RobustYourCast: Robust Bayesian Forecasting with YourCast[1]

Jon Bischof[2] Gary King[3] Samir Soneji[4]

April 3, 2010

[2]Department of Government, Harvard University.

[3]Albert J. Weatherhead III University Professor, Harvard University (Institute for Quantitative Social Science, 1737 Cambridge Street, Harvard University, Cambridge MA 02138; http://GKing.Harvard.Edu, King@Harvard.edu, (617) 495-2027).

[4]Robert Wood Johnson Foundation Health & Society Scholar, University of Pennsylvania

Contents

Abstract

RobustYourCast is a streamlined and user-friendly version of YourCast software, which focuses on generating forecasts for multiple cross-sections over time, limited to a single geographic region. In the YourCast framework, individual hyperprior parameters are easy to interpret, but applications often require the simultaneous use of multiple priors, such as for the smoothness of forecasts across groups, time, and in trends across groups; in this situation, hyperprior parameters can be difficult to interpret and set. RobustYourCast introduces a framework that makes setting multiple priors easy. The general strategy is to fit an objective function based on smoothness and fit to a subset of the data, and to use the results of that analysis to set the prior for all (or the remaining) data.

1 Introduction

"YourCast: Time Series Cross-Sectional Forecasting With Your Assumptions" (http://gking.harvard.edu/yourcast) implements a comprehensive approach to forecasting developed for the R Project for Statistical Computing. It can fit any member of a new class of Bayesian models proposed in Girosi and King (2008), all with a single user interface. The idea of YourCast is to fit a large set of linear regressions simultaneously with priors that tie them all together. An example application is a set of relatively short annual mortality time series with covariates, for each age group, sex, race, country, and cause of death. Estimating each regression separately would be very noisy and yield poor forecasts. This new approach allows users to put informative priors on the expected value of the outcome variable, about which users typically know a great deal, rather than on the parameters (i.e., the coefficients), about which they know very little. This approach greatly reduces the number of hyperprior parameters and enables researchers to include different covariates in the regression from each of the cross-sections (such as including tobbaco consumption as a predictor for adult mortality but not infant mortailty), but it still enables one to smooth over age groups, time trends, time trends across age groups, countries, time trends in neighboring countries, etc.

In this paper we introduce RobustYourCast, a streamlined version of the YourCast software which makes it easy for users to fit all the special cases of these models that run within a single geographic region (but still with many cross-sectional groups such as age by sex). The idea is to estimate optimal values for the hyperprior parameters from a small subset of the data, using prior information, fit, and easy methods of interacting with the data and results. With these estimates, the user would then a much better sense of how to set the priors for the rest of the data.

Consider the problem of forecasting all-cause mortality for multiple age groups of males in the United States. Rather than producing a single forecast for all males or estimating separate forecasts for each age group, YourCast incorporates one's prior beliefs about how smoothly mortality should change across age groups and time as well as the similarity of time trends across age groups. For example, one may believe that 30 and 35 year olds should die at similar rates, that the mortality rate for 30 year olds in 1980 should be similar to the rate in 1981, or that if death rates for 30 year olds are falling then rates for 35 year olds likely fell as well. These beliefs are formalized into three smoothness parameters—σ_a, σ_t, and σ_{at}—which if specified individually would indicate the average distance between neighboring age groups, time periods, or time trends respectively.

If each prior is used separately, its values are highly intuitive. For example, setting $\sigma_a = 0.1$ means that one believes that log mortality between neighboring age groups differs by about 0.1, which is even easy to set based on many earlier data sets. However, if one attempts more than one type of smoothing in a model, these parameters no longer have an intuitive interpretation. Furthermore, it is difficult to anticipate how multiple smoothness priors will interact in posterior inference—for instance, it may be that enforcing smooth time trends could be accomplished by either setting σ_{at} close to zero or slightly relaxing σ_t. Importantly, the common trends achieved in the latter are likely to be less linear. We offer a way around this difficulty.

Bayesian modeling such as this also presents a bias-variance trade-off which in this case is represented by smoothness vs in sample fit. We do not want noisy data to make our forecasts choppy across neighboring age groups or time periods, but we also do not want to force them to follow smooth trends to the point where we are ignoring systematic patterns in the data.

RobustYourCast allows analysts to disentangle the interactions between different smoothness parameters and how each combination affects prediction error. We accomplish this through cross-validation, where we omit a block of observations at the end of the observation period and then produce a forecast for those omitted years as well as our original years of interest using a set of candidate smoothing priors—σ_a^*, σ_t^*, and σ_{at}^*. For each set of candidates, we score the resulting forecasts based on how much they lack smoothness in their time and age profiles, similarity in time trends, and ability to predict outcomes in the omitted years. These four measures are then combined into a weighted average of undesireables, with the weights chosen by the user to reflect his or her relative tolerance for each. The problem of finding an "optimal" forecast can now be formalized as finding the prior parameters that minimize this objective function.

This approach allows users to directly observe relevant trade-offs when calibrating their smoothness parameters by changing the weights. For example, a forecast which minimizes an objective function with most weight on prediction error will probably fit the data well but not be very smooth. An analyst particularly dissatisfied with wildly divergent time trends can then begin shifting weight away from prediction error and toward time trend smoothness to see how much prediction error he or she would have to gain to get satisfactory trending behavior. These trade-offs can then be analyzed for all four of the optimality conditions, allowing one to carefully calibrate the choice of smoothness parameters for application in the current of similar forecasting exercises. For example, if one wanted to predict mortality for males in the 50 US states, one could calibrate the prior parameters using a handful of representative states and then use those priors for each state in the country.

2 Objective function measurement

In this section we briefly discuss how the four components of the objective function—prediction error, smoothness of time profiles, smoothness of age profiles, and deviance in time trends—are measured. For all diagnostics any missing observations are removed from the calculation. These are the four components:

- *Prediction error* for all of the omitted years in all groups, we measure sum of the root mean square error of the out-of-sample forecasts.
- *Age smoothness* is measured as the arc lengths of age profiles from the validation period once the mean age profile from the validation period is removed. Arc length will be shortest when these demeaned profiles are flat—i.e., the same as the mean or differing by a constant. In contrast, profiles which change constantly with respect to the mean will be receive high scores. (Other measures may be substituted for the arc length if desired.)
- *Time smoothness* is measured as the arc length of each time profile's deviation from its own trend line. Here we allow the user to choose the degree of the polynomial to which the time profiles are smoothed. For example, if a first degree polynomial is chosen, linear time profiles will have the lowest score; zero degree polynomials will give the lowest score to the flattest time profiles.
- *Trend deviations* are measured by removing the constant from all the time profiles (so that each has mean zero) and measuring the arc length of their deviations from the mean time profile. The result is very similar to our measure of age profile smoothness: the profiles scored lowest will follow or be parallel to the mean time profile.

