

# NAG Library Function Document

## nag\_rand\_garchGJR (g05pfc)

### 1 Purpose

nag\_rand\_garchGJR (g05pfc) generates a given number of terms of a GJR GARCH( $p, q$ ) process (see Glosten *et al.* (1993)).

### 2 Specification

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

void nag_rand_garchGJR (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 GJR GARCH( $p, q$ ) process is represented by:

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

where  $I_t = 1$  if  $\epsilon_t < 0$ ,  $I_t = 0$  if  $\epsilon_t \geq 0$ , and  $\epsilon_t | \psi_{t-1} = N(0, h_t)$  or  $\epsilon_t | \psi_{t-1} = S_t(df, h_t)$ . 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,  $\epsilon_t$  is the observed value of the GARCH( $p, q$ ) process at time  $t$ ,  $h_t$  is the conditional variance at time  $t$ , and  $\psi_t$  the set of all information up to time  $t$ . Symmetric GARCH sequences are generated when  $\gamma$  is zero, otherwise asymmetric GARCH sequences are generated with  $\gamma$  specifying the amount by which 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\_garchGJR (g05pfc).

### 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

Glosten L, Jagannathan R and Runkle D (1993) Relationship between the expected value and the volatility of nominal excess return on stocks *Journal of Finance* **48** 1779–1801

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  $\epsilon_t$ .

**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,  $\beta_i$ , for  $i = 1, 2, \dots, p$ .  
*Constraint:* **ip** ≥ 0.
- 4: **iq** – Integer *Input*  
*On entry:* the number of coefficients,  $\alpha_i$ , for  $i = 1, 2, \dots, q$ .  
*Constraint:* **iq** ≥ 1.
- 5: **theta**[**iq** + **ip** + 1] – const double *Input*  
*On entry:* the first element must contain the coefficient  $\alpha_0$ , the next **iq** elements must contain the coefficients  $\alpha_i$ , for  $i = 1, 2, \dots, q$ . The remaining **ip** elements must contain the coefficients  $\beta_j$ , for  $j = 1, 2, \dots, p$ .  
*Constraints:*  

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

$$\mathbf{theta}[i-1] \geq 0.0, \text{ for } i = 1 \text{ and } i = \mathbf{iq} + 2, \dots, \mathbf{iq} + \mathbf{ip} + 1.$$
- 6: **gamma** – double *Input*  
*On entry:* the asymmetry parameter  $\gamma$  for the GARCH( $p, q$ ) sequence.  
*Constraint:* **gamma** + **theta**[ $i - 1$ ] ≥ 0.0, for  $i = 2, 3, \dots, \mathbf{iq} + 1$ .
- 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  $t = 1, 2, \dots, T$ , for the GARCH( $p, q$ ) sequence.
- 9: **et**[**num**] – double *Output*  
*On exit:* the observations  $\epsilon_t$ , for  $t = 1, 2, \dots, T$ , 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\_garchGJR (g05pfc), with **fcall** = Nag\_FALSE.

- 12: **lr** – Integer *Input*  
*On entry:* the dimension of the array **r**.  
*Constraint:*  $lr \geq 2 \times (ip + iq + 2)$ .
- 13: **state**[*dim*] – Integer *Communication Array*  
**Note:** the actual argument supplied must be the array **state** supplied to the initialization functions `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  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry, **df** =  $\langle value \rangle$ .

Constraint: **df**  $\geq 3$ .

On entry, **ip** =  $\langle value \rangle$ .

Constraint: **ip**  $\geq 0$ .

On entry, **iq** =  $\langle value \rangle$ .

Constraint: **iq**  $\geq 1$ .

On entry, **lr** is not large enough, **lr** =  $\langle value \rangle$ : minimum length required =  $\langle value \rangle$ .

On entry, **num** =  $\langle value \rangle$ .

Constraint: **num**  $\geq 0$ .

### 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** =  $\langle value \rangle$  and **ip** =  $\langle value \rangle$ .

Previous value of **iq** =  $\langle value \rangle$  and **iq** =  $\langle value \rangle$ .

### NE\_REAL\_2

On entry, **theta**[ $\langle value \rangle$ ] =  $\langle value \rangle$  and  $\gamma$  =  $\langle value \rangle$ .

Constraint:  $\alpha_i + \gamma \geq 0$ .

### NE\_REAL\_ARRAY

On entry, sum of **theta**[ $i$ ] =  $\langle value \rangle$ .

Constraint: sum of **theta**[ $i$ ], for  $i = 1, 2, \dots, ip + iq$  is  $< 1.0$ .

On entry, **theta**[*value*] = *value*.  
 Constraint: **theta**[*i*] ≥ 0.0.

## 7 Accuracy

Not applicable.

## 8 Further Comments

None.

## 9 Example

This example first calls `nag_rand_init_repeatable` (g05kfc) to initialize a base generator then calls `nag_rand_garchGJR` (g05pfc) to generate two realizations, each consisting of ten observations, from a GJR GARCH(1,1) model.

### 9.1 Program Text

```

/* nag_rand_garchGJR (g05pfc) 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 */
  Nag_Error    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 = 1;
  Integer      iq = 1;
  double       theta[] = { 0.4e0, 0.1e0, 0.7e0 };
  double       gamma = 0.1e0;

  /* 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_garchGJR (g05pfc) Example Program Results\n\n\n");

/* Get the length of the state array */
lstate = -1;
nag_rand_init_repeatabe(genid, subid, seed, lseed, state, &lstate, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_init_repeatabe (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_repeatabe(genid, subid, seed, lseed, state, &lstate, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_init_repeatabe (g05kfc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}

/* Generate the first realization */
fcall = Nag_TRUE;
nag_rand_garchGJR(dist, num, ip, iq, theta, gamma, df, ht, et, fcall, r,
                  lr, state, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_garchGJR (g05pfc).\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_garchGJR(dist, num, ip, iq, theta, gamma, df, ht, et,
                  fcall, r, lr, state, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_garchGJR (g05pfc).\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:
    if (et) NAG_FREE(et);
    if (ht) NAG_FREE(ht);
    if (r) NAG_FREE(r);
    if (state) NAG_FREE(state);

    return exit_status;
}

```

## 9.2 Program Data

None.

## 9.3 Program Results

nag\_rand\_garchGJR (g05pfc) Example Program Results

Realization Number 1		
I	HT(I)	ET(I)
-----		
1	1.8000	0.4679
2	1.6819	-1.6152
3	2.0991	0.9592
4	1.9614	1.1701
5	1.9099	-1.7355
6	2.3393	-0.0289
7	2.0377	-0.4201
8	1.8617	1.0865
9	1.8212	-0.0061
10	1.6749	0.5754

Realization Number 2		
I	HT(I)	ET(I)
-----		
1	1.6055	-2.0776
2	2.3872	-1.0034
3	2.2724	0.4756
4	2.0133	-2.2871
5	2.8554	0.4012
6	2.4149	-0.9125
7	2.2570	-1.0732
8	2.2102	3.7105
9	3.3239	2.3530
10	3.2804	0.1388

---