FEP – the forecast evaluation package for *gretl*

Artur Tarassow and Sven Schreiber*

Version 2.1, February 2018

Contents

*Tarassow: University of Hamburg, artur.tarassow@gmail.com, https://sites.google.com/site/arturtarassow; Schreiber: IMK Düsseldorf and Free University Berlin, econ.svens.de.

Table 1: Included functions

#	Function	Code	Description
1	applyFCtests()		Top-level function
2	doDMtest()	DM	Diebold & Mariano regression based test on forecast accuracy
3	doMZtest()	MZ	Mincer & Zarnowitz regression based test on optimal (point) forecasts
4	doHPtest()	HP	Holden & Peel extension of the Mincer & Zarnowitz regression based test
5	doCGtest()	CG	Campbell & Ghysels sign or signed rank (Wilcoxon type) tests for unbiasedness or efficiency of forecasts
6	doEKTtest_series()	EKT	Elliott, Komunjer & Timmermann test of forecast rationality under flexible loss
7	doDLtest()	DL	Diebold & Lopez direction of change test
8	doKS()	KS	Computes the (Hanssen-) Kuipers Score for forecasts of binary outcomes
9	doPTtest()	PT	Pesaran & Timmermann test on market timing
10	DrawLoss()		draws a single loss function and its associated confidence interval
11	DrawLoss2()		draws two loss functions and their associated confidence intervals
12	getLoss()		calculate the (forecast error) loss series
13	doPS() / probscore()	PS	calculate the log and quadratic probability scores for forecasts of binary outcomes

1 Introduction

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 command:

```
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[1]

[1] 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).

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:

```
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.[2] 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. Note that 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 preprend the ampersand character: "&b".[3] Finally, a string code must be specified to indicate which test should be run, where the possible codes are given in table 1. There are 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(). But several string codes may be included in one 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.

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

[3]See "Function programming details", section 13.4 of the gretl user guide (as of November 2017).

Table 2: Bundle members needed for each function

	CG	DL	DM	EKT*	HP	MZ	KS	PT	PS
				series inputs ([]: optional)					
Realizations	y		y	y	y	y			
Forecast values	fc		f1, f2	fc	fc	fc			
Forecast errors (replaces y, fc)	(E)			(E)					
Binary indicator		yup					yup	yup	yup
Binary FC of yup		fcup					fcup	fcup	
FC of yup prob.									pfc
Further tested	[CGX]								
				list inputs					
Exog. regressors					z				
Instruments				z					
				scalar inputs (mostly integer / all are optional)					
FC horizon			[fhor]						
Loss type			[loss]	[loss]					
Lag (efficiency test)	[k]								
Bootstrap iter.					[nboot]	[nboot]			
Robust switch					[robust]	[robust]		[robust]	
Initial shape				[a0]					
Verbosity	[verb]	[verb]	[verb]	[verb]	[verb]	[verb]	[verb]	[verb]	
				string input (optional)					
Loss drawing				[lossdraw]					

Notes:
robust can be 0 (default) or 1 (use HAC/robust SE);
loss can be 1 (U-shape, default) or 2 (V-shape);
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).
*: EKT refers to the doEKTtest_series() variant. For the matrix-based variant doEKTtest() see the source code.

Table 3: Forecast error loss

Function	`getLoss(matrix fce, string LF[null], matrix param[null], matrix realiz[null])`
Description	Calculate the (forecast error) loss
Return type	matrix
Function arguments	matrix *fce*: T by k matrix of forecast errors; if *realiz* is given, *fce* is understood as the forecasts themselves string *LF* (optional): specify loss function: "ll" (linlin), "qq" (or "quadquad"), "sq" (or "square", default), "abs", "linex" (Varian), "dlinex" (or "dle") for double linex (Granger) matrix *param* (optional): loss function parameter(s), default 0.5; *param* must be a 1-element or 2-element vector matrix *realiz* (optional): matrix of realizations
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 its associated confidence interval. 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).[4] 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 $KS = POD - POFD$. See table 5 for the function interface and elements.