Using the shorthands `RSS`, `Age AL`, `Time AL`, and `Trend dev`, respectively, for these components, we express the objective function as:

$$f(\texttt{RSS}, \texttt{Age AL}, \texttt{Time AL}, \texttt{Trend dev}) = w_1\sqrt{\texttt{RSS}} + w_2(\texttt{Age AL}) + w_3(\texttt{Time AL}) + w_4(\texttt{Trend dev})$$

where $\vec{w}$ is chosen by the user subject to $\sum_{i=1}^{4} w_i = 1$. Since all four diagnostics take values in $[0, \infty)$, they are not possible to normalize. An important consequence is that the absolute values of the weights have no intuitive interpretation; only comparisons of different weighting schemes with the same data are meaningful. In fact, in particularly noisy data values of `RSS` will be high relative to other diagnostics and can thus be ignored.

In our experience, the objective function surface tends to be multimodal but locally well behaved. We thus use a grid search to narrow down the search. We recommend that users start with a coarse grid search and then use the minimum from that as the starting point for a more direct optimization method such as BFGS. RobustYourCast provides tools to visualize and optimize the user's objective function, as documented in the next section.

3 User's Guide

3.1 Installation

From the R command line, type

```
> install.packages("RobustYourCast", repos="http://gking.harvard.edu") .
```

3.2 Loading data

RobustYourCast uses much of the basic architecture of YourCast, including how data is formatted. For YourCast to make forecasts for multiple cross sections, a separate array with a response and covariates is needed for each cross section. With the case of US males, we require a data array for each age group with mortality rates for some observation period and covariates with observations in both the observation period and adjacent forecast period. While the observation and forecast periods must be identical for each age group (allowing for missing data), one of the main advantages of YourCast is that each age group may have different covariates (see Girosi and King (2008))

These arrays, along with other identifying information, must be concatenated into a single list object we call a `dataobj`. The `yourprep` function in the YourCast package can assist users to load data from multiple formats, lag covariates as needed, and concatenate this information into an R object `yourcast` can read. For detailed instructions, please see the YourCast manual and the online help pages for `yourprep` and `yourcast`.

3.3 Setting up a YourCast call

In order to illustrate how functions in RobustYourCast are used, we will focus on the specific problem of forecasting breast cancer mortality in Belgium. We observe mortality rates from

1950-2000 and would like to forecast future mortality from 2001 to 2030. To inform our predictions, we include five covariates: human capital (hc), GDP (gdp), tobacco use (tobacco3), obesity rates (fat), and a time trend (time). While our ultimate goal is to find optimal smoothing priors to get a reasonable forecast, it is often helpful to first look at the least squares fit, one of the models that can be run in yourcast. Besides a formula object and the dataobj described in the previous section, the only other argument required to yourcast is a vector with the start and end dates our observation and forecast periods, called sample.frame.

We then make our call to yourcast to get an output object:

```
ff <- log(brst3/popu3) ~ log(hc) + log(gdp) + log(tobacco3) + log(fat)+ time
out.ols <- yourcast(formula=ff,
                    model="ols"
                    dataobj=dataobj.belgium,
                    sample.frame=c(1950,2000,2001,2030))
```

We now have an output object called out.ols. We can easily visualize our forecasts with a simple call to plot:

```
plot(out.ols,print="pdf",file="belgium_ols.pdf")
```

where extra options have been specified to save a PDF file in the working directory. We have reproduced this plot in Figure 1. Here we can see a significant increase in the variance of mortality across age groups in the forecast period, and well as unlikely intersections in the age profiles; e.g., 50 year olds eventually dying at lower rates than 40 year olds. Therefore the introduction of informative smoothing priors with the Bayesian models seem promising.

The "map" model in yourcast yourcast that allows us to introduce smoothing priors, with each type of smoothing a separate argument to the function: σ_a is Ha.sigma, σ_t is Ht.sigma, and σ_{at} is Hat.sigma. Thus if we wanted fairly strong smoothing we could run the model:

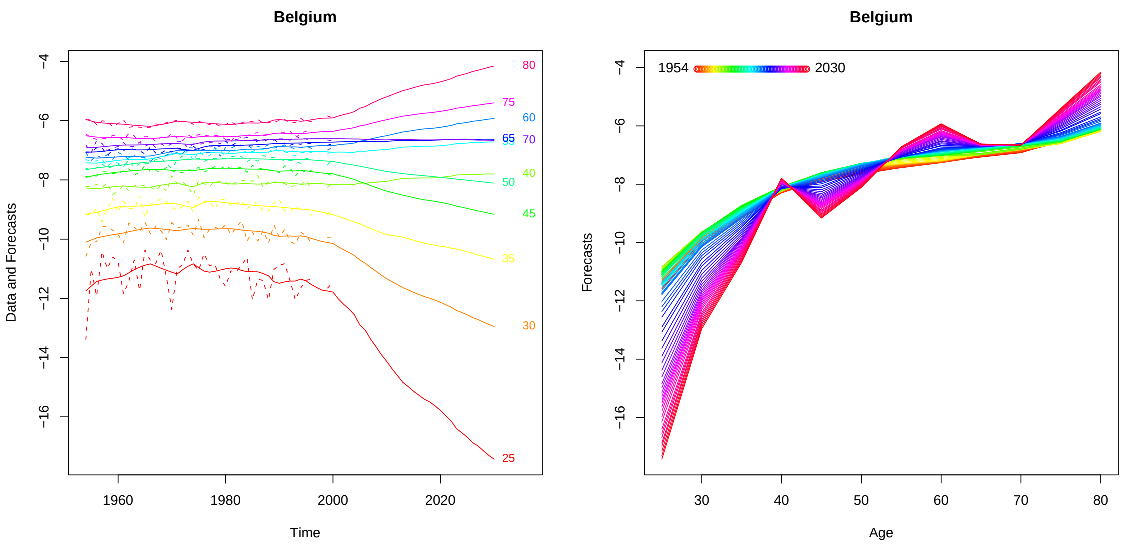
```
out.map <- yourcast(formula=ff,
                    sample.frame=c(1950,2000,2001,2030),
                    dataobj=dataobj.belgium,
                    model="map",
                    Ha.sigma=0.1,Ht.sigma=0.1,Hat.sigma=0.1)
plot(out.map)
```

The results look promising, but how can we see different possibilities that emphasize greater fit to the data or more similar time trends? By forming an objective function using RobustYourCast we will be able to find optimal forecasts given our personal preferences for different optimality criteria.

