The `naiveFC` function package for Gretl

Artur Tarassow

Version 0.6

Changelog

- Version 0.6 (May, 2019)
  - Fully revamped framework heavily exploiting the idea of bundles
  - Fix a bug related to some seasonal frequencies
  - Add the 'seasonal median' forecasting method
  - Add further sample datasets of different frequencies to the sample script
  - Make use of the user-contributed gretl package *CvDataSplitter* for computing rolling and recursive forecasts
  - Eventual missing values of the series passed to naiveFC() will be handled internally now
  - Further minor changes
- Version 0.4 (Jan, 2019)
  - correct bug in smeanf() which caused referencing the forecast to the wrong minor period (quarter, month, day) under some circumstances
  - add option to compute median value for both meanf() and smeanf()
  - speed-up: loop in stack_fc() replaced by matrix operation
  - speed-up: use aggregate() in get_mean_obsminor() and avoid a loop
  - use `--contiguous` instead of `--missing` option to ensure correct time-series pattern in case of missing values
- Version 0.3 (Oct, 2018)
  - add NaiveThroughTime() for computing rolling/ recursive forecasts
  - add nttplot() for computing plotting rolling/ recursive forecasts
- Version 0.11 (Oct, 2018)
  - initial public version
  - add AR(1)-based forecasts
  - minor changes and corrections
  - set minimum gretl version to 2018a
- Version 0.1 (Oct, 2018)
  - initial non-public version

Contents

1 Introduction

The `naiveFC` package is a collection of gretl scripts for computing forecasts using simple forecasting methods. These may yield surprisingly good (in terms of forecast accuracy) results, and may serve as benchmarks for more sophisticated approaches. For details on the methods read Rob J Hyndman and George Athanasopoulos' book "Forecasting: Principles and Practice" and especially chapter 3.1. there (URL: https://otexts.org/fpp2/simple-methods.html). The functions implemented in `naiveFC` are inspired by Hyndman's well-known *forecast* package for *R*.

The package currently comprises the following features:

- Point estimates for 9 simple forecasting methods, namely the 'Average' (mean & median), 'Seasonal Average' (seasonal mean & median), 'Naive' (Random-Walk), the "Naive plus drift' the 'Seasonal Naive' and the AR(1) model, are implemented.
- Easy computation of an average of forecasts by taking the mean point forecasts of all available simple forecast methods (depending on whether the underlying time-series has seasonality, or not).
- Computation of moving-window based sequences of forecasts using either the rolling or recursive method.

- Simple plotting of the h-step ahead forecast results as well as moving-window based sequences of forecasts.
- Easy GUI access through the gret menu "Model –> Time series –> naive forecast(s)".

NOTE: Currently, no prediction intervals or elaborated bootstrap methods for computing the whole forecast density are implemented. These may be introduced at some later stage though.

2 Install and load the package

The `naiveFC` package is publicly available on the gretl server. The package must be downloaded once, and loaded into memory each time gretl is started.

```
# turn extra output off
set verbose off
# Download package (only once need)
pkg install naiveFC
# Load the package into memory
include naiveFC.gfn
# Get the help file
help naiveFC
```

3 Example

For illustration we use the AWM-macroeconomic data set comprising various series observed at a quarterly frequency over 28 years. The data set ends in 1998Q4. The objective is to forecast the output gap (series named `YGA`). Note, the last valid observation for `YGA` is 1998Q2. Hence, the 1-step ahead forecast will be computed for period 1998Q3.

The sample script opens the data set, computes and plots some forecasts. Lastly, an average of forecasts is computed which is an average of the (currently) nine forecasting methods implemented (for details see below).

```
open AWM.gdt --quiet

series y = YGA           # output gap
scalar h = 11            # set max. forecast horizon

# Mean forecast
bundle b = null                  # initialize an empty bundle
b = naiveFC(y, "meanFC")         # compute forecasts
naivePlot(&b)                    # plot forecast values
eval b.fc                        # print matrix hodling forecast values

# RW with drift
bundle b = naiveFC(y, "rwdFC")
naivePlot(&b)
eval b.fc

# Average of Forecasts
bundle b = null
b = naiveFC(y, "avgFC")
naivePlot(&b)
eval b.fc                        # 1st col: point forecast, 2nd col: std. deviation
```

