FEP – the forecast evaluation package for *gretl*

Artur Tarassow and Sven Schreiber*

version 2.41, January 2020

Abstract

The FEP function package for the *gretl* program is a collection of functions for computing different types of forecast evaluation statistics as well as tests. For ease of use a common scripting interface framework is provided, which is flexible enough to accommodate future additions. Most of the functionality is also accessible through a graphical dialog window interface within *gretl*. This documentation explains the usage and capabilities as well as providing some econometric background where necessary. An illustration with expert forecasts of euro area growth is also provided.

Contents

1 Introduction 2

2 The applyFCTests() function 2

3 Forecast descriptive statistics 4

3.1 Calculate the (forecast error) loss 4

3.2 Draw some loss functions 7

3.3 Tools for binary outcomes 7

3.3.1 The Kuipers Score (KS) 7

3.3.2 Probability scores – doPS() and probscore() 8

3.4 Tools for point forecasts – ForecastMetrics() 8

4 Evaluation of individual forecasts 9

4.1 Mincer-Zarnowitz test of forecast unbiasedness – doMZtest() 9

4.2 Holden-Peel test of forecast efficiency – doHPtest() 9

4.3 Campbell-Ghysels efficiency tests – doCGtest() 12

4.4 Elliott/Komunjer/Timmermann test (asymmetric loss) – doEKTtest() 14

4.5 Pesaran-Timmermann test of market timing – doPTtest() 15

4.6 Diebold-Lopez direction-of-change test – doDLtest() 17

*Tarassow: atecon@posteo.de, https://sites.google.com/site/arturtarassow; Schreiber: IMK Düsseldorf and Free University Berlin, econ.svens.de.
The FEP function package is a collection of gretl functions for computing different types of forecast evaluation statistics as well as tests. The FEP package currently comprises the functions listed in Table 1. In Section 6 we explain the easy usage from gretl’s menu-driven interface (GUI), but until then we focus on the scripting way of performing the analysis. A function package can be downloaded and installed simply by invoking the install option of the pkg command:

```
pkg install FEP
```

Then, in each work session, as with all function packages the FEP package needs to be loaded by:

```
include FEP.gfn
```

### 2 The applyFCtests() function

This is a convenient top-level function which calls the other test functions in the background as needed. The standard workflow is to define the needed gretl bundle with your input data and choices, and then call this function with the appropriate string code. Here’s an example with the Mincer-Zarnowitz test:

---

We gratefully acknowledge useful comments from Michał Rubaszek.

A “bundle” is a flexible gretl datatype which is basically a collection of other gretl objects. See Section 10.7 of the gretl user guide (as of November 2017).
<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>applyFCtests</td>
<td></td>
<td>Top-level function</td>
</tr>
<tr>
<td>2</td>
<td>doDMtest()</td>
<td>DM</td>
<td>Diebold-Mariano regression based test of forecast accuracy</td>
</tr>
<tr>
<td>3</td>
<td>doGWtest()</td>
<td>GW</td>
<td>Giacomini-White regression based test of forecast accuracy</td>
</tr>
<tr>
<td>4</td>
<td>doCWtest()</td>
<td>CW</td>
<td>Clark-West regression based test of forecast accuracy</td>
</tr>
<tr>
<td>5</td>
<td>doMZtest()</td>
<td>MZ</td>
<td>Mincer-Zarnowitz regression based test of optimal (point) forecasts</td>
</tr>
<tr>
<td>6</td>
<td>doHPtest()</td>
<td>HP</td>
<td>Holden-Peel extension of the Mincer-Zarnowitz regression based test</td>
</tr>
<tr>
<td>7</td>
<td>doCGtest()</td>
<td>CG</td>
<td>Campbell-Ghysels sign or signed rank (Wilcoxon type) tests for unbiasedness or efficiency of forecasts</td>
</tr>
<tr>
<td>8</td>
<td>doEKTtest()</td>
<td>EKT</td>
<td>Elliott, Komunjer &amp; Timmermann test of forecast rationality under flexible loss</td>
</tr>
<tr>
<td>9</td>
<td>doDLtest()</td>
<td>DL</td>
<td>Diebold-Lopez direction of change test</td>
</tr>
<tr>
<td>10</td>
<td>doKS()</td>
<td>KS</td>
<td>Computes the (Hanssen-) Kuipers Score for forecasts of binary outcomes</td>
</tr>
<tr>
<td>11</td>
<td>doPTtest()</td>
<td>PT</td>
<td>Pesaran-Timmermann test of market timing</td>
</tr>
<tr>
<td>12</td>
<td>DrawLoss()</td>
<td></td>
<td>draws a single loss function and its associated confidence interval</td>
</tr>
<tr>
<td>13</td>
<td>DrawLoss2()</td>
<td></td>
<td>draws two loss functions and their associated confidence intervals</td>
</tr>
<tr>
<td>14</td>
<td>getLoss()</td>
<td></td>
<td>calculate the (forecast error) loss series</td>
</tr>
<tr>
<td>15</td>
<td>doPS() / probscore</td>
<td>PS</td>
<td>calculate the log and quadratic probability scores for forecasts of binary outcomes</td>
</tr>
</tbody>
</table>
bundle b
series b.y = growth # assuming 'growth' exists in current workfile
series b.fc = myforecast # ditto
applyFCtests(&b, "MZ")

The first three lines set up the bundle and put some relevant data in it. Note that the names “y” and “fc” need to match exactly and are case sensitive. See Table 2 for the possible and/or necessary bundle elements and their names, which depends on the test that you wish to perform. It is possible to store additional elements in the bundle that are not used by a certain test. Therefore you can set up the bundle once and then pass it around as an input to various tests.

In the last line some things are noteworthy: The applyFCtests() function does not have any return value, so the function call stands for itself without any assignment to a variable. The results of the tests are instead added to the bundle (“b” here) that is passed to the function. To actually enable the function to change the input bundle we need to pass it in so-called pointer form, which just means to prepend the ampersand character: “&b”. Finally, a string code must be specified to indicate which test should be run, where the possible codes are given in Table 1. There appear also some supplementary functions in that table which do not have a string code; those functions have to be called directly and cannot be accessed through applyFCtests(). On the other hand, several string codes may be included in a single call to applyFCtests(), separated by spaces.

If the function is called like this, then there will typically be some printed output showing the test results. The details depend on the respective background function, see the corresponding documentation below. The other possibility to access the results is to inspect the new members that are added to the bundle. Again, see the documentation below to learn which new members each function adds to the bundle.

3 Forecast descriptive statistics

3.1 Calculate the (forecast error) loss

The function getLoss() helps to calculate the forecast losses (disutilities) implied by the given forecast errors, according to various loss function assumptions such as lin-lin, square, linex, double linex. See Table 3.

---

3If you’re obsessed with saving lines of code, you might instead use something like the following: bundle b = defbundle("y",growth, "fc",myforecast).

