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
  - Ad the the 'seasonal median' forecasting method
  - Add the option to run further sample datasets of different frequencies
  - Make use the user-contributed gretl package *CvDataSplitter* for computing rolling or recursive forecasts.
  - Eventual missing values of the series passed to naiveFC() will be handled internally.
  - 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 very simple forecasting methods. These may yield surprisingly good (in terms of forecast accuracy) results, and may serve as benchmarks for more sophisticated methods. 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 `naiveFC` function package comprises currently the following features:

- Point estimates for nine simple forecasting methods, namely the 'Average' (mean & median), 'Seasonal Average' (seasonal mean & median), 'Naive' (Random-Walk), the 'Seasonal Naive', the "Naive plus drift' 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).

- Plotting of the h-step ahead forecasts.
- Computation of either rolling or recursive 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 for 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        0.98
1998:10    0.99 0.002 0.99   0.99  0.99    0.99  0.99        0.98      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
```

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              # generate an empty bundle
bundle opts = null           # additional bundle holding options
opts.type_roll = "rolling"  # "rolling"/"recursive": type of moving-window forecasting
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 0.995, 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. The plot nicely illustrates the 10-steaph 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

Here a brief description of the implemented forecasting methods follows. The user calls the respective method, by passing a string variable (`which`) with the respective name, e.g. `string which=''meanFC''`.

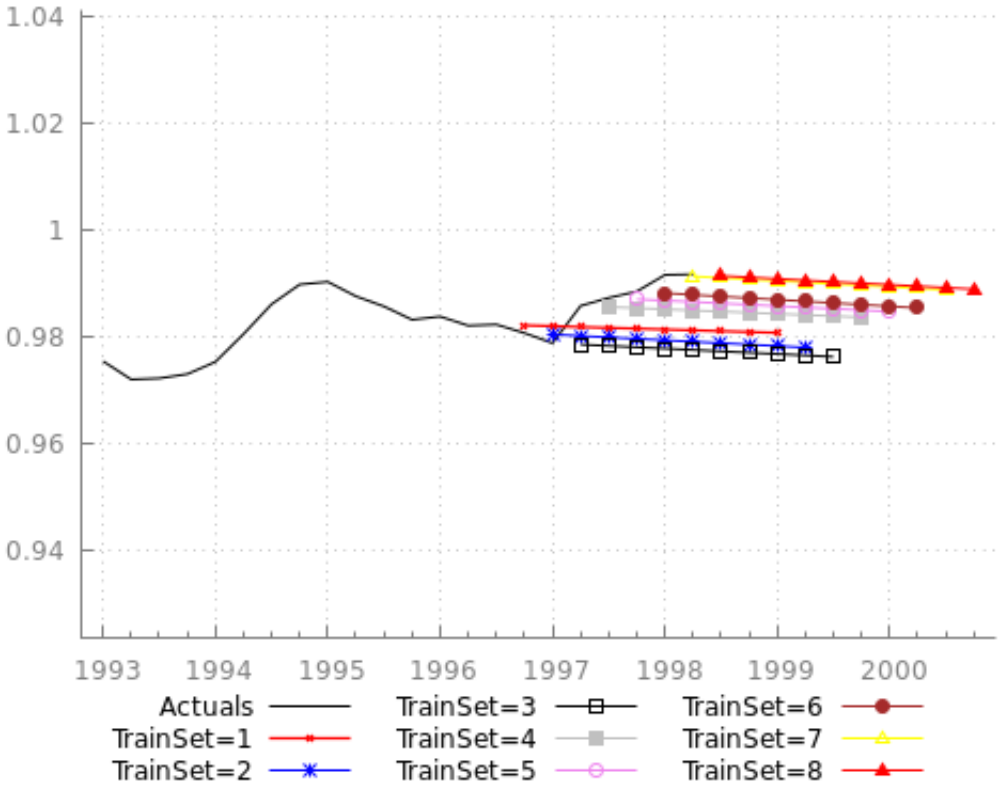

Figure 1: Rolling-window 10-steps ahead random-walk plus drift forecasts.

4.1 meanFC

Forecasts of all future values are equal to the "mean" of the historical data. The forecast is computed by

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

4.2 medianFC

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 February values 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.

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:

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

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:

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

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

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.

5 List of public functions

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 naiveFC

Main function for setting up the forecasting method and conducting the computation.

