NAG Library Function Document

nag_rand_agarchI (g05pdc)

1 Purpose

nag_rand_agarchI (g05pdc) generates a given number of terms of a type I AGARCH(p,q) process (see Engle and Ng (1993)).

2 Specification

```
#include <nag.h>
#include <nagg05.h>
```

```
void nag_rand_agarchI (Nag_ErrorDistn dist, Integer num, Integer ip,
     Integer iq, const double theta[], double gamma, Integer df, double ht[],
     double et[], Nag_Boolean fcall, double r[], Integer lr, Integer state[],
     NagError *fail)
```

3 Description

A type I AGARCH(p,q) process can be represented by:

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

where $\epsilon_t \mid \psi_{t-1} = N(0, h_t)$ or $\epsilon_t \mid \psi_{t-1} = S_t(df, h_t)$. Here S_t is a standardized Student's t-distribution with df degrees of freedom and variance h_t, T is the number of observations in the sequence, ϵ_t is the observed value of the GARCH(p,q) process at time t, h_t is the conditional variance at time t, and ψ_t the set of all information up to time t. Symmetric GARCH sequences are generated when γ is zero, otherwise asymmetric GARCH sequences are generated with γ specifying the amount by which positive (or negative) shocks are to be enhanced.

One of the initialization functions nag_rand_init_repeatable (g05kfc) (for a repeatable sequence if computed sequentially) or nag_rand_init_nonrepeatable (g05kgc) (for a non-repeatable sequence) must be called prior to the first call to nag_rand_agarchI (g05pdc).

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

1: **dist** – Nag_ErrorDistn *Input*

On entry: the type of distribution to use for ϵ_t.

dist = Nag_NormalDistn
A Normal distribution is used.

dist = Nag_Tdistn
A Student's t-distribution is used.

Constraint: **dist** = Nag_NormalDistn or Nag_Tdistn.

2: **num** – Integer *Input*

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

Constraint: **num** ≥ 0.

3: **ip** – Integer *Input*

On entry: the number of coefficients, β_i, for $i = 1, 2, \ldots, p$.

Constraint: **ip** ≥ 0.

4: **iq** – Integer *Input*

On entry: the number of coefficients, α_i, for $i = 1, 2, \ldots, q$.

Constraint: **iq** ≥ 1.

5: **theta**[**iq** + **ip** + **1**] – const double *Input*

On entry: the first element must contain the coefficient α_o, the next **iq** elements must contain the coefficients α_i, for $i = 1, 2, \ldots, q$. The remaining **ip** elements must contain the coefficients β_j, for $j = 1, 2, \ldots, p$.

Constraints:

$$\sum_{i=2}^{\mathbf{iq}+\mathbf{ip}+1} \mathbf{theta}[i-1] < 1.0;$$

theta$[i-1] \geq 0.0$, for $i = 2, 3, \ldots,$ **ip** + **iq** + 1.

6: **gamma** – double *Input*

On entry: the asymmetry parameter γ for the GARCH(p, q) sequence.

7: **df** – Integer *Input*

On entry: the number of degrees of freedom for the Student's t-distribution.

If **dist** = Nag_NormalDistn, **df** is not referenced.

Constraint: if **dist** = Nag_Tdistn, **df** > 2.

8: **ht**[**num**] – double *Output*

On exit: the conditional variances h_t, for $t = 1, 2, \ldots, T$, for the GARCH(p, q) sequence.

9: **et**[**num**] – double *Output*

On exit: the observations ϵ_t, for $t = 1, 2, \ldots, T$, for the GARCH(p, q) sequence.

10: **fcall** – Nag_Boolean *Input*

On entry: if **fcall** = Nag_TRUE, a new sequence is to be generated, otherwise a given sequence is to be continued using the information in **r**.

11: **r**[**lr**] – double *Input/Output*

On entry: the array contains information required to continue a sequence if **fcall** = Nag_FALSE.

On exit: contains information that can be used in a subsequent call of nag_rand_agarchI (g05pdc), with **fcall** = Nag_FALSE.

12: **lr** – Integer *Input*

On entry: the dimension of the array **r**.

Constraint: $\mathbf{lr} \geq 2 \times (\mathbf{ip} + \mathbf{iq} + 2)$.

13: **state**$[dim]$ – Integer *Communication Array*

Note: the dimension, dim, of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument **state** in the previous call to nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc).

On entry: contains information on the selected base generator and its current state.

On exit: contains updated information on the state of the generator.

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

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

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **df** = ⟨*value*⟩.
Constraint: **df** ≥ 3.