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

[4] The terminology is not universal it seems. 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 4: Draw losses

Function	`DrawLoss(int p, scalar aT, scalar V, string fpath[null])`
Description	Draw (forecast error) loss
Return type	none (void)
Function arguments	int *p:* loss function type parameter, 1 = lin-lin, 2 = quad-quad scalar aT*:* loss function shape parameter $\alpha \in [0, 1]$ scalar V: estimated variance of aT string *fpath* (optional): null = display figure (default) or provide complete "path+file name" to store figure
Output	displays plot (or saves to file path)

Function	`DrawLoss2(int p, scalar aT1, scalar V1, scalar aT2, scalar V2, string fpath[null])`
Description	Draw two (forecast error) losses
Return type	none (void)
Function arguments	int *p:* loss function type parameter, 1 = lin-lin, 2 = quad-quad scalar $aT1$*:* loss function shape parameter $\alpha \in [0, 1]$ scalar $V1$: estimated variance of $aT1$ scalar $aT2$*:* loss function shape parameter $\alpha \in [0, 1]$ scalar $V2$: estimated variance of $aT2$ string *fpath* (optional): null = display figure (default) or provide complete "path+file name" to store figure
Output	displays plot (or saves to file path)

Table 5: Kuipers Score

Function	`doKS(bundle *b)`
Description	compute the Kuipers Score
Return type	none (void)
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 and 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

Function	`probscore(matrix y, matrix Pr)`
Description	computes the log (LPS) and quadratic (QPS) probability scores
Return type	matrix
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 of
Output	2-element (1 by 2) vector with QPS, LPS

just a thin wrapper around probscore that is harmonized with the evaluation test functions in the package; see table 7.

4 Evaluation of individual forecasts

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

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

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

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

Table 7: Probability scores wrapper

Function	`doPS(bundle *b)`
Description	computes the log (LPS) and quadratic (QPS) probability scores
Return type	none (void)
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 and 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

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.

See table 8 for a synopsis.

4.2 Holden-Peel test on 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)

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

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.

Again the user can compute HAC robust standard errors and/or bootstrap p-values; see table 9.

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

Table 8: Mincer-Zarnowitz test

Function	`doMZtest(bundle *b)`
Description	computes the Mincer-Zarnowitz test regression on unbiasedness
Return type	none (void)
Function arguments	Pointer to the model bundle. This bundle includes as members: series *y*: actual observations series *fc*: h-step ahead forecast 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 MZstat: test statistics scalar MZpval: p-value
Reference	Mincer and Zarnowitz [1969]

Table 9: Holden-Peel test

Function	`doHPtest(bundle *b)`
Description	computes the Holden-Peel variant of the Mincer-Zarnowitz test regression on unbiasedness
Return type	none (void)
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]

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 $|e_{1t}|$,..., $|e_{1T}|$ 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 you set $k = 0$ (or leave it out, as this is the default; see table 10).

This test idea can also be employed to test for serial correlation in the forecast errors. Construct the product series $Z_{1\mathrm{t}}^{k} = e_{1t}e_{1(t-k)}$ 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 $Z_t^k = e_{ht}X_{t-k}^{c}$ with $k > 0$ based on the recursively re-centered series $X_t^c = X_t - \text{median}(X_1, X_2, \ldots, X_t)$.[5] 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

$$S_{ok} = \sum_{t=k+1}^{T} u(Z_t^k) \quad \text{and} \quad W_{ok} = \sum_{t=k+1}^{T} u(Z_t^k) R_{1t}^{+},$$

noting that the ranks used refer to the forecast errors, not to Z_t^k, 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.[6] This orthogonality test can

[5]"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 eforecast evaluation sample. Otherwise the centering would contradict the real-time information flow and would not be feasible for actual forecasting.

[6]When X_t is strictly exogenous without any feedback occurring from the forecast errors to future realizations of X the ranks of Z_t^k could also be used, and this would then indeed equal a Wilcoxon

Table 10: Campbell & Ghysels sign and signed rank

Function	`doCGtest(bundle *b)`
Description	Campbell & Ghysels sign and signed rank (Wilcoxon-type) tests for unbiasedness or efficiency of forecasts
Return type	none (void)
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* (optional): 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* and *CGWILCstat*: test statistic scalar *CGSIGNpval* and *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.

be achieved by passing the series as CGX in the bundle. See table 10.

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

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.[7]

See table 11 for the synopsis, and note that the instruments z should not contain an intercept, it will be added automatically.

4.5 Pesaran-Timmermann test on 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 (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 β_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, ρ. The test statistics 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 12 for a synopsis.

signed rank test on Z_t^k. This variant is not implemented, however, because the assumption appears very strong and hardly relevant in practice.