```
naiveFc(const series y "Historic data", string which "Select method", bundle
opts[null])
```

Return type: `bundle`

5.2 naivePlot

Function for plotting results obtained from calling the `naiveFC()` function. The user passes a bundle in pointer form.

```
naivePlot(bundle *self)
```

Return type: `void`

5.3 meanf

Forecasts of all future values are equal to the "mean" or "median" of the historical data.

```
meanf(const series y "Actuals", int h[1::10] "Forecast horizon", bool
use_median[0] "compute median", scalar level[64:99:90] "Confidence level", bool
fan[0], int nboot[0::0], int blength[2::4] "Block length bootstrap")
```

Return type: `matrix`

The function arguments are:

1. `series y`: series of actual historical outcomes

2. `int h[1::10]`: Max. forecast horizon to compute (default 10)
3. `bool use_median[0]`: Compute the median of the historical data instead (default no)
4. `scalar level[64:90:90]`: Set the confidence level for prediction intervals (default 90). NOT supported yet!
5. `bool fan[0]`: If 1, level is set to seq(51,99,3). This is suitable for fan plots (default no). NOT supported yet!
6. `int nboot[0::0]`: If >0 , use a bootstrap method with *nboot* iterations to compute prediction intervals. NOT supported yet!
7. `int blength[2::4]`: If `nboot`>0, set the block-length of the stationary block-bootstrap (default 4). NOT supported yet!

An $h \times 1$ matrix holding the h-step ahead point forecasts will be returned.

5.4 ar1f

This function simply estimates an AR(1) model incl. an intercept using standard OLS. The out-of-sample forecasts are recursively (also known as *iterated forecast*) 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.

```
ar1f(const series y "Actuals", int h[1::10] "Forecast horizon", scalar
level[64:99:90] "Confidence level", bool fan[0], int nboot[0::0], int
blength[2::4] "Block length bootstrap")
```

Return type: `matrix`

The function arguments are the same as for `meanf()`, apart from the 3rd option:

An $h \times 1$ matrix holding the h-step ahead point forecasts will be returned.

5.5 rwf

For naïve forecasts, we simply set all forecasts to be the value of the last observation. A variation on the naïve 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 time $T + h$ is given by:

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

```
rwf(const series y "Actuals", int h[1::10] "Forecast horizon", bool drift[0]
"0=Random-Walk wo drift, 1=w drift", scalar level[64:99:90] "Confidence level",
bool fan[0], int nboot[0::0], int blength[2::4] "Block length bootstrap")
```

Return type: `matrix`

The function arguments are the same as for `meanf()`, apart from the 3rd option::

1. `series y`: series of actual historical outcomes
2. `int h[1::10]`: Max. forecast horizon to compute (default 10)
3. bool drift[0]: If '1', fits a random walk with drift term (default '0': no drift)
4. `scalar level[64:90:90]`: Set the confidence level for prediction intervals (default 90). NOT supported yet!
5. `bool fan[0]`: If 1, level is set to seq(51,99,3). This is suitable for fan plots (default no). NOT supported yet!
6. `int nboot[0::0]`: If >0 , use a bootstrap method to compute prediction intervals. NOT supported yet!
7. `int blength[2::4]`: If `nboot`>0, set the block-length of the stationary block-bootstrap (default 4). NOT supported yet!

An $h \times 1$ matrix holding the h-step ahead point forecasts will be returned.

5.6 snaive

A similar method compared to naïve forecasts using `rwf()` is useful for seasonal data. Each forecast is equal to the last observed value from the same season of the year (e.g., the same month of the previous year).

