NAG Library Function Document

nag_estimate_agarchI (g13fac)

1 Purpose

nag_estimate_agarchI (g13fac) estimates the parameters of a standard univariate regression-GARCH(p, q) or a univariate regression-type I AGARCH(p, q) process (see Engle and Ng (1993)).

2 Specification

```
#include <nag.h>
#include <nagg13.h>
```

```
void nag_estimate_agarchI (const double yt[], const double x[], Integer tdx,
     Integer num, Integer p, Integer q, Integer nreg, Integer mn,
     Integer isym, double theta[], double se[], double sc[], double covar[],
     Integer tdc, double *hp, double et[], double ht[], double *lgf,
     Nag_Garch_Stationary_Type stat_opt, Nag_Garch_Est_Initial_Type est_opt,
     Integer max_iter, double tol, NagError *fail)
```

3 Description

When **isym** $= 0$, nag_estimate_agarchI (g13fac) models a standard $(\gamma = 0)$ univariate regression-GARCH(p, q) process, with p coefficients α_i, for $i = 1, 2, \ldots, p$, q coefficients, β_i, for $i = 1, 2, \ldots, q$, mean b_o, and k linear regression coefficients b_i, for $i = 1, 2, \ldots, k$, which can be represented by:

$$y_t = b_o + x_t^{\mathrm{T}} b + \epsilon_t \tag{1}$$

$$\epsilon_t \mid \psi_{t-1} \sim N(0, h_t)$$

$$h_t = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{i=1}^{p} \beta_i h_{t-i}, \quad t = 1, \ldots, T.$$

When **isym** $= 1$, nag_estimate_agarchI (g13fac) models an asymmetric GARCH(p, q) process where the conditional variance h_t is given by:

$$h_t = \alpha_0 + \sum_{i=1}^{q} \alpha_i (\epsilon_{t-i} + \gamma)^2 + \sum_{i=1}^{p} \beta_i h_{t-i}, \quad t = 1, \ldots, T.$$

Here T the number of terms in the sequence, y_t denotes the endogenous variables, x_t the exogenous variables, b_o the mean, b the regression coefficients, ϵ_t the residuals, γ the asymmetry parameter, h_t the conditional variance, and ψ_t the information set of all information up to time t.

When **isym** $= 1$, nag_estimate_agarchI (g13fac) provides an estimate for $\hat{\theta}$, the $(p + q + k + 3) \times 1$ parameter vector $\theta = (b_o, b^{\mathrm{T}}, \omega^{\mathrm{T}})$ where $\omega^{\mathrm{T}} = (\alpha_0, \alpha_1, \ldots, \alpha_q, \beta_1, \ldots, \beta_p, \gamma)$ and $b^{\mathrm{T}} = (b_1, \ldots, b_k)$.

isym, **mn**, **nreg** can be used to simplify the GARCH(p, q) expression in (1) as follows:

No Regression or Mean

$y_t = \epsilon_t$,

isym $= 0$,

mn $= 0$,

nreg $= 0$, and

θ is a $(p+q+1) \times 1$ vector.

No Regression

$y_t = b_o + \epsilon_t$,

isym $= 0$,

mn $= 1$,

nreg $= 0$, and

θ is a $(p+q+2) \times 1$ vector.

Note: if the $y_t = \mu + \epsilon_t$, where μ is known (not to be estimated by nag_estimate_agarchI (g13fac)) then (1) can be written as $y_t^\mu = \epsilon_t$, where $y_t^\mu = y_t - \mu$. This corresponds to the case **No Regression or Mean**, with y_t replaced by $y_t - \mu$.

No Mean

$y_t = x_t^{\mathrm{T}} b + \epsilon_t$,

isym $= 0$,

mn $= 0$,

nreg $= k$ and

θ is a $(p+q+k+1) \times 1$ vector.

4 References

Bollerslev T (1986) Generalised autoregressive conditional heteroskedasticity *Journal of Econometrics* **31** 307–327

Engle R (1982) Autoregressive conditional heteroskedasticity with estimates of the variance of United Kingdom inflation *Econometrica* **50** 987–1008

Engle R and Ng V (1993) Measuring and testing the impact of news on volatility *Journal of Finance* **48** 1749–1777

Hamilton J (1994) *Time Series Analysis* Princeton University Press

5 Arguments

Note: for convenience $npar$ will be used here to denote the expression $1 + \mathbf{q} + \mathbf{p} + \mathbf{isym} + \mathbf{mn} + \mathbf{nreg}$ representing the number of model parameters.