3.4 Grid search

Once we decide on a set of weights for our objective function, RobustYourCast offers two optimization methods: a grid search function called robust.yourcast and a general purpose optimization function called optim.yourcast that can call optim or rgenoud. In practice, these

Figure 1: Least squares forecast for breast cancer mortality in Belgium (no smoothing)



two methods are best used in conjunction, building up a picture of the objective function surface with `robust.yourcast` and then using the minimum from the grid search as a starting point for one of the optimization algorithms implemented in `optim.yourcast`. Since the objective function surface is almost never unimodal, a thorough grid search is critical.

We start with the grid search function `robust.yourcast`. We first need decide at which points in the space of positive real numbers $\mathbb{R}^3_+$ we want to evaluate the function. The arguments `H*.sigma.range` sets the range for each parameter and the arguments `N.*` set the number of points to test in that range. By default, these points are evenly spaced out on the log scale to increase the number of lower values tested—changing σ_a from 0.1 to 0.01 has much more impact than a change from 10 to 9.9. We then set the desired weights with the `weights` argument, which will be a length-four vector of positive numbers that will be normalized by the function if they do not sum to one already.

Since `robust.yourcast` works through cross-validation, we also have to decide which block of observations to omit and try to predict with our model. The argument `length.block` specifies the number of years in the validation block; then we then only need to specify the last year in the block. By default, `end.block` is set to `"last"`, the last observed year, since in practice omitting a block of observations and the end is the only meaningful proxy for an actual forecast.

It is helpful to use the `runs.save` argument, which stores the output from the computationally intensive part of the grid search so that changes to the weights can be made with little additional computation. Finally, we also pass on the basic arguments to `yourcast`:

```
out.rob <- robust.yourcast(# Set up sigma grid
                           Ha.sigma.range=c(0.01,10),
                           Hat.sigma.range=c(0.01,10),
```

```
                    Ht.sigma.range=c(0.01,10),

                    # How many points to test in each range?
                    N.Ha=15,
                    N.Ht=15,
                    N.Hat=15,

                    # Weights
                    weights=c(0.5,0.25,0.25,0),

                    # Length and positions of blocks
                    length.block=7,
                    end.block="last",

                    runs.save="belgium-runs.RData",

                    # yourcast() call
                    formula=ff,
                    dataobj=dataobj.belgium,
                    model="map",
                    sample.frame=c(1950,2000,2001,2030))
```

We then see the number of runs and a notification each time one completes:

```
[1] "Starting 3375 yourcast() runs"
[1] "Done with run 1"
[1] "Done with run 2"
[1] "Done with run 3"
[1] "Done with run 4"
...
[1] "Done with run 3375"
```

Increasing the total number of points to evaluate is always better, but at the obvious cost of computational intensity. Usually a $10 \times 10 \times 10$ grid is sufficient, but here we move to a $15 \times 15 \times 15$ grid since the objective function surface is poorly behaved for many choices of weights in this example.

Once the grid search is complete, we can visualize the objective function surface for our choice of weights using the `plot` command:

```
plot(out.rob,print="pdf",filename="belgium_objplot.pdf")
```

The resulting plot is presented in Figure 2. Here we can see that the surface of the function is not particularly well behaved. Since the parameters are constrained to the positive octant, we often find the optimum lodged in the corner. Plots of the objective function are also helpful to tease out relationships between the smoothing parameters; for example, in this plot apparently σ_a has little impact given that $\sigma_{at} = 0.05$. `plot.robust.yourcast` can generate plots for the individual components of the objective function by changing the `family` argument from its default.

Figure 2: Plot of objective function for breast cancer case with weights = $(0.5, 0.25, 0.25, 0)$

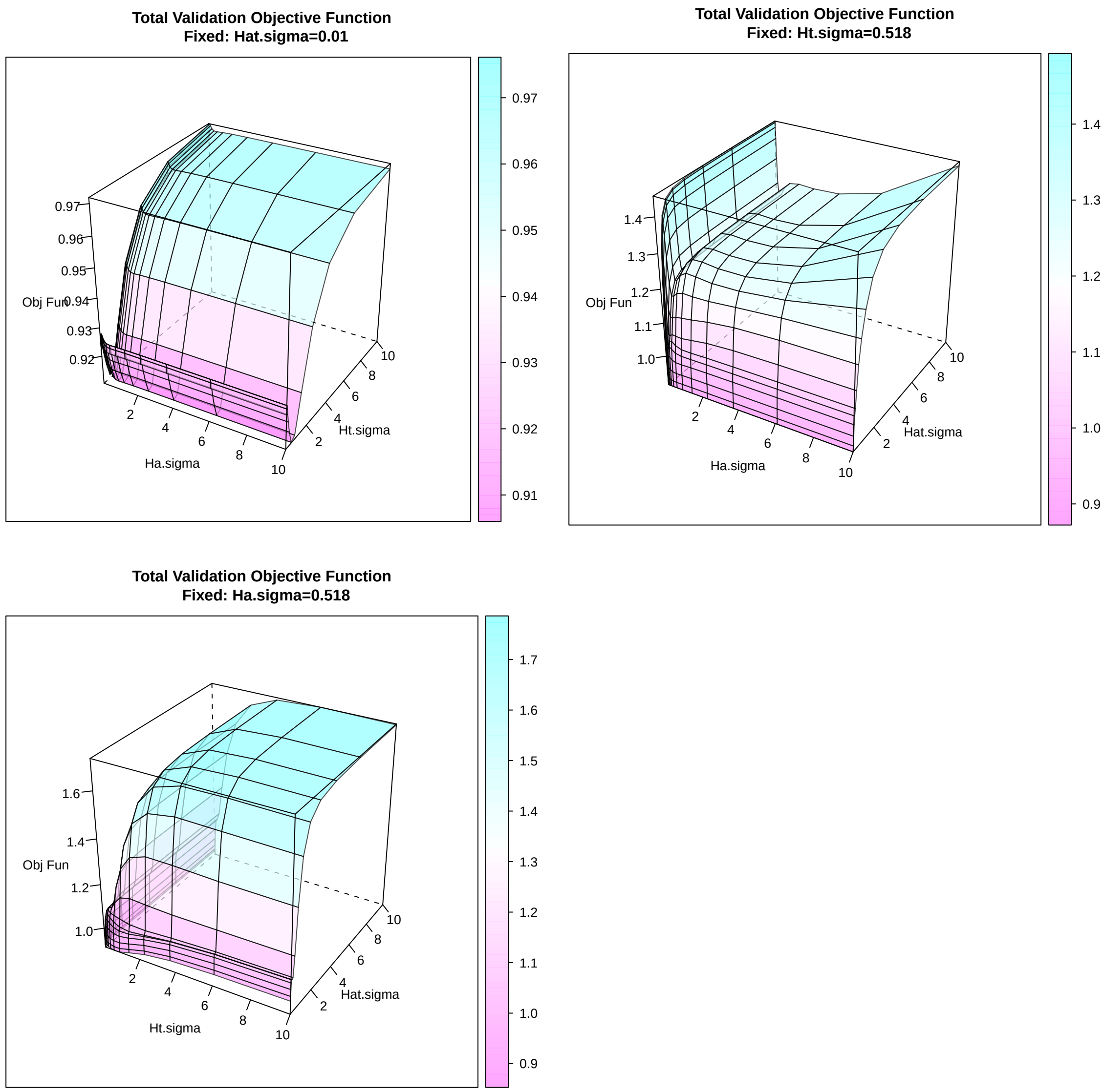
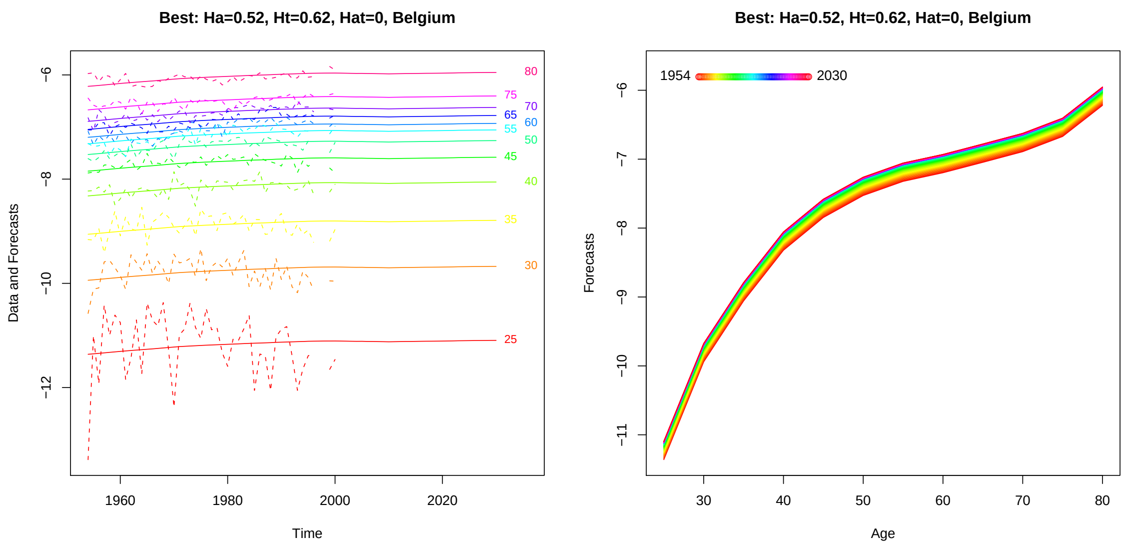