```
snaive(const series y "Actuals", int h[1::10] "Forecast horizon", scalar
level[64:99:90] "Confidence level", bool fan[0], int nboot[0::0], int
blength[2::4] "Block length bootstrap")
```

Return type: `matrix`

The function arguments are the same as for `meanf()`, apart from the 3rd option:

An $h \times 1$ matrix holding the h-step ahead point forecasts will be returned.

5.7 fcplot

```
fcplot(const series y, matrix fc, string title[null], string ylab[null], string
xlab[null], string filename[null] "'display' OR 'Path+filename'")
```

Return type: `void`

The function arguments are:

1. `series y`: series of actual historical outcomes
2. `matrix fc`: $h \times 1$ matrix holding the h-step ahead forecasts
3. `string title`[null]: provide a string for the title.
4. `string ylab`[null]: provide a string for the y-label.
5. `string xlab`[null]: provide a string for the x-label.

6. `string filename`[null]: provide a string for the x-label (default 'display' which plots the forecast directly on the screen).

5.8 avgfc

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 simple forecasting methods available. For annual data only the **(i)** mean-forecast (`meanf`), the **(ii)** median-forecast (`meanf`), **(iii)** Random-Walk without drift (`rwf`), **(iv)** Random-Walk with drift (`rwf`) and (**v**) the AR(1)-forecast will be computed. For data with seasonality (e.g., monthly, quarterly etc.) we also add the **(vi)** the seasonal-mean (`smean`), **(vii)** the seasonal-median (`smean`) and the **(viii)** seasonal-naive forecast (`snaive`) to collection of forecasts.

```
avgfc(const series y "Actuals", int h[1::10] "Forecast horizon", scalar
level[64:99:90] "Confidence level", bool fan[0], int nboot[0::0], int
blength[2::4] "Block length bootstrap")
```

Return type: `matrix`

The function arguments are the same as for `meanf()`, apart from the 3rd option.

An $h \times (2 + k)$ matrix holding the h-step ahead point forecasts in the 1st column and the cross-sectional standard-deviation across all (k) forecasts will be returned.

5.9 NaiveThroughTime

This function computes a selected naive forecast method through time for specific forecast horizons. Either "rolling" (fixed window length shifting through time) or "recursive" (expanding window through time) are supported. This allows you to analyze the h-step ahead forecast made at specific observations.

```
NaiveThroughTime(bundle *b, string which, bool verbose[1])
```

Return type: adds matrix FC which is of dimension $R \times h$ where R refers to the no. of rolling/recursive forecasts and h to the max. forecast horizon.

The function arguments are:

1. `bundle b`: provide some at least required information in a bundle
2. `string which`: provide a string for selecting the underlying forecast method ("meanf", "medianf", "smean", "smedianf", "snaive", "rwf", "rwfd" (random-walk with drift), "avgfc")
3. `bool verbose`[1]: print some information (default: print)

The user needs to pass at least the following information into the bundle:

1. `series b.y`: series to be forecasted

Furthermore, the user can set the following additional parameters:

1. `int b.wsize`: length of the (initial) window (default $0.25 \times T$ where T is the total no. of observations of series y)

2. `int h`: max. forecast horizon (default 12)

3. `string type_roll`: Choose between "rolling" or "recursive" window forecasts (default “rolling”)

After having called `NaiveThroughTime()`, the bundle will contain a matrix named “FC“ of dimension $R \times h$ where R denotes the total number of forecasts made and h to the max. forecast horizon selected.

5.10 nttplot

This function plots the rolling/ recursive h-step ahead forecasts computed by function `NaiveThroughTime()` before.

```
nttplot(bundle *b, string title[null], string ylab[null], string xlab[null],
string filename[null] "'display' OR 'Path+filename'")
```

Return type: `void`

The function arguments are:

1. `bundle b:` provide bundle after having called `NaiveThroughTime()` before.

2. `string title`[null]: provide a string for the title.

3. `string ylab`[null]: provide a string for the y-label.

4. `string xlab`[null]: provide a string for the x-label.

5. `string filename`[null]: provide a string for the x-label (default 'display' which plots the forecast directly on the screen).