1: **yt**[**num**] – const double *Input*

On entry: the sequence of observations, y_t, for $t = 1, 2, \ldots, T$.

2: **x**[**num** $\times$ **tdx**] – const double *Input*

Note: the ith element of the jth vector X is stored in $\mathbf{x}[(i-1) \times \mathbf{tdx} + j - 1]$.

On entry: row t of **x** must contain the time dependent exogenous vector x_t, where $x_t^{\mathrm{T}} = \left(x_t^1, \ldots, x_t{}^k\right)$, for $t = 1, 2, \ldots, T$.

3: **tdx** – Integer *Input*

On entry: the stride separating matrix column elements in the array **x**.

Constraint: **tdx** $\geq$ **nreg**.

4: **num** – Integer *Input*

On entry: the number of terms in the sequence, T.

Constraint: **num** $\geq npar$.

5: **p** – Integer *Input*

On entry: the GARCH(p, q) parameter p.

Constraint: **p** ≥ 0.

6: **q** – Integer *Input*

On entry: the GARCH(p, q) parameter q.

Constraint: **q** ≥ 1.

7: **nreg** – Integer *Input*

On entry: k, the number of refression coefficients.

Constraint: **nreg** ≥ 0.

8: **mn** – Integer *Input*

On entry: if **mn** $= 1$, the mean term b_0 will be included in the model.

Constraint: **mn** $= 0$ or 1.

9: **isym** – Integer *Input*

On entry: if **isym** $= 1$, the asymmetry term γ will be included in the model.

Constraint: **isym** $= 0$ or 1.

10: **theta**$[npar]$ – double *Input/Output*

On entry: the initial parameter estimates for the vector θ.

The first element contains the coefficient α_o, the next **q** elements contain the autoregressive coefficients α_i, for $i = 1, 2, \ldots, q$.

The next **p** elements are the moving average coefficients β_j, for $j = 1, 2, \ldots, p$.

The next element contains the asymmetry parameter γ.

If **est_opt** $=$ Nag_Garch_Est_Initial_False, (when **mn** $= 1$) the next term contains an initial estimate of the mean term b_o and the remaining **nreg** elements are taken as initial estimates of the linear regression coefficients b_i, for $i = 1, 2, \ldots, k$.

On exit: the estimated values $\hat{\theta}$ for the vector θ.

The first element contains the coefficient α_o, the next **q** elements contain the coefficients α_i, for $i = 1, 2, \ldots, q$.

The next **p** elements are the coefficients β_j, for $j = 1, 2, \ldots, p$.

The next element contains the estimate for the asymmetry parameter γ.

If **mn** $= 1$, the next element contains an estimate for the mean term b_o.

The final **nreg** elements are the estimated linear regression coefficients b_i, for $i = 1, 2, \ldots, k$.

11: **se**$[npar]$ – double *Output*

On exit: the standard errors for $\hat{\theta}$.

The first element contains the standard error for α_o.

The next **q** elements contain the standard errors for α_i, for $i = 1, 2, \ldots, q$.

The next **p** elements are the standard errors for β_j, for $j = 1, 2, \ldots, p$.

If **isym** $= 1$, the next element contains the standard error for γ.

If **mn** $= 1$, the next element contains the standard error for b_o.

The final **nreg** elements are the standard errors for b_j, for $j = 1, 2, \ldots, k$.

12: **sc**[$npar$] – double *Output*

On exit: the scores for $\hat{\theta}$.

The first element contains the score for α_o.

The next **q** elements contain the score for α_i, for $i = 1, 2, \ldots, q$.

The next **p** elements are the scores for β_j, for $j = 1, 2, \ldots, p$.

If **isym** $= 1$, the next element contains the score for γ.

If **mn** $= 1$, the next element contains the score for b_o.

The final **nreg** elements are the scores for b_j, for $j = 1, 2, \ldots, k$.

13: **covar**[$npar \times$ **tdc**] – double *Output*

Note: the (i, j)th element of the matrix is stored in **covar**[$(i - 1) \times$ **tdc** $+ j - 1$].

