ race + educate, model = "probit.gee",
+     id = "cluster", data = sorted.turnout, robust = TRUE, corstr = "stat_M_dep",
+     Mv = 3)
```

Setting values for the explanatory variables to their default values:

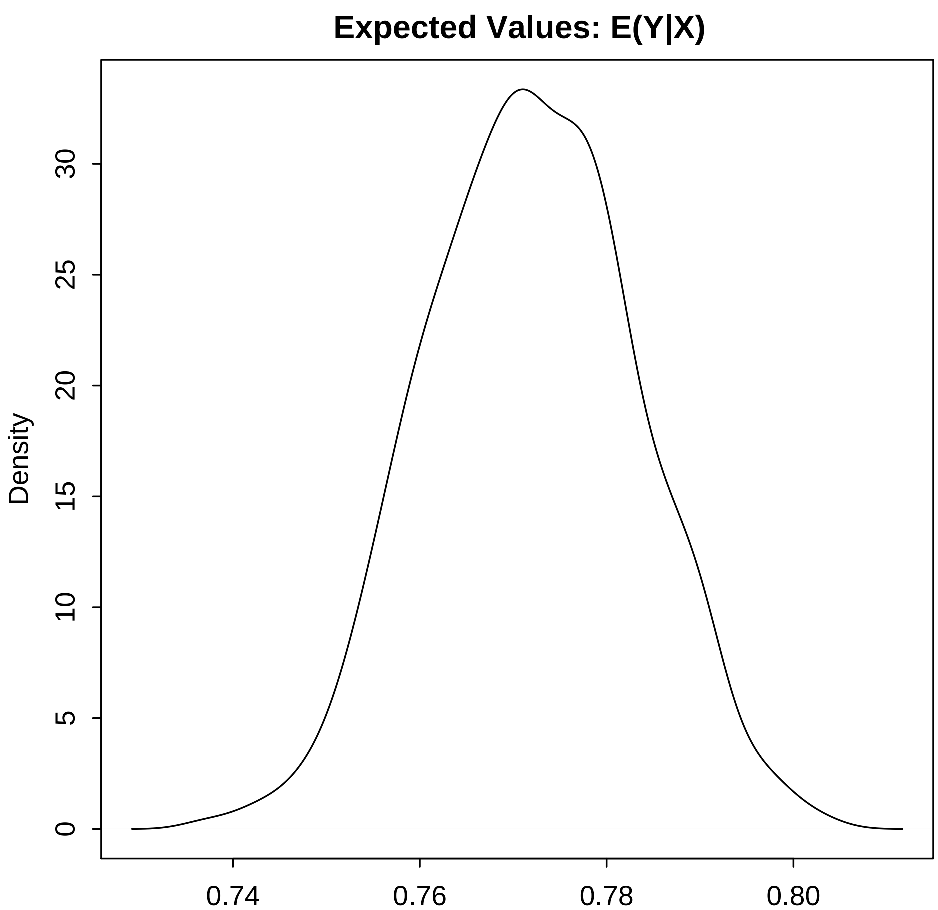
```
> x.out1 <- setx(z.out1)
```

Simulating quantities of interest:

```
> s.out1 <- sim(z.out1, x = x.out1)
```

```
> summary(s.out1)
```

```
> plot(s.out1)
```



2. Simulating First Differences

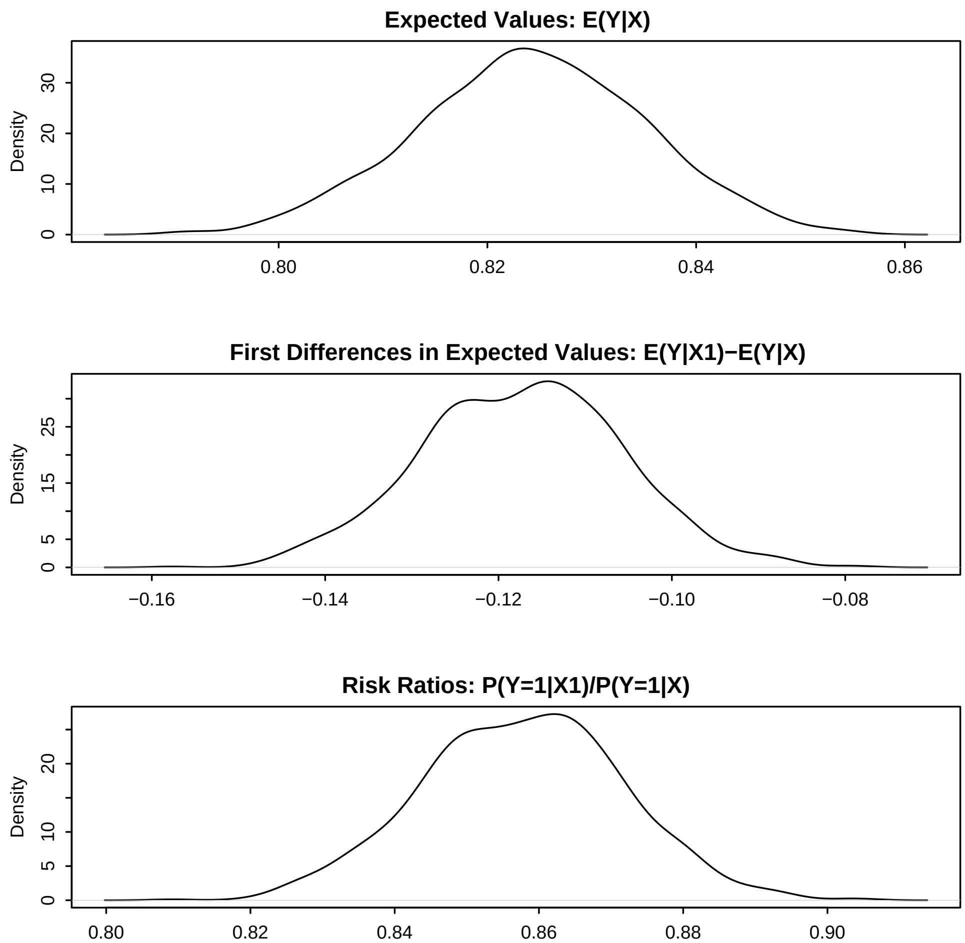
   Estimating the risk difference (and risk ratio) between low education (25th percentile) and high education (75th percentile) while all the other variables held at their default values.

```
> x.high <- setx(z.out1, educate = quantile(turnout$educate, prob = 0.75))
> x.low <- setx(z.out1, educate = quantile(turnout$educate, prob = 0.25))

> s.out2 <- sim(z.out1, x = x.high, x1 = x.low)

> summary(s.out2)

> plot(s.out2)
```



3. Example with Fixed Correlation Structure

   User-defined correlation structure

```
> corr.mat <- matrix(rep(0.5, 100), nrow = 10, ncol = 10)
> diag(corr.mat) <- 1
```

Generating empirical estimates:

```
> z.out2 <- zelig(vote ~ race + educate, model = "probit.gee",
+      id = "cluster", data = sorted.turnout, robust = TRUE,
+      corstr = "fixed", R = corr.mat)
```

Viewing the regression output:

```
> summary(z.out2)
```

The Model

Suppose we have a panel dataset, with Y_{it} denoting the binary dependent variable for unit i at time t. Y_i is a vector or cluster of correlated data where y_{it} is correlated with $y_{it'}$ for some or all t, t'. Note that the model assumes correlations within i but independence across i.

- The *stochastic component* is given by the joint and marginal distributions

$$\begin{aligned} Y_i &\sim f(y_i \mid \pi_i) \\ Y_{it} &\sim g(y_{it} \mid \pi_{it}) \end{aligned}$$

  where f and g are unspecified distributions with means π_i and π_{it}. GEE models make no distributional assumptions and only require three specifications: a mean function, a variance function, and a correlation structure.

- The *systematic component* is the *mean function*, given by:

$$\pi_{it} = \Phi(x_{it}\beta)$$

  where $\Phi(\mu)$ is the cumulative distribution function of the Normal distribution with mean 0 and unit variance, x_{it} is the vector of k explanatory variables for unit i at time t and β is the vector of coefficients.

- The *variance function* is given by:

$$V_{it} = \pi_{it}(1 - \pi_{it})$$

- The *correlation structure* is defined by a $T \times T$ "working" correlation matrix, where T is the size of the largest cluster. Users must specify the structure of the "working" correlation matrix *a priori*. The "working" correlation matrix then enters the variance term for each i, given by:

$$V_i = \phi \, A_i^{\frac{1}{2}} R_i(\alpha) A_i^{\frac{1}{2}}$$

  where A_i is a $T \times T$ diagonal matrix with the variance function $V_{it} = \pi_{it}(1 - \pi_{it})$ as the tth diagonal element, $R_i(\alpha)$ is the "working" correlation matrix, and ϕ is a scale parameter. The parameters are then estimated via a quasi-likelihood approach.

- In GEE models, if the mean is correctly specified, but the variance and correlation structure are incorrectly specified, then GEE models provide consistent estimates of the parameters and thus the mean function as well, while consistent estimates of the standard errors can be obtained via a robust "sandwich" estimator. Similarly, if the mean and variance are correctly specified but the correlation structure is incorrectly specified, the parameters can be estimated consistently and the standard errors can be estimated consistently with the sandwich estimator. If all three are specified correctly, then the estimates of the parameters are more efficient.

- The robust "sandwich" estimator gives consistent estimates of the standard errors when the correlations are specified incorrectly only if the number of units i is relatively large and the number of repeated periods t is relatively small. Otherwise, one should use the "naïve" model-based standard errors, which assume that the specified correlations are close approximations to the true underlying correlations. See **?** for more details.

Quantities of Interest

- All quantities of interest are for marginal means rather than joint means.

- The method of bootstrapping generally should not be used in GEE models. If you must bootstrap, bootstrapping should be done within clusters, which is not currently supported in Zelig. For conditional prediction models, data should be matched within clusters.

- The expected values (`qi$ev`) for the GEE probit model are simulations of the predicted probability of a success:
$$E(Y) = \pi_c = \Phi(x_c \beta),$$
given draws of β from its sampling distribution, where x_c is a vector of values, one for each independent variable, chosen by the user.

- The first difference (`qi$fd`) for the GEE probit model is defined as
$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as
$$\text{RR} = \Pr(Y = 1 \mid x_1) \ / \ \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is
$$\frac{1}{\sum_{i=1}^{n} \sum_{t=1}^{T} tr_{it}} \sum_{i:tr_{it}=1}^{n} \sum_{t:tr_{it}=1}^{T} \left\{ Y_{it}(tr_{it} = 1) - E[Y_{it}(tr_{it} = 0)] \right\},$$

where tr_{it} is a binary explanatory variable defining the treatment ($tr_{it} = 1$) and control ($tr_{it} = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_{it}(tr_{it} = 0)]$, the counterfactual expected value of Y_{it} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $tr_{it} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "probit.gee", id, data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_{it}.
  - `linear.predictors`: the vector of $x_{it}\beta$
  - `max.id`: the size of the largest cluster.
- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and z-statistics.
  - `working.correlation`: the "working" correlation matrix
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How To Cite

To cite the *probit.gee* Zelig model:

> Patrick Lam. 2007. "probit.gee: Generalized Estimating Equation for Probit Regression," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

See also

The gee function is part of the gee package by Vincent J. Carey, ported to R by Thomas Lumley and Brian Ripley. Advanced users may wish to refer to `help(gee)` and `help(family)`. Sample data are from King et al. (2000).

12.59 `probit.mixed`: Mixed effects probit Regression

Use generalized multi-level linear regression if you have covariates that are grouped according to one or more classification factors. The probit model is appropriate when the dependent variable is dichotomous.

While generally called multi-level models in the social sciences, this class of models is often referred to as mixed-effects models in the statistics literature and as hierarchical models in a Bayesian setting. This general class of models consists of linear models that are expressed as a function of both *fixed effects*, parameters corresponding to an entire population or certain repeatable levels of experimental factors, and *random effects*, parameters corresponding to individual experimental units drawn at random from a population.

Syntax

```
z.out <- zelig(formula= y ~ x1 + x2 + tag(z1 + z2 | g),
               data=mydata, model="probit.mixed")

z.out <- zelig(formula= list(mu=y ~ xl + x2 + tag(z1, gamma | g),
               gamma= ~ tag(w1 + w2 | g)), data=mydata, model="probit.mixed")
```

Inputs

`zelig()` takes the following arguments for `mixed`:

- `formula`: a two-sided linear formula object describing the systematic component of the model, with the response on the left of a ˜ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1 + ... + zn | g)` with `z1 + ... + zn` specifying the model for the random effects and `g` the grouping structure. Random intercept terms are included with the notation `tag(1 | g)`.
  Alternatively, `formula` may be a list where the first entry, `mu`, is a two-sided linear formula object describing the systematic component of the model, with the repsonse on the left of a ˜ operator and the fixed effects terms, separated by `+` operators, on the right. Any random effects terms are included with the notation `tag(z1, gamma | g)` with `z1` specifying the individual level model for the random effects, `g` the grouping structure and `gamma` references the second equation in the list. The `gamma` equation is one-sided linear formula object with the group level model for the random effects on the right side of a ˜ operator. The model is specified with the notation `tag(w1 + ... + wn | g)` with `w1 + ... + wn` specifying the group level model and `g` the grouping structure.

Additional Inputs

In addition, `zelig()` accepts the following additional arguments for model specification:

- `data`: An optional data frame containing the variables named in `formula`. By default, the variables are taken from the environment from which `zelig()` is called.
- `method`: a character string. The criterion is always the log-likelihood but this criterion does not have a closed form expression and must be approximated. The default approximation is `"PQL"` or penalized quasi-likelihood. Alternatives are `"Laplace"` or `"AGQ"` indicating the Laplacian and adaptive Gaussian quadrature approximations respectively.
- `na.action`: A function that indicates what should happen when the data contain `NA`s. The default action (`na.fail`) causes `zelig()` to print an error message and terminate if there are any incomplete observations.

Additionally, users may with to refer to `lmer` in the package `lme4` for more information, including control parameters for the estimation algorithm and their defaults.

Examples

1. Basic Example with First Differences

   Attach sample data:

```
> data(voteincome)
```

   Estimate model:

```
> z.out1 <- zelig(vote ~ education + age + female + tag(1 |
+     state), data = voteincome, model = "probit.mixed")
```

   Summarize regression coefficients and estimated variance of random effects:

```
> summary(z.out1)
```

   Set explanatory variables to their default values, with high (80th percentile) and low (20th percentile) values for education:

```
> x.high <- setx(z.out1, education = quantile(voteincome$education,
+     0.8))
> x.low <- setx(z.out1, education = quantile(voteincome$education,
+     0.2))
```

   Generate first differences for the effect of high versus low education on voting:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
> summary(s.out1)
```

Mixed effects probit Regression Model

Let Y_{ij} be the binary dependent variable, realized for observation j in group i as y_{ij} which takes the value of either 0 or 1, for $i = 1, \ldots, M$, $j = 1, \ldots, n_i$.

- The *stochastic component* is described by a Bernoulli distribution with mean vector π_{ij}.
$$Y_{ij} \sim \text{Bernoulli}(y_{ij}|\pi_{ij}) = \pi_{ij}^{y_{ij}}(1 - \pi_{ij})^{1-y_{ij}}$$
where
$$\pi_{ij} = \Pr(Y_{ij} = 1)$$

- The q-dimensional vector of *random effects*, b_i, is restricted to be mean zero, and therefore is completely characterized by the variance covarance matrix Ψ, a $(q \times q)$ symmetric positive semi-definite matrix.
$$b_i \sim Normal(0, \Psi)$$

- The *systematic component* is
$$\pi_{ij} \equiv \Phi(X_{ij}\beta + Z_{ij}b_i)$$
where $\Phi(\mu)$ is the cumulative distribution function of the Normal distribution with mean 0 and unit variance, and
where X_{ij} is the $(n_i \times p \times M)$ array of known fixed effects explanatory variables, β is the p-dimensional vector of fixed effects coefficients, Z_{ij} is the $(n_i \times q \times M)$ array of known random effects explanatory variables and b_i is the q-dimensional vector of random effects.

Quantities of Interest

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value, π_{ij} for
$$\pi_{ij} = \Phi(X_{ij}\beta + Z_{ij}b_i)$$
given X_{ij} and Z_{ij} and simulations of of β and b_i from their posterior distributions. The estimated variance covariance matrices are taken as correct and are themselves not simulated.

- The expected values (`qi$ev`) are simulations of the predicted probability of a success given draws of β from its posterior:
$$E(Y_{ij}|X_{ij}) = \pi_{ij} = \Phi(X_{ij}\beta).$$

- The first difference (`qi$fd`) is given by the difference in predicted probabilities, conditional on X_{ij} and X'_{ij}, representing different values of the explanatory variables.

$$FD(Y_{ij}|X_{ij}, X'_{ij}) = Pr(Y_{ij} = 1|X_{ij}) - Pr(Y_{ij} = 1|X'_{ij})$$

- The risk ratio (`qi$rr`) is defined as

$$RR(Y_{ij}|X_{ij}, X'_{ij}) = \frac{Pr(Y_{ij} = 1|X_{ij})}{Pr(Y_{ij} = 1|X'_{ij})}$$

- In conditional prediction models, the average predicted treatment effect (`qi$att.pr`) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij} = 1) - \widehat{Y_{ij}(t_{ij} = 0)}\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $Y_{ij}(t_{ij} = 0)$, the counterfactual predicted value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is given by

$$\frac{1}{\sum_{i=1}^{M}\sum_{j=1}^{n_i} t_{ij}} \sum_{i=1}^{M} \sum_{j:t_{ij}=1}^{n_i} \{Y_{ij}(t_{ij} = 1) - E[Y_{ij}(t_{ij} = 0)]\},$$

  where t_{ij} is a binary explanatory variable defining the treatment ($t_{ij} = 1$) and control ($t_{ij} = 0$) groups. Variation in the simulations is due to uncertainty in simulating $E[Y_{ij}(t_{ij} = 0)]$, the counterfactual expected value of Y_{ij} for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_{ij} = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. You may examine the available information in `z.out` by using `slotNames(z.out)`, see the fixed effect coefficients by using `summary(z.out)@coefs`, and a default summary of information through `summary(z.out)`. Other elements available through the operator are listed below.

- From the `zelig()` output stored in `summary(z.out)`, you may extract:
  - `fixef`: numeric vector containing the conditional estimates of the fixed effects.

- `ranef`: numeric vector containing the conditional modes of the random effects.
- `frame`: the model frame for the model.

- From the `sim()` output stored in `s.out`, you may extract quantities of interest stored in a data frame:
  - `qi$pr`: the simulated predicted values drawn from the distributions defined by the expected values.
  - `qi$ev`: the simulated expected values for the specified values of x.
  - `qi$fd`: the simulated first differences in the expected values for the values specified in x and x1.
  - `qi$ate.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.
  - `qi$ate.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *probit.mixed* Zelig model:

> Delia Bailey, Ferdinand Alimadhi. 2007. "probit.mixed: Mixed effects probit regression" in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Mixed effects probit regression is part of `lme4` package by Douglas M. Bates (Bates 2007). For a detailed discussion of mixed-effects models, please see **?**

12.60 probit.net: Network Probit Regression for Dichotomous Proximity Matrix Dependent Variables

Use network probit regression analysis for a dependent variable that is a binary valued proximity matrix (a.k.a. sociomatricies, adjacency matrices, or matrix representations of directed graphs).