4See “Function programming details”, Section 13.4 of the gretl user guide (as of November 2017).
Table 2: Bundle members needed for each function

<table>
<thead>
<tr>
<th></th>
<th>CG</th>
<th>DL</th>
<th>DM</th>
<th>GW</th>
<th>CW</th>
<th>EKT*</th>
<th>HP</th>
<th>MZ</th>
<th>KS</th>
<th>PT</th>
<th>PS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>series inputs</strong> ([f]: optional)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Realizations</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>Forecast values</td>
<td>fc</td>
<td>f1</td>
<td>f2</td>
<td>f1</td>
<td>f2</td>
<td>fc</td>
<td>fc</td>
<td>fc</td>
<td>fc</td>
<td>fc</td>
<td>fc</td>
</tr>
<tr>
<td>Forecast errors (replaces y, fc)</td>
<td>(E)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary indicator</td>
<td>yup</td>
<td>yup</td>
<td>yup</td>
<td>yup</td>
<td>yup</td>
<td>yup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary FC of yup</td>
<td>fcup</td>
<td>fcup</td>
<td>fcup</td>
<td>fcup</td>
<td>fcup</td>
<td>fpc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC of yup prob.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Further tested</td>
<td>[CGX]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>list inputs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exog. regressors</td>
<td>[z]</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruments</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>scalar inputs</strong> (mostly integer / all are optional)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC horizon</td>
<td>[fhor]</td>
<td>[fhor]</td>
<td>[fhor]</td>
<td>[fhor]</td>
<td>[fhor]</td>
<td>[fhor]</td>
<td>[fhor]</td>
<td>[fhor]</td>
<td>[fhor]</td>
<td>[fhor]</td>
<td>[fhor]</td>
</tr>
<tr>
<td>Loss type</td>
<td>[loss]</td>
<td>[loss]</td>
<td>[loss]</td>
<td>[loss]</td>
<td>[loss]</td>
<td>[loss]</td>
<td>[loss]</td>
<td>[loss]</td>
<td>[loss]</td>
<td>[loss]</td>
<td>[loss]</td>
</tr>
<tr>
<td>Lag (efficiency test)</td>
<td>[k]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bootstrap iter.</td>
<td></td>
<td>[nboot]</td>
<td>[nboot]</td>
<td>[nboot]</td>
<td>[nboot]</td>
<td>[nboot]</td>
<td>[nboot]</td>
<td>[nboot]</td>
<td>[nboot]</td>
<td>[nboot]</td>
<td>[nboot]</td>
</tr>
<tr>
<td>Robust switch</td>
<td></td>
<td>[robust]</td>
<td>[robust]</td>
<td>[robust]</td>
<td>[robust]</td>
<td>[robust]</td>
<td>[robust]</td>
<td>[robust]</td>
<td>[robust]</td>
<td>[robust]</td>
<td>[robust]</td>
</tr>
<tr>
<td>Initial shape</td>
<td></td>
<td>[a0]</td>
<td>[a0]</td>
<td>[a0]</td>
<td>[a0]</td>
<td>[a0]</td>
<td>[a0]</td>
<td>[a0]</td>
<td>[a0]</td>
<td>[a0]</td>
<td>[a0]</td>
</tr>
<tr>
<td>Conditional</td>
<td></td>
<td>[cond]</td>
<td>[cond]</td>
<td>[cond]</td>
<td>[cond]</td>
<td>[cond]</td>
<td>[cond]</td>
<td>[cond]</td>
<td>[cond]</td>
<td>[cond]</td>
<td>[cond]</td>
</tr>
<tr>
<td>Verbosity</td>
<td>[verb]</td>
<td>[verb]</td>
<td>[verb]</td>
<td>[verb]</td>
<td>[verb]</td>
<td>[verb]</td>
<td>[verb]</td>
<td>[verb]</td>
<td>[verb]</td>
<td>[verb]</td>
<td>[verb]</td>
</tr>
</tbody>
</table>

**string input (optional)**

| Loss drawing | [lossdraw] | [lossdraw] |

**Notes:** (*: EKT refers to the doEKTtest() variant. For the matrix-based variant doEKTtest_matrix() see the source code.)

- **robust**: can be 0 (default) or 1 (use HAC/robust SE);
- **loss**: can be 1 (U-shape, default) or 2 (V-shape);
- **cond**: can be 0 (unconditional test, default) or 1 (conditional test);
- **verb**: can be 0 (no details, default) or 1 (print details);
- **lossdraw**: can be “no” (default), “display”, or consist of path + filename;
- **fcup**: is binary, not a probability;
- **k**: can be 0 (default, no test on lags) or a positive (not too large) integer; note that only a single lag is tested;
- **a0**: can be between 0 and 1 (non-integer, default 0.5).
### Table 3: Forecast error loss

<table>
<thead>
<tr>
<th>Function</th>
<th>getLoss(matrix fce, string LF[null], matrix param[null], matrix realiz[null])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Calculate the (forecast error) loss</td>
</tr>
<tr>
<td>Return type</td>
<td>matrix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function arguments</th>
<th>matrix fce</th>
<th>T by k matrix of forecast errors; if realiz is given, fce is understood as the forecasts themselves</th>
</tr>
</thead>
<tbody>
<tr>
<td>string LF</td>
<td>(optional)</td>
<td>specify loss function: &quot;ll&quot; (or “linlin”), &quot;qq&quot; (or &quot;quadquad&quot;), &quot;sq&quot; (or &quot;square&quot;, default), &quot;abs&quot;, &quot;linex&quot; (Varian), &quot;dlinex&quot; (or &quot;dle&quot;) for double linex (Granger)</td>
</tr>
<tr>
<td>matrix param</td>
<td>(optional)</td>
<td>loss function parameter(s), default 0.5; param must be a 1-element or 2-element vector</td>
</tr>
<tr>
<td>matrix realiz</td>
<td>(optional)</td>
<td>matrix of realizations</td>
</tr>
</tbody>
</table>

| Output | Returns a matrix with forecast error losses, of the same (T x k) dimension as the input matrix. |

| Reference | Varian [1975], Granger [1999] |

Notes: About input format for param: For gretl versions until 2017a scalar values must explicitly be provided as a pseudo matrix (e.g [0.4]) while for later versions, gretl accepts a scalar as a 1x1 matrix.
3.2 Draw some loss functions

Draw a single (DrawLoss) or two (DrawLoss2) loss functions and their associated confidence intervals. See Table 4.

3.3 Tools for binary outcomes

3.3.1 The Kuipers Score (KS)

The (Hanssen-) Kuipers Score (KS) is used to evaluate binary outcomes. Let \( f = \{0, 1\} \) be a forecast of the binary variable \( y = \{0, 1\} \). The KS is defined as the difference between the probability of detection (POD) and the probability of false detection (POFD).\(^5\) The POD is the proportion of times where \( y = 1 \) is correctly predicted. The POFD is the proportion of times where \( y = 1 \) is wrongly predicted. Thus, KS is defined as \( \text{KS} = \text{POD} - \text{POFD} \). See Table 5 for the function interface and elements.

\(^5\)It seems that this terminology is not universal. Sometimes the POD might be called hit rate, whereas usually the hit rate denotes something else. Similarly with the POFD and the false alarm rate.
Table 5: Kuipers Score

<table>
<thead>
<tr>
<th>Function</th>
<th>doKS(bundle *b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>compute the Kuipers Score</td>
</tr>
<tr>
<td>Return type</td>
<td>none (void)</td>
</tr>
</tbody>
</table>

Function arguments

- Pointer to the model bundle. This bundle includes as members:
  - series yup: binary-valued series of actuals that takes the value of unity for ups, otherwise zero
  - series fcup: binary-valued forecast that takes the value of unity for ups, otherwise zero
  - scalar verb: 0 = no printout, 1 = print details

Output

The following new elements are stored into the model bundle:

- scalar KSstat: Kuipers score
- matrix KSmat: matrix holding all classifications

Reference Pesaran [2015, p. 396]

Table 6: Probability scores

<table>
<thead>
<tr>
<th>Function</th>
<th>probscore(matrix y, matrix Pr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>computes the log (LPS) and quadratic (QPS) probability scores</td>
</tr>
<tr>
<td>Return type</td>
<td>matrix</td>
</tr>
</tbody>
</table>

Function arguments

- matrix y: binary-valued vector of actuals that takes the value of unity for ups and otherwise zero
- matrix Pr: vector of forecast probabilities

Output

2-element (1 by 2) vector with QPS, LPS

3.3.2 Probability scores – doPS() and probscore()

The probscore() function computes forecast accuracy statistics used for probability forecasts. The observed outcome is still binary, but in contrast to the Kuipers score the forecast here is a continuous probability. See Table 6. The doPS() function is just a thin wrapper around probscore that is harmonized with the evaluation test functions in the package; see Table 7.