On exit: the covariance matrix of the parameter estimates $\hat{\theta}$, that is the inverse of the Fisher Information Matrix.

14: **tdc** – Integer *Input*

On entry: the stride separating matrix column elements in the array **covar**.

Constraint: **tdc** $\geq npar$.

15: **hp** – double * *Input/Output*

On entry: if **est_opt** = Nag_Garch_Est_Initial_False, **hp** is the value to be used for the pre-observed conditional variance.

If **est_opt** = Nag_Garch_Est_Initial_True, **hp** is not referenced.

On exit: if **est_opt** = Nag_Garch_Est_Initial_True, **hp** is the estimated value of the pre-observed of the conditional variance.

16: **et**[**num**] – double *Output*

On exit: the estimated residuals, ϵ_t, for $t = 1, 2, \ldots, T$.

17: **ht**[**num**] – double *Output*

On exit: the estimated conditional variances, h_t, for $t = 1, 2, \ldots, T$.

18: **lgf** – double * *Output*

On exit: the value of the log likelihood function at $\hat{\theta}$.

19: **stat_opt** – Nag_Garch_Stationary_Type *Input*

On entry: if **stat_opt** = Nag_Garch_Stationary_True, Stationary conditions are enforced.

If **stat_opt** = Nag_Garch_Stationary_False, Stationary conditions are not enforced.

Constraint: **stat_opt** = Nag_Garch_Stationary_True or Nag_Garch_Stationary_False.

20: **est_opt** – Nag_Garch_Est_Initial_Type *Input*

On entry: if **est_opt** = Nag_Garch_Est_Initial_True, the function provides initial parameter estimates of the regression terms (b_0, b^{T}).

If **est_opt** = Nag_Garch_Est_Initial_False, you must supply the initial estimations of the regression parameters (b_0, b^{T}).

Constraint: **est_opt** = Nag_Garch_Est_Initial_True or Nag_Garch_Est_Initial_False.

21: **max_iter** – Integer *Input*

On entry: the maximum number of iterations to be used by the optimization function when estimating the GARCH(p, q) parameters. If **max_iter** is set to 0, the standard errors, score vector and variance-covariance are calculated for the input value of θ in **theta**; however the value of θ is not updated.

Constraint: **max_iter** ≥ 0.

22: **tol** – double *Input*

On entry: the tolerance to be used by the optimization function when estimating the GARCH(p, q) parameters.

23: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_2_INT_ARG_LT

On entry, **num** = ⟨*value*⟩ while 1 + **q** + **p** + **isym** + **mn** + **nreg** = ⟨*value*⟩. These parameters must satisfy **num** ≥ 1 + **q** + **p** + **isym** + **mn** + **nreg**.

On entry, **tdc** = ⟨*value*⟩ while 1 + **q** + **p** + **isym** + **mn** + **nreg** = ⟨*value*⟩. These parameters must satisfy **tdc** ≥ 1 + **q** + **p** + **isym** + **mn** + **nreg**.

On entry, **tdx** = ⟨*value*⟩ while **nreg** = ⟨*value*⟩. These parameters must satisfy **tdx** ≥ **nreg**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument **est_opt** had an illegal value.

On entry, argument **stat_opt** had an illegal value.

NE_INT_ARG_LT

On entry, **max_iter** must not be less than 0: **max_iter** = ⟨*value*⟩.

On entry, **nreg** = ⟨*value*⟩.
Constraint: **nreg** ≥ 0.

On entry, **p** = ⟨*value*⟩.
Constraint: **p** ≥ 0.

On entry, **q** = ⟨*value*⟩.
Constraint: **q** ≥ 1.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

NE_INVALID_INT_RANGE_2

Value ⟨*value*⟩ given to **isym** is not valid. Correct range is 0 to 1.

Value ⟨*value*⟩ given to **mn** is not valid. Correct range is 0 to 1.

NE_MAT_NOT_FULL_RANK

Matrix X does not give a model of full rank.

NE_MAT_NOT_POS_DEF

Attempt to invert the second derivative matrix needed in the calculation of the covariance matrix of the parameter estimates has failed. The matrix is not positive definite, possibly due to rounding errors.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

This example program illustrates the use of nag_estimate_agarchI (g13fac) to model a GARCH(1,1) sequence generated by nag_rand_agarchI (g05pdc), a three step forecast is then calculated using nag_forecast_agarchI (g13fbc).