Syntax

```
> z.out <- zelig(y ~ x1 + x2, model = "probit.net", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Examples

1. Basic Example

   Load the sample data (see ?friendship for details on the structure of the network dataframe):

   ```
   > data(friendship)
   ```

   Estimate model:

   ```
   > z.out <- zelig(friends ~ advice + prestige + perpower, model = "probit.net",
   +     data = friendship)
   > summary(z.out)
   ```

   Setting values for the explanatory variables to their default values:

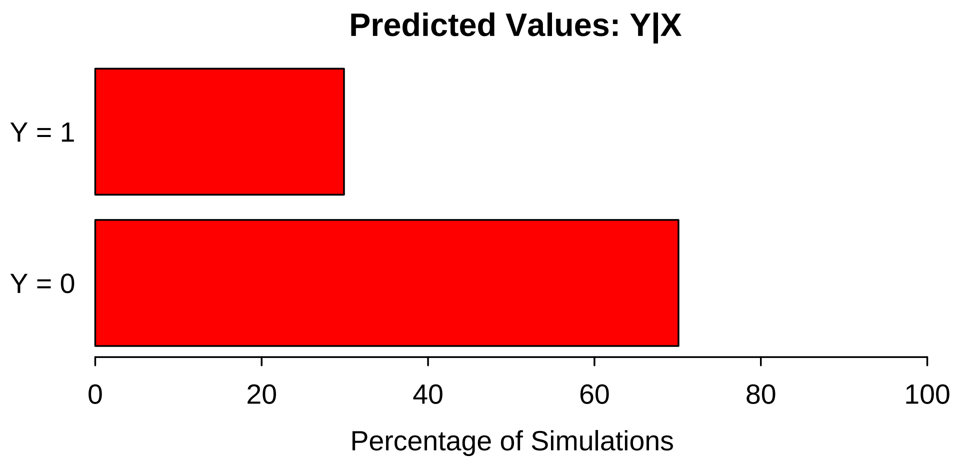
   ```
   > x.out <- setx(z.out)
   ```

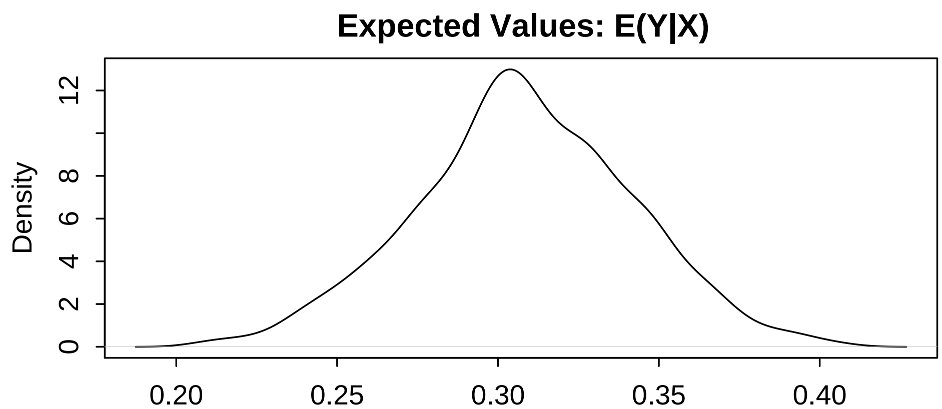
   Simulating quantities of interest from the posterior distribution.

   ```
   > s.out <- sim(z.out, x = x.out)
   > summary(s.out)
   > plot(s.out)
   ```

2. Simulating First Differences

   Estimating the risk difference (and risk ratio) between low personal power (25th percentile) and high personal power (75th percentile) while all the other variables are held at their default values.

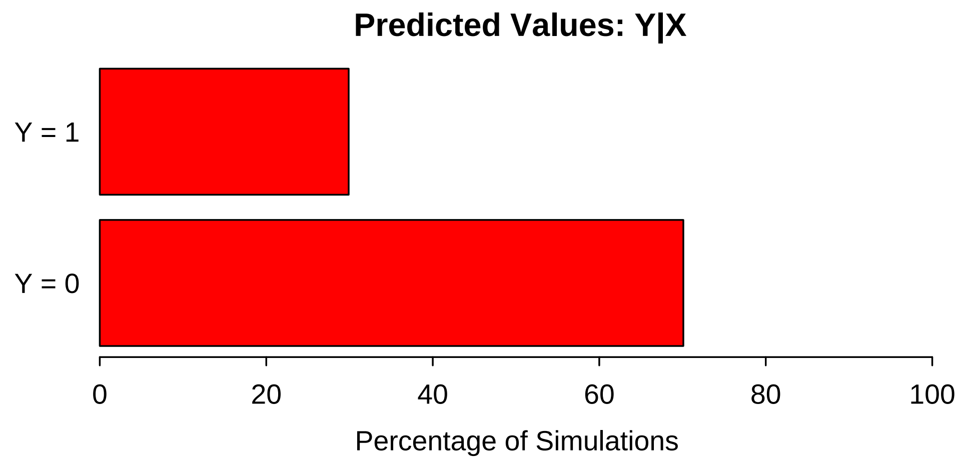


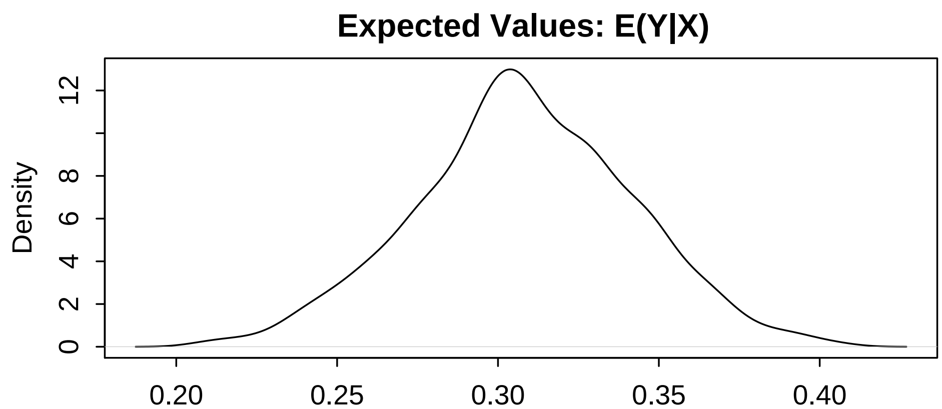



```
> x.high <- setx(z.out, perpower = quantile(friendship$perpower,
+     prob = 0.75))
> x.low <- setx(z.out, perpower = quantile(friendship$perpower,
+     prob = 0.25))
> s.out2 <- sim(z.out, x = x.high, x1 = x.low)
> summary(s.out2)
> plot(s.out2)
```

Model

The `probit.net` model performs a probit regression of the proximity matrix $\mathbf{Y}$, a $m \times m$ matrix representing network ties, on a set of proximity matrices $\mathbf{X}$. This network regression model is directly analogous to standard probit regression element-wise on the appropriately vectorized matrices. Proximity matrices are vectorized by creating Y, a $m^2 \times 1$ vector to represent the proximity matrix. The vectorization which produces the Y vector from the $\mathbf{Y}$





matrix is performed by simple row-concatenation of $\mathbf{Y}$. For example, if $\mathbf{Y}$ is a 15×15 matrix, the $\mathbf{Y}_{1,1}$ element is the first element of Y, and the $\mathbf{Y}_{2,1}$ element is the second element of Y and so on. Once the input matrices are vectorized, standard probit regression is performed.

Let Y_i be the binary dependent variable, produced by vectorizing a binary proximity matrix, for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$Y_i \sim \text{Bernoulli}(\pi_i)$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is given by:

$$\pi_i = \mathbf{\Phi}(x_i \beta).$$

  where $\mathbf{\Phi}(\mu)$ is the cumulative distribution function of the Normal distribution with mean 0 and unit variance.

Quantities of Interest

The quantities of interest for the network probit regression are the same as those for the standard probit regression.

- The expected values (`qi$ev`) for the `probit.net` model are simulations of the predicted probability of a success:
  $$E(Y) = \pi_i = \mathbf{\Phi}(x_i \beta),$$
  given draws of β from its sampling distribution.

- The predicted values (`qi$pr`) are draws from the Binomial distribution with mean equal to the simulated expected value π_i.

- The first difference (`qi$fd`) for the network probit model is defined as
  $$FD = \Pr(Y = 1|x_1) - \Pr(Y = 1|x)$$

Output Values

The output of each Zelig command contains useful information which you may view. For example, you run `z.out <- zelig(y ~ x, model = "probit.net", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the coefficients by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output stored in `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `fitted.values`: the vector of fitted values for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `linear.predictors`: the vector of $x_i \beta$.
  - `aic`: Akaikeś Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `bic`: the Bayesian Information Criterion (minus twice the maximized log-likelihood plus the number of coefficients times log n).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`

- From `summary(z.out)`(as well as from `zelig()`), you may extract:
  - `mod.coefficients`: the parameter estimates with their associated standard errors, p-values, and t statistics.

– `cov.scaled`: a $k \times k$ matrix of scaled covariances.
– `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output stored in `s.out`, you may extract:
  – `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  – `qi$pr`: the simulated predicted values for the specified values of `x`.
  – `qi$fd`: the simulated first differences in the expected probabilities simulated from `x` and `x1`.

How to Cite

To cite the *probit.net* Zelig model:

> Skyler J. Cranmer. 2007. "probit.net: Network Probit Regression for Dichotomous Proximity Matrix Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The network probit regression is part of the `netglm` package by Skyler J. Cranmer and is built using some of the functionality of the `sna` package by Carter T. Butts (Butts and Carley 2001).In addition, advanced users may wish to refer to `help(netpoisson)`. Sample data are fictional.

12.61 `probit.survey`: Survey-Weighted Probit Regression for Dichotomous Dependent Variables

The survey-weighted probit regression model is appropriate for survey data obtained using complex sampling techniques, such as stratified random or cluster sampling (e.g., not simple random sampling). Like the conventional probit regression models (see Section 12.55), survey-weighted probit regression specifies a dichotomous dependent variable as function of a set of explanatory variables. The survey-weighted probit model reports estimates of model parameters identical to conventional probit estimates, but uses information from the survey design to correct variance estimates.

The `probit.survey` model accommodates three common types of complex survey data. Each method listed here requires selecting specific options which are detailed in the "Additional Inputs" section below.

1. **Survey weights**: Survey data are often published along with weights for each observation. For example, if a survey intentionally over-samples a particular type of case, weights can be used to correct for the over-representation of that type of case in the dataset. Survey weights come in two forms:

   (a) *Probability* weights report the probability that each case is drawn from the population. For each stratum or cluster, this is computed as the number of observations in the sample drawn from that group divided by the number of observations in the population in the group.

   (b) *Sampling* weights are the inverse of the probability weights.

2. **Strata/cluster identification**: A complex survey dataset may include variables that identify the strata or cluster from which observations are drawn. For stratified random sampling designs, observations may be nested in different strata. There are two ways to employ these identifiers:

   (a) Use *finite population corrections* to specify the total number of cases in the stratum or cluster from which each observation was drawn.

   (b) For stratified random sampling designs, use the raw strata ids to compute sampling weights from the data.

3. **Replication weights**: To preserve the anonymity of survey participants, some surveys exclude strata and cluster ids from the public data and instead release only precomputed replicate weights.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "probit.survey", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Additional Inputs

In addition to the standard `zelig` inputs (see Section **??**), survey-weighted probit models accept the following optional inputs:

1. Datasets that include survey weights:.

    - `probs`: An optional formula or numerical vector specifying each case's probability weight, the probability that the case was selected. Probability weights need not (and, in most cases, will not) sum to one. Cases with lower probability weights are weighted more heavily in the computation of model coefficients.
    - `weights`: An optional numerical vector specifying each case's sample weight, the inverse of the probability that the case was selected. Sampling weights need not (and, in most cases, will not) sum to one. Cases with higher sampling weights are weighted more heavily in the computation of model coefficients.

2. Datasets that include strata/cluster identifiers:

    - `ids`: An optional formula or numerical vector identifying the cluster from which each observation was drawn (ordered from largest level to smallest level). For survey designs that do not involve cluster sampling, `ids` defaults to `NULL`.
    - `fpc`: An optional numerical vector identifying each case's frequency weight, the total number of units in the population from which each observation was sampled.
    - `strata`: An optional formula or vector identifying the stratum from which each observation was sampled. Entries may be numerical, logical, or strings. For survey designs that do not involve cluster sampling, `strata` defaults to `NULL`.
    - `nest`: An optional logical value specifying whether primary sampling unites (PSUs) have non-unique ids across multiple strata. `nest=TRUE` is appropriate when PSUs reuse the same identifiers across strata. Otherwise, `nest` defaults to `FALSE`.
    - `check.strata`: An optional input specifying whether to check that clusters are nested in strata. If `check.strata` is left at its default, `!nest`, the check is not performed. If `check.strata` is specified as `TRUE`, the check is carried out.

3. Datasets that include replication weights:

    - `repweights`: A formula or matrix specifying replication weights, numerical vectors of weights used in a process in which the sample is repeatedly re-weighted and parameters are re-estimated in order to compute the variance of the model parameters.
    - `type`: A string specifying the type of replication weights being used. This input is required if replicate weights are specified. The following types of replication weights are recognized: `"BRR"`, `"Fay"`, `"JK1"`, `"JKn"`, `"bootstrap"`, or `"other"`.

- `weights`: An optional vector or formula specifying each case's sample weight, the inverse of the probability that the case was selected. If a survey includes both sampling weights and replicate weights separately for the same survey, both should be included in the model specification. In these cases, sampling weights are used to correct potential biases in in the computation of coefficients and replication weights are used to compute the variance of coefficient estimates.
- `combined.weights`: An optional logical value that should be specified as `TRUE` if the replicate weights include the sampling weights. Otherwise, `combined.weights` defaults to `FALSE`.
- `rho`: An optional numerical value specifying a shrinkage factor for replicate weights of type `"Fay"`.
- `bootstrap.average`: An optional numerical input specifying the number of iterations over which replicate weights of type `"bootstrap"` were averaged. This input should be left as `NULL` for `"bootstrap"` weights that were not were created by averaging.
- `scale`: When replicate weights are included, the variance is computed as the sum of squared deviations of the replicates from their mean. `scale` is an optional overall multiplier for the standard deviations.
- `rscale`: Like `scale`, `rscale` specifies an optional vector of replicate-specific multipliers for the squared deviations used in variance computation.
- `fpc`: For models in which `"JK1"`, `"JKn"`, or `"other"` replicates are specified, `fpc` is an optional numerical vector (with one entry for each replicate) designating the replicates' finite population corrections.
- `fpctype`: When a finite population correction is included as an `fpc` input, `fpctype` is a required input specifying whether the input to `fpc` is a sampling fraction (`fpctype="fraction"`) or a direct correction (`fpctype="correction"`).
- `return.replicates`: An optional logical value specifying whether the replicates should be returned as a component of the output. `return.replicates` defaults to `FALSE`.

Examples

1. A dataset that includes survey weights:

   Attach the sample data:

   ```
   > data(api, package = "survey")
   ```

   Suppose that a dataset included a dichotomous indicator for whether each public school attends classes year round (`yr.rnd`), a measure of the percentage of students at each school who receive subsidized meals (`meals`), a measure of the percentage of students at each school who are new to to the school (`mobility`), and sampling weights computed by the survey house (`pw`). Estimate a model that regresses the year-round schooling indicator on the `meals` and `mobility` variables:

```
> z.out1 <- zelig(yr.rnd ~ meals + mobility, model = "probit.survey",
+     weights = ~pw, data = apistrat)
```

Summarize regression coefficients:

```
> summary(z.out1)
```

Set explanatory variables to their default (mean/mode) values, and set a high (80th percentile) and low (20th percentile) value for "meals":

```
> x.low <- setx(z.out1, meals = quantile(apistrat$meals, 0.2))
> x.high <- setx(z.out1, meals = quantile(apistrat$meals, 0.8))
```

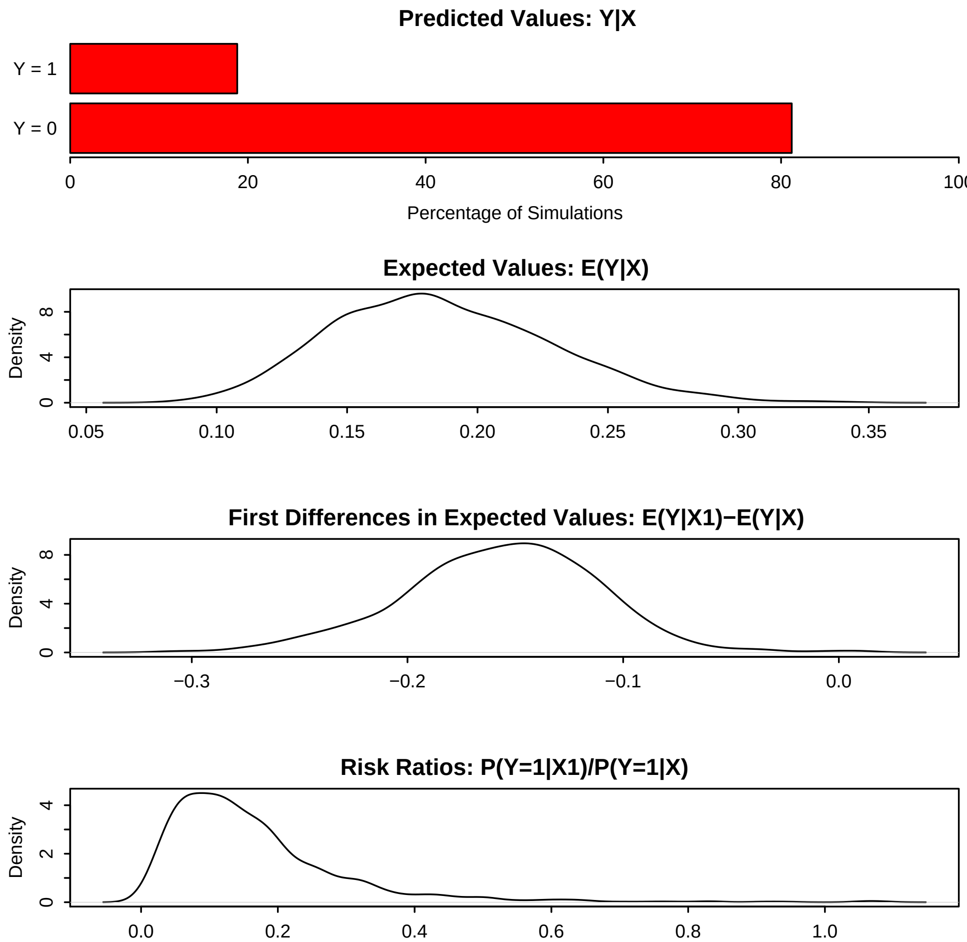
Generate first differences for the effect of high versus low concentrations of children receiving subsidized meals on the probability of holding school year-round:

```
> s.out1 <- sim(z.out1, x = x.high, x1 = x.low)
```

```
> summary(s.out1)
```

Generate a visual summary of the quantities of interest:

```
> plot(s.out1)
```



2. A dataset that includes strata/cluster identifiers:

   Suppose that the survey house that provided the dataset used in the previous example excluded sampling weights but made other details about the survey design available. A user can still estimate a model without sampling weights that instead uses inputs that identify the stratum and/or cluster to which each observation belongs and the size of the finite population from which each observation was drawn.

   Continuing the example above, suppose the survey house drew at random a fixed number of elementary schools, a fixed number of middle schools, and a fixed number of high schools. If the variable `stype` is a vector of characters (`"E"` for elementary schools, `"M"` for middle schools, and `"H"` for high schools) that identifies the type of school each case represents and `fpc` is a numerical vector that identifies for each case the total number of schools of the same type in the population, then the user could estimate the following model:

```
> z.out2 <- zelig(yr.rnd ~ meals + mobility, model = "probit.survey",
+     strata = ~stype, fpc = ~fpc, data = apistrat)
```

Summarize the regression output:

```
> summary(z.out2)
```

The coefficient estimates for this example are identical to the point estimates in the first example, when pre-existing sampling weights were used. When sampling weights are omitted, they are estimated automatically for "probit.survey" models based on the user-defined description of sampling designs.

Moreover, because the user provided information about the survey design, the standard error estimates are lower in this example than in the previous example, in which the user omitted variables pertaining to the details of the complex survey design.

3. A dataset that includes replication weights:

   Consider a dataset that includes information for a sample of hospitals about the number of out-of-hospital cardiac arrests that each hospital treats and the number of patients who arrive alive at each hospital:

   ```
   > data(scd, package = "survey")
   ```

   Survey houses sometimes supply replicate weights (in lieu of details about the survey design). For the sake of illustrating how replicate weights can be used as inputs in probit.survey models, create a set of balanced repeated replicate (BRR) weights and an (artificial) dependent variable to simulate an indicator for whether each hospital was sued:

   ```
   > BRRrep <- 2 * cbind(c(1, 0, 1, 0, 1, 0), c(1, 0, 0, 1, 0,
   +     1), c(0, 1, 1, 0, 0, 1), c(0, 1, 0, 1, 1, 0))
   > scd$sued <- as.vector(c(0, 0, 0, 1, 1, 1))
   ```

   Estimate a model that regresses the indicator for hospitals that were sued on the number of patients who arrive alive in each hospital and the number of cardiac arrests that each hospital treats, using the BRR replicate weights in BRRrep to compute standard errors.

   ```
   > z.out3 <- zelig(formula = sued ~ arrests + alive, model = "probit.survey",
   +     repweights = BRRrep, type = "BRR", data = scd)
   ```

   Summarize the regression coefficients:

   ```
   > summary(z.out3)
   ```

   Set alive at its mean and set arrests at its 20th and 80th quantiles:

   ```
   > x.low <- setx(z.out3, arrests = quantile(scd$arrests, 0.2))
   > x.high <- setx(z.out3, arrests = quantile(scd$arrests, 0.8))
   ```

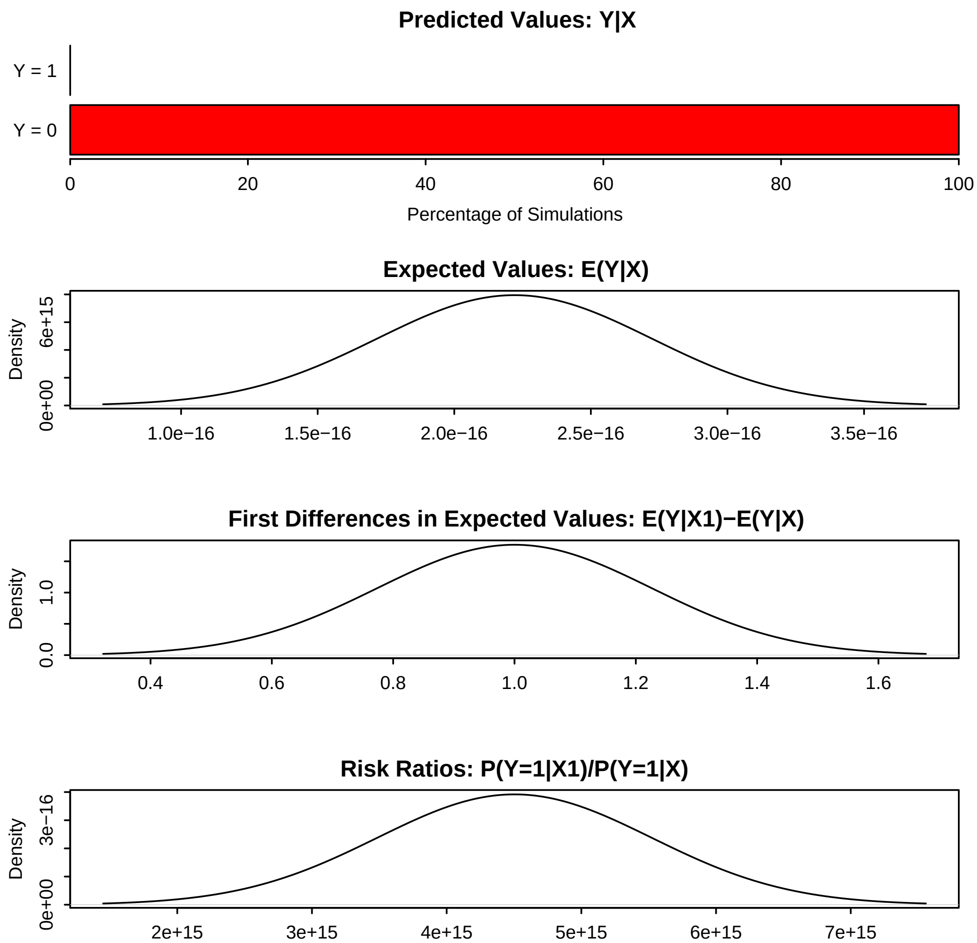
Generate first differences for the effect of high versus low cardiac arrests on the probability that a hospital will be sued:

```
> s.out3 <- sim(z.out3, x = x.high, x1 = x.low)
```

```
> summary(s.out3)
```

Generate a visual summary of quantities of interest:

```
> plot(s.out3)
```



Model

Let Y_i be the observed binary dependent variable for observation i which takes the value of either 0 or 1.

- The *stochastic component* is given by

$$Y_i \sim \text{Bernoulli}(\pi_i),$$

  where $\pi_i = \Pr(Y_i = 1)$.

- The *systematic component* is

$$\pi_i = \Phi(x_i \beta)$$

  where $\Phi(\mu)$ is the cumulative distribution function of the Normal distribution with mean 0 and unit variance.

Variance

When replicate weights are not used, the variance of the coefficients is estimated as

$$\hat{\mathbf{\Sigma}} \left[\sum_{i=1}^{n} \frac{(1-\pi_i)}{\pi_i^2} (\mathbf{X}_i (Y_i - \mu_i))' (\mathbf{X}_i (Y_i - \mu_i)) + 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{(\pi_{ij} - \pi_i \pi_j)}{\pi_i \pi_j \pi_{ij}} (\mathbf{X}_i (Y_i - \mu_i))' (\mathbf{X}_j (Y_j - \mu_j)) \right] \hat{\mathbf{\Sigma}}$$

where π_i is the probability of case i being sampled, $\mathbf{X}_i$ is a vector of the values of the explanatory variables for case i, Y_i is value of the dependent variable for case i, $\hat{\mu}_i$ is the predicted value of the dependent variable for case i based on the linear model estimates, and $\hat{\mathbf{\Sigma}}$ is the conventional variance-covariance matrix in a parametric glm. This statistic is derived from the method for estimating the variance of sums described in Binder (1983) and the Horvitz-Thompson estimator of the variance of a sum described in Horvitz and Thompson (1952).

When replicate weights are used, the model is re-estimated for each set of replicate weights, and the variance of each parameter is estimated by summing the squared deviations of the replicates from their mean.

Quantities of Interest

- The expected value (`qi$ev`) is a simulation of predicted probability of success

$$E(Y) = \pi_i = \Phi(x_i \beta),$$

  given a draw of β from its sampling distribution.

- The predicted value (`qi$pr`) is a draw from a Bernoulli distribution with mean π_i.

- The first difference (`qi$fd`) in expected values is defined as

$$\text{FD} = \Pr(Y = 1 \mid x_1) - \Pr(Y = 1 \mid x).$$

- The risk ratio (`qi$rr`) is defined as

$$\text{RR} = \Pr(Y = 1 \mid x_1) / \Pr(Y = 1 \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \{Y_i(t_i = 1) - E[Y_i(t_i = 0)]\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)}\right\},$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "probit.survey", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_i.
  - `linear.predictors`: the vector of $x_i \beta$
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.

  - `data`: the name of the input data frame.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.
  - `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected probabilities for the specified values of `x`.
  - `qi$pr`: the simulated predicted values for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the expected probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

When users estimate `probit.survey` models with replicate weights in `Zelig`, an object called `.survey.prob.weights` is created in the global environment. `Zelig` will over-write any existing object with that name, and users are therefore advised to re-name any object called `.survey.prob.weights` before using `probit.survey` models in `Zelig`.

How to Cite

To cite the *probit.survey* Zelig model:

> Nicholas Carnes. 2008. "probit.survey: Survey-weighted Probit Regression for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Survey-weighted linear models and the sample data used in the examples above are a part of the `survey` package by Thomas Lumley. Users may wish to refer to the help files for the three functions that Zelig draws upon when estimating survey-weighted models, namely, `help(svyglm)`, `help(svydesign)`, and `help(svrepdesign)`. The probit model is part of the stats package by Venables and Ripley (2002). Advanced users may wish to refer to `help(glm)` and `help(family)`, as well as McCullagh and Nelder (1989).

12.62 `relogit`: Rare Events Logistic Regression for Dichotomous Dependent Variables

The `relogit` procedure estimates the same model as standard logistic regression (appropriate when you have a dichotomous dependent variable and a set of explanatory variables; see Section 12.22), but the estimates are corrected for the bias that occurs when the sample is small or the observed events are rare (i.e., if the dependent variable has many more 1s than 0s or the reverse). The `relogit` procedure also optionally uses prior correction for case-control sampling designs.

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, model = "relogit", tau = NULL,
                 case.correct = c("prior", "weighting"),
                 bias.correct = TRUE, robust = FALSE,
                 data = mydata, ...)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Arguments

The `relogit` procedure supports four optional arguments in addition to the standard arguments for `zelig()`. You may additionally use:

- `tau`: a vector containing either one or two values for τ, the true population fraction of ones. Use, for example, `tau = c(0.05, 0.1)` to specify that the lower bound on `tau` is 0.05 and the upper bound is 0.1. If left unspecified, only finite-sample bias correction is performed, not case-control correction.

- `case.correct`: if `tau` is specified, choose a method to correct for case-control sampling design: `"prior"` (default) or `"weighting"`.

- `bias.correct`: a logical value of `TRUE` (default) or `FALSE` indicating whether the intercept should be corrected for finite sample (rare events) bias.

- `robust`: defaults to `FALSE` (except when `case.control = "weighting"`; the default in this case becomes `robust = TRUE`). If `TRUE` is selected, `zelig()` computes robust standard errors via the `sandwich` package (see Zeileis (2004)). The default type of robust standard error is heteroskedastic and autocorrelation consistent (HAC), and assumes that observations are ordered by time index.

  In addition, `robust` may be a list with the following options:

  - `method`: Choose from
    * `"vcovHAC"`: (default if `robust = TRUE`) HAC standard errors.

    * "kernHAC": HAC standard errors using the weights given in Andrews (1991).
    * "weave": HAC standard errors using the weights given in Lumley and Heagerty (1999).
  - order.by: defaults to NULL (the observations are chronologically ordered as in the original data). Optionally, you may specify a vector of weights (either as order.by = z, where z exists outside the data frame; or as order.by = ~z, where z is a variable in the data frame) The observations are chronologically ordered by the size of z.
  - ...: additional options passed to the functions specified in method. See the sandwich library and Zeileis (2004) for more options.

Note that if tau = NULL, bias.correct = FALSE, robust = FALSE, the relogit procedure performs a standard logistic regression without any correction.

Example 1: One Tau with Prior Correction and Bias Correction

Due to memory and space considerations, the data used here are a sample drawn from the full data set used in King and Zeng, 2001, The proportion of militarized interstate conflicts to the absence of disputes is $\tau = 1,042/303,772 \approx 0.00343$. To estimate the model,

```
> data(mid)

> z.out1 <- zelig(conflict ~ major + contig + power + maxdem +
+     mindem + years, data = mid, model = "relogit", tau = 1042/303772)
```

Summarize the model output:

```
> summary(z.out1)
```

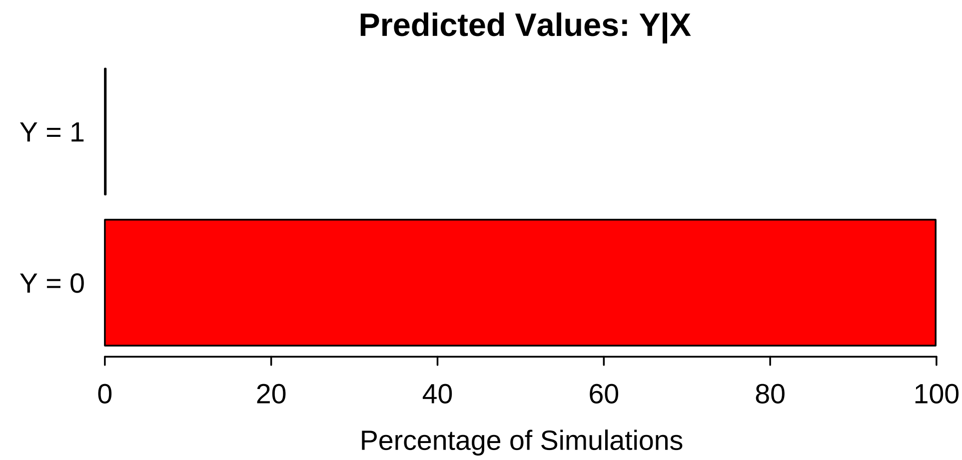
Set the explanatory variables to their means:

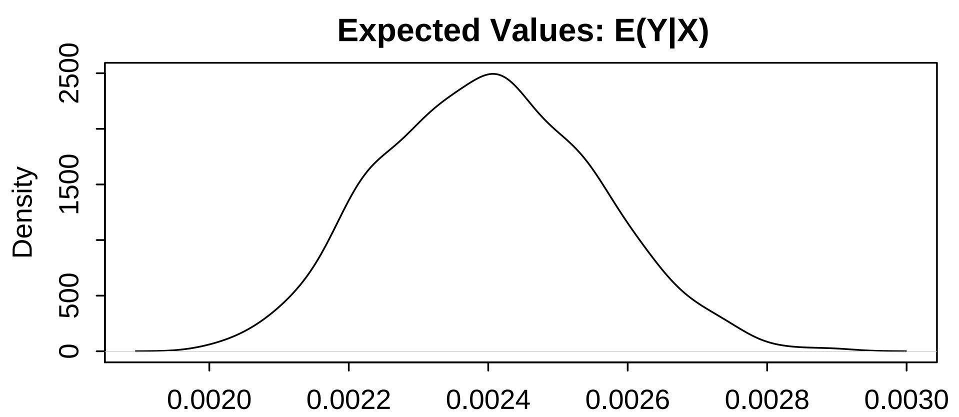
```
> x.out1 <- setx(z.out1)
```

Simulate quantities of interest:

```
> s.out1 <- sim(z.out1, x = x.out1)
> summary(s.out1)

> plot(s.out1)
```





Example 2: One Tau with Weighting, Robust Standard Errors, and Bias Correction

Suppose that we wish to perform case control correction using weighting (rather than the default prior correction). To estimate the model:

```
> z.out2 <- zelig(conflict ~ major + contig + power + maxdem +
+      mindem + years, data = mid, model = "relogit", tau = 1042/303772,
+      case.control = "weighting", robust = TRUE)
```

Summarize the model output:

```
> summary(z.out2)
```

Set the explanatory variables to their means:

```
> x.out2 <- setx(z.out2)
```

Simulate quantities of interest:

```
> s.out2 <- sim(z.out2, x = x.out2)
> summary(s.out2)
```

Example 3: Two Taus with Bias Correction and Prior Correction

Suppose that we did not know that $\tau \approx 0.00343$, but only that it was somewhere between $(0.002, 0.005)$. To estimate a model with a range of feasible estimates for τ (using the default prior correction method for case control correction):

```
> z.out2 <- zelig(conflict ~ major + contig + power + maxdem +
+      mindem + years, data = mid, model = "relogit", tau = c(0.002,
+      0.005))
```

Summarize the model output:

```
> summary(z.out2)
```

Set the explanatory variables to their means:

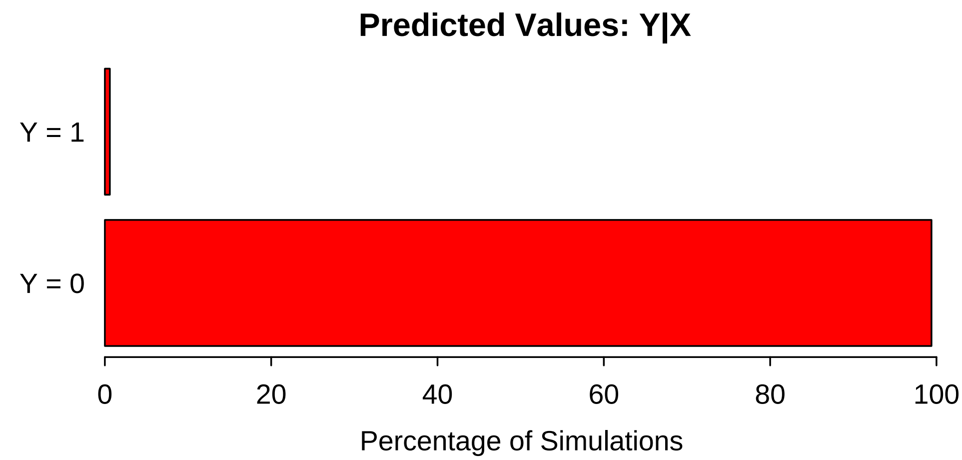
```
> x.out2 <- setx(z.out2)
```

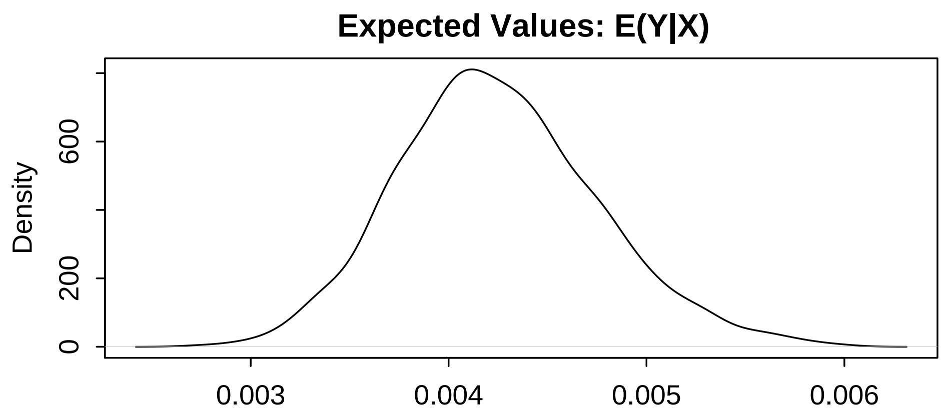
Simulate quantities of interest:

```
> s.out <- sim(z.out2, x = x.out2)

> summary(s.out2)

> plot(s.out2)
```





The cost of giving a range of values for τ is that point estimates are not available for quantities of interest. Instead, quantities are presented as confidence intervals with significance less than or equal to a specified level (e.g., at least 95% of the simulations are contained in the nominal 95% confidence interval).

Model

- Like the standard logistic regression, the *stochastic component* for the rare events logistic regression is:

$$Y_i \sim \text{Bernoulli}(\pi_i),$$

  where Y_i is the binary dependent variable, and takes a value of either 0 or 1.

- The *systematic component* is:

$$\pi_i = \frac{1}{1 + \exp(-x_i\beta)}.$$

- If the sample is generated via a case-control (or choice-based) design, such as when drawing all events (or "cases") and a sample from the non-events (or "controls") and going backwards to collect the explanatory variables, you must correct for selecting on the dependent variable. While the slope coefficients are approximately unbiased, the constant term may be significantly biased. Zelig has two methods for case control correction:

  1. The "prior correction" method adjusts the intercept term. Let τ be the true population fraction of events, $\bar{y}$ the fraction of events in the sample, and $\hat{\beta}_0$ the uncorrected intercept term. The corrected intercept β_0 is:
  $$\beta = \hat{\beta}_0 - \ln\left[\left(\frac{1-\tau}{\tau}\right)\left(\frac{\bar{y}}{1-\bar{y}}\right)\right].$$
  2. The "weighting" method performs a weighted logistic regression to correct for a case-control sampling design. Let the 1 subscript denote observations for which the dependent variable is observed as a 1, and the 0 subscript denote observations for which the dependent variable is observed as a 0. Then the vector of weights w_i
  $$\begin{aligned} w_1 &= \frac{\tau}{\bar{y}} \\ w_0 &= \frac{(1-\tau)}{(1-\bar{y})} \\ w_i &= w_1 Y_i + w_0(1-Y_i) \end{aligned}$$

  If τ is unknown, you may alternatively specify an upper and lower bound for the possible range of τ. In this case, the `relogit` procedure uses "robust Bayesian" methods to generate a confidence interval (rather than a point estimate) for each quantity of interest. The nominal coverage of the confidence interval is at least as great as the actual coverage.

- By default, estimates of the the coefficients β are bias-corrected to account for finite sample or rare events bias. In addition, quantities of interest, such as predicted probabilities, are also corrected of rare-events bias. If $\widehat{\beta}$ are the uncorrected logit coefficients and bias($\widehat{\beta}$) is the bias term, the corrected coefficients $\tilde{\beta}$ are
  $$\widehat{\beta} - \text{bias}(\widehat{\beta}) = \tilde{\beta}$$
  The bias term is
  $$\text{bias}(\widehat{\beta}) = (X'WX)^{-1}X'W\xi$$
  where
  $$\begin{aligned} \xi_i &= 0.5Q_{ii}\Big((1+w-1)\widehat{\pi}_i - w_1\Big) \\ Q &= X(X'WX)^{-1}X' \end{aligned}$$
  $$W = \text{diag}\{\widehat{\pi}_i(1-\widehat{\pi}_i)w_i\}$$

where w_i and w_1 are given in the "weighting" section above.

Quantities of Interest

- For either one or no τ:
  - The expected values (`qi$ev`) for the rare events logit are simulations of the predicted probability
  $$E(Y) = \pi_i = \frac{1}{1 + \exp(-x_i\beta)},$$
  given draws of β from its posterior.
  - The predicted value (`qi$pr`) is a draw from a binomial distribution with mean equal to the simulated π_i.
  - The first difference (`qi$fd`) is defined as
  $$\text{FD} = \Pr(Y = 1 \mid x_1, \tau) - \Pr(Y = 1 \mid x, \tau).$$
  - The risk ratio (`qi$rr`) is defined as
  $$\text{RR} = \Pr(Y = 1 \mid x_1, \tau) \ / \ \Pr(Y = 1 \mid x, \tau).$$

- For a range of τ defined by $[\tau_1, \tau_2]$, each of the quantities of interest are $n \times 2$ matrices, which report the lower and upper bounds, respectively, for a confidence interval with nominal coverage at least as great as the actual coverage. At worst, these bounds are conservative estimates for the likely range for each quantity of interest. Please refer to King and Zeng (2002) for the specific method of calculating bounded quantities of interest.

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is
$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$
where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is
$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "relogit", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `bias.correct`: `TRUE` if bias correction was selected, else `FALSE`.
  - `prior.correct`: `TRUE` if prior correction was selected, else `FALSE`.
  - `weighting`: `TRUE` if weighting was selected, else `FALSE`.
  - `tau`: the value of `tau` for which case control correction was implemented.
  - `residuals`: the working residuals in the final iteration of the IWLS fit.
  - `fitted.values`: the vector of fitted values for the systemic component, π_i.
  - `linear.predictors`: the vector of $x_i\beta$
  - `aic`: Akaike's Information Criterion (minus twice the maximized log-likelihood plus twice the number of coefficients).
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.

  Note that for a range of τ, each of the above items may be extracted from the `"lower.estimate"` and `"upper.estimate"` objects in your `zelig` output. Use `lower <- z.out$lower.estimate`, and then `lower$coefficients` to extract the coefficients for the empirical estimate generated for the smaller of the two τ.

- From `summary(z.out)`, you may extract:
  - `coefficients`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
  - `cov.scaled`: a $k \times k$ matrix of scaled covariances.

- `cov.unscaled`: a $k \times k$ matrix of unscaled covariances.

- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:

  - `qi$ev`: the simulated expected values, or predicted probabilities, for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from Binomial distributions given the predicted probabilities.
  - `qi$fd`: the simulated first difference in the predicted probabilities for the values specified in `x` and `x1`.
  - `qi$rr`: the simulated risk ratio for the predicted probabilities simulated from `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

Differences with Stata Version

The Stata version of ReLogit and the R implementation differ slightly in their coefficient estimates due to differences in the matrix inversion routines implemented in R and Stata. Zelig uses orthogonal-triangular decomposition (through `lm.influence()`) to compute the bias term, which is more numerically stable than standard matrix calculations.

How to Cite

To cite the *relogit* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "relogit: Rare Events Logistic Regression for Dichotomous Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

For more information see King and Zeng (2001a),King and Zeng (2001b),King and Zeng (2002a). Sample data are from King and Zeng (2001a).

12.63 sur: Seemingly Unrelated Regression

sur extends ordinary least squares analysis to estimate system of linear equations with correlated error terms. The seemingly unrelated regression model can be viewed as a special case of generalized least squares.

Syntax

```
> fml <- list ("mu1" = Y1 ~ X1,
               "mu2" = Y2 ~ X2,
               "mu3" = Y3 ~ X3)
> z.out<-zelig(formula = fml, model = "2sls", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Inputs

sur regression specification has at least M equations ($M \geq 2$) corresponding to the dependent variables ($Y_1, Y_2, \ldots, Y_M$).

- formula:a list whose elements are formulae corresponding to the M equations and their respective dependent and explanatory variables. For example, when there are no constraints on the coefficients:

  ```
  > fml <- list(mu1 = Y1 ~ X1, mu2 = Y2 ~ X2, mu3 = Y3 ~ X3)
  ```

  "mu1" is the label for the first equation with Y1 as the dependent variable and X1 as the explanatory variable. Similarly "mu2" and "mu3" are the labels for the Y2 and Y3 equations.