3.4 Tools for point forecasts – ForecastMetrics()

The ForecastMetrics() function comprises the computation of various evaluation metrics for point forecasts. For computation of five standard measures, we apply gretl’s built-in function fcstats(). In total twelve measures are currently implemented – for details see Table 8. For further readings we refer to Hyndman and Koehler [2006]. Details on the ForecastMetrics() function can be found in Table 9.
Table 7: Probability scores wrapper

<table>
<thead>
<tr>
<th>Function</th>
<th>doPS(bundle *b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>computes the log (LPS) and quadratic (QPS) probability scores</td>
</tr>
<tr>
<td>Return type</td>
<td>none (void)</td>
</tr>
</tbody>
</table>

Function arguments
- Pointer to the model bundle. This bundle includes as members:
  - series yup: binary-valued series of actuals that takes the value of unity for ups, otherwise zero
  - series pfc: probability forecast (between 0 and 1) that yup takes the value of unity

Output
- The following new elements are stored into the model bundle:
  - scalar qps: quadratic probability score
  - scalar lps: log probability score

4 Evaluation of individual forecasts

4.1 Mincer-Zarnowitz test of forecast unbiasedness – doMZtest()

Define the $h$-step ahead forecast made in $t$ as $f_{t+h|t}$ and the actual outcome as $y_{t+h}$. Mincer&Zarnowitz suggest to run the following regression:

$$ y_{t+h} = \beta_0 + \beta_1 f_{t+h|t} + u_{t+h} $$

Unbiasedness is viewed as a joint test of $\beta_0 = 0$ and $\beta_1 = 1$. (H0: Forecast is unbiased and efficient. H1: Forecast is biased and/or inefficient.) Usually the corresponding test statistics is compared with asymptotic critical values from the F-distribution.

However, remaining serial correlation in $u_{t+h}$ yields inefficient estimates. Also the small sample properties are unknown. To account for these two potential sources of inefficiency, the user can compute HAC robust standard errors as well as bootstrap p-values. An intercept will be automatically inserted.

See Table 10 for a synopsis.

4.2 Holden-Peel test of forecast efficiency – doHPtest()

The Mincer-Zarnowitz regression can be extended to include another regressor (or a whole vector of additional regressors), $Z_t$, such that (assuming for simplicity that $Z_t$ is a scalar value)

$$ y_{t+h} = \beta_0 + \beta_1 f_{t+h|t} + \beta_2 Z_t + u_{t+h} $$

The hypothesis to test for a strong form of unbiasedness involves the null $\beta_0 = 0$, $\beta_1 = 1$ and $\beta_2 = 0$. An intercept will be automatically inserted.
Table 8: Computations of point forecast evaluation measures

<table>
<thead>
<tr>
<th>Measure</th>
<th>Abbreviation</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean error</td>
<td>ME</td>
<td>$\text{mean}(y_t - f_t)$</td>
</tr>
<tr>
<td>Media error</td>
<td>MdE</td>
<td>$\text{median}(y_t - f_t)$</td>
</tr>
<tr>
<td>Root mean squared error</td>
<td>RMSE</td>
<td>$\sqrt{\text{mean}((y_t - f_t)^2)}$</td>
</tr>
<tr>
<td>Mean absolute error</td>
<td>MAE</td>
<td>$\text{mean}(</td>
</tr>
<tr>
<td>Median absolute error</td>
<td>MdAE</td>
<td>$\text{median}(</td>
</tr>
<tr>
<td>Mean percentage error</td>
<td>MPE</td>
<td>$\text{mean}\left(\frac{</td>
</tr>
<tr>
<td>Mean absolute percentage error</td>
<td>MAPE</td>
<td>$\text{mean}\left(\frac{</td>
</tr>
<tr>
<td>Median absolute percentage error</td>
<td>MdAPE</td>
<td>$\text{median}\left(\frac{</td>
</tr>
<tr>
<td>Mean absolute error relative to the mean of realizations</td>
<td>MAM</td>
<td>$\text{mean}\left(\frac{</td>
</tr>
<tr>
<td>Median absolute error relative to the median of realizations</td>
<td>MdAMd</td>
<td>$\text{median}\left(\frac{</td>
</tr>
<tr>
<td>Root mean square percentage error</td>
<td>RMSPE</td>
<td>$\sqrt{\text{mean}\left(\frac{</td>
</tr>
<tr>
<td>Root median square percentage error</td>
<td>RMdSPE</td>
<td>$\sqrt{\text{median}\left(\frac{</td>
</tr>
<tr>
<td>Symmetric mean absolute percentage error</td>
<td>sMAPE</td>
<td>$\text{mean}\left(\frac{</td>
</tr>
<tr>
<td>Symmetric median absolute percentage error</td>
<td>sMdAPE</td>
<td>$\text{median}\left(\frac{</td>
</tr>
<tr>
<td>Symmetric mean absolute absolute percentage error</td>
<td>sMAAPE</td>
<td>$\text{mean}\left(\frac{</td>
</tr>
<tr>
<td>Symmetric median absolute absolute percentage error</td>
<td>sMdAAPE</td>
<td>$\text{median}\left(\frac{</td>
</tr>
</tbody>
</table>
### Table 9: Point forecast measures

<table>
<thead>
<tr>
<th>Function</th>
<th>ForecastMetrics(series y, list fc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Computes various evaluation statistics for point forecasts</td>
</tr>
<tr>
<td>Return type</td>
<td>matrix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function arguments</th>
<th>series y: series of actual outcomes</th>
<th>list fc: list of k alternative forecasts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>16 by k matrix with 16 measures for k different forecasts</td>
<td></td>
</tr>
</tbody>
</table>

### Table 10: Mincer-Zarnowitz test

<table>
<thead>
<tr>
<th>Function</th>
<th>doMZtest(bundle *b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>computes the Mincer-Zarnowitz test regression of unbiasedness</td>
</tr>
<tr>
<td>Return type</td>
<td>none (void)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function arguments</th>
<th>Pointer to the model bundle. This bundle includes as members:</th>
</tr>
</thead>
<tbody>
<tr>
<td>series y</td>
<td>actual observations</td>
</tr>
<tr>
<td>series fc</td>
<td>h-step ahead forecast</td>
</tr>
<tr>
<td>scalar robust</td>
<td>0 = no robust SEs (default), 1 = compute HAC SEs</td>
</tr>
<tr>
<td>scalar nboot</td>
<td>0 = no bootstrap (default), or number of bootstrap iterations</td>
</tr>
<tr>
<td>scalar verb</td>
<td>0 = no printout (default), 1 = print details</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>The following new elements are stored into the model bundle:</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalar MZstat</td>
<td>test statistics</td>
</tr>
<tr>
<td>scalar MZpval</td>
<td>p-value</td>
</tr>
</tbody>
</table>

| Reference          | Mincer and Zarnowitz [1969]                                  |
Table 11: Holden-Peel test

<table>
<thead>
<tr>
<th>Function</th>
<th>doHPtest(bundle *b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>computes the Holden-Peel variant of the Mincer-Zarnowitz test regression of unbiasedness</td>
</tr>
<tr>
<td>Return type</td>
<td>none (void)</td>
</tr>
</tbody>
</table>

Function arguments
- Pointer to the model bundle. This bundle includes as members:
  - series y: actual observations
  - series fc: h-step ahead forecast
  - list z: gretl list of additional control variables
  - scalar robust: 0 = no robust SEs (default), 1 = compute HAC SEs
  - scalar nboot: 0 = no bootstrap (default), or number of bootstrap iterations
  - scalar verb: 0 = no printout (default), 1 = print details

Output
The following new elements are stored into the model bundle:
- scalar HPstat: test statistics
- scalar HPpval: p-value

Reference
Holden and Peel [1990]

Again the user can compute HAC robust standard errors and/or bootstrap p-values; see Table 11.