Figure 3: Optimal forecast with weights = $(50, 25, 25, 00)$



Finally, we can easily plot the forecast generated by the grid search minimum with the `run.opt` function:

```
run.opt(out.rob)
```

The resulting forecast can be found in Figure 3. The user can always override any of the "optimum" parameters with the `H*.set` argument to `run.opt`.

Users can get information about the basic information about any `robust.yourcast` output object with the `summary` command:

```
> summary(out.rob)
Observed period: 1950-2000
Forecast period: 2001-2030

Validation blocks:
      year.1 year.2 year.3 year.4 year.5 year.6 year.7
run.1   1994   1995   1996   1997   1998   1999   2000

Weights:
      RSS     Age AL    Time AL Trend Dev
     0.50       0.25       0.25       0.00

Grid search range:
Ha.sigma:  0.01 - 10
Ht.sigma:  0.01 - 10
```

```
Hat.sigma: 0.01 - 10

Optimal sigma combination:
 Ha.sigma  Ht.sigma Hat.sigma
    0.518     0.518     0.010
```

Similar output is generated when `summary` is used on a `optim.yourcast` output object.

3.5 Direct optimization

RobustYourCast includes the function `optim.yourcast`, which can directly optimize the objective function by making calls to `optim` or `rgenoud`. The arguments to `optim.yourcast` are almost identical to `robust.yourcast` except that the user must specify the type of optimization to employ rather than the grid points.

However, direct optimization rarely works without a good starting point provided by a grid search. RobustYourCast makes it very easy to pass the result of a grid search on to an optimizer with the call:

```
out.opt <- optim.yourcast(out.rob)
```

Here `optim.yourcast` will automatically transfer all relevant information about the `yourcast` model from the `robust.yourcast` output object. Optimization will be started at the optimum point identified in the grid search with default method `BFGS` from `optim`. Since the parameter space is constrained, the function by default reparameterizes the search into log space, although these and several other options can be tweaked by the user.

Once optimization is complete, the resulting forecast can be plotted with `run.opt`:

```
run.opt(out.opt,print="pdf",filename="belgium_forecast_50-25-25-00.pdf")
```

3.6 Refinements

Once an initial grid search is completed, RobustYourCast allows users to experiment with different objective function weights with minimal additional computation. One can test a different set of weights with another call to `robust.yourcast` specifying only the `.RData` object where the runs are stored and the new weights. For example, a user interested in forecasts with linear time profiles could try

```
out.rob20.00.80.00 <- robust.yourcast(runs.load="belgium-runs.RData",
                                      weights=c(0.20,0,0.8,0))
run.opt(out.rob20.00.80.00)
```

which will launch a plot with the new forecast in less than a minute. A user can then add more emphasis on smooth time trends or prediction error with the calls

```
out.rob20.00.40.40 <- robust.yourcast(runs.load="belgium-runs.RData",
```

```
                                weights=c(0.20,0,0.4,0.4))
run.opt(out.rob20.00.40.40)

out.rob70.20.10.00 <- robust.yourcast(runs.load="belgium-runs.RData",
                                weights=c(0.70,0.2,0.1,0))
run.opt(out.rob70.20.10.00)
```