[7] For historical reasons and backward compatibility there is a doEKTtest() function that operates with matrices. The preferred function today is the bundle and series-based doEKTtest_series().

Table 11: Elliott, Komunjer and Timmermann test

Function	`doEKTtest_series(bundle *b)`
Description	Elliott, Komunjer and Timmermann test for forecast rationality that allows for asymmetric loss
Return type	none (void)
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 conditional on symmetry (as if alpha = 0.5)
Reference	Elliott et al. [2005]

Table 12: Pesaran & Timmermann test on market timing

Function	`doPTtest(bundle *b)`
Description	Pesaran & Timmermann test on market timing based on binary outcomes
Return type	none (void)
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 and otherwise zero series *fcup*: binary-valued forecast that takes the value of unity for ups otherwise zero scalar *robust*: 0 = correlation-based PT test, 1 = regression-based with HAC robust SEs scalar *verb*: 0 = no printout, 1 = print details
Output	The following new elements are stored into the model bundle: scalar $PTstat$: test statistic scalar $PTpval$: p-value
Reference	Pesaran and Timmermann [1992], Pesaran [2015, p. 398]

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

Diebold & Lopez direction of change test which is in principle just Pearson's χ- square test. H0: The direction-of-change forecast has no value meaning that the forecasts and realizations are independent. H1: 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 13 for a synopsis.

5 Evaluation and comparison of multiple forecasts

5.1 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 $d_{12,t+h} = L(e_{1,t+h|t}) - L(e_{2,t+h|t})$. Specifically, one

Table 13: Diebold and Lopez test

Function	`doDLtest(bundle *b)`
Description	Diebold and Lopez test for predictive accuracy of a direction-of-change forecast
Return type	none (void)
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 and 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]

can test the null hypothesis of equal predictive accuracy

$$H_0: \quad E(d_{12,t+h}) = 0$$

against either an one-sided alternative or against a two-sided alternative. Under the null, the test statistics is $DM = \bar{d}_{12,t+h}/\widehat{\sigma}_{\bar{d}_{12,t+h}} \rightarrow N(0,1)$ where $\bar{d}_{12,t+h} = F^{-1}\sum_{j=1}^{F} d_{12,j,t+h}$ refers to the sample mean loss differential and $\widehat{\sigma}_{\bar{d}_{12,t+h}}$ 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, Leybourne and Newbold have suggested the following small-sample corrected degress-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 14 for a synopsis.

Table 14: Diebold and Mariano test

Function	`doDMtest(bundle *b)`
Description	Diebold and Mariano test for predictive accuracy, compute regression-based with HAC robust SEs
Return type	none (void)
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 statistics (not small-sample adjusted) scalar *DMpvaln*: p-value based on standard normal scalar *DMpvalt*: p-value based on t-distribution scalar *DMpvalssc*: p-value using small-sample correction
Reference	Diebold and Mariano [1995], Harvey et al. [1997]

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.[8]

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

[8] This will be transferred internally to the various function arguments "z", "ref", or "CGX", respectively.

