

## NAG Library Function Document

### nag\_rand\_egarch (g05pgc)

#### 1 Purpose

nag\_rand\_egarch (g05pgc) generates a given number of terms of an exponential GARCH( $p, q$ ) process (see Engle and Ng (1993)).

#### 2 Specification

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

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

#### 3 Description

An exponential GARCH( $p, q$ ) process is represented by:

$$\ln(h_t) = \alpha_0 + \sum_{i=1}^q \alpha_i z_{t-i} + \sum_{i=1}^q \phi_i (|z_{t-i}| - E[|z_{t-i}|]) + \sum_{j=1}^p \beta_j \ln(h_{t-j}), \quad t = 1, 2, \dots, T;$$

where  $z_t = \frac{\epsilon_t}{\sqrt{h_t}}$ ,  $E[|z_{t-i}|]$  denotes the expected value of  $|z_{t-i}|$ , 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  $t$ .

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\_egarch (g05pgc).

#### 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**  $\geq 0$ .
- 3: **ip** – Integer *Input*  
*On entry:* the number of coefficients,  $\beta_i$ , for  $i = 1, 2, \dots, p$ .  
*Constraint:* **ip**  $\geq 0$ .
- 4: **iq** – Integer *Input*  
*On entry:* the number of coefficients,  $\alpha_i$ , for  $i = 1, 2, \dots, q$ .  
*Constraint:* **iq**  $\geq 1$ .
- 5: **theta**[ $2 \times \mathbf{iq} + \mathbf{ip} + 1$ ] – const double *Input*  
*On entry:* the initial parameter estimates for the vector  $\theta$ . The first element must contain the coefficient  $\alpha_0$  and the next **iq** elements must contain the autoregressive coefficients  $\alpha_i$ , for  $i = 1, 2, \dots, q$ . The next **iq** elements must contain the coefficients  $\phi_i$ , for  $i = 1, 2, \dots, q$ . The next **ip** elements must contain the moving average coefficients  $\beta_j$ , for  $j = 1, 2, \dots, p$ .  
*Constraints:*
- $$\sum_{i=1}^p \beta_i \neq 1.0;$$
- $$\frac{\alpha_0}{1 - \sum_{i=1}^p \beta_i} \leq -\log(\text{nag\_real\_safe\_small\_number}).$$
- 6: **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**  $> 2$ .
- 7: **ht**[**num**] – double *Output*  
*On exit:* the conditional variances  $h_t$ , for  $t = 1, 2, \dots, T$ , for the GARCH( $p, q$ ) sequence.
- 8: **et**[**num**] – double *Output*  
*On exit:* the observations  $\epsilon_t$ , for  $t = 1, 2, \dots, T$ , for the GARCH( $p, q$ ) sequence.
- 9: **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**.

- 10: **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\_egarch (g05pgc), with **fcall** = Nag\_FALSE.
- 11: **lr** – Integer *Input*  
*On entry:* the dimension of the array **r**.  
*Constraint:*  $lr \geq 2 \times (ip + 2 \times iq + 2)$ .
- 12: **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\_repeatabl (g05kfc) or nag\_rand\_init\_nonrepeatabl (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.
- 13: **fail** – NagError \* *Input/Output*  
 The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

On entry, argument *<value>* had an illegal value.

### NE\_INT

On entry, **df** = *<value>*.

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

On entry, **ip** = *<value>*.

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

On entry, **iq** = *<value>*.

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

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

On entry, **num** = *<value>*.

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.

An unexpected error has been triggered by this function. Please contact NAG.

See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

**NE\_INVALID\_STATE**

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

**NE\_NO\_LICENCE**

Your licence key may have expired or may not have been installed correctly.  
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

**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\_ARRAY**

Invalid sequence generated, use different parameters.

**7 Accuracy**

Not applicable.

**8 Parallelism and Performance**

nag\_rand\_egarch (g05pgc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also 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\_egarch (g05pgc) to generate two realizations, each consisting of ten observations, from an exponential GARCH(1,1) model.

**10.1 Program Text**

```

/* nag_rand_egarch (g05pgc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* 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 = 1;
Integer iq = 1;
double theta[] = { 0.1e0, -0.3e0, 0.1e0, 0.9e0 };

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

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

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

printf("nag_rand_egarch (g05pgc) 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 * (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;
}

/* Initialize 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_egarch(dist, num, ip, iq, theta, df, ht, et, fcall, r, lr,
    state, &fail);
if (fail.code != NE_NOERROR) {

```

```

    printf("Error from nag_rand_egarch (g05pgc).\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(" %5" NAG_IFMT " %16.4f %16.4f\n", i + 1, ht[i], et[i]);
printf("\n");

/* Generate a second realization */
fcall = Nag_FALSE;
nag_rand_egarch(dist, num, ip, iq, theta, df, ht, et, fcall, r, lr,
                state, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_egarch (g05pgc).\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(" %5" NAG_IFMT " %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\_egarch (g05pgc) Example Program Results

Realization Number 1		
I	HT(I)	ET(I)
-----		
1	2.5098	0.5526
2	2.1785	-1.8383
3	3.3844	1.2180
4	2.6780	1.3672
5	2.0953	-1.8178
6	3.2813	-0.0343
7	2.9958	-0.5094
8	3.0815	1.3978
9	2.3961	-0.0070
10	2.2445	0.6661

  

Realization Number 2		
I	HT(I)	ET(I)
-----		
1	1.9327	-2.2795
2	3.5577	-1.2249
3	4.1461	0.6424
4	3.4455	-2.9920
5	5.9199	0.5777

6	4.8221	-1.2894
7	5.3174	-1.6473
8	6.1095	6.1689
9	3.1579	2.2935
10	2.2189	0.1141

---