- tag: Users can also put constraints on the coefficients by using the special function tag. tag takes two parameters. The first parameter is the variable whose coefficient needs to be constrained and the second parameter is label for the constrained coefficient. Each label uniquely identifies the constrained coefficient. For example:

  ```
  > fml <- list(mu1 = Y1 ~ tag(Xc, "constrain1") + X1, mu2 = Y2 ~
  +      tag(Xc, "constrain1") + X2, mu3 = Y3 ~ X3)
  ```

Additional Inputs

sur takes the following additional inputs for model specifications:

- TX: an optional matrix to transform the regressor matrix and, hence, also the coefficient vector (see details). Default is NULL.

- maxiter: maximum number of iterations.

- `tol`: tolerance level indicating when to stop the iteration.
- `rcovformula`: formula to calculate the estimated residual covariance matrix (see details). Default is equal to 1.
- `probdfsys`: use the degrees of freedom of the whole system (in place of the degrees of freedom of the single equation to calculate probability values for the t-test of individual parameters.
- `solvetol`: tolerance level for detecting linear dependencies when inverting a matrix or calculating a determinant. Default is `solvetol`=

  `.Machine\$double.eps.`

- `saveMemory`: logical. Save memory by omitting some calculation that are not crucial for the basic estimate (e.g McElroy's R^2).

Details

The matrix `TX` transforms the regressor matrix (X) by $X* = X \times TX$. Thus, the vector of coefficients is now $b = TX \times b*$ where b is the original(stacked) vector of all coefficients and $b*$ is the new coefficient vector that is estimated instead. Thus, the elements of vector b and $b_i = \sum_j TX_{ij} \times b_j*$. The TX matrix can be used to change the order of the coefficients and also to restrict coefficients (if TX has less columns than it has rows). If iterated (with `maxit`>1), the covergence criterion is

$$\sqrt{\frac{\sum_i (b_{i,g} - b_{i,g-1})^2}{\sum_i b_{i,g-1}^2}} < tol$$

where $b_{i,g}$ is the ith coefficient of the gth iteration step. The formula (`rcovformula` to calculate the estimated covariance matrix of the residuals($\hat{\Sigma}$)can be one of the following (see Judge et al., 1955, p.469): if `rcovformula`= 0:

$$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{T}$$

if `rcovformula`= 1 or `rcovformula`='geomean':

$$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{\sqrt{(T - k_i) \times (T - k_j)}}$$

if `rcovformula`= 2 or `rcovformula`='Theil':

$$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{T - k_i - k_j + tr[X_i(X_i\prime X_i)^{-1}X_i\prime X_j(X_j\prime X_j)^{-1}X_j\prime]}$$

if `rcovformula`= 3 or `rcovformula`='max':

$$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{T - max(k_i, k_j)}$$

If $i = j$, formula 1, 2, and 3 are equal. All these three formulas yield unbiased estimators for the diagonal elements of the residual covariance matrix. If $ineqj$, only formula 2 yields an unbiased estimator for the residual covariance matrix, but it is not necessarily positive semidefinit. Thus, it is doubtful whether formula 2 is really superior to formula 1 (Theil, 1971, p.322).

Examples

Attaching the example dataset:

```
> data(grunfeld)
```

Formula:

```
> formula <- list(mu1 = Ige ~ Fge + Cge, mu2 = Iw ~ Fw + Cw)
```

Estimating the model using `sur`:

```
> z.out <- zelig(formula = formula, model = "sur", data = grunfeld)
```

```
> summary(z.out)
```

Set explanatory variables to their default (mean/mode) values

```
> x.out <- setx(z.out)
```

Simulate draws from the posterior distribution:

```
> s.out <- sim(z.out, x = x.out)
```

```
> summary(s.out)
```

Model

The basic seemingly unrelated regression model assumes that for each individual observation i there are M dependent variables $(Y_{ij}, j = 1, \ldots, M)$ each with its own regression equation:

$$Y_{ij} = X_{ij}'\beta_j + \epsilon_{ij}, \quad \text{for} \quad i = 1, \ldots, N \quad \text{and} \quad j = 1, \ldots, M$$

when X_{ij} is a k-vector of explanatory variables, β_j is the coefficients of the explanatory variables,

- The *stochastic component* is:

  $$\epsilon_{ij} \sim \mathcal{N}(0, \sigma_{ij})$$

  where within each j equation, $epsilon_{ij}$ is identically and independently distributed for $i = 1, \ldots, M$,

  $$\text{Var}(\epsilon_{ij}) = \sigma_j \quad \text{and} \quad \text{Cov}(\epsilon_{ij}, \epsilon_{i\prime j}) = 0, \quad \text{for} \quad i \neq i\prime, \quad \text{and} \quad j = 1, \ldots, M$$

  However, the error terms for the *ith* observation can be correlated across equations

  $$\text{Cov}(\epsilon_{ij}, \epsilon_{ij\prime}) \neq 0, \quad \text{for} \quad j \neg j\prime, \quad \text{and} \quad i = 1, \ldots, N$$

- The *systematic component* is:

  $$\mu_{ij} = E(Y_i j) = X_{ij}\beta_j, \quad \text{for} \quad i = 1, \ldots, N, \quad \text{and} \quad j = 1, \ldots, M$$

See Also

For information about two stage least squares regression, see Section **??** and `help(2sls)`. For information about three stage least squares regression, see Section **??** and `help(3sls)`.

Quantities of Interest

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(formula=fml, model = "sur", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below:

- `rcovest`: residual covariance matrix used for estimation.

- `mcelr2`: McElroys R-squared value for the system.
- `method`: Estimation method.
- `g`: number of equations.
- `n`: total number of observations.
- `k`: total number of coefficients.
- `ki`: total number of linear independent coefficients.
- `df`: degrees of freedom of the whole system.
- `iter`: number of iteration steps.
- `b`: vector of all estimated coefficients.
- `t`: t values for b.
- `se`: estimated standard errors of b.
- `bt`: coefficient vector transformed by TX.
- `p`: p values for b.
- `bcov`: estimated covariance matrix of b.
- `btcov`: covariance matrix of bt.
- `rcov`: estimated residual covariance matrix.
- `drcov`: determinant of `rcov`.
- `rcor`: estimated residual correlation matrix.
- `olsr2`: system OLS R-squared value.
- `y`: vector of all (stacked) endogenous variables.
- `x`: matrix of all (diagonally stacked) regressors.
- `data`: data frame of the whole system (including instruments).
- `TX`: matrix used to transform the regressor matrix.
- `rcovformula`: formula to calculate the estimated residual covariance matrix.
- `probdfsys`: system degrees of freedom to calculate probability values?.
- `solvetol`: tolerance level when inverting a matrix or calculating a determinant.

- `eq`: a list that contains the results that belong to the individual equations.
- `eqnlabel*`: the equation label of the ith equation (from the labels list).
- `formula*`: model formula of the ith equation.
- `n*`: number of observations of the ith equation.
- `k*`: number of coefficients/regressors in the ith equation (including the constant).
- `ki*`: number of linear independent coefficients in the ith equation (including the constant differs from k only if there are restrictions that are not cross equation).
- `df*`: degrees of freedom of the ith equation.
- `b*`: estimated coefficients of the ith equation.
- `se*`: estimated standard errors of b of the ith equation.
- `t*`: t values for b of the ith equation.
- `p*`: p values for b of the ith equation.
- `covb*`: estimated covariance matrix of b of the ith equation.
- `y*`: vector of endogenous variable (response values) of the ith equation.
- `x*`: matrix of regressors (model matrix) of the ith equation.
- `data*`: data frame (including instruments) of the ith equation.
- `fitted*`: vector of fitted values of the ith equation.
- `residuals*`: vector of residuals of the ith equaiton.
- `ssr*`: sum of squared residuals of the ith equation.
- `mse*`: estimated variance of the residuals (mean of squared errors) of the ith equation.
- `s2*`: estimated variance of the residents($\hat{sigma}^2$) of the ith equation.
- `rmse*`: estimated standard error of the reiduals (square root of mse) of the ith equation.
- `s*`: estimated standard error of the residuals ($\hat{\sigma}$) of the ith equation.
- `r2*`: R-squared (coefficient of determination).
- `adjr2*`: adjusted R-squared value.
- `maxiter`: maximum number of iterations.
- `tol`: tolerance level indicating when to stop the iteration.

How to Cite

To cite the *sur* Zelig model:

> Ferdinand Alimadhi, Ying Lu, and Elena Villalon. 2007. "sur: Seemingly Unrelated Regression," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The sur function is adapted from the `systemfit` library (Hamann and Henningsen 2005).

12.64 `threesls`: Three Stage Least Squares

`threesls` is a combination of two stage least squares and seemingly unrelated regression. It provides consistent estimates for linear regression models with explanatory variables correlated with the error term. It also extends ordinary least squares analysis to estimate system of linear equations with correlated error terms

Syntax

```
> fml <- list(mu1 = Y1 ~ X1 + Z1, mu2 = Y2 ~ X2 + Z2, inst1 = Z1 ~
+     W1 + X1, inst2 = Z2 ~ W2 + X2)
```

```
> z.out <- zelig(formula = fml, model = "treesls", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Inputs

`threesls` regression specification requires at least two sets of equations. The first set of M euqations corresponds to the M dependent variables $(Y_1, \ldots, Y_M)$ to be estimated. The second set of equations (Z) corresponds to the instrumental variables in the M equations.

- `formula`:a list of the system of equations and instrumental variable equations. The system of equations is listed first as `mu`s. The equations for the instrumental variables are listed next as `inst`s. For example:

  ```
  > fml <- list(mu1 = Y1 ~ X1 + Z1, mu2 = Y2 ~ X2 + Z2, inst1 = Z1 ~
  +     W1 + X1, inst2 = Z2 ~ W2 + X2)
  ```

  `"mu1"` is the first equation in the two equation model with `Y1` as the dependent variable and `X1` and `Z1` as the explanatory variables. `"mu2"` is the second equation with `Y2` as the dependent variable and `X2` and `Z2` as the explanatory variables. `Z1` and `Z2` are also problematic endogenous variables, so they are estimated through instruments in the `"inst1"` and `"inst2"` equations.

- `Y`: dependent variables of interest in the system of equations.

- `Z`: the problematic explanatory variables correlated with the error term.

- `W`: exogenous instrument variables used to estimate the problematic explanatory variables (`Z`)

Additional Inputs

`threesls` takes the following additional inputs for model specifications:

- `TX`: an optional matrix to transform the regressor matrix and, hence, also the coefficient vector (see details). Default is `NULL`.
- `maxiter`: maximum number of iterations.
- `tol`: tolerance level indicating when to stop the iteration.
- `rcovformula`: formula to calculate the estimated residual covariance matrix (see details). Default is equal to 1.
- `formulathreesls`: formula for calculating the threesls estimator, one of "GLS", "IV", "GMM", "Schmidt", or "Eviews" (see details.)
- `probdfsys`: use the degrees of freedom of the whole system (in place of the degrees of freedom of the single equation to calculate probability values for the t-test of individual parameters.
- `single.eq.sigma`: use different σ^2 for each single equation to calculate the covariance matrix and the standard errors of the coefficients.
- `solvetol`: tolerance level for detecting linear dependencies when inverting a matrix or calculating a determinant. Default is `solvetol`=.Machine$double.eps.
- `saveMemory`: logical. Save memory by omitting some calculation that are not crucial for the basic estimate (e.g McElroy's R^2).

Details

The matrix `TX` transforms the regressor matrix (X) by $X* = X \times TX$. Thus, the vector of coefficients is now $b = TX \times b*$ where b is the original(stacked) vector of all coefficients and $b*$ is the new coefficient vector that is estimated instead. Thus, the elements of vector b and $b_i = \sum_j TX_{ij} \times b_j*$. The TX matrix can be used to change the order of the coefficients and also to restrict coefficients (if TX has less columns than it has rows). If iterated (with `maxit`>1), the covergence criterion is

$$\sqrt{\frac{\sum_i (b_{i,g} - b_{i,g-1})^2}{\sum_i b_{i,g-1}^2}} < tol$$

where $b_{i,g}$ is the ith coefficient of the gth iteration step. The formula (`rcovformula` to calculate the estimated covariance matrix of the residuals($\hat{\Sigma}$)can be one of the following (see Judge et al., 1955, p.469): if `rcovformula`= 0:

$$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{T}$$

if `rcovformula`= 1 or `rcovformula`='geomean':

$$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{\sqrt{(T-k_i)\times(T-k_j)}}$$

if `rcovformula`= 2 or `rcovformula`='Theil':

$$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{T-k_i-k_j+tr[X_i(X_i\prime X_i)^{-1}X_i\prime X_j(X_j\prime X_j)^{-1}X_j\prime]}$$

if `rcovformula`= 3 or `rcovformula`='max':

$$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{T-max(k_i,k_j)}$$

If $i = j$, formula 1, 2, and 3 are equal. All these three formulas yield unbiased estimators for the diagonal elements of the residual covariance matrix. If $ineqj$, only formula 2 yields an unbiased estimator for the residual covariance matrix, but it is not necessarily positive semidefinit. Thus, it is doubtful whether formula 2 is really superior to formula 1 (Theil, 1971, p.322). The formulas to calculate the threesls estimator lead to identical results if the same instruments are used in all equations. If different instruments are used in the different equations, only the GMM-threesls estimator ("GMM") and the threesls estimator proposed by Schmidt (1990) ("Schmidt") are consistent, whereas "GMM" is efficient relative to "Schmidt" (see Schmidt, 1990).

Examples

Attaching the example dataset:

```
> data(kmenta)
```

Formula:

```
> formula <- list(mu1 = q ~ p + d, mu2 = q ~ p + f + a, inst = ~d +
+     f + a)
```

Estimating the model using `threesls`:

```
> z.out <- zelig(formula = formula, model = "threesls", data = kmenta)
> summary(z.out)
```

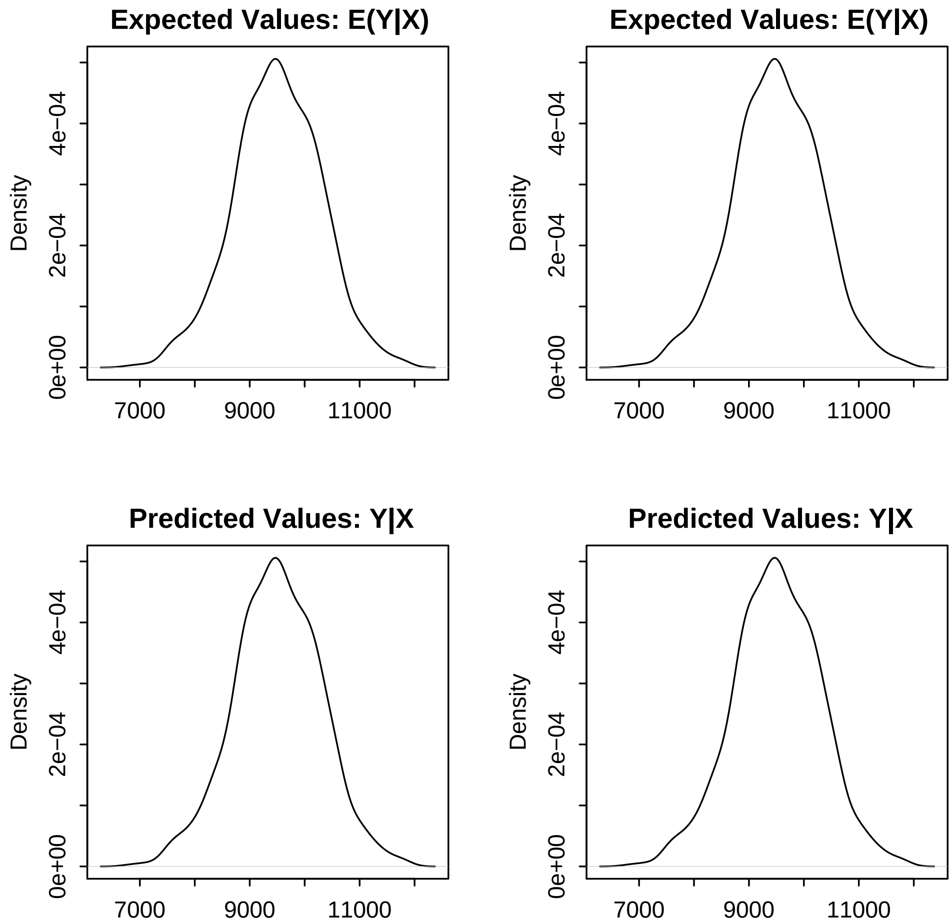
Set explanatory variables to their default (mean/mode) values

```
> x.out <- setx(z.out)
```

Simulate draws from the posterior distribution:

```
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
```

Plot the quantities of interest



Model

See Also

For information about two stage least square regression, see Section ?? and `help(2sls)`. For information about seemingly unrelated regression, see Section 12.63 and `help(sur)`.

Quantities of Interest

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(formula=fml, model = "threesls", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below:

- `rcovest`: residual covariance matrix used for estimation.
- `mcelr2`: McElroys R-squared value for the system.
- `h`: matrix of all (diagonally stacked) instrumental variables.
- `formulathreesls`: formula for calculating the threesls estimator
- `method`: Estimation method.
- `g`: number of equations.
- `n`: total number of observations.
- `k`: total number of coefficients.
- `ki`: total number of linear independent coefficients.
- `df`: degrees of freedom of the whole system.
- `iter`: number of iteration steps.
- `b`: vector of all estimated coefficients.
- `t`: t values for b.
- `se`: estimated standard errors of b.
- `bt`: coefficient vector transformed by TX.

- `p`: p values for b.
- `bcov`: estimated covariance matrix of b.
- `btcov`: covariance matrix of bt.
- `rcov`: estimated residual covariance matrix.
- `drcov`: determinant of `rcov`.
- `rcor`: estimated residual correlation matrix.
- `olsr2`: system OLS R-squared value.
- `y`: vector of all (stacked) endogenous variables.
- `x`: matrix of all (diagonally stacked) regressors.
- `data`: data frame of the whole system (including instruments).
- `TX`: matrix used to transform the regressor matrix.
- `rcovformula`: formula to calculate the estimated residual covariance matrix.
- `probdfsys`: system degrees of freedom to calculate probability values?.
- `solvetol`: tolerance level when inverting a matrix or calculating a determinant.
- `eq`: a list that contains the results that belong to the individual equations.
- `eqnlabel*`: the equation label of the ith equation (from the labels list).
- `formula*`: model formula of the ith equation.
- `n*`: number of observations of the ith equation.
- `k*`: number of coefficients/regressors in the ith equation (including the constant).
- `ki*`: number of linear independent coefficients in the ith equation (including the constant differs from k only if there are restrictions that are not cross equation).
- `df*`: degrees of freedom of the ith equation.
- `b*`: estimated coefficients of the ith equation.
- `se*`: estimated standard errors of b of the ith equation.
- `t*`: t values for b of the ith equation.
- `p*`: p values for b of the ith equation.

- `covb*`: estimated covariance matrix of b of the ith equation.
- `y*`: vector of endogenous variable (response values) of the ith equation.
- `x*`: matrix of regressors (model matrix) of the ith equation.
- `data*`: data frame (including instruments) of the ith equation.
- `fitted*`: vector of fitted values of the ith equation.
- `residuals*`: vector of residuals of the ith equaiton.
- `ssr*`: sum of squared residuals of the ith equation.
- `mse*`: estimated variance of the residuals (mean of squared errors) of the ith equation.
- `s2*`: estimated variance of the residents($\hat{sigma}^2$) of the ith equation.
- `rmse*`: estimated standard error of the reiduals (square root of mse) of the ith equation.
- `s*`: estimated standard error of the residuals ($\hat{\sigma}$) of the ith equation.
- `r2*`: R-squared (coefficient of determination).
- `adjr2*`: adjusted R-squared value.
- `inst*`: instruments of the ith equation.
- `h*`: matrix of instrumental variables of the ith equation.
- `zelig.data`: the input data frame if `save.data = TRUE`.

How to Cite

To cite the *threesls* Zelig model:

> Ferdinand Alimadhi, Ying Lu, and Elena Villalon. 2007. "threesls: Three Stage Least Squares," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The threesls function is adapted from the `systemfit` library (Hamann and Henningsen 2005).

12.65 `tobit`: Linear Regression for a Left-Censored Dependent Variable

Tobit regression estimates a linear regression model for a left-censored dependent variable, where the dependent variable is censored from below. While the classical tobit model has values censored at 0, you may select another censoring point. For other linear regression models with fully observed dependent variables, see Bayesian regression (Section 12.40), maximum likelihood normal regression (Section 12.39), or least squares (Section 12.32).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, below = 0, above = Inf,
                 model = "tobit", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Inputs

`zelig()` accepts the following arguments to specify how the dependent variable is censored.

- `below`: (defaults to 0) The point at which the dependent variable is censored from below. If any values in the dependent variable are observed to be less than the censoring point, it is assumed that that particular observation is censored from below at the observed value. (See Section 12.66 for a Bayesian implementation that supports both left and right censoring.)

- `robust`: defaults to `FALSE`. If `TRUE`, `zelig()` computes robust standard errors based on sandwich estimators (see Huber (1981) and White (1980)) and the options selected in `cluster`.

- `cluster`: if `robust = TRUE`, you may select a variable to define groups of correlated observations. Let `x3` be a variable that consists of either discrete numeric values, character strings, or factors that define strata. Then