4.3 Campbell-Ghysels efficiency tests – doCGtest()

Here we understand “efficiency” in a broad sense, comprising unbiasedness for example.

Let us define the one-period forecast errors as $e_{1t} = y_{t+1} - f_{t+1|t}$. An indicator function indicates whether the forecast error is positive or negative such that $u(z) = 1$ if $z \geq 0$ and $u(z) = 0$ otherwise. The test statistics of the sign test of unbiasedness of forecast errors is

$$S = \sum_{t=1}^{T} u(e_{1t}),$$

where $T$ is the number of available forecast errors. While the signed rank test (see below) is defined as

$$W = \sum_{t=1}^{T} u(e_{1t}) R_{1t}^+$$

with $R_{1t}^+$ referring to the rank of each forecast error when $|e_{11}|, ..., |e_{1T}|$ are placed in ascending order. The forecast errors are independent with zero median. The sign statistic $S$ is binomially distributed with $Bi(T, 0.5)$. Under the additional assumption of symmetrically distributed forecast errors around zero, the test statistics $W$ follows a Wilcoxon signed rank distribution. Note that this test is performed once
Table 12: Campbell-Ghysels sign and signed rank

<table>
<thead>
<tr>
<th>Function</th>
<th>doCGtest (bundle *b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Campbell-Ghysels sign and signed rank (Wilcoxon-type) tests for unbiasedness or efficiency of forecasts</td>
</tr>
<tr>
<td>Return type</td>
<td>none (void)</td>
</tr>
</tbody>
</table>

**Function arguments**
- Pointer to the model bundle. This bundle includes as members:
  - series $y$: actual observations
  - series $fc$: forecast (or series $E$: forecast error)
  - scalar $verb$: 0 = no printout (default), 1 = print details
  - series $CGX$: variable for orthogonality (see also $k$)
  - scalar $k$: set $k > 0$ to test efficiency with respect to information at lag $k$ (own lag of forecast error if $X$ is absent), else $k = 0$ (or omit) to test for unbiasedness

**Output**
- The following new elements are stored into the model bundle:
  - scalar $CGSIGNstat$: test statistic
  - scalar $CGWILCstat$: test statistic
  - scalar $CGSIGNpval$: p-value
  - scalar $CGWILCpval$: p-value

**Reference**
- Campbell and Ghysels [1995], Campbell and Ghysels [1997], Dufour [1981]

**Notes:** The calculation of $CGWILCpval$, the p-value of the Wilcoxon-type signed rank tests, depends on the $WSRpvalue$ function in the special package “extra” for gretl ($\geq v0.41$). Please install that package manually from the gretl package repository if the automatic loading fails.

You set $k = 0$ (or leave it out, as this is the default; see Table 12).

This test idea can also be employed to test for serial correlation in the forecast errors. Construct the product series $Z_{1t}^k = e_{1t}e_{1(t-k)}$ with $k \geq 1$, and compute the statistics:

$$S_k = \sum_{t=k+1}^{T} u(Z_{1t}^k) \quad \text{and} \quad W_k = \sum_{t=k+1}^{T} u(Z_{1t}^k)R_{2t}^+$

where $R_{2t}^+$ is the signed rank of the product $Z_{1t}^k$, $t = 1, \ldots, T$. These location tests were proposed by Dufour [1981]. Serial correlation in the forecast errors will move the centre of their product away from zero. The sign statistic $S_k$ is binomially distributed with $Bi(T - k, 0.5)$. The test statistics $W_k$ follows a Wilcoxon signed rank distribution of size $T - k$. Note, both tests on serial correlation require to set the option $k > 0$, and the necessary product series $Z_{1t}^k$ will be constructed automatically.

Lastly, one can use this framework to assess whether the forecast has made efficient use of the available information represented by the series $X$ in $t$. For this,
one needs to construct the product series $Z^k_t = e_{ht}X^c_{t-k}$ with $k > 0$ based on the recursively re-centered series $X^c_t = X_t - \text{median}(X_1, X_2, \ldots, X_t)$.

This way of re-centering requires, however, that the series $X_t$ is stationary and has no trend.

The sign and signed rank statistics are given by

$$S_{ok} = \sum_{t=k+1}^{T} u(Z^k_t) \quad \text{and} \quad W_{ok} = \sum_{t=k+1}^{T} u(Z^k_t)R^+_{1t},$$

noting that the ranks used refer to the forecast errors, not to $Z^k_t$, to obtain a statistic with a known and valid distribution; see Campbell and Ghysels [1995, pp. 3] or Campbell and Ghysels [1997, p. 560] for a discussion.

4.4 Elliott, Komunjer, and Timmermann test with asymmetric loss – doEKTTest()

This is a test for forecast rationality that allows for asymmetric loss. As a side product the parameters of a general class of loss functions can be estimated.

Elliott et al. [2005] propose a flexible class of loss functions, the so-called EKT loss function:

$$L(e) = [\alpha + (1-2\alpha)I(e < 0)] |e|^p, \quad \alpha \in [0, 1]$$

where $I$ is an indicator function which equals one if the forecast error $e = y - f < 0$, and otherwise zero. Asymmetric loss is given for $\alpha \neq 0.5$, and where values exceeding 0.5 indicate greater aversion to positive forecast errors. Parameter $p$ is a positive integer. Setting $p = 1$ results in a lin-lin loss function with possibly different slopes under asymmetric loss. The quadratic loss function is given for the special setting $p = 2$ and $\alpha = 0.5$. Asymmetric quadratic loss functions are produced under $p = 2$ and $\alpha \neq 0.5$.

The unknown loss function parameter $\alpha$ is estimated by means of a linear instrumental variable (IV) approach. The computation of the linear IV estimator $\hat{\alpha}$ is done iteratively. Using the same notation as in Elliott et al. [2005] we set the initial weighting matrix $S$ to be the identity matrix $I_d$, where $d$ refers to the number of

---

6"Recursive" in the sense that for the calculation of the median in each period $t$ only realizations up to $t$ are used, not all observations from the forecast evaluation sample. Otherwise the centering would contradict the real-time information flow and would not be feasible for actual forecasting.

7When $X_t$ is strictly exogenous without any feedback occurring from the forecast errors to future realizations of $X$ the ranks of $Z^k_t$ could also be used, and this would then indeed equal a Wilcoxon signed rank test on $Z^k_t$. This variant is not implemented, however, because the assumption appears very strong and hardly relevant in practice.