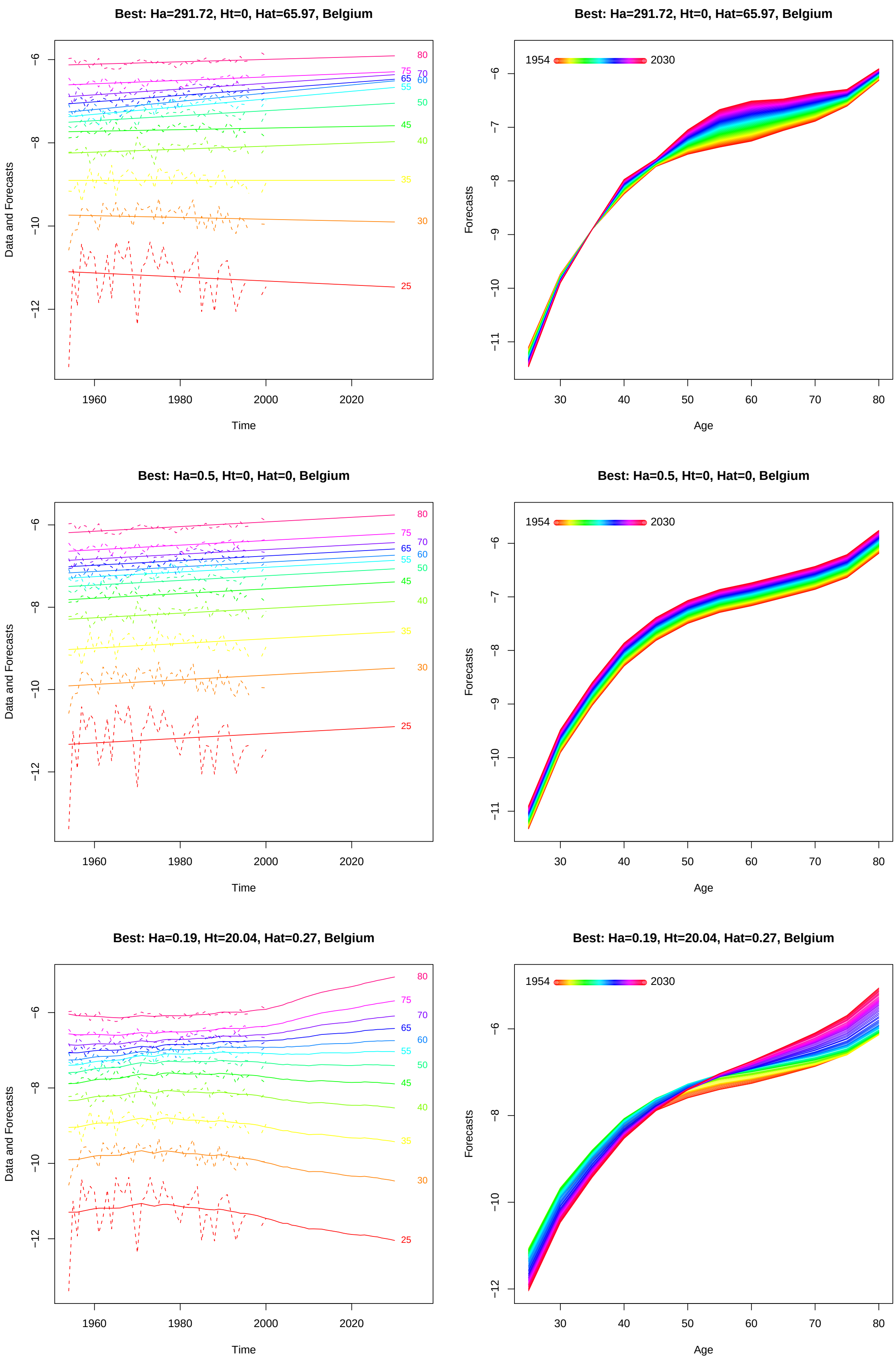
If desired, each `robust.yourcast` output can also be further refined by `optim.yourcast` before being sent to `run.opt`.

The plots discussed in this section are displayed in Figure 4.

4 Reference

The following pages list the main functions in RobustYourCast with detailed reference information. These can also be loaded from R with the standard help command, such as `help(robust.yourcast)`.

Figure 4: Optimal forecasts with weights (20,00,80,00), (20,00,40,40), and (70,20,10,00)



4.1 `robust.yourcast`: Robust Bayesian Forecasting with YourCast

Description

Generate Yourcasts under range of prior specifications and evaluates predictions with cross validation

Usage

```
robust.yourcast(
                              # Pararmeter space to search for priors
                              Ha.sigma.range=c(0.01,20),
                              Ha.list=NULL,
                              Ht.sigma.range=c(0.01,20),
                              Ht.list=NULL,
                              Hat.sigma.range=NA,
                              Hat.list=NULL,
                              N.Ha=5, N.Ht=5,N.Hat=5,
                              logscale=FALSE,

                              # Weights for objective function
                              weights=c(0.5,0.25,0.25,0),
                              time.degree=1,

                              # Set up blocks
                              length.block=5,
                              end.block="last",

                              # Store validation data for future use?
                              # (changing weights)
                              runs.save=NULL,
                              # Use stored data from previous run?
                              runs.load=NULL,

                              # Verbose sapply() loop?
                              print.runs=TRUE,

                              # Use condor to process runs?
                              condor=FALSE,
                              condor.dir=getwd(),
                              condor.fld=NULL,
                              condor.comp=NULL,

                              # Inputs to yourcast()
                              # See help(yourcast) for details
                              ...)
```

Arguments

`Ha.sigma.range`
: Two element vector of non-negative numbers. Range of Ha.sigma values to search over. If do not want to use age smoothing, set as `NA` and leave `Ha.list` as `NULL`.

`Ha.list`
: Vector. If changed from `NULL`, a vector of additional values of Ha.sigma to include in the grid search.

`Ht.sigma.range`
: Two element vector of non-negative numbers. Range of Ht.sigma values to search over. If do not want to use time smoothing, set as `NA` and leave `Ht.list` as `NULL`.

`Ht.list`
: Vector. If changed from `NULL`, a vector of additional values of Ht.sigma to include in the grid search.

`Hat.sigma.range`
: Two element vector of non-negative numbers. Range of Hat.sigma values to search over. If do not want to use trend smoothing, set as `NA` and leave `Hat.list` as `NULL`.

`Hat.list`
: Vector. If changed from `NULL`, a vector of additional values of Hat.sigma to include in the grid search.

`N.Ha`
: Scalar. Number of values to test (evenly spaced out) along Ha.sigma.range. Set to `NA` if do not want to use this parameter.

`N.Ht`
: Scalar. Number of values to test (evenly spaced out) along Ht.sigma.range. Set to `NA` if do not want to use this parameter.

`N.Hat`
: Scalar. Number of values to test (evenly spaced out) along Hat.sigma.range. Set to `NA` if do not want to use this parameter.

`logscale`
: Logical. Should sigma values be even spaced on a log scale and then exponentiated? If `FALSE`, sigma values will be equally spaced out on normal scale.

`weights`
: Vector of length four. Provides weights for the four components of the objective function. See 'Details'.

`time.degree`
: Non-negative integer. Specifies the degree of the baseline polynomial to which time profiles are smoothed. For example, if `time.degree=1`, then the forecasts closest to a straight line be scored highest. If `time.degree=0`, then forecasts closest to a flat line will be scored highest.

`length.block`
: Numeric. How many years should be omitted at a time in validation blocks?

`end.block`
: Vector. Specifies years in which validation blocks should end. Length of vector determines how many vaidation exercises done. Alternatively, if set to `"last"`, will choose the last possible block in the observed data period only.