  ```
  > z.out <- zelig(y ~ x1 + x2, robust = TRUE, cluster = "x3",
                   model = "tobit", data = mydata)
  ```

  means that the observations can be correlated within the strata defined by the variable `x3`, and that robust standard errors should be calculated according to those clusters. If `robust = TRUE` but `cluster` is not specified, `zelig()` assumes that each observation falls into its own cluster.

Zelig users may wish to refer to `help(survreg)` for more information.

Examples

1. Basic Example
   Attaching the sample dataset:

   ```
   > data(tobin)
   ```

   Estimating linear regression using `tobit`:

   ```
   > z.out <- zelig(durable ~ age + quant, model = "tobit", data = tobin)
   ```

   Setting values for the explanatory variables to their sample averages:

   ```
   > x.out <- setx(z.out)
   ```

   Simulating quantities of interest from the posterior distribution given `x.out`.

   ```
   > s.out1 <- sim(z.out, x = x.out)
   ```

   ```
   > summary(s.out1)
   ```

2. Simulating First Differences
   Set explanatory variables to their default(mean/mode) values, with high (80th percentile) and low (20th percentile) liquidity ratio (`quant`):

   ```
   > x.high <- setx(z.out, quant = quantile(tobin$quant, prob = 0.8))
   > x.low <- setx(z.out, quant = quantile(tobin$quant, prob = 0.2))
   ```

   Estimating the first difference for the effect of high versus low liquidity ratio on duration(`durable`):

   ```
   > s.out2 <- sim(z.out, x = x.high, x1 = x.low)
   ```

   ```
   > summary(s.out2)
   ```

Model

- Let Y_i^* be a latent dependent variable which is distributed with *stochastic* component

  $$Y_i^* \sim \text{Normal}(\mu_i, \sigma^2)$$

  where μ_i is a vector means and σ^2 is a scalar variance parameter. Y_i^* is not directly observed, however. Rather we observed Y_i which is defined as:

  $$Y_i = \begin{cases} Y_i^* & \text{if} \quad c < Y_i^* \\ c & \text{if} \quad c \ge Y_i^* \end{cases}$$

  where c is the lower bound below which Y_i^* is censored.

- The *systematic component* is given by

$$\mu_i = x_i \beta,$$

  where x_i is the vector of k explanatory variables for observation i and β is the vector of coefficients.

Quantities of Interest

- The expected values (`qi$ev`) for the tobit regression model are the same as the expected value of $Y*$:

$$E(Y^*|X) = \mu_i = x_i \beta$$

- The first difference (`qi$fd`) for the tobit regression model is defined as

$$\text{FD} = E(Y^* \mid x_1) - E(Y^* \mid x).$$

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is

$$\frac{1}{\sum t_i} \sum_{i:t_i=1} [E[Y_i^*(t_i = 1)] - E[Y_i^*(t_i = 0)]],$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(y ~ x, model = "tobit.bayes", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated parameters. The first k columns contain the posterior draws of the coefficients β, and the last column contains the posterior draws of the variance σ^2.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
  - `seed`: the random seed used in the model.

- From the `sim()` output object `s.out`:
  - `qi$ev`: the simulated expected value for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected values given the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *oprobit.bayes* Zelig model use:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "oprobit.bayes: Bayesian Ordered Probit Regression," in Kosuke Imai,Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The tobit function is part of the survival library by Terry Therneau, ported to R by Thomas Lumley. Advanced users may wish to refer to `help(survfit)` in the survival library and Venables and Ripley (2002).Sample data are from King et al. (1990a).

12.66 `tobit.bayes`: Bayesian Linear Regression for a Censored Dependent Variable

Bayesian tobit regression estimates a linear regression model with a censored dependent variable using a Gibbs sampler. The dependent variable may be censored from below and/or from above. For other linear regression models with fully observed dependent variables, see Bayesian regression (Section 12.40), maximum likelihood normal regression (Section 12.39), or least squares (Section 12.32).

Syntax

```
> z.out <- zelig(Y ~ X1 + X2, below = 0, above = Inf,
                 model = "tobit.bayes", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Inputs

`zelig()` accepts the following arguments to specify how the dependent variable is censored.

- `below`: point at which the dependent variable is censored from below. If the dependent variable is only censored from above, set `below = -Inf`. The default value is 0.
- `above`: point at which the dependent variable is censored from above. If the dependent variable is only censored from below, set `above = Inf`. The default value is `Inf`.

Additional Inputs

Use the following arguments to monitor the convergence of the Markov chain:

- `burnin`: number of the initial MCMC iterations to be discarded (defaults to 1,000).
- `mcmc`: number of the MCMC iterations after burnin (defaults to 10,000).
- `thin`: thinning interval for the Markov chain. Only every `thin`-th draw from the Markov chain is kept. The value of `mcmc` must be divisible by this value. The default value is 1.
- `verbose`: defaults to `FALSE`. If `TRUE`, the progress of the sampler (every 10%) is printed to the screen.
- `seed`: seed for the random number generator. The default is `NA` which corresponds to a random seed of 12345.
- `beta.start`: starting values for the Markov chain, either a scalar or vector with length equal to the number of estimated coefficients. The default is `NA`, such that the least squares estimates are used as the starting values.

Use the following parameters to specify the model's priors:

- `b0`: prior mean for the coefficients, either a numeric vector or a scalar. If a scalar, that value will be the prior mean for all coefficients. The default is 0.
- `B0`: prior precision parameter for the coefficients, either a square matrix (with the dimensions equal to the number of the coefficients) or a scalar. If a scalar, that value times an identity matrix will be the prior precision parameter. The default is 0, which leads to an improper prior.
- `c0`: `c0/2` is the shape parameter for the Inverse Gamma prior on the variance of the disturbance terms.
- `d0`: `d0/2` is the scale parameter for the Inverse Gamma prior on the variance of the disturbance terms.

Zelig users may wish to refer to `help(MCMCtobit)` for more information.

Convergence

Users should verify that the Markov Chain converges to its stationary distribution. After running the `zelig()` function but before performing `setx()`, users may conduct the following convergence diagnostics tests:

- `geweke.diag(z.out$coefficients)`: The Geweke diagnostic tests the null hypothesis that the Markov chain is in the stationary distribution and produces z-statistics for each estimated parameter.
- `heidel.diag(z.out$coefficients)`: The Heidelberger-Welch diagnostic first tests the null hypothesis that the Markov Chain is in the stationary distribution and produces p-values for each estimated parameter. Calling `heidel.diag()` also produces output that indicates whether the mean of a marginal posterior distribution can be estimated with sufficient precision, assuming that the Markov Chain is in the stationary distribution.
- `raftery.diag(z.out$coefficients)`: The Raftery diagnostic indicates how long the Markov Chain should run before considering draws from the marginal posterior distributions sufficiently representative of the stationary distribution.

If there is evidence of non-convergence, adjust the values for `burnin` and `mcmc` and rerun `zelig()`.

Advanced users may wish to refer to `help(geweke.diag)`, `help(heidel.diag)`, and `help(raftery.diag)` for more information about these diagnostics.

Examples

1. Basic Example
   Attaching the sample dataset:

   ```
   > data(tobin)
   ```

   Estimating linear regression using `tobit.bayes`:

   ```
   > z.out <- zelig(durable ~ age + quant, model = "tobit.bayes",
   +      data = tobin, verbose = TRUE)
   ```

   Checking for convergence before summarizing the estimates:

   ```
   > geweke.diag(z.out$coefficients)
   ```

   ```
   > heidel.diag(z.out$coefficients)
   ```

   ```
   > raftery.diag(z.out$coefficients)
   ```

   ```
   > summary(z.out)
   ```

   Setting values for the explanatory variables to their sample averages:

   ```
   > x.out <- setx(z.out)
   ```

   Simulating quantities of interest from the posterior distribution given `x.out`.

   ```
   > s.out1 <- sim(z.out, x = x.out)
   ```

   ```
   > summary(s.out1)
   ```

2. Simulating First Differences
   Set explanatory variables to their default(mean/mode) values, with high (80th percentile) and low (20th percentile) liquidity ratio (`quant`):

   ```
   > x.high <- setx(z.out, quant = quantile(tobin$quant, prob = 0.8))
   > x.low <- setx(z.out, quant = quantile(tobin$quant, prob = 0.2))
   ```

   Estimating the first difference for the effect of high versus low liquidity ratio on duration(`durable`):

   ```
   > s.out2 <- sim(z.out, x = x.high, x1 = x.low)
   ```

   ```
   > summary(s.out2)
   ```

Model

Let Y_i^* be the dependent variable which is not directly observed. Instead, we observe Y_i which is defined as following:

$$Y_i = \begin{cases} Y_i^* & \text{if} \quad c_1 < Y_i^* < c_2 \\ c_1 & \text{if} \quad c_1 \geq Y_i^* \\ c_2 & \text{if} \quad c_2 \leq Y_i^* \end{cases}$$

where c_1 is the lower bound below which Y_i^* is censored, and c_2 is the upper bound above which Y_i^* is censored.

- The *stochastic component* is given by

  $$\epsilon_i \sim \text{Normal}(0, \sigma^2)$$

  where $\epsilon_i = Y_i^* - \mu_i$.

- The *systematic component* is given by

  $$\mu_i = x_i \beta,$$

  where x_i is the vector of k explanatory variables for observation i and β is the vector of coefficients.

- The *semi-conjugate priors* for β and σ^2 are given by

  $$\begin{aligned} \beta &\sim \text{Normal}_k \left(b_0, B_0^{-1}\right) \\ \sigma^2 &\sim \text{InverseGamma}\left(\frac{c_0}{2}, \frac{d_0}{2}\right) \end{aligned}$$

  where b_0 is the vector of means for the k explanatory variables, B_0 is the $k \times k$ precision matrix (the inverse of a variance-covariance matrix), and $c_0/2$ and $d_0/2$ are the shape and scale parameters for σ^2. Note that β and σ^2 are assumed *a priori* independent.

Quantities of Interest

- The expected values (`qi$ev`) for the tobit regression model is calculated as following. Let

  $$\begin{aligned} \Phi_1 &= \Phi\left(\frac{(c_1 - x\beta)}{\sigma}\right) \\ \Phi_2 &= \Phi\left(\frac{(c_2 - x\beta)}{\sigma}\right) \\ \phi_1 &= \phi\left(\frac{(c_1 - x\beta)}{\sigma}\right) \\ \phi_2 &= \phi\left(\frac{(c_2 - x\beta)}{\sigma}\right) \end{aligned}$$

where $\Phi(\cdot)$ is the (cumulative) Normal density function and $\phi(\cdot)$ is the Normal probability density function of the standard normal distribution. Then the expected values are

$$\begin{aligned} E(Y|x) &= P(Y^* \leq c_1|x)c_1 + P(c_1 < Y^* < c_2|x)E(Y^* \mid c_1 < Y^* < c_2, x) + P(Y^* \geq c_2)c_2 \\ &= \Phi_1 c_1 + x\beta(\Phi_2 - \Phi_1) + \sigma(\phi_1 - \phi_2) + (1 - \Phi_2)c_2, \end{aligned}$$

- The first difference (`qi$fd`) for the tobit regression model is defined as

$$\text{FD} = E(Y \mid x_1) - E(Y \mid x).$$

- In conditional prediction models, the average expected treatment effect (`qi$att.ev`) for the treatment group is

$$\frac{1}{\sum t_i} \sum_{i:t_i=1} [Y_i(t_i = 1) - E[Y_i(t_i = 0)]],$$

  where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(y ~ x, model = "tobit.bayes", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: draws from the posterior distributions of the estimated parameters. The first k columns contain the posterior draws of the coefficients β, and the last column contains the posterior draws of the variance σ^2.
  - `zelig.data`: the input data frame if `save.data = TRUE`.
  - `seed`: the random seed used in the model.
- From the `sim()` output object `s.out`:
  - `qi$ev`: the simulated expected value for the specified values of `x`.
  - `qi$fd`: the simulated first difference in the expected values given the values specified in `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *tobit.bayes* Zelig model:

> Ben Goodrich and Ying Lu. 2007. "tobit.bayes: Bayesian Linear Regression for a Censored Dependent Variable," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

Bayesian tobit regression is part of the MCMCpack library by Andrew D. Martin and Kevin M. Quinn (Martin and Quinn 2005). The convergence diagnostics are part of the CODA library by Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines (Plummer et al. 2005).

12.67 `twosls`: Two Stage Least Squares

`twosls` provides consistent estimates for linear regression models with some explanatory variable correlated with the error term using instrumental variables. In this situation, ordinary least squares fails to provide consistent estimates. The name two-stage least squares stems from the two regressions in the estimation procedure. In stage one, an ordinary least squares prediction of the instrumental variable is obtained from regressing it on the instrument variables. In stage two, the coefficients of interest are estimated using ordinary least square after substituting the instrumental variable by its predictions from stage one.

Syntax

```
> fml <- list("mu1"  = Y ~ X + W, "mu2" = W ~ X + Z,
              "inst" = ~ X + Z)
> z.out <- zelig(formula = fml, model = "twosls", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Inputs

`twosls` regression take the following inputs:

- `formula`: A list of the formula for the main equation, the formula for the endogenous variable, and the (one-sided) formula for instrumental variables (including covariates). The first object in the list `mu` corresponds to the main regression model needs to be estimated. Alternatively, a system of simultaneous equations can be used. See the help file of `systemfit` for more information. For example:

  ```
  > fml <- list(mu1 = Y ~ X + W, mu2 = W ~ X + Z, inst = ~X +
  +     Z)
  ```

  - `Y`: the dependent variable of interest.
  - `X`: the covariate.
  - `W`: the endogenous variable.
  - `Z`: the exogenous instrumental variable.

Additional Inputs

`twosls` takes the following additional inputs for model specifications:

- `TX`: an optional matrix to transform the regressor matrix and, hence, also the coefficient vector (see 12.67). Default is `NULL`.

- `rcovformula`: formula to calculate the estimated residual covariance matrix (see 12.67). Default is equal to 1.

- `probdfsys`: use the degrees of freedom of the whole system (in place of the degrees of freedom of the single equation to calculate probability values for the t-test of individual parameters.

- `single.eq.sigma`: use different σ^2 for each single equation to calculate the covariance matrix and the standard errors of the coefficients.

- `solvetol`: tolerance level for detecting linear dependencies when inverting a matrix or calculating a determinant. Default is `solvetol`=.Machine$double.eps.

- `saveMemory`: logical. Save memory by omitting some calculation that are not crucial for the basic estimate (e.g McElroy's R^2).

Details

- `TX`: The matrix `TX` transforms the regressor matrix (X) by $X* = X \times TX$. Thus, the vector of coefficients is now $b = TX \times b*$ where b is the original(stacked) vector of all coefficients and $b*$ is the new coefficient vector that is estimated instead. Thus, the elements of vector b and $b_i = \sum_j TX_{ij} \times b_j*$. The TX matrix can be used to change the order of the coefficients and also to restrict coefficients (if TX has less columns than it has rows).

- `rcovformula`: The formula to calculate the estimated covariance matrix of the residuals($\hat{\Sigma}$)can be one of the following (see Judge et al., 1955, p.469): if `rcovformula`= 0:

  $$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{T}$$

  if `rcovformula`= 1 or `rcovformula`='geomean':

  $$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{\sqrt{(T - k_i) \times (T - k_j)}}$$

  if `rcovformula`= 2 or `rcovformula`='Theil':

  $$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{T - k_i - k_j + tr[X_i(X_i\prime X_i)^{-1}X_i\prime X_j(X_j\prime X_j)^{-1}X_j\prime]}$$

  if `rcovformula`= 3 or `rcovformula`='max':

  $$\hat{\sigma_{ij}} = \frac{\hat{e_i}\prime\hat{e_j}}{T - max(k_i, k_j)}$$

  If $i = j$, formula 1, 2, and 3 are equal. All these three formulas yield unbiased estimators for the diagonal elements of the residual covariance matrix. If $ineqj$, only formula 2 yields an unbiased estimator for the residual covariance matrix, but it is not necessarily positive semidefinit. Thus, it is doubtful whether formula 2 is really superior to formula 1

Examples

Attaching the example dataset:

```
> data(klein)
```

Formula:

```
> formula <- list(mu1 = C ~ Wtot + P + P1, mu2 = I ~ P + P1 +
+     K1, mu3 = Wp ~ X + X1 + Tm, inst = ~P1 + K1 + X1 + Tm +
+     Wg + G)
```

Estimating the model using `twosls`:

```
> z.out <- zelig(formula = formula, model = "twosls", data = klein)
> summary(z.out)
```

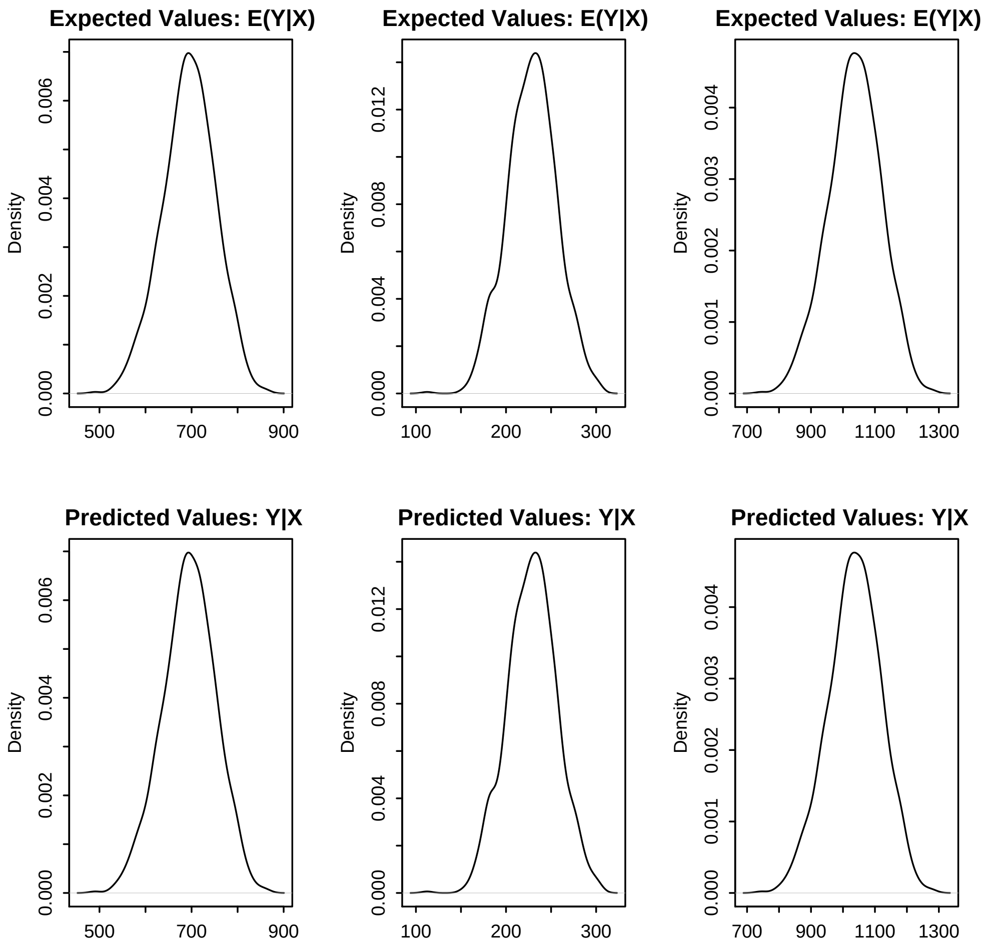
Set explanatory variables to their default (mean/mode) values

```
> x.out <- setx(z.out)
```

Simulate draws from the posterior distribution:

```
> s.out <- sim(z.out, x = x.out)
> summary(s.out)
```

Plot the quantities of interest


Expected Values: E(Y|X)
Expected Values: E(Y|X)
Expected Values: E(Y|X)
Density
0.000
0.002
0.004
0.006
500
700
900
Density
0.000
0.004
0.008
0.012
100
200
300
Density
0.000
0.001
0.002
0.003
0.004
700
900
1100
1300
Predicted Values: Y|X
Predicted Values: Y|X
Predicted Values: Y|X
Density
0.000
0.002
0.004
0.006
500
700
900
Density
0.000
0.004
0.008
0.012
100
200
300
Density
0.000
0.001
0.002
0.003
0.004
700
900
1100
1300

Model

Let's consider the following regression model,

$$Y_i = X_i\beta + Z_i\gamma + \epsilon_i, \quad i = 1, \ldots, N$$

where Y_i is the dependent variable, $X_i = (X_{1i}, \ldots, X_{Ni})$ is the vector of explanatory variables, β is the vector of coefficients of the explanatory variables X_i, Z_i is the problematic explanatory variable, and γ is the coefficient of Z_i. In the equation, there is a direct dependence of Z_i on the structural disturbances of ϵ.

- The *stochastic component* is given by

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2), \quad \text{and} \quad \text{cov}(Z_i, \epsilon_i) \neq 0,$$

- The *systematic component* is given by:

$$\mu_i = E(Y_i) = X_i\beta + Z_i\gamma,$$

To correct the problem caused by the correlation of Z_i and ϵ, two stage least squares utilizes two steps:

- *Stage 1*: A new instrumental variable $\hat{Z}$ is created for Z_i which is the ordinary least squares predictions from regressing Z_i on a set of exogenous instruments W and X.

$$\widehat{Z}_i = \widetilde{W}_i[(\widetilde{W}^\top\widetilde{W})^{-1}\widetilde{W}^\top Z]$$

  where $\widetilde{W} = (W, X)$

- *Stage 2*: Substitute for $\hat{Z}_i$ for Z_i in the original equation, estimate β and γ by ordinary least squares regression of Y on X and $\hat{Z}$ as in the following equation.

$$Y_i = X_i\beta + \widehat{Z}_i\gamma + \epsilon_i, \quad \text{for} \quad i = 1, \ldots, N$$

See Also

For information about three stage least square regression, see Section **??** and `help(3sls)`. For information about seemingly unrelated regression, see Section 12.63 and `help(sur)`.

Quantities of Interest

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run:

```
z.out <- zelig(formula=fml, model = "twosls", data)
```

then you may examine the available information in `z.out` by using `names(z.out)`, see the draws from the posterior distribution of the `coefficients` by using `z.out$coefficients`, and view a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below:

- `h`: matrix of all (diagonally stacked) instrumental variables.
- `single.eq.sigma`: different σ^2s for each single equation?.
- `zelig.data`: the input data frame if `save.data = TRUE`.
- `method`: Estimation method.
- `g`: number of equations.
- `n`: total number of observations.
- `k`: total number of coefficients.
- `ki`: total number of linear independent coefficients.
- `df`: degrees of freedom of the whole system.
- `iter`: number of iteration steps.
- `b`: vector of all estimated coefficients.
- `t`: t values for b.
- `se`: estimated standard errors of b.
- `bt`: coefficient vector transformed by TX.
- `p`: p values for b.
- `bcov`: estimated covariance matrix of b.
- `btcov`: covariance matrix of bt.
- `rcov`: estimated residual covariance matrix.
- `drcov`: determinant of `rcov`.
- `rcor`: estimated residual correlation matrix.
- `olsr2`: system OLS R-squared value.
- `y`: vector of all (stacked) endogenous variables.
- `x`: matrix of all (diagonally stacked) regressors.

- `data`: data frame of the whole system (including instruments).
- `TX`: matrix used to transform the regressor matrix.
- `rcovformula`: formula to calculate the estimated residual covariance matrix.
- `probdfsys`: system degrees of freedom to calculate probability values?.
- `solvetol`: tolerance level when inverting a matrix or calculating a determinant.
- `eq`: a list that contains the results that belong to the individual equations.
- `eqnlabel*`: the equation label of the ith equation (from the labels list).
- `formula*`: model formula of the ith equation.
- `n*`: number of observations of the ith equation.
- `k*`: number of coefficients/regressors in the ith equation (including the constant).
- `ki*`: number of linear independent coefficients in the ith equation (including the constant differs from k only if there are restrictions that are not cross equation).
- `df*`: degrees of freedom of the ith equation.
- `b*`: estimated coefficients of the ith equation.
- `se*`: estimated standard errors of b of the ith equation.
- `t*`: t values for b of the ith equation.
- `p*`: p values for b of the ith equation.
- `covb*`: estimated covariance matrix of b of the ith equation.
- `y*`: vector of endogenous variable (response values) of the ith equation.
- `x*`: matrix of regressors (model matrix) of the ith equation.
- `data*`: data frame (including instruments) of the ith equation.
- `fitted*`: vector of fitted values of the ith equation.
- `residuals*`: vector of residuals of the ith equaiton.
- `ssr*`: sum of squared residuals of the ith equation.
- `mse*`: estimated variance of the residuals (mean of squared errors) of the ith equation.
- `s2*`: estimated variance of the residents($\hat{sigma}^2$) of the ith equation.

- `rmse*`: estimated standard error of the reiduals (square root of mse) of the ith equation.
- `s*`: estimated standard error of the residuals ($\hat{\sigma}$) of the ith equation.
- `r2*`: R-squared (coefficient of determination).
- `adjr2*`: adjusted R-squared value.
- `inst*`: instruments of the ith equation.
- `h*`: matrix of instrumental variables of the ith equation.

How to Cite

To cite the *twosls* Zelig model:

> Ferdinand Alimadhi, Ying Lu, and Elena Villalon. 2007. "twosls: Two Stage Least Squares," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The twosls function is adapted from the `systemfit` library (Hamann and Henningsen 2005).

12.68 `weibull`: Weibull Regression for Duration Dependent Variables

Choose the Weibull regression model if the values in your dependent variable are duration observations. The Weibull model relaxes the exponential model's (see Section 12.11) assumption of constant hazard, and allows the hazard rate to increase or decrease monotonically with respect to elapsed time.

Syntax

```
> z.out <- zelig(Surv(Y, C) ~ X1 + X2, model = "weibull", data = mydata)
> x.out <- setx(z.out)
> s.out <- sim(z.out, x = x.out)
```

Weibull models require that the dependent variable be in the form `Surv(Y, C)`, where `Y` and `C` are vectors of length n. For each observation i in 1, ..., n, the value y_i is the duration (lifetime, for example), and the associated c_i is a binary variable such that $c_i = 1$ if the duration is not censored (*e.g.*, the subject dies during the study) or $c_i = 0$ if the duration is censored (*e.g.*, the subject is still alive at the end of the study). If c_i is omitted, all Y are assumed to be completed; that is, time defaults to 1 for all observations.

Input Values

In addition to the standard inputs, `zelig()` takes the following additional options for weibull regression:

- `robust`: defaults to `FALSE`. If `TRUE`, `zelig()` computes robust standard errors based on sandwich estimators (see Huber (1981) and White (1980)) based on the options in `cluster`.

- `cluster`: if `robust = TRUE`, you may select a variable to define groups of correlated observations. Let `x3` be a variable that consists of either discrete numeric values, character strings, or factors that define strata. Then

  ```
  > z.out <- zelig(y ~ x1 + x2, robust = TRUE, cluster = "x3",
                   model = "exp", data = mydata)
  ```

  means that the observations can be correlated within the strata defined by the variable `x3`, and that robust standard errors should be calculated according to those clusters. If `robust = TRUE` but `cluster` is not specified, `zelig()` assumes that each observation falls into its own cluster.

Example

Attach the sample data:

```
> data(coalition)
```

Estimate the model:

```
> z.out <- zelig(Surv(duration, ciep12) ~ fract + numst2, model = "weibull",
+     data = coalition)
```

View the regression output:

```
> summary(z.out)
```

Set the baseline values (with the ruling coalition in the minority) and the alternative values (with the ruling coalition in the majority) for X:

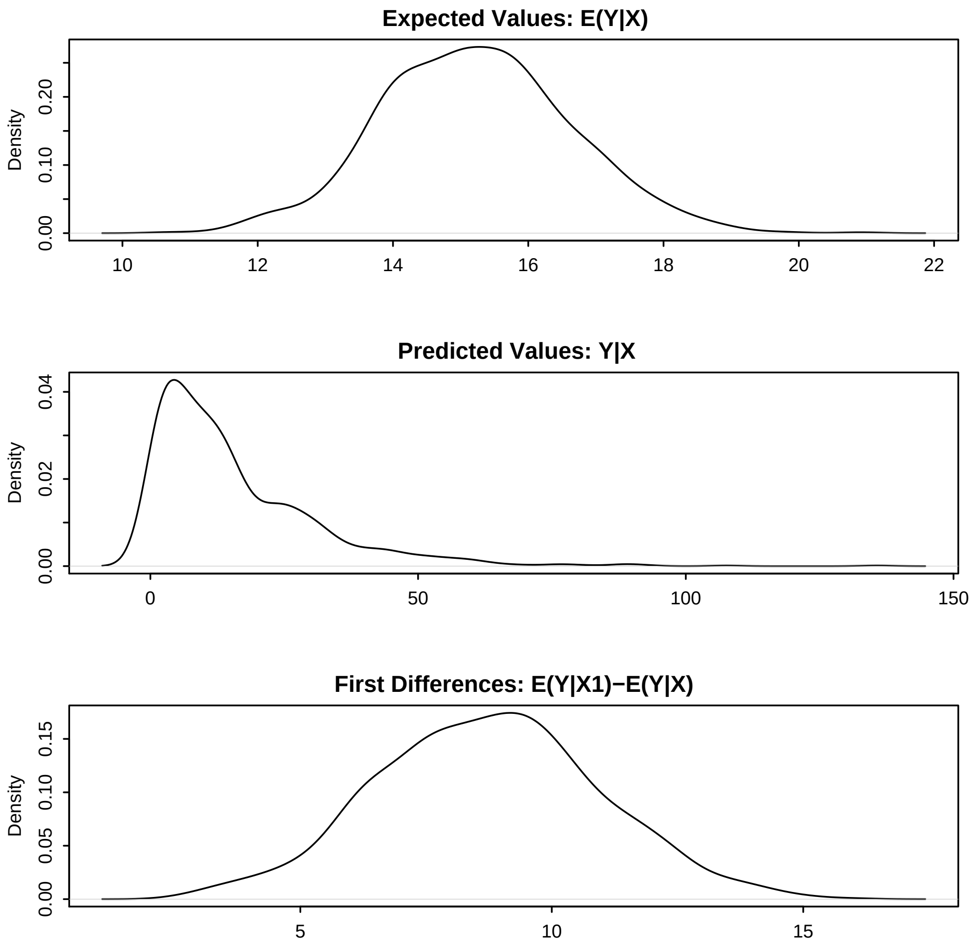
```
> x.low <- setx(z.out, numst2 = 0)
> x.high <- setx(z.out, numst2 = 1)
```

Simulate expected values (`qi$ev`) and first differences (`qi$fd`):

```
> s.out <- sim(z.out, x = x.low, x1 = x.high)
```

```
> summary(s.out)
```

```
> plot(s.out)
```



Model

Let Y_i^* be the survival time for observation i. This variable might be censored for some observations at a fixed time y_c such that the fully observed dependent variable, Y_i, is defined as

$$Y_i = \begin{cases} Y_i^* & \text{if } Y_i^* \leq y_c \\ y_c & \text{if } Y_i^* > y_c \end{cases}$$

- The *stochastic component* is described by the distribution of the partially observed variable Y^*. We assume Y_i^* follows the Weibull distribution whose density function is given by

$$f(y_i^* \mid \lambda_i, \alpha) = \frac{\alpha}{\lambda_i^\alpha} y_i^{*\alpha-1} \exp\left\{-\left(\frac{y_i^*}{\lambda_i}\right)^\alpha\right\}$$

for $y_i^* \geq 0$, the scale parameter $\lambda_i > 0$, and the shape parameter $\alpha > 0$. The mean of this distribution is $\lambda_i \Gamma(1 + 1/\alpha)$. When $\alpha = 1$, the distribution reduces to the exponential distribution (see Section 12.11). (Note that the output from `zelig()` parameterizes `scale`$= 1/\alpha$.)

In addition, survival models like the Weibull have three additional properties. The hazard function $h(t)$ measures the probability of not surviving past time t given survival up to t. In general, the hazard function is equal to $f(t)/S(t)$ where the survival function $S(t) = 1 - \int_0^t f(s)ds$ represents the fraction still surviving at time t. The cumulative hazard function $H(t)$ describes the probability of dying before time t. In general, $H(t) = \int_0^t h(s)ds = -\log S(t)$. In the case of the Weibull model,

$$\begin{aligned} h(t) &= \frac{\alpha}{\lambda_i^\alpha} t^{\alpha-1} \\ S(t) &= \exp\left\{-\left(\frac{t}{\lambda_i}\right)^\alpha\right\} \\ H(t) &= \left(\frac{t}{\lambda_i}\right)^\alpha \end{aligned}$$

For the Weibull model, the hazard function $h(t)$ can increase or decrease monotonically over time.

- The *systematic component* λ_i is modeled as

$$\lambda_i = \exp(x_i \beta),$$

where x_i is the vector of explanatory variables, and β is the vector of coefficients.

Quantities of Interest

- The expected values (`qi$ev`) for the Weibull model are simulations of the expected duration:

$$E(Y) = \lambda_i \, \Gamma(1 + \alpha^{-1}),$$

given draws of β and α from their sampling distributions.

- The predicted value (`qi$pr`) is drawn from a distribution defined by (λ_i, α).

- The first difference (`qi$fd`) in expected value is

$$\text{FD} = E(Y \mid x_1) - E(Y \mid x).$$

- In conditional prediction models, the average expected treatment effect (`att.ev`) for the treatment group is

$$\frac{1}{\sum_{i=1}^n t_i} \sum_{i:t_i=1}^n \left\{ Y_i(t_i = 1) - E[Y_i(t_i = 0)] \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. When $Y_i(t_i = 1)$ is censored rather than observed, we replace it with a simulation from the model given available knowledge of the censoring process. Variation in the simulations are due to uncertainty in simulating $E[Y_i(t_i = 0)]$, the counterfactual expected value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

- In conditional prediction models, the average predicted treatment effect (`att.pr`) for the treatment group is

$$\frac{1}{\sum_{i=1}^{n} t_i} \sum_{i:t_i=1}^{n} \left\{ Y_i(t_i = 1) - \widehat{Y_i(t_i = 0)} \right\},$$

where t_i is a binary explanatory variable defining the treatment ($t_i = 1$) and control ($t_i = 0$) groups. When $Y_i(t_i = 1)$ is censored rather than observed, we replace it with a simulation from the model given available knowledge of the censoring process. Variation in the simulations are due to uncertainty in simulating $\widehat{Y_i(t_i = 0)}$, the counterfactual predicted value of Y_i for observations in the treatment group, under the assumption that everything stays the same except that the treatment indicator is switched to $t_i = 0$.

Output Values

The output of each Zelig command contains useful information which you may view. For example, if you run `z.out <- zelig(y ~ x, model = "weibull", data)`, then you may examine the available information in `z.out` by using `names(z.out)`, see the `coefficients` by using `z.out$coefficients`, and a default summary of information through `summary(z.out)`. Other elements available through the `$` operator are listed below.

- From the `zelig()` output object `z.out`, you may extract:
  - `coefficients`: parameter estimates for the explanatory variables.
  - `icoef`: parameter estimates for the intercept and "scale" parameter $1/\alpha$.
  - `var`: the variance-covariance matrix.
  - `loglik`: a vector containing the log-likelihood for the model and intercept only (respectively).
  - `linear.predictors`: a vector of the $x_i\beta$.
  - `df.residual`: the residual degrees of freedom.
  - `df.null`: the residual degrees of freedom for the null model.
  - `zelig.data`: the input data frame if `save.data = TRUE`.

- Most of this may be conveniently summarized using `summary(z.out)`. From `summary(z.out)`, you may additionally extract:
  - `table`: the parameter estimates with their associated standard errors, p-values, and t-statistics.
- From the `sim()` output object `s.out`, you may extract quantities of interest arranged as matrices indexed by simulation $\times$ `x`-observation (for more than one `x`-observation). Available quantities are:
  - `qi$ev`: the simulated expected values for the specified values of `x`.
  - `qi$pr`: the simulated predicted values drawn from a distribution defined by (λ_i, α).
  - `qi$fd`: the simulated first differences between the simulated expected values for `x` and `x1`.
  - `qi$att.ev`: the simulated average expected treatment effect for the treated from conditional prediction models.
  - `qi$att.pr`: the simulated average predicted treatment effect for the treated from conditional prediction models.

How to Cite

To cite the *weibull* Zelig model:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "weibull: Weibull Regression for Duration Dependent Variables," in Kosuke Imai, Gary King, and Olivia Lau, "Zelig: Everyone's Statistical Software," `http://gking.harvard.edu/zelig`.

To cite Zelig as a whole, please reference these two sources:

> Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software," `http://GKing.harvard.edu/zelig`.

> Kosuke Imai, Gary King, and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development," *Journal of Computational and Graphical Statistics*, forthcoming, `http://gking.harvard.edu/files/abs/z-abs.shtml`.

See also

The Weibull model is part of the survival library by Terry Therneau, ported to R by Thomas Lumley. Advanced users may wish to refer to `help(survfit)` in the survival library, and Venables and Ripley (2002). Sample data are from King et al. (1990a).

Chapter 13

Commands for Programmers and Contributors

13.1 `model.end`: Cleaning up after optimization

Description

The `model.end()` function creates a list of regression output from `optim()` output. The list includes coefficients (from the `optim() par` output), a variance-covariance matrix (from the `optim()` Hessian output), and any terms, contrasts, or xlevels (from the model frame). Use `model.end()` after calling `optim()`, but before assigning a class to the regression output.

Syntax

```
model.end(res, mf)
```

Arguments

- `res`: the output from `optim()` or another fitting-algorithm.
- `mf`: the model frame output by `model.frame()`.

Output Values

A list of regression output, including:

- `coefficients`: the optimized parameters.
- `variance`: the variance-covariance matrix (the negative inverse of the Hessian matrix returned from the optimization procedure).
- `terms`: the terms object. See `help(terms.object)` for more information.
- `...`: additional elements passed from `res`.

See Also

- Section 8 for an overview of how to write a new model.

Contributors

Kosuke Imai, Gary King, Olivia Lau, and Ferdinand Alimadhi.

13.2 `model.frame.multiple`: Extracting the "environment" of a model formula

Description

Use `model.frame.multiple()` after `parse.par()` to create a data frame of the unique variables identified in the formula (or list of formulas).

Syntax

```
model.frame.multiple(formula, data, eqn = NULL, ...)
```

Arguments

- `formula`: a list of formulas of class `"multiple"`, returned from `parse.par()`.
- `data`: a data frame containing all the variables used in `formula`.
- `eqn`: an optional character string or vector of character strings specifying the equations for which you would like to extract variables. Defaults to `NULL`, which pulls out all the variables for all equations in `formula`.
- `...`: additional arguments passed to `model.frame.default()`.

Output Values

The output is a data frame (with a terms attribute) containing all the unique explanatory and response variables identified in the list of formulas. By default, missing (`NA`) values are listwise deleted.

If `as.factor()` appears on the left-hand side, the response variables will be returned as an indicator (0/1) matrix with columns corresponding to the unique levels in the factor variable.

If any formula contains a `tag()`, `model.frame.multiple()` will return the original variable in the data frame and use the `tag()` information in the terms attribute only.

Examples

```
formulae <- list(import ~ coop + cost + target,
                 export ~ coop + cost + target)
fml <- parse.formula(formulae, model = "bivariate.logit")
D <- model.frame(fml, data = mydata)
```

Since the output from `parse.formula()` is of class `"multiple"`, you do not need to call `model.frame.multiple()` explicitly, but can use the generic `model.frame()` instead.

See Also

- Section 13.4 for `parse.formula()`
- Section 8.1 for an overview of the user-interface.

Contributors

Kosuke Imai, Gary King, Olivia Lau, and Ferdinand Alimadhi.

13.3 `model.matrix.multiple`: Design matrix for multivariate models

Description

Use `model.matrix.multiple()` after `parse.formula()` to create a design matrix for multiple-equation models.

Syntax

```
model.matrix(object, data, shape = "compact", eqn = NULL, ...)
```

Arguments

- `object`: the list of formulas output from `parse.formula()`.
- `data`: a data frame created with `model.frame.multiple()`.
- `shape`: a character string specifying the shape of the outputed matrix. Available options are:
  - `"compact"`: (default) the output matrix will be an $n \times v$, where v is the number of unique variables in all of the equations (including the intercept term).
  - `"array"`: the output is an $n \times K \times J$ array where J is the total number of equations and K is the total number of parameters across all the equations. If a variable is not in a certain equation, it is observed as a vector of 0s.
  - `"stacked"`: the output will be a $2n \times K$ matrix where K is the total number of parameters across all the equations.
- `eqn`: a character string or a vector of character strings identifying the equations from which to construct the design matrix. The defaults to `NULL`, which only uses the systematic parameters (for which `DepVar = TRUE` in the appropriate `describe.model()`) .
- `...`: additional arguments passed to `model.matrix.default()`.

Output Values

A design matrix or array, depending on the options chosen in `shape`, with appropriate terms attributes.

Examples

Let's say that the name of the model is `"bivariate.probit"`, and the corresponding describe function is `describe.bivariate.probit()`, which identifies `mu1` and `mu2` as systematic components, and an ancillary parameter `rho`, which may be parameterized, but is estimated as a scalar by default. Let `par` be the parameter vector (including parameters for `rho`), `formulae` a user-specified formula given in one of the formats in Table 13.1, and `mydata` the user specified data frame.

Acceptable combinations of `parse.par()` and `model.matrix()` are as follows:

```
## Setting up
fml <- parse.formula(formulae, model = "bivariate.probit")
D <- model.frame(fml, data = mydata)
terms <- terms(D)

## Intuitive option
Beta <- parse.par(par, terms, shape = "vector", eqn = c("mu1", "mu2"))
X <- model.matrix(fml, data = D, shape = "stacked", eqn = c("mu1", "mu2")
eta <- X %*% Beta

## Memory-efficient (compact) option (default)
Beta <- parse.par(par, terms, eqn = c("mu1", "mu2"))
X <- model.matrix(fml, data = D, eqn = c("mu1", "mu2"))
eta <- X %*% Beta

## Computationally-efficient (array) option
Beta <- parse.par(par, terms, shape = "vector", eqn = c("mu1", "mu2"))
X <- model.matrix(fml, data = D, shape = "array", eqn = c("mu1", "mu2"))
eta <- apply(X, 3, '%*%', Beta)
```

In each case, `eta` is an $n \times 2$ matrix with columns corresponding to the linear predictors for `mu1` and `mu2`, respectively.

See Also

- Section 13.5 for selecting and shaping parameter vectors.
- Section 8 for examples of how to write new models.

Contributors

Kosuke Imai, Gary King, Olivia Lau, and Ferdinand Alimadhi.

13.4 parse.formula: Parsing the inputs

Description

Parse the input formula (or list of formulas) into the standard format described below. Since labels for this format will vary by model, `parse.formula()` will evaluate a function `describe.model()`, where `model` is given as an input to `parse.formula()`.

If the `check.model()` function has more than one parameter for which `ExpVar = TRUE` and `DepVar = TRUE`, then the user-specified equations must have labels to match those parameters, else `parse.formula()` should return an error. In addition, if the formula entries are not unambiguous, then `parse.formula()` should return an error.

Syntax