10.1 Program Text

```
/* nag_estimate_agarchI (g13fac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 6, 2000.
 *
 *
 * Mark 8 revised, 2004
 *
 */
#include <nag.h>
#include <nag_stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <nagg05.h>
#include <nagg13.h>

#define X(I, J) x[(I) *tdx + (J)]

int main(void)
{
  /* Integer scalar and array declarations */
```

```
  Integer                     exit_status = 0;
  Integer                     i, j, k, npar, tdc, tdx, lstate, lr;
  Integer                     *state = 0;

  /* NAG structures and data types */
  NagError                    fail;
  Nag_Boolean                 fcall;

  /* Double scalar and array declarations */
  double                      *covar = 0, *cvar = 0, *et = 0, *ht = 0, *r = 0;
  double                      *sc = 0, *se = 0, *theta = 0, *x = 0, *yt = 0;
  double                      fac1, hp, lgf, xterm;

  /* Choose the base generator */
  Nag_BaseRNG                 genid = Nag_Basic;
  Integer                     subid = 0;

  /* Set the seed */
  Integer                     seed[] = { 1762543 };
  Integer                     lseed = 1;

  /* Set parameters for the (randomly generated) time series ... */
  /* Generate data assuming normally distributed errors */
  Nag_ErrorDistn              dist = Nag_NormalDistn;
  double                      df = 0;

  /* Size of the time series */
  Integer                     num = 1000;

  /* MA and AR parameters */
  Integer                     ip = 1;
  Integer                     iq = 1;
  double                      param[] = { 0.15, 0.1, 0.8, 0.1 };

  /* Asymmetry parameter */
  double                      gamma = -0.3;

  /* Regression parameters */
  Integer                     nreg = 2;
  double                      mean = 3.0;
  double                      bx[] = { 1.5, 2.5 };
  /* ... end of parameters for (randomly generated) time series */

  /* When fitting a model to the time series ... */
  /* Include asymmetry parameter in the model */
  Integer                     isym = 1;

  /* Include mean in the model */
  Integer                     mn = 1;

  /* Use the following maaximum number of iterations and tolerance */
  Integer                     maxit = 50;
  double                      tol = 1e-12;

  /* Enforce stationary conditions */
  Nag_Garch_Stationary_Type  stat_opt = Nag_Garch_Stationary_True;

  /* Estimate initial values for regression parameters */
  Nag_Garch_Est_Initial_Type est_opt = Nag_Garch_Est_Initial_True;

  /* Set the number of values to forecast from the fitted model */
  Integer                     nt = 3;
  /* ... end of model fitting options */


  /* Initialise the error structure */
  INIT_FAIL(fail);

  printf("nag_estimate_agarchI (g13fac) Example Program Results \n\n");

  /* Get the length of the state array */
```

```
  lstate = -1;
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Derive various amounts */
  npar = iq + ip + 1;
  tdc = npar + mn + isym + nreg;
  tdx = nreg;

  /* Calculate the size of the reference vector */
  lr = 2 * (iq + ip + 2);

  /* Allocate arrays */
  if (!(covar = NAG_ALLOC((npar + mn + isym + nreg) * tdc, double)) ||
      !(et = NAG_ALLOC(num, double)) ||
      !(ht = NAG_ALLOC(num, double)) ||
      !(sc = NAG_ALLOC(npar + mn + isym + nreg, double)) ||
      !(se = NAG_ALLOC(npar + mn + isym + nreg, double)) ||
      !(theta = NAG_ALLOC(npar + mn + isym + nreg, double)) ||
      !(state = NAG_ALLOC(lstate, Integer)) ||
      !(r = NAG_ALLOC(lr, double)) ||
      !(x = NAG_ALLOC(num * tdx, double)) ||
      !(cvar = NAG_ALLOC(nt, double)) ||
      !(yt = NAG_ALLOC(num, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Initialise the generator to a repeatable sequence */
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Set up the time dependent exogenous matrix x */
  for (i = 0; i < num; ++i)
    {