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

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

On entry, **lr** is not large enough, **lr** = ⟨*value*⟩: minimum length required = ⟨*value*⟩.

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

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_STATE

On entry, **state** vector has been corrupted or not initialized.

NE_PREV_CALL

ip or **iq** is not the same as when **r** was set up in a previous call.
Previous value of **ip** = ⟨*value*⟩ and **ip** = ⟨*value*⟩.
Previous value of **iq** = ⟨*value*⟩ and **iq** = ⟨*value*⟩.

NE_REAL_ARRAY

On entry, sum of **theta**$[i-1]$, for $i = 2, 3, \ldots, \mathbf{ip} + \mathbf{iq} + 1$ is ≥ 1.0: sum = ⟨*value*⟩.

On entry, **theta**[⟨*value*⟩] = ⟨*value*⟩.
Constraint: **theta**$[i-1] \geq 0.0$.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_rand_agarchI (g05pdc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

This example first calls nag_rand_init_repeatable (g05kfc) to initialize a base generator then calls nag_rand_agarchI (g05pdc) to generate two realizations, each consisting of ten observations, from a symmetric GARCH$(1,1)$ model.

10.1 Program Text

```
/* nag_rand_agarchI (g05pdc) Example Program.
 *
 * Copyright 2008, Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>

int main(void)
{
  /* Integer scalar and array declarations */
  Integer        exit_status = 0;
  Integer        lr, i, lstate;
  Integer        *state = 0;

  /* NAG structures */
  NagError       fail;
  Nag_Boolean    fcall;

  /* Double scalar and array declarations */
  double         *et = 0, *ht = 0, *r = 0;

  /* Number of terms to generate */
  Integer        num = 10;

  /* Normally distributed errors */
  Nag_ErrorDistn dist = Nag_NormalDistn;
  Integer        df = 0;

  /* Set up the parameters for the series being generated */
  Integer        ip = 0;
  Integer        iq = 3;
  double         theta[] = { 0.8e0, 0.6e0, 0.2e0, 0.1e0 };
  double         gamma = -0.4e0;

  /* Choose the base generator */
  Nag_BaseRNG    genid = Nag_Basic;
```

```
Integer        subid = 0;

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

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

printf("nag_rand_agarchI (g05pdc) Example Program Results\n\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;
  }

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

/* Allocate arrays */
if (!(et = NAG_ALLOC(num, double)) ||
    !(ht = NAG_ALLOC(num, double)) ||
    !(r = NAG_ALLOC(lr, double)) ||
    !(state = NAG_ALLOC(lstate, Integer)))
  {
    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;
  }

/* Generate the first realization */
fcall = Nag_TRUE;
nag_rand_agarchI(dist, num, ip, iq, theta, gamma, df, ht, et, 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;
  }

/* Display the results */
printf("  Realization Number 1\n");
printf("    I            HT(I)            ET(I)\n");
printf("   --------------------------------------\n");
for (i = 0; i < num; i++)
  printf(" %5ld %16.4f %16.4f\n", i+1, ht[i], et[i]);
printf("\n");

/* Generate a second realization */
fcall = Nag_FALSE;
nag_rand_agarchI(dist, num, ip, iq, theta, gamma, df, ht, et, 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;
    }

  /* Display the results */
  printf("  Realization Number 2\n");
  printf("    I           HT(I)           ET(I)\n");
  printf("   -------------------------------------\n");
  for (i = 0; i < num; i++)
    printf(" %5ld %16.4f %16.4f\n", i+1, ht[i], et[i]);

 END:
  NAG_FREE(et);
  NAG_FREE(ht);
  NAG_FREE(r);
  NAG_FREE(state);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_rand_agarchI (g05pdc) Example Program Results


  Realization Number 1
    I           HT(I)           ET(I)
   -------------------------------------
     1          0.9440           0.3389
     2          0.8502          -1.1484
     3          2.2553           0.9943
     4          1.4918           1.0204
     5          1.3413          -1.4544
     6          2.9757          -0.0326
     7          1.6386          -0.3767
     8          1.5433           0.9892
     9          1.1477          -0.0049
    10          1.0281           0.4508

  Realization Number 2
    I           HT(I)           ET(I)
   -------------------------------------
     1          0.8691          -1.5286
     2          3.0485          -1.1339
     3          2.9558           0.5424
     4          1.6547          -2.0734
     5          4.7100           0.5153
     6          2.0336          -0.8373
     7          2.3331          -1.0912
     8          2.4417           3.8999
     9          8.7473           3.8171
    10         10.4783           0.2480
```