`runs.save`
: String. If changed from `NULL`, specifies a file name in the form of `*.RData` in which the raw output from validation exercises will be stored. Saving this information allows users to evaluate the objective function using different weights without having to recompute forecasts with yourcast function.

`runs.load`	String. If changed from `NULL`, specifies a file name in the form of `*.RData` from which previous runs of yourcast (based on a specific model, dataset and grid of points to evaluate) will be loaded. If provided, all other arguments except `weights` will be ignored.
`print.runs`	Logical. If `TRUE`, will print notification each time run of `yourcast` completed.
`condor`	Logical. Use the Condor batch processing software for parallel processing of `yourcast` runs on the RCE servers. Note: this is only available to members of Harvard's IQSS working on the RCE servers.
`condor.dir`	String. Directory in which to write condor files. Generated folder will be deleted before function exits.
`condor.fld`	String. A name for the folder in which condor output is stored in the `condor.dir`. If left as `NULL`, folder will be given a random name starting with `tmp_` and will be deleted after the runs are completed and loaded in R. NOTE: This will not delete the `condor.dir`, but a folder created within it.
`condor.comp`	If changed from `NULL`, specifies a folder in the `condor.dir` in which already completed runs of condor are stored. All other arguments supplied in the first run of `robust.yourcast` must again be supplied, but the function will skip sending the jobs to condor and instead load them from the specified folder as if they had just been completed. NOTE: All other condor-related arguments will be ignored when this is changed from `NULL` since condor is never called. NOTE: This feature exists mainly for debugging purposes and will not be useful to most users.
`...`	Arguments to be passed to `yourcast`. See `help(yourcast)` for more details. *Not clear what prior arguments might also be set by user apart from ones here.*

Details

Function to evaluate predictions using `yourcast` under a range of prior specifications. Given the different values (specified by the user) of the three sigma priors to test, the function will perform a validation exercise for `yourcast` using the blocks of time periods specified by the user.

This validation blocks are specified by indicating the year each block should end (`end.block`) and the number of years in each block (`length.block`). The number of years in `end.block` will determine the number of validation periods. Alternatively, if `end.block` is set to `"last"`, the function will choose the last possible block in the observed data period only.

For each block to be omitted, `robust.yourcast` generates a `yourcast` input object with those years marked as `NA` and generates a prediction for that block of responses under each of the prior combinations.

The total number of runs of `yourcast` is the product of the non-`NA` `N.*` arguments and the number of blocks to be omitted. `robust.yourcast` processes the `yourcast` runs locally, or, if `condor=TRUE`, parallel on the RCE servers with the Condor batch processing software.

After predictions for the validation blocks under each sigma combination are generated, `robust.yourcast` calculates values of RSS, age profile arc length, time profile arc length, and time trend deviation for each. These diagnostics are then used as inputs into a univariate objective function that employs weights specified by the user to evaluate each set of forecasts.

Given a length-four vector of weights, the function is

f(RSS, age AL, time AL, trend dev) = weights[1]*sqrt(RSS) + weights[2]*age AL + weights[3]*time AL + weights[4]*trend dev

The optimal combination of prior values will minimize this function over the specified validation periods.

Value

`list` A list object of class 'robust.yourcast' with the following components:

- **par.opt** A vector of the optimal value for each sigma parameter as indicated by the objective function.
- **sigma** A matrix of all combinations of sigma parameters compared in validation exercise.
- **rss.valid** A matrix that lists the RSS value estimated for each combination of sigma parameters in the `sigma` matrix. Results are broken down by validation period.
- **arc.age.valid** A matrix that lists the age profile arc length value estimated for each combination of sigma parameters in the `sigma` matrix. Results are broken down by validation period.
- **arc.time.valid** A matrix that lists the time profile arc length value estimated for each combination of sigma parameters in the `sigma` matrix.
- **trend.dev.valid** A matrix that lists the trend deviation arc length value estimated for each combination of sigma parameters in the `sigma` matrix.
- **diag.valid** A matrix that lists all diagnostic values estimated for each combination of sigma parameters in the `sigma` matrix, including the objective function using the specified weights. Results are summed over all specified validation periods.
- **aux.robust** A list of information about the run of `robust.yourcast` used by other functions in the `RobustYourCast` library.

Author(s)

Jon Bischof `<jbischof@fas.harvard.edu>`

References

`http://gking.harvard.edu/yourcast`

See Also

`yourcast`, `plot.robust.yourcast`

4.2 `optim.yourcast`: Robust Bayesian Forecasting with YourCast Using Optimization

Description

Finds sigma parameters that produce optimal forecast using optimization algorithms

Usage

```
optim.yourcast(# Robust.yourcast output object?
                      robust.out=NULL,

                      # Starting values for optim
                      # Vector order is c(Ha,Ht,Hat)
                      # Set any dimension to 'NA' if don't want
                      # to use
                      par=ifelse(rep(is.null(robust.out),3),
                                 c(1,1,1),robust.out$par.opt),

                      # Constrained or unconstrained optimization?
                      reparam=TRUE,

                      # Arguments for optim
                      method="BFGS",
                      args.optim=list(),

                      # Use rgenoud?
                      rgenoud=FALSE,
                      upper.bound=100,
                      args.rgenoud=list(),

                      # Weights for objective function
                      weights=c(0.5,0.25,0.25,0),
                      time.degree=1,

                      # Set up blocks
                      length.block=5,
                      end.block="last",

                      # Inputs to yourcast()
                      # See help(yourcast) for details
                      ...)
```

Arguments

`robust.out` A object of class ‘robust.yourcast’. If changed from `NULL`, will take information from previous run of `robust.yourcast` to continue optimization where that function left off. Specifically, the function will use the optimal sigma combination identified for the `par` argument and recover all arguments pertaining to the weights, blocks, and yourcast input from the output object. However, a different starting point can be specified if the `par` argument is changed from its default. All arguments after `args.rgenoud` will be ignored.

`par` Vector of length three. Provides starting values for the optimization algorithm. The entries in the vector coorespond to Ha.sigma, Ht.sigma, and Hat.sigma, respectively.

`reparam` Logical. Since objective function lives in positive quadrant/octant, should parameters be transformed to log-space to allow unconstrained optimization? If `FALSE`, only `"L-BFGS-B"` method will be allowed for `optim`. For both `optim` and `rgenoud` lower boundries close to zero will be enforced; the upper boundary for `rgenoud` can be set with the `upper.bound` argument.

`args.optim` List. If `rgenoud=FALSE`, a list of arguments (must be labeled) to be passed to `optim`. For example, if wanted to turn off verbose option, could add `control=list(trace=0)`.