The estimated h-step ahead average of forecasts, named 'Average-FC', and the cross-sectional standard deviation ('SD') as well as the point forecasts of the separate methods are reported in the output below for the first three horizons (only 2 digits are shown here):

```
****************************************************
                Naive Forecasting Method
Forecasting method:             avgFC
Start valid data set:           1971:4
End valid data set:             1998:2
Number of observations:         106
Forecast horizon:               10
First observation forecasted:   1998:3
****************************************************
Average-FC SD Mean Median RW RW+Drift AR(1) AR(1)+Trend Seas-Mean Seas-Median Seas-Naive
1998:07  0.99  0.002   0.99   0.99   0.99   0.99   0.99   0.99   0.99  0.99  0.98
1998:10  0.99  0.002   0.99   0.99   0.99   0.99   0.99   0.98   0.99  0.99  0.98
1999:01  0.99  0.001   0.99   0.99   0.99   0.99   0.99   0.98   0.99  0.99  0.99
```

The *private* `NaiveThroughTime()` function implements the easy computation of either (i) "rolling" (fixed window length) or (ii) "recursive" (expanding window length) forecasts for a specific naive forecast type. The user just needs to set the string variable `type_roll` which actually triggers the `NaiveThroughTime()` function. The following sample script computes the 1- to 10-steps ahead random-walk plus drift (*rwdFC*) forecasts based on a rolling window of length `wsize=100`.

```
bundle b = null
bundle opts = null           # additional bundle holding options
opts.type_roll = "rolling"  # "rolling"/"recursive": type of moving-window (optional)
opts.wsize = 100             # moving-window length (optional)

bundle b = naiveFC(y, "rwdFC", opts)
eval b.fc
# Plot outcome
b.preobs_fc = 10        # set no. of pre.-forecast periods obs. to plot (optional)
naivePlot(&b)           # call plotting function
```

Given the length of the current data set and the chosen window length, eight training sets (moving window samples) on which separate forecasts are made, are internally defined. For each of these eight sets, a 10-steps ahead forecast is computed resulting in a sequence of h-step ahead forecasts which are stored in matrix `fc` in bundle `b`. The first 1-step ahead forecast for series y is made conditional on information up to 1996Q3 for period 1996Q4 and is 0.982, and so on (see output below).

The public `naivePlot()` function grabs all relevant information from bundle `b`, and allows for an easy illustration of the moving forecast exercise, as depicted below in Plot 1. The plot nicely illustrates the 10-step ahead forecasts for each of the eight training sets.

```
****************************************************
               Naive Forecasting Method
Forecasting method:                rwdFC
Start valid data set:              1971:4
End valid data set:                1998:2
Number of observations:            106
Forecast horizon:                  10
Moving window length:              100
Number of rolling forecasts:       8
First observation forecasted (h=1): 1996:4
****************************************************
         h=1    h=2     h=3    h=4     h=5    h=6     h=7    h=8     h=9    h=10
1996:3   0.982 0.981   0.981  0.981   0.981  0.981   0.981  0.980   0.980  0.980
1996:4   0.980 0.980   0.979  0.979   0.979  0.978   0.971  0.978   0.978  0.977
1997:1   0.978 0.978   0.977  0.977   0.977  0.977   0.976  0.976   0.976  0.976
```

4 Forecasting methods

A brief description of the implemented forecasting methods follows in this section. The user calls the respective method, by passing a string variable with the respective name, e.g. `string which=”meanFC”` to the main function `naiveFC()` as the 2nd function argument (see subsection 5.1 for details).