8For historical reasons and backward compatibility the doEKTTest function up to FEP version 2.1 has operated on matrices. The preferred interface now is based on a bundle and series as described in this document. The doEKTTest_series function has fulfilled that role since version 2.0. Starting with version 2.3 the doEKTTest function will follow the preferred interface (be an alias for doEKTTest_series), and a new wrapper function doEKTTest_matrix will provide the old access for those who need it.
instruments used. Based on the initial $S$ one can compute the initial $\hat{\alpha}_1$ which in turn can be used to update the precision matrix $S^{-1} = S^{-1}(\hat{\alpha}_1)$. These steps are repeated until some convergence criteria is met.

The user can use the framework to (i) test for symmetric loss, and (ii) for rationality. As $\hat{\alpha}$ is asymptotically normal and its variance is identical to the one obtained by the standard GMM estimator, one can test the null of symmetry, $H_0 : \alpha = 0.5$, against the alternative, $H_0 : \alpha \neq 0.5$, by a t-test.

For a given $p_0$ the user can test for rationality if the number of instruments $d > 1$. Testing for over-identification by means of a $J$-type test provides a joint test for rationality of the forecasts. The EKT approach allows to test for rationality (i) either for an unknown $\alpha$ or (ii) by imposing the restriction of symmetry $\alpha = 0.5$.

See Table 13 for the synopsis, and note that the instruments $z$ should not contain an intercept, it will be added automatically.

4.5 Pesaran-Timmermann test of market timing – doPTtest()

While the Kuipers score just computes the difference between the hit rate, $H$, and the false alarm rate, $F$, it is not a proper statistical test. Pesaran and Timmermann [1992, PT] have proposed a simple test on market timing using binary outcomes. The basic idea is to test whether predicted ups are independent of actual ups or not.

Let $f = \{0, 1\}$ be a forecast of the binary variable $Y = \{0, 1\}$. Denote the corresponding time series of binary predictions as $x_t$ and actual realizations of “ups” (or unity values) as $y_t$. How the forecaster obtains the forecasts $x_t$ –e.g. through a model of a latent variable in the background– is irrelevant here.

The PT test statistic can be approximated by the t-ratio of the $\beta_1$ coefficient of the following OLS regression

$$y_t = \beta_0 + \beta_1 x_t + u_t.$$  

Under the null hypothesis that predictions and realizations are independently distributed, the restriction $\beta_1 = 0$ holds. The test statistic follows asymptotically a standard normal distribution. A rejection of the null hypothesis indicates predictive failure. Serial correlation in the errors, $u_t$, are likely to occur but can be dealt with by using Bartlett weights to compute HAC standard errors.

A second –non-regression based– approach to compute the test instead is to rely on the correlation coefficient between forecasts and predictions, $\rho$. The test statistic is computed by $\rho \times \sqrt{T}$ where $T$ refers to the number of forecasts available. The test statistic also follows a standard normal distribution asymptotically and cannot be robustified.

See Table 14 for a synopsis.
Table 13: Elliott, Komunjer and Timmermann test

<table>
<thead>
<tr>
<th>Function</th>
<th>doEKTtest_series(bundle *b) &lt;doEKTtest(bundle *b) from v2.2&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Elliott, Komunjer and Timmermann test for forecast rationality that allows for asymmetric loss</td>
</tr>
<tr>
<td>Return type</td>
<td>none (void)</td>
</tr>
</tbody>
</table>

**Function arguments**
- Pointer to the model bundle. This bundle includes as members:
  - series `y`: actual observations
  - series `fc`: h-step ahead forecast
  - (or series `E`): forecast error
  - list `z`: instruments
  - scalar `a0`: initial value of shape parameter `alpha` (between 0 and 1, default 0.5)
  - scalar `loss`: Loss function type: 1 = U-shape (default), 2 = V-shape
  - scalar `verb`: 0 = no printout (default), 1 = print details
  - string `lossdraw`: “no” = no draw (default), “display” = immediate plot, “Path+filename”

**Output**
The following new elements are stored into the model bundle:
- scalar `alpha`: estimated shape parameter
- scalar `V`: estimated variance of `alpha`
- scalar `SymTest`: test statistics of test for symmetric loss function
- scalar `SymPval`: p-value of test for symmetric loss function
- scalar `RatTest`: test statistics of test for rationality conditional on estimated `alpha`
- scalar `RatPv`: p-value of test for rationality conditional on estimated `alpha`
- scalar `RatTest05`: test statistics of test for rationality conditional on symmetry (as if `alpha` = 0.5)
- scalar `RatPv05`: p-value of test for rationality conditional on symmetry (as if `alpha` = 0.5)

**Reference**
Elliott et al. [2005]
Diebold-Lopez direction-of-change test – doDLtest()

Performs the Diebold-Lopez direction of change test which is in principle just Pearson’s $\chi^2$-square test. H$_0$: The direction-of-change forecast has no value meaning that the forecasts and realizations are independent. H$_1$: The direction-of-change forecast has some value. (The side output of $DLinfo$ is closely related to the Kuipers score, namely by adding unity to it.)

See Table 15 for a synopsis.

Evaluation and comparison of multiple forecasts

Diebold-Mariano test – doDMtest()

First, note that there also exists a dedicated contributed gretl function package called “DiebMar.gfn” by Giulio Palomba. It only partly overlaps with the features of the doDMtest() function, so it might be useful for you, too.

The Diebold-Mariano (DM) test of equivalent expected loss takes explicitly into account the underlying loss function as well as sampling variation in the average losses.

We define the $h$-step ahead forecast error and its associated loss function by $e_{t+h|t}$ and $L(e_{t+h|t})$, respectively. The loss differential of two non-nested forecasts for observation $t+h$ is given by $d_{12,t+h|t} = L(e_{1,t+h|t}) - L(e_{2,t+h|t})$. Specifically, one can test the null hypothesis of equal predictive accuracy

$$H_0 : \ E(d_{12,t+h|t}) = 0$$
Table 15: Diebold-Lopez test

<table>
<thead>
<tr>
<th>Function</th>
<th>doDLtest (bundle *b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Diebold-Lopez test for predictive accuracy of a direction-of-change forecast</td>
</tr>
<tr>
<td>Return type</td>
<td>none (void)</td>
</tr>
</tbody>
</table>

**Function arguments**

- **series yup**: binary-valued series of actuals that takes the value of unity for ups, otherwise zero
- **series fcup**: binary-valued forecast that takes the value of unity for ups otherwise zero
- **scalar verb**: 0 = no printout (default), 1 = print details

**Output**

The following new elements are stored into the model bundle:

- **scalar DLstat**: test statistic
- **scalar DLpval**: p-value
- **scalar DLinfo**: additional info value

**Reference**

Diebold and Lopez [1996]

against either an one-sided alternative or against a two-sided alternative. Under the null, the test statistics is $DM = \bar{d}_{12,t+h|t} / \hat{\sigma}_{d_{12,t+h|t}} \overset{D}{\rightarrow} N(0,1)$ where $\bar{d}_{12,t+h|t} = F^{-1} \sum_{j=1}^{F} d_{12,j,t+h|t}$ refers to the sample mean loss differential and $\hat{\sigma}_{d_{12,t+h|t}}$ is a consistent estimate of its standard deviation based on $F$ forecasts available.

However, due to serial correlation in the forecast errors, we compute alternatively the following regression-based version using OLS combined with HAC robust standard errors:

$$d_{12,t+h|t} = \beta + u_t$$

where $u_t$ is an i.i.d. zero-mean error term. The null of equal forecast accuracy between the two point forecasts is defined as $\beta = 0$. The test statistics, $t_{DM}$, follows asymptotically a standard normal distribution.