`rgenoud` Logical. Should `rgenoud` be used instead of `optim` for the optimization?

`upper.bound` Numeric. If `rgenoud=TRUE` and `reparam=FALSE`, specifies an upper bound for parameters. If `reparam=FALSE`, the lower bound is set close to zero. Whether or not `reparam=TRUE`, the user can set his or her own bounds for `rgenoud` by adding the `Domains` argument to `args.rgenoud`.

`weights` Vector of length four. Provides weights for the four components of the objective function. See ‘Details’.

`time.degree` Non-negative integer. Specifies the degree of the baseline polynomial to which time profiles are smoothed. For example, if `time.degree=1`, then the forecasts closest to a straight line be scored highest. If `time.degree=0`, then forecasts closest to a flat line will be scored highest.

`length.block` Numeric. How many years should be omitted at a time in the validation block?

`end.block` Numeric. Specifies year in which validation block should end. Alternatively, if set to `"last"`, will choose the last possible block in the observed data period. Unlike `robust.yourcast`, in this function only one validation block can be used at a time.

`...` Arguments to be passed to `yourcast`.

Details

Function to find the optimal sigma parameters for `yourcast` forecasts using an optimization algorithm. Starting from values for the sigma parameters specified in `par`, `optim.yourcast` will call either `optim` or `rgenoud` to search over the parameter space for the point that performs best in a validation exercise.

`optim.yourcast` only allows a single validation period to be used at a time. This period is specified by indicating in which year the block should end (`end.block`) and the number of years in the block (`length.block`). Alternatively, if `end.block` is set to `"last"`, the function will choose the last possible block in the observed data period.

To set up the validation, `optim.yourcast` generates a `yourcast` input object with a block of validation years specified by the user marked as `NA` and then for each set of parameter values guessed by the optimization routine generates a forecast for that block of responses.

The quality of the forecast is quantified by an objective function which considers four diagnostics: the sum of squares of the prediction error for the block (RSS), the arc length of the age profile for that block (age AL), the arc length of the time profile for that block (time AL), and the deviations from the mean time trend (trend dev). Thus the ideal forecast will produce the most linear trends possible while minimizing prediction error.

Given a length-four vector of weights specified in the `weights` argument, the objective function is

f(RSS, age AL, time AL) = weights[1]*sqrt(RSS) + weights[2]*age AL + weights[3]*time AL + weights[4]*trend dev

The optimal combination of sigma parameters will minimize this function for the validation period.

Since the parameter space is restricted to positive values for Ha.sigma, Ht.sigma, and Hat.sigma, it is necessary to use constrained optimization (on [0,Inf]) or reparameterize to log-space (so that the algorithm guesses of log(*.sigma) cannot be negative when transformed back to the original space). By default, `optim.yourcast` uses reparameterization since unconstrained optimization is more straightforward.

In practice, the objective function is rarely unimodal. Thus most optimization algorithms will fail to find the global minima if started in the wrong place. We recommend that users first perform a grid search with `robust.yourcast` and then start `optim.yourcast` at the best guess found by the grid search. If users adopt this strategy, they can pass their `robust.yourcast` output object to `optim.yourcast` with the `robust.out` argument. Then only arguments pertaining to the optimization will need to be considered; arguments pertaining to weights, validation blocks, and `yourcast` inputs will be recovered from the output object.

Value

`list` A list object of class 'optim.yourcast' with the following components:

- **par.opt** A vector of the optimal value for each sigma parameter as indicated by the objective function.
- **aux.robust** A list of information about the run of `optim.yourcast` used by other functions in the `RobustYourCast` library.

Author(s)

Jon Bischof `<jbischof@fas.harvard.edu>`

References

`http://gking.harvard.edu/yourcast`

See Also

`yourcast`,`robust.yourcast`,`run.opt`

4.3 `plot.robust.yourcast`: Plot generation tool for RobustYourCast

Usage

```
plot.robust.yourcast(x,nparam=2,
                                    screen1=list(z =-30,x=-60),
                                    screen2=list(z =-30,x=-60),
                                    screen3=list(z =-30,x=-60),
                                    screen4=list(z =-30,x=-60),
                                    print="device",
                                    filename="diagplots.pdf",...)
```

Arguments

`x`	'robust.yourcast' output object or equivalent
`nparam`	Integer. Number of parameters to be varied at a time in the diagnostic plots. If `nparam=1`, two dimensional plots using `plot` will be used; if `nparam=2`, three dimensional plots using `wireframe` will be used.
`screen1`	List. List with three elements 'x', 'y', and 'z' that rotate the viewing angle for three dimensional plots (passed to `wireframe`). Optimal viewing angles can often by found by increasing the 'z' element by 90 or 180 degrees. This argument pertains to the first plot (top left) only.
`screen2`	List. Same as `screen1`, but applies to the second plot (top right) only.
`screen3`	List. Same as `screen1`, but applies to the third plot (bottom left) only.
`screen4`	List. Same as `screen1`, but applies to the fourth plot (bottom right) only.
`print`	String. Specifies whether graphical output should be displayed on a device window (`"device"`) or saved directly to a '.pdf' file (`"pdf"`).
`filename`	String. If `"pdf"=TRUE`, the filename of the '`.pdf`' to be created.
`...`	Additional arugments to be passed to plotting method. If one parameter allowed to vary will be `plot`; if two parameters allowed to vary will be `wireframe`.

Details

Uses the output of `robust.yourcast` to produce plots of the objective function and its component diagnostics. Since the function is four-dimensional if no sigma parameters are set to `NA`, only a subset of the parameters can be allowed to vary in each plot. Therefore if all parameters were varied in the grid search from `robust.yourcast`, the function will produce three plots on the same device showing different conditional responses.

Parameters not allowed to vary will be held at their optimum value as identified by the grid search in `robust.yourcast`. Users who want more flexibility in creating diagnostic plots should call the `plot.diag` function directly; this function is intended to give users a quick summary of the `robust.yourcast` output.

The function by default plots the value of the objective function against the sigma parameters. However, by adding the `family` argument to `plot.diag` in the function call, users can also see graphs of its three components specified by strings "rss", "arc.age", or "age.time".

If nparam=1, the three plots will each show the conditional relationship of the diagnostic and one sigma parameter at a time, with the other parameters held constant at their optimum. If nparam=2, the three plots will vary two of the parameters at a time with the third held constant at its optimum. A fourth plot with the all three parameters varied at once will also be displayed.

The three dimensional plots produced when nparam=2 are sometimes not shown at an ideal viewing angle. For that reason users are provided with three screen* arguments to rotate each of the plots.

With the exception of the screen* argument, arguments passed to plot.diag will be applied to all the plots produced by this function.

Value

Device windows with requested plots or '.pdf' files saved in the working directory.