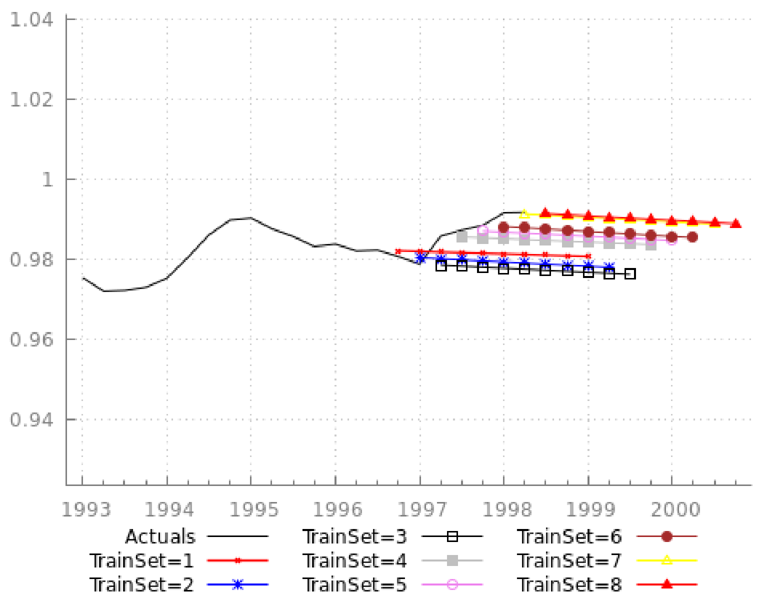

Figure 1: Rolling-window 10-steps ahead random-walk plus drift forecasts.

4.1 meanFC

Forecasts of all future values are equal to the "mean" of the historical data. The h-step ahead forecast conditional on using data up to period T is computed by the historical mean of the time-series:

$$\hat{y}_{T+h|T} = \frac{1}{T} \sum_{i=1}^{T} y_i$$

4.2 medianFC

This method works similar as the meanFC approach, but instead forecasts of all future values are equal to the "median" of the historical data.

4.3 smeanFC

Forecasts of all future values are equal to the "mean" of the historical data for each specific seasonality (e.g. daily, monthly, or quarterly). For example, with monthly data, the forecast for all future for February is equal to mean value of all past February values. With quarterly data, the forecast of all future 2nd quarter values is equal to mean of all past Q2 values. Similar rules apply for other months and quarters, and for other seasonal periods. For daily frequency data one can either have a 7d (standard week) or 5d (business week) season.

4.4 smedianFC

Works as the "smeanFC" method but computes the seasonal median instead of the mean statistics.

4.5 ar1FC

Forecasts are based on an AR(1) model including an intercept where parameters are estimated by standard OLS. Out-of-sample forecasts are recursively (also known as *the iterated forecasting method*) constructed. The 1-step ahead forecast is based on realized values from the previous period. The h-step ahead (for $h > 1$) forecast is based on the forecast value from horizon $h-1$, respectively. We rely on gretl's built-in `ols` and `fcast` commands for estimating the parameters of the model and computing the dynamic forecast, respectively, based on the following time-series model:

$$y_t = \beta_0 + \beta_1 y_{t-1} + u_t \quad u_t \sim iid(0, \sigma^2)$$

4.6 ar1trendFC

Works similar as the *ar1FC* forecasting method but adds a linear trend to the specification. We rely on gretl's built-in `ols` and `fcast` commands for estimating the parameters of the model and computing the dynamic forecast, respectively, based on the following time-series model:

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 T + u_t \quad u_t \sim iid(0, \sigma^2)$$

4.7 rwFC

The random-walk forecast for period $T + h$ equals the value of the last valid observation:

$$\hat{y}_{T+h|T} = y_T$$

4.8 rwdFC

A variation of the *rwFC* method is to allow the forecasts to increase or decrease over time, where the amount of change over time (called the drift) is set to be the average change seen in the historical data. Thus the forecast for period $T + h$ is given by:

$$\hat{y}_{T+h|T} = y_T + h\left(\frac{y_T - y_1}{T-1}\right)$$

4.9 snaiveFC

This is a similar method as the random-walk forecast but especially useful for seasonal data. Forecasts are based on an ARIMA(0,0,0)(0,1,0)[m] model where m is the seasonal period. We rely on gretl's built-in `arima` and `fcast` commands for computation.

4.10 avgFC

This is meant to be a convenience function. This function computes the mean and cross-sectional (across forecast methods) standard deviation of the respective point forecasts at each forecast horizon using all k forecasting methods available. For annual data the **(i)** mean-forecast (`meanFC`), the **(ii)** median-forecast (`medianFC`), **(iii)** random-walk without drift (`rw`FC), **(iv)** random-walk with drift (`rwdFC`), (**v**) the AR(1)-forecast without linear trend (`ar1FC`) and (**vi**) with linear trend (`ar1trendFC`) will be computed. For data with seasonality (e.g., monthly, quarterly etc.) we also add the **(vii)** the seasonal-mean (`smeanFC`), **(viii)** the seasonal-median (`smedianFC`) and the **(ix)** seasonal-naive forecast (`snaiveFC`) to the collection of forecasts.