Harvey et al. [1997] have suggested the following small-sample corrected degrees-of-freedom adjusted t-statistics

$$t_{HLN} = [1 + F^{-1}(1 - 2 \times h) + F^{-2}h \times (h - 1)]^{0.5} \times t_{DM}$$

where $F$ and $h$ refer to the number of forecasts and the $h$-step forecast horizon. See Table 16 for a synopsis.

At the moment we only support two loss functions, namely symmetric linear (V-shape) and symmetric quadratic (U-shape) loss but in principle other loss functions could be applied and may be incorporated into the package in future versions.
Table 16: Diebold-Mariano test

<table>
<thead>
<tr>
<th>Function</th>
<th>doDMtest(bundle *b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Diebold-Mariano test for predictive accuracy, computation regression-based with HAC robust SEs</td>
</tr>
<tr>
<td>Return type</td>
<td>none (void)</td>
</tr>
</tbody>
</table>

Function arguments:
- Pointer to the model bundle. This bundle includes as members:
  - series y: realized values
  - series f1: forecast values of model 1
  - series f2: forecast values of model 2
  - scalar fhor: forecast horizon (default 1)
  - scalar loss: Loss function type: 1 = U-shape (default), 2 = V-shape
  - scalar verb: 0 = print no details (default), 1 = print details

Output:
The following new elements are stored into the model bundle:
- series L: loss differentials
- scalar DMstat: test statistic (not small-sample adjusted)
- scalar DMpvaln: p-value based on standard normal
- scalar DMpvalt: p-value based on t-distribution
- scalar DMstatssc: test statistic w/ small-sample correction
- scalar DMpvalssc: p-value using small-sample correction

Reference:
Diebold and Mariano [1995], Harvey et al. [1997]

5.2 Giacomini-White test – doGWtest()

The Giacomini-White (GW) test is a test on equal conditional predictive ability. Given that the population parameters are not known and must be estimated, it is the actual future loss that is of interest to the forecaster, rather than that based on some population value. The function doGWtest() implements two test approaches: (i) The unconditional test restricts attention to the forecast model whereas (ii) the conditional approach allows the evaluation of the forecasting method, which includes the model, the estimation procedure and the possible choice of estimation window.

The conditional test for forecasting performance tests whether the loss-differential between two forecasts is conditional on current information, \( z_t \). The null hypothesis is then altered to

\[
E \left[ d_{12,t+h} | \hat{y}_{1,t}, \hat{y}_{2,t}, y_{t+h} | z_t \right] = 0
\]

where \( z_t \) is the information set (in Gretl terms a list of potential variables) available at time \( t \). In case the user selects the conditional approach, per default the lagged loss differential, \( d_{12,t-h} \), which is available in period \( t \) for the \( t+h \) forecast, enters the information set. Additional conditioning variables may be included into the list ‘\( z \)’.

At the moment we only support two loss functions, namely symmetric linear (V-shape) and symmetric quadratic (U-shape) loss but in principle other loss func-
Table 17: Giacomini-White test

<table>
<thead>
<tr>
<th>Function</th>
<th>doGWtest(bundle *b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Giacomini-White test on equal conditional predictive ability, computation regression-based</td>
</tr>
<tr>
<td>Return type</td>
<td>none (void)</td>
</tr>
</tbody>
</table>

Function arguments
- Pointer to the model bundle. This bundle includes as members:
  - series y: realized values
  - series f1: forecast values of model 1
  - series f2: forecast values of model 2
  - scalar fhor: forecast horizon (default 1)
  - scalar loss: Loss function type: 1 = U-shape (default), 2 = V-shape
  - scalar cond: 0 = unconditional test (default), 1 = conditional test
  - list z: series to condition on (only if cond = 1)
  - scalar verb: 0 = print no details (default), 1 = print details

Output
- The following new elements are stored into the model bundle:
  - series L: loss differentials
  - scalar GWstat: test statistics (not small-sample adjusted)
  - scalar GWpval: p-value based on \( \chi^2 \)-distribution

Reference
- Giacomini and White [2006]

5.3 Clark-West test – doCWtest()\(^9\)

Implementation of Clark and West [2007], testing equal predictive accuracy in nested models (one-sided test); see Table 18. If the test rejects, there is evidence that the smaller nested model is significantly worse at forecasting. It is the user’s responsibility to ensure that the models are indeed nested, since this cannot be determined from the forecast realizations. Note that even though the test setup is one-sided, reverse outcomes may occur in reality (finite samples); in those cases the given p-value is still computed mechanically as the right-tail probability mass.

Two test statistics are provided, where the second one is an autocorrelation-robust variant of the first. For example, if the forecast horizon exceeds 1 it is expected that the forecast time series display serial correlation and thus robust inference is usually advisable. For a one-step forecast standard inference may be sufficient, depending on the context and the application.

\(^9\)We gratefully acknowledge useful comments from Michał Rubaszek.
Table 18: Clark-West test

<table>
<thead>
<tr>
<th>Function</th>
<th>doCWtest(bundle *b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Clark-West test of equal predictive ability in nested models</td>
</tr>
<tr>
<td>Return type</td>
<td>none (void)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function arguments</th>
<th>Pointer to the model bundle. This bundle includes as members:</th>
</tr>
</thead>
<tbody>
<tr>
<td>series y</td>
<td>realized values</td>
</tr>
<tr>
<td>series f1</td>
<td>forecast values of model 1</td>
</tr>
<tr>
<td>series f2</td>
<td>forecast values of model 2 (larger nesting model)</td>
</tr>
<tr>
<td>scalar verb</td>
<td>0 = print no details (default), 1 = print details</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>The following new elements are stored into the model bundle:</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalar CWstat</td>
<td>test statistic</td>
</tr>
<tr>
<td>scalar CWpval</td>
<td>p-value based on t-distribution (one-sided)</td>
</tr>
<tr>
<td>scalar CWstat_rob</td>
<td>robustified test statistic</td>
</tr>
<tr>
<td>scalar CWpval_rob</td>
<td>p-value based on t-distribution (one-sided)</td>
</tr>
</tbody>
</table>

| Reference | Clark and West [2007] |

6 Menu-driven (GUI) usage

To install the FEP package using the GUI one follows the usual steps for contributed function packages: Open the function package list window for example via the menu File / Function packages / On server, then find the FEP entry and click the install button (or mouse right-click and choose Installation in the context pop-up menu).

For the precise meaning of the inputs to the respective functions please see the function documentation above, but ideally, using this package from gretl’s menu interface should be mostly self-explanatory. From the menus you can only execute one test at a time.

Figure 1 shows the layout of the central window where choices and specifications are entered. In order to keep the number of argument fields in this window within reasonable bounds, some fields have different overloaded meanings depending on which test is chosen. For example, the penultimate entry field expects a gretl list, and this input is relevant for the HP, DL, EKT, and CG variants.\(^{10}\)

The output from executing the function from the GUI is presented to the user in a new window, mostly simply with the printed test result. At the top of that output there is a toolbar including “save” and “bundle” icons. The save button allows you to save first the textual output, and secondly the whole produced bundle to the current gretl session. The bundle button in turn gives you direct access to the

\(^{10}\) This will be transferred internally to the various function arguments “z”, “ref”, or “CGX”, respectively.
various produced bundle members which can also be saved.\footnote{The concrete images of those icons will depend on your operating system and/or your desktop theme, but they are the first two buttons from the left.}

Note that only the actual test functions are choosable from the GUI, not the helper functions from Section 3.