Author(s)

Jon Bischof <jbischof@fas.harvard.edu>

References

http://gking.harvard.edu/yourcast

See Also

robust.yourcast, plot.diag

4.4 `run.opt`: Generate forecasts using optimal sigma combinations

Description

Uses output from robust.yourcast or optim.yourcast function to call yourcast using the identified optimal sigma combination and other arguments sent to yourcast in original call to robust.yourcast or optim.yourcast. Will produce a plot of the forecasts if requested using plot.yourcast.

Usage

```
run.opt(x,quant="best",plot=TRUE,
               Ha.set=NULL,Ht.set=NULL,Hat.set=NULL,
               create.main=TRUE,...)
```

Arguments

`x`	'robust.yourcast' or 'optim.yourcast' output object or equivalent
`quant`	Numeric. If using 'robust.yourcast' output object, specifies the sigma combination to be plotted by its quantile of the objective function among those tested. For example, if `quant=0.5`, the function will use the median sigma combination considered. If left as `"best"`, will use combination with lowest objective function value. If using 'optim.yourcast' output object, will be fixed to `"best"`.
`plot`	Logical. Should yourcast output object be plotted?
`Ha.set`	Numeric. If changed from `NULL`, specifies an alternate value of Ha.sigma to be used in yourcast() run.
`Ht.set`	Numeric. If changed from `NULL`, specifies an alternate value of Ht.sigma to be used in yourcast() run.
`Hat.set`	Numeric. If changed from `NULL`, specifies an alternate value of Hat.sigma to be used in yourcast() run.
`create.main`	Logical. If `plot=TRUE`, should a title for the plots be created that lists the sigma combination used and its quantile (or 'Best') of the objective function?
`...`	Additional arugments to be passed to plot.yourcast(). Commonly used arguments are `print` and `filename`.

Details

Extracts the optimal sigma combination from a `robust.yourcast` or `optim.yourcast` output object and then generates predictions with those sigma values by calling `yourcast`. Other arguments to `yourcast`, including the original data, are also extracted from the output object.

If `plot=TRUE`, the function will also produce a plot of the resulting forecasts by sending the `yourcast` output object to `plot.yourcast`.

Value

`yourcast` object called with same arguments supplied to `robust.yourcast` or `optim.yourcast` and most desireable sigma combination identified by the respective function. If `plot=TRUE` will also create a plot of the forecast using plot.yourcast printed to the device window or to a '`.pdf`' file. If `create.main=TRUE`, the function will create an informative main title for the plots the lists the optimal sigma combination used.

Author(s)

Jon Bischof `<jbischof@fas.harvard.edu>`

References

`http://gking.harvard.edu/yourcast`

See Also

`robust.yourcast`, `optim.yourcast`, `plot.yourcast`

4.5 `plot.diag`: Plot objective function or component diagnostics

Description

Uses output from robust.yourcast function to plot the surface of the objective function or its component diagnostics over different combinations of the three sigma parameters.

Usage

```
plot.diag(x,fix.param=list("vary","vary","opt"),
                      family="obj.fun",
                      lattice.plot="wireframe",
                      screen=list(z=-30,x=-60),
                      print="device",
                      filename="objplot.pdf",
                      args.par=list(),
                      args.print.trellis=list(),...)
```

Arguments

`x` ‘robust.yourcast’ or ‘optim.yourcast’ output object or equivalent

`fix.param` List. A list of length three that specifies which of the sigma parameters will be varied and how the others will be fixed. The three elements of the list coorespond to Ha.sigma, Ht.sigma, and Hat.sigma, respectively. List elements may take value `"vary"` if the parameter is to be varied, `"opt"` if the parameter is to be held fixed at is optimum value (as evaluated by the objective function), or an arbitrary numeric value at which that parameter is to be fixed. If the element is a numeric value, the function will look for the closest value at which the objective function was evaluated to hold the parameter constant. Naturally, at least one parameter must be varied. If any of the parameters was left as `NA` in the grid search, the function will automatically hold it fixed at `NA` regardless of the value of its corresponding list element.

`family` String. Indicates the surface to be plotted. The default, `"obj.fun"`, indicates the objective function, and its four components are specified by strings `"rss"`, `"arc.age"`, `"arc.time"`, or `"trend.deviate"`.

`lattice.plot` String. Type of plot in `lattice` package to be used. The default is `wireframe`, but `levelplot`, `cloud`, and `contourplot` will also work. Note: this only applies when exactly two of the parameters are not fixed.

`screen` List. List with three elements ‘x’, ‘y’, and ‘z’ that rotate the viewing angle for three dimensional plots (passed to `wireframe`). Optimal viewing angles can often by found by increasing the ‘z’ element by 90 or 180 degrees.

`print` String. Specifies whether graphical output should be displayed on a device window (`"device"`) or saved directly to a ‘`.pdf`’ file (`"pdf"`).

`filename` If `print="pdf"`, specifies the filename of the ‘`.pdf`’ created.

`args.par` List. If only one variable is allowed to vary (so that `plot` is the plotting method), a list of arguments (must be labeled) to be passed to `par` such as `col="blue"`, `cex=0.8`, etc.

`args.print.trellis`
List. If two variables allowed to vary (so that `wireframe` is the plotting method), a list of arguments (must be labeled) to be passed to `print.trellis` function used to print `wireframe` plot to the device. Used by `plot.robust.yourcast` to print multiple plots to the same device.

`...` Additional arugments to be passed to plotting method. If one parameter allowed to vary will be `plot`; if two parameters allowed to vary will be `wireframe`.

Details

Function plots the surface of the objective function or any of its three component diagnostics using the grid search output from `robust.yourcast`. The `fix.param` argument specifies which of the three smoothing parameters should be allowed to vary and which should be fixed at a specific value (either the optimum or one chosen by the user). If the user requests that one parameter be varied, the function makes a call to the `plot` function; if two varied, it makes a call to the function specified in `lattice.plot`; if three are varied, it makes a call to `cloud`.

`plot.diag` calls several functions in order to create a plot. To ensure maximum flexibility, the user can pass additional arguments to these functions through use of the `...` argument (for the plotting method) or with the 'args' lists. While `plot.diag` changes some of the defaults of these functions, the user can override these changes by specifying a value for that argument. For example, if the user fails to supply an argument to `main`, the function will create an informative title for the plot. If a value for `main` is supplied, that value will be used.

Value

None. Prints a plot either to the device window or to a '`.pdf`' file.

Author(s)

Jon Bischof `<jbischof@fas.harvard.edu>`

References

`http://gking.harvard.edu/yourcast`

See Also

`robust.yourcast`

References

Girosi, Federico and Gary King. 2008. *Demographic Forecasting*. Princeton: Princeton University Press. http://gking.harvard.edu/files/smooth/.