The resulting matrix `fc` stored in the bundle will be of dimension $h \times (2 + k)$ comprising the h-step ahead combined cross-sectional mean forecasts in the 1st column and the cross-sectional standard-deviations across all (k) separate forecasting methods.

Note, the `avgFC` method will not be computed when combined with either the *rolling* or *recursive* moving-window method.

5 Public functions and parameter values

The following public functions are intended to be used for scripting purposes only. Hence, calling these functions through the GUI will not return any printout. Instead, the user can access the function `avgfc_gui()` through the gretl GUI menu "Model –> Time series –> naive forecast(s)" which allows steering key functions by *point and click*.

5.1 The naiveFC() function

The naiveFC() function marks the main function. The function arguments are:

```
naiveFc(const series y, string which "Select forecasting method", bundle
opts[null] "Bundle incl.  optional parameters"))
```

Return type: `bundle`

The user must pass a time-series (potentially with missing values) and a string variable selecting one of the supported forecasting methods. Optionally, the user pass another bundle element including further parameters which are described below. In case the function is successfully called, a bundle including various items will be returned to the user.

The additional parameters which can be passed by means of the `opts` bundle to `naiveFC()` are:

Parameter	**Data type**	**Description**	**Default value**
`h`	int	Forecast horizon: Must be > 0	10
`type_roll`	str	Moving-window type: "static", "rolling", "recursive"	"static"
`wsize`	int	Length of the moving window in case `type_roll` takes either "rolling" or "recursive".	25% of the total sample of series y
`verbose`	bool	Print details or not: either 0 or 1	1 (True)
`preobs_fc`	int	No. of pre-forecast observations to plot when calling the `naivePlot()` function	25% of the total sample of series y
`title`	str	Title string for the plot when calling `naivePlot()`	empty
`ylab`	str	String of the y-label of the plot	empty
`xlab`	str	String of the x-label of the plot	empty
`filename`	str	String of the path+filename+file-extension (either png, pdf or eps) for storing the plot	"display"

Table 1: Parameters which can be set through the optional bundle `opts`.

In case the user sets `type_roll` to either “rolling” or “recursive”, the internal private function `NaiveThroughTime()` is called. This function computes point forecasts for some method either in *(i)* a rolling or *(ii)* a recursive manner. For details see the user-contributed package *CvDataSplitter* written by this author (URL:).

The bundle returned by `naiveFC()` includes the following items:

Item	**Data type**	**Description**
`T_all`	int	No. of observation of series y originally passed to `naiveFC()` by the user.
`T`	int	No. of effective observations after omitting eventual missing values in series y.
`t1_label`	str	Date string of the 1st valid obs. of the training set.
`t2_label`	str	Date string of the last valid obs. of the (last) training set.
`fc`	matrix	If `type_roll`=”static”: $h \times m$ matrix holding h-step ahead forecasts made by m methods. For `type_roll`=”rolling/ recursive”: $tr \times h$ matrix holding h-step ahead forecasts for tr training sets using a single method.
`h`	int	No. of maximum forecast horizon
`wsize`	int	Length of the moving-window scheme.
`type_roll`	str	String of the chosen moving-window scheme
`verbose`	bool	Flag for printing details or not
`w_drift`	bool	Flag whether a drift-term is incl. or not.
`w_trend`	bool	Flag whether a linear trend is incl. or nor
`is_seas`	bool	Flag whether the data set has seasonal frequency or not
`which`	str	String of the selected forecasting method
`fcperiods`	string array	Array of date strings referring the last training date on which each of the moving-window forecasts is based on.

Table 2: Parameters which are returned by the function `naiveFC()`.

5.2 The naivePlot() function

The `naivePlot()` function is a convenience function for plotting the results initially compiled by having successfully called `naiveFC()`. This user function accepts a bundle argument in pointer form as returned by `naiveFC()`:

```
naivePlot(bundle *self "Bundle returned from naiveFC()")
```

Return type: `void`