\newpage

\section{Illustrative example}

In the following an applied example using survey-based forecasts for annual growth of euro area (EA) real GDP is presented. Forecasts are averages based on surveys among professional forecasters, $f_{ct}$, conducted by the Economist magazine and obtained through the Macrobond database. The realizations, $y_t$, are from the AMECO database provided by the European Commission. Both series’ values are reported in the data appendix A, and Figure 2 depicts the forecast errors, $e_t = y_t - f_{ct}$. For the following code to work we assume that a gretl annual time-series datafile is loaded with the series named $y$ and $f_{c}$.

In the following the main (but not all) functions of the FEP package are applied. Assuming the package is already installed on the local machine, at the beginning of the session the package must be loaded with:
7.1 Tests of unbiasedness

We run the parametric Mincer-Zarnowitz test on forecast unbiasedness using bootstrap p-values (999 iterations, \(iid\)) and HAC standard errors, see Section 4.1. First an empty bundle is defined, then the series and parameters are added:

```plaintext
bundle b = null
series b.y = y  # realizations
series b.fc = fc # forecasts
b.nboot = 999
b.robust = 1 # 1= HAC robust VCV
b.verb = 1 # 1= print details
applyFCtests(&b, "MZ") # call the top-level function
```

Yields as output:

```
***********************************************
*** Mincer & Zarnowitz test
*** on forecast unbiasedness
Method: Approach using HAC robust VCV.
Bootstrap p-value using 999 iterations.
H0: forecasts are unbiased
Test stat.: 5.6758
```
As can be seen, the null hypothesis cannot be rejected at standard levels which indicates that the forecast errors are unbiased. (Please bear in mind that your exact numerical values may differ slightly due to different initializations of the bootstrap. To avoid this you would have to give gretl’s random number generator a seed: set seed <whatever>)

A non-parametric alternative test approach for forecast unbiasedness was proposed by Campbell and Ghysels, as discussed in Section 4.3. Assuming that the previous code has been executed and thus b, b.y, b.fc and b.verb are already defined, the CG test can be performed as follows:\(^\text{12}\)

```gretl
applyFCtests(&b, "CG")
printf "P-value Wilcoxon-type rank sign test = %.3f\n", b.CGWILCpval
printf "P-value sign test = %.3f\n", b.CGSIGNpval
```

Yields as output:

```
********************************************************
You selected the test(s) of unbiasedness.  
********************************************************
P-value Wilcoxon-type rank sign test = 0.122
P-value sign test = 0.096
```

These nonparametric tests do not provide strong evidence against unbiasedness, either. However, if a researcher assumed an underlying asymmetric distribution the only adequate test would be the nonparametric sign test, which might be considered borderline significant here.\(^\text{13}\)

### 7.2 Tests of efficiency

The Holden-Peel test is a parametric test of efficiency, see Section 4.2. We simply use lagged forecasts, \(f_{t-1}\), as the conditional variable. The HP test with bootstrap HAC standard errors is easily executed by:

```gretl
list b.z = fc(-1) # use lagged forecast as conditional variable
applyFCtests(&b, "HP")
```

\(^{12}\) Formatting the number printout as %.3f in the printf command is already slightly advanced, rounding to three decimal digits. A straightforward alternative that usually also works fine for floating point numbers is %g.

\(^{13}\) The empirical skewness in this sample is -0.6. A skewed distribution would obviously imply non-Gaussianity; the Shapiro-Wilk test of the null of a normal distribution yields a p-value of 0.096, the Doornik-Hansen test in turn produces 0.152, thus somewhat borderline again.
Yields as output:

******************************************************************************
*** Holden & Peel test
*** on forecast efficiency
Method: Approach using HAC robust VCV.
Bootstrap p-value using 999 iterations.
H0: forecasts are efficient
Test stat.: 8.1013
p-value.: 0.2763
******************************************************************************

Hence, this parametric test result does not indicate any issue with forecast efficiency conditional on $f_{C_t-1}$.

A non-parametric version is also provided by the CG test approach which can be used as follows. Note that here $k$ already specifies the lag to be applied to CGX, so the series for CGX should be given contemporaneously (here: $f_c$), not with a lag (not: $f_c(-1)$); otherwise errors due to missing values may occur.

b.k = 1
series b.CGX = fc
applyFCtests(&b, "CG")
printf "P-value Wilcoxon-type rank sign test = %.3f\n", b.CGWILCpval
printf "P-value sign test = %.3f\n", b.CGSIGNpval

Yields as output:

******************************************************************************
You selected the test(s) of orthogonality at lag 1.
******************************************************************************
P-value Wilcoxon-type rank sign test = 0.109
P-value sign test = 0.210

Thus, the non-parametric test results broadly agree with the parametric ones, although we might add that for lag 2 the result is quite different. (The reader is invited to verify this statement using the FEP package and the data in the appendix.)

7.3 Tests of (a)symmetric loss and forecast rationality

The framework proposed by Elliott et al. [2005] extends parts of the previous analysis to the case of asymmetric loss functions, cf. Section 4.4. We will use lagged forecast errors as an additional instrument apart from an intercept for illustration, and assume a quad-quad loss function. (Strictly speaking, the explicit specification of $b.loss$ is redundant, since the quadratic U-shape is the default, see Table 2.)
b.loss = 1  # Loss function (1=quad-quad, 2=lin-lin)
series e = y - fc  # forecast error
series e1 = e(-1)  # lagged forecast error
b.z = e1  # instrumental variable
smpl e e1 --no-missing  # avoid mismatch
b.a0 = 0.5  # Initial value of shape param aT
string b.lossdraw = "display"
applyFCtests(&b, "EKT")
smpl full

Yields as output:

Number of iterations before convergence = 8
************************************************
Test for Symmetry: H0: aT=0.5 vs. H1: aT!=0.5
Estim. alpha = 0.236
Test stat. = -2.41
P-value = 0.0158
************************************************
Rationality Test
Estim. alpha = 0.236
J-statistics = 1.46
P-value = 0.227
************************************************
Rationality Test
Alpha fixed to 0.5
J-statistics = 7.28
P-value = 0.0262
************************************************

The null of symmetric loss can be rejected at the 5% level, and after imposing symmetry anyway one would indeed reject the null of forecast rationality at the 5% level. However, under the estimated asymmetric loss function one cannot reject the null of forecast rationality. Here the suggested conclusion is that the forecasts may be regarded as “rational”, but only if one departs from the assumption of a symmetric forecast error loss function.

The estimated \( \hat{\alpha} \) is 0.236 which indicates greater aversion to negative forecast errors. This is perhaps not surprising considering the sequence of positive forecast errors in the first part of the sample. Apart from the reported test results, a plot of
the estimated loss function is returned (see Figure 3). The upper and lower lines represent a confidence band due to the estimation uncertainty of the asymmetry parameter $\hat{\alpha}$.

![Figure 3: Estimated loss function using the EKT approach](image)

### 7.4 Forecast comparison

The Diebold-Mariano approach tests for equal predictive accuracy of two competing forecasts (Section 5.1). We compare the current forecast for simplicity with the pre-year (naive) forecast assuming linear loss. The test is called by:

```r
b.f1 = fc  # FC 1 series
b.f2 = y(-1)  # FC 2 series: naive pre-year realization
b.loss = 2  # 1="U-shape", 2="V-shape"
applyFCtests(&b, "DM")
```

Yields as output:

```
***********************************************
*** Diebold & Mariano (1995) test
*** on equal forecast accuracy
Loss: V-shape linear.
Forecast horizon: 1
H0: forecasts are equally accurate
Test stat.: -2.5611
p-value (stand. normal): 0.0104
p-value (t-distributed): 0.0209
p-value (small-sample): 0.0178
```
The null hypothesis can be rejected at the 5% significance level for all three versions computed. The negative test statistic (-2.56) indicates that the “real” survey-based forecast is more accurate compared to the naive one.

