The naiveFC function package for Gretl

Artur Tarassow

Version 0.11

Changelog

- 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 six simple forecasting methods, namely the 'Average' (mean), 'Seasonal Average' (seasonal mean), '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.

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) five forecast methods implemented (for details see below).

```
open AWM.gdt --quiet

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

# SMEANF
smean = smeanf(x, h)
print smean
fcplot(x, smean, "Seasonal Mean Forecast")

# RW with drift
rwd = rwf(x, h, 1)
print rwd
fcplot(x, rwd, "Random-Walk + Drift Forecast")

# Average of Forecasts
avg_fc = avgfc(x, h)
print avg_fc             # 1st col: point forecast, 2nd col: std. deviation
fcplot(x, avg_fc[,1], "Avg. of Forecasts")
```

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:

```
        average-fc      sd            meanf         rwf          rwf+drift
AR(1)          smean          snaive
h=1       0.99176     0.0025708       0.99446       0.99160       0.99154
0.99181       0.99397       0.98720
h=2       0.99203     0.0022136       0.99446       0.99160       0.99149
0.99201       0.99426       0.98840
h=3       0.99264     0.0015085       0.99446       0.99160       0.99143
0.99219       0.99465       0.99150
```

4 List of public functions

4.1 meanf

Forecasts of all future values are equal to the average (or "mean") of the historical data.

```
meanf(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:

1. `series y`: series of actual historical outcomes
2. `int` h[1::10]: Max. forecast horizon to compute (default 10)

3. `scalar level[64:90:90]`: Set the confidence level for prediction intervals (default 90). NOT supported yet!

4. `bool fan[0]`: If 1, level is set to seq(51,99,3). This is suitable for fan plots (default no). NOT supported yet!

5. `int nboot[0::0]`: If >0 , use a bootstrap method with *nboot* iterations to compute prediction intervals. NOT supported yet!

6. `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.

4.2 smeanf

Forecasts of all future values are equal to the average 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 the last observed February value. With quarterly data, the forecast of all future Q2 values is equal to the last observed Q2 value (where Q2 means the second quarter). Similar rules apply for other months and quarters, and for other seasonal periods.

```
smeanf(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().

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

4.3 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().

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

4.4 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(), despite 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.

4.5 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().

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

4.6 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).

4.7 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)** Random-Walk without drift (*rwf*), **(iii)** Random-Walk with drift (*rwf*) and (**iv**) the AR(1)-forecast will be computed. For data with seasonality (e.g., monthly, quarterly) we also add the **(v)** the seasonal-mean (*smean*) and the **(vi)** 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().

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.