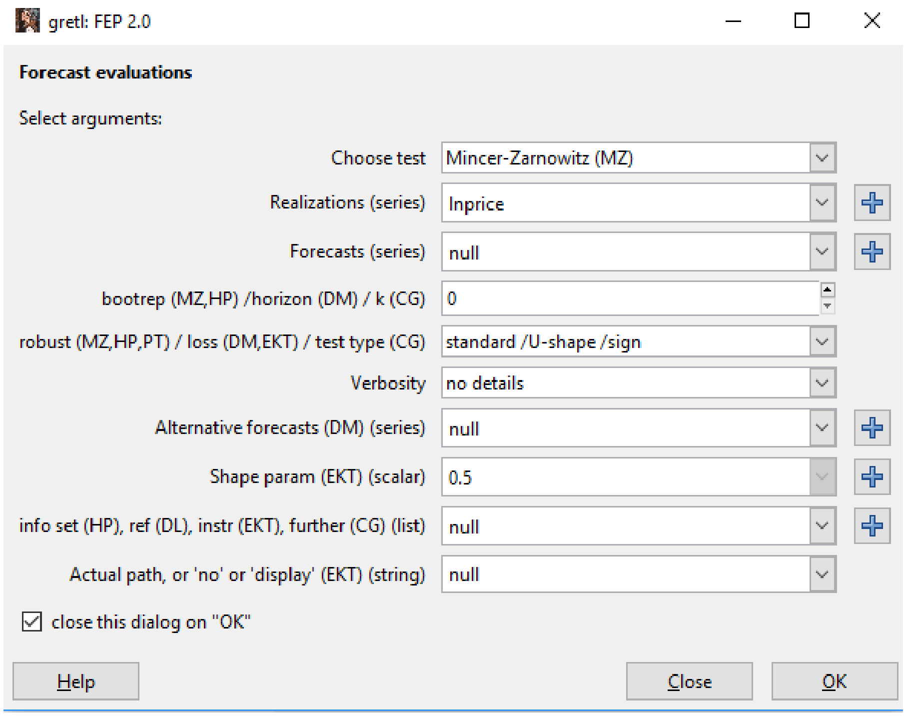

Figure 1: Screenshot of the FEP graphical interface (under Windows 10)

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 various produced bundle members which can also be saved.[9]

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

References

B. Campbell and E. Ghysels. Federal budget projections: A nonparametric assessment of bias and efficiency. *The Review of Economics and Statistics*, 77(1):17 – 31, 1995.

Bryan Campbell and Eric Ghysels. An Empirical Analysis of the Canadian Budget Process. *Canadian Journal of Economics*, 30(3):553–76, August 1997. URL `https://ideas.repec.org/a/cje/issued/v30y1997i3p553-76.html`.

Francis X Diebold and Jose A. Lopez. Forecast evaluation and combination. In G. S. Maddala and C. R. Rao, editors, *Handbook of Statistics*, volume 14 of *Statistical Methods in Finance*, pages 241–268. 1996.

Francis X. Diebold and Roberto S. Mariano. Comparing predictive accuracy. *Journal of Business and Economic Statistics*, 20(1):134–144, 1995.

[9]The concrete images of those icons will depend on your OS and/or your desktop theme, but they are the first two buttons from the left.

Jean-Marie Dufour. Rank tests for serial dependence. *Journal of Time Series Analysis*, 2(3):117–128, 1981. ISSN 1467-9892. doi: 10.1111/j.1467-9892.1981.tb00317.x. URL `http://dx.doi.org/10.1111/j.1467-9892.1981.tb00317.x`.

Graham Elliott, Ivana Komunjer, and Allan Timmermann. Estimation and testing of forecast rationality under flexible loss. *Review of Economic Studies*, 72:1107–1125, 2005.

C.W.J. Granger. Outline of forecast theory using generalized cost functions. *Spanish Economic Review*, 1:161–173, 1999.

David Harvey, Stephen Leybourne, and Paul Newbold. Testing the equality of prediction mean squared errors. *International Journal of Forecasting*, 13(2):281–291, 1997. URL `https://ideas.repec.org/a/eee/intfor/v13y1997i2p281-291.html`.

K Holden and David Peel. On Testing for Unbiasedness and Efficiency of Forecasts. *The Manchester School of Economic & Social Studies*, 58(2):120–27, 1990. URL `http://EconPapers.repec.org/RePEc:bla:manch2:v:58:y:1990:i:2:p:120-27`.

J. Mincer and V. Zarnowitz. The evaluation of economic forecasts. In J. Mincer, editor, *Economic Forecasts and Expectations: Analysis of Forecasting Behavior and Performance*, pages 1 – 46. NBER, NBER, 1969.

M. Hashem Pesaran. *Time Series and Panel Data Econometrics*. Number 9780198759980 in OUP Catalogue. Oxford University Press, 2015. ISBN ARRAY(0x79444480). URL `https://ideas.repec.org/b/oxp/obooks/9780198759980.html`.

M Hashem Pesaran and Allan Timmermann. A Simple Nonparametric Test of Predictive Performance. *Journal of Business & Economic Statistics*, 10(4):561–65, 1992. URL `https://ideas.repec.org/a/bes/jnlbes/v10y1992i4p561-65.html`.

H.R. Varian. A Bayesian approach to real estate assessment. In S.E. Fienberg and A. Zellner, editors, *Studies in Bayesian Econometrics and Statistics in Honor of Leonard J. Savage*, pages 195–208. North-Holland, Amsterdam, 1975.

A Changelog

- 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.