```
> fml <- parse.formula(formula, model, data = NULL)
```

Arguments

- `formula`: either a single formula or a list of formula objects.
- `model`: a character string specifying the name of the model.
- `data`: an optional data frame for models that require a factor response variable.

Output Values

The output is a list of formula objects with class `("multiple", "list")`. Let's say that the name of the model is `"bivariate.probit"`, and the corresponding describe function is `describe.bivariate.probit()`, which identifies `mu1` and `mu2` as systematic components, and an ancillary parameter `rho`, which may be parameterized, but is estimated as a scalar by default. Given this model, Table 13.1 gives acceptable user inputs.

Examples

```
formulae <- list(cbind(import, export) ~ coop + cost + target)
fml <- parse.formula(formulae, model = "bivariate.probit")
D <- model.frame(fml, data = mydata)
```

See Also

- Section 8.1 for commented examples of how `parse.formula()` and `describe.model()` work together.
- Section 13.8 for constraints between coefficients in a multiple equation context.

Contributors

Kosuke Imai, Gary King, Olivia Lau, and Ferdinand Alimadhi.

Table 13.1: Examples of acceptable short-hand for user-specified formulas, using bivariate probit as an example

	User Input	Output from `parse.formula()`

Same covariates, separate effects	`cbind(y1, y2) ~ x1 + x2 * x3`	`list(mu1 = y1 ~ x1 + x2 * x3,`
		`mu2 = y2 ~ x1 + x2 * x3,`
		`rho = ~ 1)`

With ρ as a systematic equation	`list(cbind(y1, y2) ~ x1 + x2,`
	`rho = ~ x4 + x5)`
		`list(mu1 = y1 ~ x1 + x2,`
		`mu2 = y2 ~ x1 + x2,`
		`rho = ~ x4 + x5)`

With constraints (same variable)	`list(mu1 = y1 ~ x1 + tag(x2, "x2"),`
	`mu2 = y2 ~ x3 + tag(x2, "x2"))`
		`list(mu1 = y1 ~ x1 + tag(x2, "x2"),`
		`mu2 = y2 ~ x3 + tag(x2, "x2"),`
		`rho = ~ 1)`

With constraints (different variables)	`list(mu1 = y1 ~ x1 + tag(x2, "z1"),`
	`mu2 = y2 ~ x3 + tag(x4, "z1"))`
		`list(mu1 = y1 ~ x1 + tag(x2, "z1"),`
		`mu2 = y2 ~ x3 + tag(x4, "z1"),`
		`rho = ~ 1)`

13.5 parse.par: Select and reshape parameter vectors

Description

The `parse.par()` function reshapes parameter vectors for compatability with the output matrix from `model.matrix.multiple()`. (SeeSection 13.3.) Use `parse.par()` to identify sets of parameters; for example, within optimization functions that require vector input, or within `qi()` functions that take matrix input of all parameters as a lump.

Syntax

```
parse.par(par, terms, shape = "matrix", eqn = NULL)
```

Arguments

- `par`: the vector (or matrix) of parameters.
- `terms`: the terms from either `model.frame.*()` or `model.matrix.*()`.
- `shape`: a character string (either `"matrix"` or `"vector"`) that identifies the type of output structure.
- `eqn`: a character string (or strings) that identify the parameters that you would like to subset from the larger `par` structure.

Output Values

A matrix or vector of the sub-setted (and reshaped) parameters for the specified parameters given in `eqn`. By default, `eqn = NULL`, such that all systematic components are selected. (Systematic components have `ExpVar = TRUE` in the appropriate `describe.model()` function.)

If an ancillary parameter (for which `ExpVar = FALSE` in `describe.model()`) is specified in `eqn`, it is always returned as a vector (ignoring `shape`). (Ancillary parameters are all parameters that have intercept only formulas.)

Examples

Let's say that the name of the model is `"bivariate.probit"`, and the corresponding describe function is `describe.bivariate.probit()`, which identifies `mu1` and `mu2` as systematic components, and an ancillary parameter `rho`, which may be parameterized, but is estimated as a scalar by default. Let `par` be the parameter vector (including parameters for `rho`), `formulae` a user-specified formula given in one of the formats in Table 13.1, and `mydata` the user specified data frame.

Acceptable combinations of `parse.par()` and `model.matrix()` are as follows:

```
## Setting up
fml <- parse.formula(formulae, model = "bivariate.probit")
D <- model.frame(fml, data = mydata)
terms <- terms(D)

## Intuitive option
Beta <- parse.par(par, terms, shape = "vector", eqn = c("mu1", "mu2"))
X <- model.matrix(fml, data = D, shape = "stacked", eqn = c("mu1", "mu2")
eta <- X %*% Beta

## Memory-efficient (compact) option (default)
Beta <- parse.par(par, terms, eqn = c("mu1", "mu2"))
X <- model.matrix(fml, data = D, eqn = c("mu1", "mu2"))
eta <- X %*% Beta

## Computationally-efficient (array) option
Beta <- parse.par(par, terms, shape = "vector", eqn = c("mu1", "mu2"))
X <- model.matrix(fml, data = D, shape = "array", eqn = c("mu1", "mu2"))
eta <- apply(X, 3, '%*%', Beta)
```

In each case, `eta` is an $n \times 2$ matrix with columns corresponding to the linear predictors for `mu1` and `mu2`, respectively.

See Also

- Section 13.3 for a description of how multiple equation models work with `model.matrix()`.
- Section 8.2 for more detail on the three combinations (intuitive, memory-efficient, and computationally-efficient) methods of multiplying matrices of parameters and variables.

Contributors

Kosuke Imai, Gary King, Olivia Lau, and Ferdinand Alimadhi.

13.6 `put.start`: Set specific starting values for certain parameters

Description

After calling `set.start()` to create default starting values, use `put.start()` to change starting values for specific parameters or parameter sets.

Syntax

```
put.start(start.val, value, terms, eqn)
```

Arguments

- `start.val`: the vector of starting values created by `set.start()`.
- `value`: the scalar or vector of replacement starting values.
- `terms`: the terms output from `model.frame.multiple()`.
- `eqn`: the parameters for which you would like to replace the default values with `value`.

Output Values

A vector of starting values (of the same length as `start.val`).

See Also

- Section 13.7 to set default starting values.
- Section 8 for an overview of the procedure to add models to Zelig.

Contributors

Kosuke Imai, Gary King, Olivia Lau, and Ferdinand Alimadhi.

13.7 `set.start`: Set starting values for all parameters

Description

After using `parse.par()` and `model.matrix()`, use `set.start()` to set starting values for all parameters. By default, starting values are set to 0. If you wish to select alternative starting values for certain parameters, use `put.start()` after `set.start()`.

Syntax

```
set.start(start.val = NULL, terms)
```

Arguments

- `start.val`: user-specified starting values. If `NULL` (default), the default starting values for all parameters are set to 0.
- `terms`: the terms output from `model.frame.multiple()`.

Output Values

A named vector of starting values for all parameters specified in `terms`, defaulting to 0.

Example

```
fml <- parse.formula(formula, model = "bivariate.probit")
D <- model.frame(fml, data = data)
terms <- terms(D)
start.val <- set.start(start.val = NULL, terms)
```

See Also

- Section 13.6 to change starting values for specific parameter sets.
- Section 8 for detailed examples of writing new models.

Contributors

Kosuke Imai, Gary King, Olivia Lau and Ferdinand Alimadhi.

13.8 `tag`: Constrain parameter effects across equations

Description

Use `tag()` to identify parameters and constrain their effects across equations in multiple-equation models.

Syntax

```
tag(x, label)
```

Arguments

- `x`: the variable to be constrained.
- `label`: the name that the constrained variable takes.

Output Values

While there is no specific output from `tag()` itself, `parse.formula()` uses `tag()` to identify parameter constraints across equations, when a model takes more than one systematic component.

Examples

See Also

- Section 8.1 for an overview of the multiple-equation user-interface.
- Section 13.4 for more examples of acceptable uses for `tag()` in formulas.

Contributors

Kosuke Imai, Gary King, Olivia Lau, and Ferdinand Alimadhi.

Part IV

Appendices

Appendix A

What's New? What's Next?

A.1 What's New: Zelig Release Notes

Releases listed as "stable releases" have been tested against prior versions of Zelig for consistency and accuracy. Testing distributions may contain bugs, but are usually replaced by stable releases within a few days.

- **3.3-1** (June 12, 2008): Bug-fix release for R.2.7.0. A bug fix for plot.ci() so that it works with mixed effects models (thanks to Keith Schnakenberg).

- **3.3-0** (June 03, 2008): Stable release for R.2.7.0. Updated coxph so that it handles time-varying covariates (contributed by Patrick Lam). A new plot function for survival models (contributed by John Graves). First version dependencies are as follows:

"MASS"	"7.2-41"
"nlme"	"3.1-87"
"survival"	"2.34"
"coda"	"0.13-1"
"sna"	"1.5"
"boot"	"1.2-31"
"nnet"	"7.2-41"
"zoo"	"1.5-0"
"sandwich"	"2.1-0"
"lme4"	"0.99875-9"
"systemfit"	"1.0-2"
"VGAM"	"0.7-5"
"MCMCpack"	"0.8-2"
"mvtnorm"	"0.8-3"
"gee"	"4.13-13"
"mgcv"	"1.3-29"
"anchors"	"1.9-2"
"survey"	"3.6-13"

- **3.2-1** (April 10, 2008): Bug-fix release for R.2.6.0-2.6.2. Fixed the setx() bug for multiply imputed data sets. (Thanks to Steve Shewfelt and Keith Schnakenberg.)

- **3.2** (April 3, 2008): Stable release for R 2.6.0-2.6.2. Adding models for survey data – `normal.survey`, `logit.survey`, `probit.survey`, `poisson.survey`, `gamma.survey`. First version dependencies are as follows:

survey	3.6-13
MASS	7.2-34
nlme	3.1-86
survival	2.34
boot	1.2-30
nnet	7.2-34
zoo	1.4-0
sandwich	2.0-2
sna	1.4
lme4	0.99875-9
coda	0.12-1
systemfit	0.8-5
VGAM	0.7-4
MCMCpack	0.8-2
mvtnorm	0.8-1
gee	4.13-13
mgcv	1.3-29
anchors	2.0

- **3.1-1** (January 10, 2008): Bug-fix release for R 2.6.0-2.6.1. Fixed bugs, improved the code and the documentation for mixed effects models. Thanks to Gregor Gorjanc. Fixed systemfit models due to some API changes in systemfit package. Added some other models (including *.mixed models) in plot.ci

- **3.1** (November 30, 2007): Stable release for R 2.6.0-2.6.1. Adding many new models such as `aov`, `chopit`, `coxph`, generalized linear mixed-effects models, and gee models. Also, several bugs are fixed. First version dependencies are as follows:

MASS	7.2-34
nlme	3.1-86
survival	2.34
boot	1.2-30
nnet	7.2-34
zoo	1.4-0
sandwich	2.0-2
sna	1.4
lme4	0.99875-9
coda	0.12-1
systemfit	0.8-5
VGAM	0.7-4
MCMCpack	0.8-2
mvtnorm	0.8-1
gee	4.13-13
mgcv	1.3-29
anchors	2.0

- **3.0-1** – **3.0-6**: Minor bug fixes. Stable release for R 2.5.0-2.5.1.

- **3.0** (July 20, 2007): Stable release for R 2.5.0-2.5.1. Introducing vignettes for each model. Improving documentation in the Zelig web site, improving citation style, improving `help.zelig()` function, adding gam models, social network methods, logit gee model, adding support for cross-validation procedures and diagnostics tools, etc.

- **2.8-3** (May 29, 2007): Stable release for R 2.4.0-2.5.0. Fixed bugs in `help.zelig()`, and summary for multinomial logit, bivariate probit, and bivariate logit with multiple imputation. (Thanks to Brant Inman and Javier Marquez.) First version dependencies are as follows:

MASS	7.2-34
boot	1.2-27
VGAM	0.7-1
MCMCpack	0.8-2
mvtnorm	0.7-5
survival	2.31
sandwich	2.0-0
zoo	1.2-1
coda	0.10-7
nnet	7.2-34
sna	1.4

- **2.8-2** (March 3, 2007): Stable release for R 2.4.0-2.4.1. Fixed bug in ARIMA simulation process.

- **2.8-1** (February 21, 2007): Stable release for R 2.4.0-2.4.1. Made `setx()` compatible with ordred factor variables (thanks to Mike Ward and Kirill Kalinin). First order dependencies as in version 2.8-1.

- **2.8-0** (February 12, 2007): Stable release for R 2.4.0-2.4.1. Released ARIMA models and network analysis models (least squares and logit) for sociomatrices. First level dependencies are as follows:

MASS	7.2-31
boot	1.2-27
VGAM	0.7-1
MCMCpack	0.7-4
mvtnorm	0.7-5
survival	2.31
sandwich	2.0-0
zoo	1.2-1
coda	0.10-7
nnet	7.2-31
sna	1.4

- **2.7-5** (December 25, 2006): Stable release for R 2.4.0-2.4.1. Fixed bug related to `names.default()`, summary for multiple imputation methods, and prediction for ordinal response models (thanks to Brian Ripley, Chris Lawrence, and Ian Yohai).

- **2.7-4** (November 10, 2006): Stable release for R 2.4.0. Fixed bugs related to R check.

- **2.7-3** (November 9, 2006): Stable release for R 2.4.0. Fixed bugs related to R check.

- **2.7-2** (November 5, 2006): Stable release for R 2.4.0. Temporarily removed ARIMA models.

- **2.7-1** (November 3, 2006): Stable release for R 2.4.0. Made changes regarding the S4 classes in VGAM. The ARIMA (`arima`) model for time series data added by Justin Grimmer. First level dependencies are as follows:

MASS	7.2-29
boot	1.2-26
VGAM	0.7-1
MCMCpack	0.7-4
mvtnorm	0.7-5
survival	2.29
sandwich	2.0-0
zoo	1.2-1
coda	0.10-7

- **2.6-5** (September 14, 2006): Stable release for R 2.3.0-2.3.1. Fixed bugs in bivariate logit, bivariate probit, multinomial logit, and model.matrix.multiple (related to changes in version 2.6-4, but not previous versions, thanks to Chris Lawrence). First level dependencies are as follows:

MASS	7.2-27.1
boot	1.2-26
VGAM	0.6-9
MCMCpack	0.7-1
mvtnorm	0.7-2
survival	2.28
sandwich	1.1-1
zoo	1.0-6
coda	0.10-5

- **2.6-4** (September 8, 2006): Stable release for R 2.3.0-2.3.1. Fixed bugs in `setx()`, and bugs related to `multiple` and the multinomial logit model. Added instructions for installing Fortran tools for Intel macs. Added the R×C ecological inference model. (thanks to Kurt Hornik, Luke Keele, Joerg Mueller-Scheessel, and B. Dan Wood)

- **2.6-3** (June 19, 2006): Stable release for R 2.0.0-2.3.1. Fixed bug in VDC interface functions, and `parse.formula()`. (thanks to Micah Altman, Christopher N. Lawrence, and Eric Kostello)

- **2.6-2** (June 7, 2006): Stable release for R 2.0.0-2.3.1. Removed R × C EI. Changed `data = list()` to `data = mi()` for multiply-imputed data frames. First level version compatibilities are as for version 2.6-1.

- **2.6-1** (April 29, 2006): Stable release for R 2.0.0-2.2.1. Fixed major bug in ordinal logit and ordinal probit expected value simulation procedure (does not affect Bayesian ordinal probit). (reported by Ian Yohai) Added the following ecological inference EI models: Bayesian hierarchical EI, Bayesian dynamic EI, and R × C EI. First level version compatibilities (at time of release) are as follows:

MASS	7.2-24
boot	1.2-24
VGAM	0.6-8
MCMCpack	0.7-1
mvtnorm	0.7-2
survival	2.24
sandwich	1.1-1
zoo	1.0-6
coda	0.10-5

- **2.5-4** (March 16, 2006): Stable release for R 2.0.0-2.2.1. Fixed bug related to windows build. First-level dependencies are the same as in version 2.5-1.

- **2.5-3** (March 9, 2006): Stable release for R 2.0.0-2.2.1. Fixed bugs related to VDC GUI. First level dependencies are the same as in version 2.5-1.

- **2.5-2** (February 3, 2006): Stable release for R 2.0.0-2.2.1. Fixed bugs related to VDC GUI. First level dependencies are the same as in version 2.5-1.

- **2.5-1** (January 31, 2006): Stable release for R 2.0.0-2.2.1. Added methods for multiple equation models. Added tobit regression. Fixed bugs related to robust estimation and upgrade of sandwich and zoo packages. Revised `setx()` to use environments. Added `current.packages()` to retrieve version of packages upon which Zelig depends. First level version compatibilities (at time of release) are as follows:

MASS	7.2-24
boot	1.2-24
VGAM	0.6-7
mvtnorm	0.7-2
survival	2.20
sandwich	1.1-0
zoo	1.0-4
MCMCpack	0.6-6
coda	0.10-3

- **2.4-7** (December 10, 2005): Stable release for R 2.0.0-2.2.2. Fixed the environment of `eval()` called within `setx.default()` (thanks to Micah Altman).

- **2.4-6** (October 27, 2005): Stable release for R 2.0.0-2.2.2. Fixed bug related to simulation for Bayesian Normal regression.

- **2.4-5** (October 18, 2005): Stable release for R 2.0.0-2.2.0. Fixed installation instructions.

- **2.4-4** (September 29, 2005): Stable release for R 2.0.0-2.2.0. Fixed `help.zelig()` links.

- **2.4-3** (September 29, 2005): Stable release for R 2.0.0-2.2.0. Revised `matchit()` documentation.

- **2.4-2** (August 30, 2005): Stable release for R 2.0.0-2.1.1. Fixed bug in `setx()` related to `as.factor()` and `I()`. Streamlined `qi.survreg()`.

- **2.4-1** (August 15, 2005): Stable release for R 2.0.0-2.1.1. Added the following Bayesian models: factor analysis, mixed factor analysis, ordinal factor analysis, unidimensional item response theory, k-dimensional item response theory, logit, multinomial logit, normal, ordinal probit, Poisson, and tobit. Also fixed minor bug in formula (long variable names coerced to list).

- **2.3-2** (August 5, 2005): Stable release for R 2.0.0-2.1.1. Fixed bug in simulation procedure for lognormal model.

- **2.3-1** (August 4, 2005): Stable release for R 2.0.0-2.1.1. Fixed documentation errors related to model parameterization and code bugs related to first differences and conditional prediction for exponential, lognormal, and Weibull models. (reported by Alison Post)

- **2.2-4** (July 30, 2005): Stable release for R 2.0.0-2.1.1. Revised relogit, adding option for weighting in addition to prior correction. (reported by Martin Plöderl)

- **2.2-3** (July 24, 2005): Stable release for R 2.0.0-2.1.1. Fixed bug associated with robust standard errors for negative binomial.

- **2.2-2** (July 13, 2005): Stable release for R 2.0.0-2.1.1. Fixed bug in setx(). (reported by Ying Lu)

- **2.2-1** (July 11, 2005): Stable release for R 2.0.0-2.1.0. Revised ordinal probit to use MASS library. Added robust standard errors for the following regression models: exponential, gamma, logit, lognormal, least squares, negative binomial, normal (Gaussian), poisson, probit, and weibull.

- **2.1-4** (May 22, 2005): Stable release for R 1.9.1-2.1.0. Revised help.zelig() to deal with CRAN build of Windows version. Added recode of slots to lists in NAMESPACE. Revised `install.R` script to deal with changes to `install.packages()`. (reported by Dan Powers and Ying Lu)

- **2.1-3** (May 9, 2005): Stable release for R 1.9.1-2.1.0. Revised param.lm() function to work with bootstrap simulation. (reported by Jens Hainmueller)

- **2.1-2** (April 14, 2005): Stable release for R 1.9.1-2.1.0. Revised summary.zelig().

- **2.1-1** (April 7, 2005): Stable release for R 1.9.1-2.1.0. Fixed bugs in NAMESPACE and summary.vglm().

- **2.0-14** (April 5, 2005): Stable release for R 1.9.1-2.0.1. Added `summary.vglm()` to ensure the compatibility with VGAM 0.6-2.

- **2.0-13** (March 11, 2005): Stable release for R 1.9.1-2.0.1. Fixed bugs in `NAMESPACE` and R-help file for `rocplot()`.

- **2.0-12** (February 20, 2005): Stable release for R 1.9.1-2.0.1. Added `plot = TRUE` option to `rocplot()`.

- **2.0-11** (January 14, 2005): Stable release for R 1.9.1-2.0.1. Changed class name for subsettted models from `"multiple"` to `"strata"`, and modified affected functions.

- **2.0-10** (January 5, 2005): Stable release for R 1.9.1 and R 2.0.0. Fixed bug in ordinal logit simulation procedure. (reported by Ian Yohai)

- **2.0-9** (October 21, 2004): Stable release for R 1.9.1 *and* R 2.0.0 (Linux and Windows). Fixed bug in NAMESPACE file.

- **2.0-8** (October 18, 2004): Stable release for R 1.9.1 *and* R 2.0.0 (Linux only). Revised for submission to CRAN.

- **2.0-7** (October 14, 2004): Stable release for R 1.9.1 *and* R 2.0.0 (Linux only). Fixed bugs in `summary.zelig()`, NAMESPACE, and assorted bugs related to new R release. Revised syntax for multiple equation models.

- **2.0-6** (October 4, 2004): Stable release for R 1.9.1. Fixed problem with NAMESPACE.

- **2.0-5** (September 25, 2004): Stable release for R 1.9.1. Changed installation procedure to source `install.R` from Zelig website.

- **2.0-4** (September 22, 2004): Stable release for R 1.9.1. Fixed typo in installation directions, implemented NAMESPACE, rationalized `summary.zelig()`, and tweaked documentation for least squares.

- **2.0-3** (September 1, 2004): Stable release for R 1.9.1. Fixed bug in conditional prediction for survival models.

- **2.0-2** (August 25, 2004): Stable release for R 1.9.1. Removed predicted values from `ls`.

- **2.0-1b** (July 16, 2004): Stable release for R 1.9.1. MD5 checksum problem fixed. Revised `plot.zelig()` command to be a generic function with methods assigned by the model. Revised entire architecture to accept multiply imputed data sets with strata. Added functions to simplify adding models. Completely restructured reference manual. Fixed bugs related to conditional prediction in setx and summarizing strata in summary.zelig.

- **1.1-2** (June 24, 2004): Stable release for R 1.9.1 (MD5 checksum problem not fixed, but does not seem to cause problems). Fixed bug in `help.zelig()`. (reported by Michael L. Levitan)

- **1.1-1** (June 14, 2004): Stable release for R 1.9.0. Revised `zelig()` procedure to use `zelig2model()` wrappers, revised `help.zelig()` to use a data file with extension `.url.tab`, and revised `setx()` procedure to take a list of `fn` to apply to variables, and such that `fn = NULL` returns the entire `model.matrix()`.

- **1.0-8** (May 27, 2004): Stable release for R 1.9.0. Fixed bug in simulation procedure for survival models. (reported by Elizabeth Stuart)

- **1.0-7** (May 26, 2004): Stable release for R 1.9.0. Fixed bug in relogit simulation procedure. (reported by Tom Vanwellingham)

- **1.0-6** (May 11, 2004): Stable release for R 1.9.0. Fixed bug in setx.default, which had previously failed to ignore extraneous variables in data frame. (reported by Steve Purpura)
- **1.0-5** (May 7, 2004): Replaced relogit procedure with memory-efficient version. (reported by Tom Vanwellingham)
- **1.0-4** (April 19, 2004): Stable release for R 1.9.0. Added vcov.lm method; changed print for summary.relogit.
- **1.0-2** (April 16, 2004): Testing distribution for R 1.9.0.
- **1.0-1** (March, 23, 2004): Stable release for R 1.8.1.

A.2 What's Next?

We have several plans for expanding and improving Zelig. Major changes slated for Version 3.0 (and beyond) include:

- Hierarchical and multi-level models
- Ecological inference models
- GEE models
- Neural network models
- Average treatment effects for everyone (treated and control units)
- Time-series cross-sectional models (via `nlme`)
- Generalized boosted regression model (via `gbm`)
- Saving random seeds to ensure exact replication

If you have suggestions, or packages that you would like to contribute to Zelig, please email our listserv at zelig@lists.gking.harvard.edu.

Appendix B

Frequently Asked Questions

B.1 For All Zelig Users

How do I cite Zelig?

We would appreciate if you would cite Zelig as:

> Imai, Kosuke, Gary King and Olivia Lau. 2006. "Zelig: Everyone's Statistical Software," `http://GKing.Harvard.Edu/zelig`.

Please also cite the contributors for the models or methods you are using. These citations can be found in the contributors section of each model or command page.

Why can't I install Zelig?

You must be connected to the internet to install packages from web depositories. In addition, there are a few platform-specific reasons why you may have installation problems:

- **On Windows**: If you are using the very latest version of R, you may not be able to install Zelig until we update Zelig to work on the latest release of R. If you wish to install Zelig in the interim, check the Zelig release notes (Section A.1) and download the appropriate version of R to work with the last release of Zelig. You may have to manually download and install Zelig.
- **On Mac**: If the latest version of Zelig is not yet available at CRAN but you would like to install it on your Mac, try typing the following at your R prompt:

  ```
  install.packages("Zelig", repos = "http://gking.harvard.edu", type = "source")
  ```

- **On Mac or Linux systems**: If you get the following warning message at the end of your installation:

  ```
  Installation of package VGAM had non-zero exit status in ...
  ```

this means that you were not able to install VGAM properly. Make sure that you have the g77 Fortran compiler. For PowerPC Macs, download g77 from http://hpc.sourceforge.net). For Intel Macs, download the xcode Apple developer tools. After installation, try to install Zelig again.

Why can't I install R?

If you have problems installing R (rather than Zelig), you should check the R FAQs for your platform. If you still have problems, you can search the archives for the R help mailing list, or email the list directly at r-help@stat.math.ethz.ch.

Why can't I load data?

When you start R, you need to specify your working directory. In linux R, this is done pretty much automatically when you start R, whether within ESS or in a terminal window. In Windows R, you may wish to specify a working directory so that you may load data without typing in long directory paths to your data files, and it is important to remember that *Windows* R uses the *Linux* directory delimiter. That is, if you right click and select the "Properties" link on a Windows file, the slashes are backslashes (\), but Windows R uses forward slashes (/) in directory paths. Thus, the Windows link may be `C:\Program Files\R\R-2.5.1\`, but you would type `C:/Program Files/R/R-2.5.1/` in Windows R.

When you start R in Windows, the working directory is by default the directory in which the R executible is located.

```
# Print your current working directory.
> getwd()

# To read data not located in your working directory.
> data <- read.table("C:/Program Files/R/newwork/mydata.tab")

# To change your working directory.
> setwd("C:/Program Files/R/newwork")

# Reading data in your working directory.
> data <- read.data("mydata.tab")
```

Once you have set the working directory, you no longer need to type the entire directory path.

Where can I find old versions of Zelig?

For some replications, you may require older versions of Zelig.

- **Windows** users may find old binaries at `http://gking.harvard.edu/bin/windows/contrib/` and selecting the appropriate version of R.

- **Linux** and **MacOSX** users may find source files at `http://gking.harvard.edu/src/contrib/`

If you want an older version of Zelig because you are using an older version of R, we strongly suggest that you update R and install the latest version of Zelig.

Some Zelig functions don't work for me!

If this is a new phenomenon, there may be functions in your namespace that are overwriting Zelig functions. In particular, if you have a function called `zelig`, `setx`, or `sim` in your workspace, the corresponding functions in Zelig will not work. Rather than deleting things that you need, R will tell you the following when you load the Zelig library:

```
Attaching package: 'Zelig'
        The following object(s) are masked _by_ .GlobalEnv :
         sim
```

In this case, simply rename your `sim` function to something else and load Zelig again:

```
> mysim <- sim
> detach(package:Zelig)
> library(Zelig)
```

Who can I ask for help? How do I report bugs?

If you find a bug, or cannot figure something out, please follow these steps: (1) Reread the relevant section of the documentation. (2) Update Zelig if you don't have the current version. (3) Rerun the same code and see if the bug has been fixed. (4) Check our list of frequently asked questions. (5) Search or browse messages to find a discussion of your issue on the zelig listserv.

If none of these work, then if you haven't already, please (6) subscribe to the Zelig listserv and (7) send your question to the listserv at `zelig@lists.gking.harvard.edu`. Please explain exactly what you did and include the full error message, including the `traceback()`. You should get an answer from the developers or another user in short order.

How do I increase the memory for R?

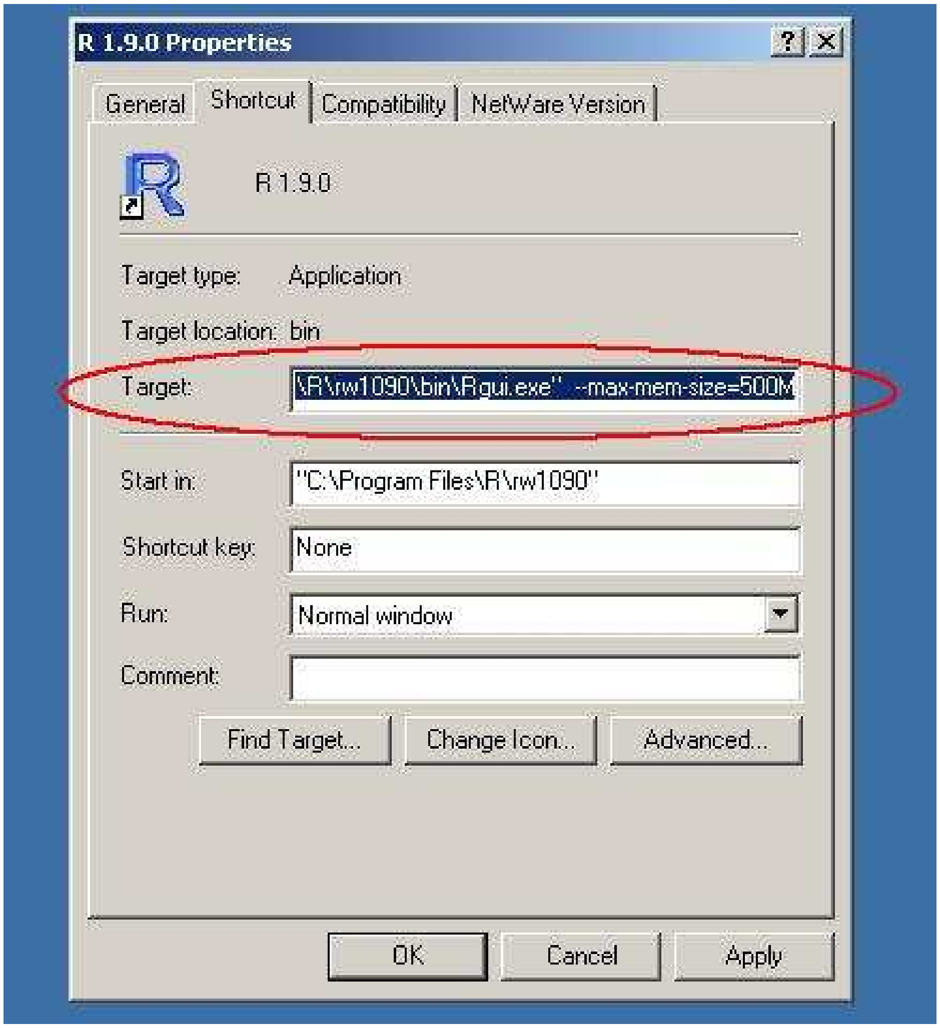
Windows users may get the error that R has run out of memory.

If you have R already installed and subsequently install more RAM, you may have to reinstall R in order to take advantage of the additional capacity.

You may also set the amount of available memory manually. Close R, then right-click on your R program icon (the icon on your desktop or in your programs directory). Select "Properties", and then select the "Shortcut" tab. Look for the "Target" field and after the closing quotes around the location of the R executible, add

```
--max-mem-size=500M
```

as shown in the figure below. You may increase this value up to 2GB or the maximum amount of physical RAM you have installed.



If you get the error that R cannot allocate a vector of length x, close out of R and add the following line to the "Target" field:

```
--max-vsize=500M
```

or as appropriate.

You can always check to see how much memory R has available by typing at the R prompt

```
> round(memory.limit()/2^20, 2)
```

which gives you the amount of available memory in MB.

Why doesn't the pdf print properly?

Zelig uses several special LaTeX environments. If the pdf looks right on the screen, there are two possible reasons why it's not printing properly:

- Adobe Acrobat isn't cleaning up the document. Updating to Acrobat Reader 6.0.1 or higher should solve this problem.
- Your printer doesn't support PostScript Type 3 fonts. Updating your print driver should take care of this problem.

R is neat. How can I find out more?

R is a collective project with contributors from all over the world. Their website (`http://www.r-project.o` has more information on the R project, R packages, conferences, and other learning material.

In addition, there are several canonical references which you may wish to peruse:

Venables, W.N. and B.D. Ripley. 2002. *Modern Applied Statistics with S.* 4th Ed. Springer-Verlag.

Venables, W.N. and B.D. Ripley. 2000. *S Programming.* Springer-Verlag.

B.2 For Zelig Contributors

Where can I find the source code for Zelig?

Zelig is distributed under the GNU General Public License, Version 2. After installation, the source code is located in your R library directory. For Linux users who have followed our installation example, this is `~/.R/library/Zelig/`. For Windows users under R 2.5.1, this is by default `C:\Program Files\R\R-2.5.1\library\Zelig\`. For Macintosh users, this is `~/Library/R/library/Zelig/`.

In addition, you may download the latest Zelig source code as a tarball'ed directory from `http://gking.harvard.edu/src/contrib/`. (This makes it easier to distinguish functions which are run together during installation.)

How can I make my R programs run faster?

Unlike most commercial statistics programs which rely on precompiled and pre-packaged routines, R allows users to program functions and run them in the same environment. If you notice a perceptible lag when running your R code, you may improve the performance of your programs by taking the following steps:

- Reduce the number of loops. If it is absolutely necessary to run loops in loops, the inside loop should have the most number of cycles because it runs faster than the outside loop. Frequently, you can eliminate loops by using vectors rather than scalars. Most R functions deal with vectors in an efficient and mathematically intuitive manner.

- Do away with loops altogether. You can vectorize functions using the `apply`, `mapply()`, `sapply()`, `lapply()`, and `replicate()` functions. If you specify the function passed to the above `*apply()` functions properly, the R consensus is that they should run significantly faster than loops in general.

- You can compile your code using C or Fortran. R is not compiled, but can use bits of precompiled code in C or Fortran, and calls that code seamlessly from within R wrapper functions (which pass input from the R function to the C code and back to R). Thus, almost every regression package includes C or Fortran algorithms, which are locally compiled in the case of Linux systems or precompiled in the case of Windows distributions. The recommended Linux compilers are gcc for C and g77 for Fortran, so you should make sure that your code is compatible with those standards to achieve the widest possible distribution.

Which compilers can I use with R and Zelig?

In general, the C or Fortran algorithms in your package should compile for any platform. While Windows R packages are distributed as compiled binaries, Linux R compiles packages locally during installation. Thus, to ensure the widest possible audience for your package, you should make sure that your code will compile on gcc (for C and C++), or on g77 (for Fortran).

Bibliography

Adolph, C., Gary King, w. M. C. H., and Shotts, K. W. (2003), "A Consensus on Second Stage Analyses in Ecological Inference Models," *Political Analysis*, 11, 86–94, http://gking.harvard.edu/files/abs/akhs-abs.shtml.

Andrews, D. W. (1991), "Heteroskedasticity and Autocorrelation Consistent Covariance Matrix Estimation," *Econometrica*, 59, 817–858.

Bates, D. (2007), *lme4: Fit linear and generalized linear mixed-effects models*.

Binder, D. A. (1983), "On the Variance of Asymptoticaly Normal Estimators from Complex Surveys," *International Statistical Review*, 51, 279–292.

Box-Steffensmeier, J. M. and Jones, B. S. (2004), *Event History Modeling: A Guide for Social Scientists*, Cambridge University Press.

Brockwell, P. J. and Davis, R. A. (1991), *Time Series: Theory and Methods*, Springer-Verlag, 2nd ed.

Butts, C. and Carley, K. (2001), "Multivariate Methods for Interstructural Analysis," Tech. rep., CASOS working paper, Carnegie Mellon University.

Enders, W. (2004), *Applied Econometric Time Series*, Wiley, 2nd ed.

Hamann, J. and Henningsen, A. (2005), *systemfit: Simultaneous Equation Systems in R Package*.

Ho, D., Imai, K., King, G., and Stuart, E. (2007), "Matching as Nonparametric Preprocessing for Reducing Model Dependence in Parametric Causal Inference," *Political Analysis*, 15, 199–236, http://gking.harvard.edu/files/abs/matchp-abs.shtml.

Horvitz, D. G. and Thompson, D. (1952), "A Generalization of Sampling without Replacement from a Finite Universe," *Journal of the American Statistical Association*, 47, 663–685.

Huber, P. J. (1981), *Robust Statistics*, Wiley.

Katz, J. and King, G. (1999), "A Statistical Model for Multiparty Electoral Data," *American Political Science Review*, 93, 15–32, http://gking.harvard.edu/files/abs/multiparty-abs.shtml.

King, G. (1989), *Unifying Political Methodology: The Likelihood Theory of Statistical Inference*, Ann Arbor: Michigan University Press.

— (1995), "Replication, Replication," *PS: Political Science and Politics*, 28, 443–499, http://gking.harvard.edu/files/abs/replication-abs.shtml.

— (1997), *A Solution to the Ecological Inference Problem: Reconstructing Individual Behavior from Aggregate Data*, Princeton: Princeton University Press, http://gking.harvard.edu/eicamera/kinroot.html.

King, G., Alt, J., Burns, N., and Laver, M. (1990a), "A Unified Model of Cabinet Dissolution in Parliamentary Democracies," *American Journal of Political Science*, 34, 846–871, http://gking.harvard.edu/files/abs/coal-abs.shtml.

King, G., Alt, J. E., Burns, N. E., and Laver, M. (1990b), "A Unified Model of Cabinet Dissolution in Parliamentary Democracies," *American Journal of Political Science*, 34, 846–871.

King, G., Honaker, J., Joseph, A., and Scheve, K. (2001), "Analyzing Incomplete Political Science Data: An Alternative Algorithm for Multiple Imputation," *American Political Science Review*, 95, 49–69, http://gking.harvard.edu/files/abs/evil-abs.shtml.

King, G., Murray, C. J., Salomon, J. A., and Tandon, A. (2004), "Enhancing the Validity and Cross-cultural Comparability of Measurement in Survey Research," *American Political Science Review*, 98, 191–207, http://gking.harvard.edu/files/abs/vign-abs.shtml.

King, G., Tomz, M., and Wittenberg, J. (2000), "Making the Most of Statistical Analyses: Improving Interpretation and Presentation," *American Journal of Political Science*, 44, 341–355, http://gking.harvard.edu/files/abs/making-abs.shtml.

King, G. and Wand, J. (2007), "Comparing Incomparable Survey Responses: New Tools for Anchoring Vignettes," *Political Analysis*, 15, 46–66, http://gking.harvard.edu/files/abs/c-abs.shtml.

King, G. and Zeng, L. (2001a), "Explaining Rare Events in International Relations," *International Organization*, 55, 693–715, http://gking.harvard.edu/files/abs/baby0s-abs.shtml.

— (2001b), "Logistic Regression in Rare Events Data," *Political Analysis*, 9, 137–163, http://gking.harvard.edu/files/abs/0s-abs.shtml.

— (2002a), "Estimating Risk and Rate Levels, Ratios, and Differences in Case-Control Studies," *Statistics in Medicine*, 21, 1409–1427, http://gking.harvard.edu/files/abs/1s-abs.shtml.

— (2002b), "Improving Forecasts of State Failure," *World Politics*, 53, 623–658, http://gking.harvard.edu/files/abs/civil-abs.shtml.

— (2006), “The Dangers of Extreme Counterfactuals,” *Political Analysis*, 14, 131–159, http://gking.harvard.edu/files/abs/counterft-abs.shtml.

— (2007), “When Can History Be Our Guide? The Pitfalls of Counterfactual Inference,” *International Studies Quarterly*, 183–210, http://gking.harvard.edu/files/abs/counterf-abs.shtml.

Lumley, T. and Heagerty, P. (1999), “Weighted Empirical Adaptive Variance Estimators for Correlated Data Regression,” *jrssb*, 61, 459–477.

Martin, A. D. and Quinn, K. M. (2005), *MCMCpack: Markov chain Monte Carlo (MCMC) Package*.

Martin, L. (1992), *Coercive Cooperation: Explaining Multilateral Economic Sanctions*, Princeton University Press, please inquire with Lisa Martin before publishing results from these data, as this dataset includes errors that have since been corrected.

McCullagh, P. and Nelder, J. A. (1989), *Generalized Linear Models*, no. 37 in Monograph on Statistics and Applied Probability, Chapman & Hall, 2nd ed.

Plummer, M., Best, N., Cowles, K., and Vines, K. (2005), *coda: Output analysis and diagnostics for MCMC*.

Rosen, O., Jiang, W., King, G., and Tanner, M. A. (2001), “Bayesian and Frequentist Inference for Ecological Inference: The $R \times C$ Case,” *Statistica Neerlandica*, 55, 134–156, http://gking.harvard.edu/files/abs/rosen-abs.shtml.

Scheve, K. and Slaughter, M. (2001), “Labor Market Competition and Individual Preferences over Immigration Policy,” *Review of Economics and Statistics*, 83, 133–145, sample data include only the first five of ten multiply imputed data sets.

Stoll, H., King, G., and Zeng, L. (2005), “WhatIf: Software for Evaluating Counterfactuals,” *Journal of Statistical Software*, 15, http://www.jstatsoft.org/index.php?vol=15.

Therneau, T. M. and Grambsch, P. M. (2000), *Modeling Survival Data: Extending the Cox Model*, Springer.

Venables, W. N. and Ripley, B. D. (2002), *Modern Applied Statistics with S*, Springer-Verlag, 4th ed.

Wand, J., King, G., and Lau, O. (2007, forthcoming), “Anchors: Software for Anchoring Vignettes Data,” *Journal of Statistical Software*.

White, H. (1980), “A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity,” *Econometrica*, 48, 817–838.

Wood, S. N. (2000), “Modeling and Smoothing Parameter Estimation wiht Multiple Quadratic penalties,” *Journal of the Royal Statistical Society*, 62, 413–428.

— (2004), “Stable and efficient multiple smoothing parameter estimation for generalized additive models,” *Journal of the American Statistical Association*, 99, 673–686.

— (2006), *Generalized Additive Models: An Introduction with R*, London: CRC Press.

Yee, T. W. and Hastie, T. J. (2003), “Reduced-rank vector generalized linear models,” *Statistical Modelling*, 3, 15–41.

Zeileis, A. (2004), “Econometric Computing with HC and HAC Covariance Matrix Estimators,” *Journal of Statistical Software*, 11, 1–17.

Zorn, C. (2001), “Generalized Estimating Equation Models for Correlated Data: A Review with Applications,” *American Journal of Political Science*, 45, 470–490.