      fac1 = (double)(i + 1) * 0.01;
      X(i, 0) = sin(fac1) * 0.7 + 0.01;
      X(i, 1) = fac1 * 0.1 + 0.5;
    }

  /* Generate a realization of a random AGARCH I time series and discard it */
  fcall = Nag_TRUE;
  nag_rand_agarchI(dist, num, ip, iq, param, gamma, df, ht, yt, fcall, r, lr,
                   state, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_agarchI (g05pdc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Generate a realization of a random AGARCH I time series to use */
  fcall = Nag_FALSE;
  nag_rand_agarchI(dist, num, ip, iq, param, gamma, df, ht, yt, fcall, r, lr,
                   state, &fail);
  if (fail.code != NE_NOERROR)
```

```
    {
      printf("Error from nag_rand_agarchI (g05pdc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Adjust the randomly generated time series to take into account for the
     exogenous matrix x */
  for (i = 0; i < num; ++i)
    {
      xterm = 0.0;
      for (k = 0; k < nreg; ++k)
        xterm += X(i, k) * bx[k];

      if (mn == 1)
        yt[i] = mean + xterm + yt[i];
      else
        yt[i] = xterm + yt[i];
    }

  /* Set initial estimates for the parameters */
  for (i = 0; i < npar; ++i)
    theta[i] = param[i] * 0.5;
  if (isym == 1)
    theta[npar + isym - 1] = gamma * 0.5;
  if (mn == 1)
    theta[npar + isym] = mean * 0.5;
  for (i = 0; i < nreg; ++i)
    theta[npar + isym + mn + i] = bx[i] * 0.5;

  /* nag_estimate_agarchI (g13fac).
   * Univariate time series, parameter estimation for either a
   * symmetric GARCH process or a GARCH process with asymmetry
   * of the form (epsilon_(t-1)+gamma)^2
   */
  nag_estimate_agarchI(yt, x, tdx, num, ip, iq, nreg, mn, isym, theta, se, sc,
                       covar, tdc, &hp, et, ht, &lgf, stat_opt, est_opt, maxit,
                       tol, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_estimate_agarchI (g13fac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Display the results */
  printf("       Parameter estimates     Standard errors       "
         "Correct values\n");
  for (j = 0; j < npar; ++j)
    printf("%20.4f             (%6.4f) %20.4f\n", theta[j], se[j],
           param[j]);

  if (isym)
    printf("%20.4f             (%6.4f) %20.4f\n",
           theta[npar + isym - 1], se[npar + isym - 1], gamma);
  if (mn)
    printf("%20.4f             (%6.4f) %20.4f\n", theta[npar + isym],
           se[npar + isym], mean);
  for (j = 0; j < nreg; ++j)
    printf("%20.4f             (%6.4f) %20.4f\n",
           theta[npar + isym + mn + j], se[npar + isym + mn + j], bx[j]);


  /* Now forecast nt steps ahead */
  if (isym)
    {
      gamma = theta[npar + isym - 1];
    }
  else
```

```
    {
      gamma = 0.0;
    }

  /* nag_forecast_agarchI (g13fbc).
   * Univariate time series, forecast function for either a
   * symmetric GARCH process or a GARCH process with asymmetry
   * of the form (epsilon_(t-1)+gamma)^2
   */
  nag_forecast_agarchI(num, nt, ip, iq, theta, gamma, cvar, ht, et, &fail);
  printf("\n%"NAG_IFMT" step forecast = %8.4f\n", nt, cvar[nt - 1]);


 END:
  NAG_FREE(covar);
  NAG_FREE(et);
  NAG_FREE(ht);
  NAG_FREE(sc);
  NAG_FREE(se);
  NAG_FREE(theta);
  NAG_FREE(x);
  NAG_FREE(cvar);
  NAG_FREE(yt);
  NAG_FREE(state);
  NAG_FREE(r);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_estimate_agarchI (g13fac) Example Program Results

       Parameter estimates     Standard errors       Correct values
              0.1045             (0.0486)               0.1500
              0.0884             (0.0235)               0.1000
              0.8196             (0.0433)               0.8000
             -0.7658             (0.2296)              -0.3000
              3.0051             (0.1309)               3.0000
              1.4478             (0.0779)               1.5000
              2.4681             (0.1229)               2.5000

3 step forecast =   1.1019
```