### 7.5 Directional forecast

Apart from point forecasts, directional forecasts may be of interest. The *Kuipers score* (KS) is a widely applied simple statistic to summarize directional forecasts, see Section 3.3.1. The following lines first compute the dummy variables `yup` and `fcup` which take the value of one if the period-change in realized (or forecast) values is positive (otherwise zero), and then execute the KS computation.

```plaintext
series b.yup = (diff(y) > 0)
series b.fcup = (diff(fc) > 0)
applyFCtests(&b, "KS") # could also use doKS(&b)
```

Yields as output:

```
*************************
*** Kuipers Score ***
Hit Rate       = 1.000
False Alarm Rate = 0.100
Kuipers Score  = 0.900
*************************
```

Accordingly, the forecasts have a perfect hit rate of 1 meaning that all “ups” are correctly predicted. The false alarm rate is only 10% such that the KS statistic is 0.9 which indicates a good directional forecast performance.

Finally, the *Diebold-Lopez* test (Section 4.6) tests whether a directional forecast is significantly different from a coin flip. For reasons of backwards compatibility with older FEP versions the bundle for the DL test currently must not contain other non-binary `y` and `fc` series, and therefore we cannot re-use the previously defined bundle. Instead we build a new bundle here.

The positive test statistic of 1.88 indicates that the actual forecast outperforms a pure coin flip which is statistically confirmed by the p-value.

```plaintext
bundle bDL = null
bDL.verb = 1
series bDL.yup = (diff(y) > 0)
series bDL.fcup = (diff(fc) > 0)
applyFCtests(&bDL, "DL")
```
Yields as output:

```
*** Diebold & Lopez test
*** on directional change
H0: y(t) and fc(t) are independent
Info-value: 1.8750
Test stat.: 13.3875
p-value.: 0.0003
```

In this context the Pesaran-Timmermann test described in Section 4.5 may be seen as an alternative test for directional forecasts.

References


A Data

For replication purposes we reproduce the data used in the illustration in Section 7. For the definitions see the text; source Macrobond and AMECO.

fc y
2000
2001 1.5 2.123350
2002 0.7 0.980211
2003 0.5 0.661216
2004 1.8 2.299620
2005 1.4 1.679013
2006 2.7 3.228410
2007 2.6 3.049257
2008 0.8 0.429603
2009 -3.9 -4.514502
2010 1.7 2.081707
2011 1.5 1.605639
2012 -0.5 -0.886516
2013 -0.4 -0.240252
2014 0.8 1.336558
2015 1.5 2.070988
2016 1.7 1.811255
2017 2.4 2.227502
2018 2.4 2.099555

B Changelog

- version 2.41 (January 2020)
  - add some documentation for the Clark-West test and also provide robust test statistics
  - fix missing printout of the CG tests

- version 2.4 (December 2019)
  - Fix ForecastingMetrics() function point forecast evaluation metrics “RMdSPE”: instead of the median the mean was computed.
  - Add additional point forecast evaluation metrics “MdE”, “MdAE”, “MAM” and MdAMd”.
• version 2.3 (November 2019)
  – Implement experimental version of Clark-West test on equal predictive accuracy of nested models.
  – New function ForecastMetrics() for computing various time-series related measures for the evaluation of point forecasts.
  – As previously announced, rename doEKTtest_series to doEKTtest and doEKTtest to doEKTtest_matrix to harmonize the interfaces. This is backwards-incompatible, but the minimal needed change is to follow the renaming to doEKTtest_matrix (or adapt your used interface).
  – remove several deprecated aliases: doCGWILCTest (use: doCGtest), doCGRANKtest (use: doCGtest), doKStest (use: doKS), doKGtest (use: doHPtest)
  – Remove deprecated compatibility handling of y and fc in doDLtest. (Need to use the documented labels “yup” and “fcup”.)

• version 2.21 (August 2019)
  – replace deprecated isnull() function
  – refactoring code

• version 2.2 (December 2018)
  – add Giacomini-White test on conditional predictive ability
  – switch to new sample data set comprising annual EA GDP growth realizations and forecasts by the Economist Poll of Forecasters
  – In case the forecast horizon is chosen to be \( f_{hor} = 0 \) for both the DM- and GW-Test, respectively, return a warning.

• version 2.11 (March 2018)
  – internal change (related to the ‘strstr’ function) to make FEP compatible with the next gretl version 2018a
  – extended and reformatted help (this document)

• version 2.1 (February 2018)
  – doCGtest():
    * re-centering of the CGX series for the orthogonality test is done properly (recursively) now
    * rewrite the entire CG test apparatus based on new function CamDufStats (referring to Campbell & Dufour 1995, not yet in references), properly taking into account the necessary different variants of constructing the ranks
* remove the method switch CGmeth and always calculate both variants
  - add print-out option for doDLtest, doHPtest and doMZtest
  - fix bug and simplify the computation of the DL test

- version 2.0 (December 2017)
  - switch the documentation to this PDF file
  - harmonize the functions’ interfaces, in a hopefully backwards-compatible way (old behavior not documented anymore)
  - introduce more default values (b.nboot = 0, b.fhor = 1, b.verb = 0...)
  - allow more than one test spec in applyFCtests()
  - For functions that deal with FC errors E, infer the respective series from the bundle input of realizations y and forecasts fc if E isn’t directly given.
  - add probscore() and doPS()
  - deprecate doKStest() in favor of doKS() because it is not a test
  - deprecate doKGtest() in favor of doHPtest() because the reference is Holden-Peel
  - fix bug with the CG tests for \( k > 1 \)
  - deprecate doCGRANKtest() and doCGWILCtest() in favor of single doCGtest() with method switch (also CGRANK didn’t really have ranks in it); and switch to gretl built-in nonparametric test (difftest)

- version 1.3 (July 2017)
  - add a GUI wrapper
  - fix another bug with p-value calculation in the PT case
  - DL test: additional info value now in <bundle>.DLinfo, test stat in <bundle>.DLstat (backward incompatible change)

- version 1.25 (12.06.2017): Correct a bug in the computation of the p-value of the Pesaran-Timmermann test when using HAC S.E.

- version 1.24 (22.05.2017)
  - Add the getLoss() function for computing forecast error losses (written by Sven Schreiber)
  - no specific data type for usage of the FEP package required now

- version 1.23 (09.05.2017): Some small improvements in both DrawLoss() and DrawLoss2() (thanks to Sven Schreiber for suggestions)
• version 1.22 (08.03.2017)
  – corrected: Pesaran-Timmermann test should be a 1-sided not 2-sided alternative
  – DrawLoss() (related to doEKTtest) now also plots the 2 S.E. of the estimated loss function based on the estimated (a-)symmetry parameter alpha
  – add the function DrawLoss2() which can be called manually (see sample script) to plot jointly 2 loss functions as well as its associated 2 S.E.

• version 1.21 (26.01.2017): return classification matrix Ksmat computed for the Kuipers Score

• version 1.20 (15.01.2017)
  – Replace the DiebMar.gfn package (vers. 1.0) for computation of Diebold-Mariano test by own regression-based procedure using HAC robust S.E.
  – *** NOTE: The function’s usage has changed from vers. 1.1 -> 1.2!

• version 1.10 (30.12.2016)
  – Correction of small bug in doDMtest() which occured for loss>2
  – add Kuipers Score
  – add Pesaran-Timmermann Test (1992)

• version 1.01 (24.11.2016): In doKGtest() the linear restriction was not correctly set.