NAG Library Function Document

nag_rand_agarchII (g05pec)

1 Purpose
nag_rand_agarchII (g05pec) generates a given number of terms of a type II AGARCH\((p,q)\) process (see Engle and Ng (1993)).

2 Specification

```c
#include <nag.h>
#include <nag05.h>

void nag_rand_agarchII (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 II AGARCH\((p,q)\) process can be represented by:

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

where \(\epsilon_t | \psi_{t-1} = N(0, h_t)\) 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 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_agarchII (g05pec).

4 References

5 Arguments
1: dist – Nag_ErrorDistn
   ```c
   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 $i = 1, 2, \ldots, p$.

*Constraint:* $ip \geq 0$.

4: iq – Integer

*Input*

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

*Constraint:* iq $\geq 1$.

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 $i = 1, 2, \ldots, q$. The remaining ip elements must contain the coefficients $\beta_j$, for $j = 1, 2, \ldots, p$.

*Constraints:*

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

\[
\theta[i-1] \geq 0.0, \text{ for } i = 2, 3, \ldots, ip + iq + 1.
\]

6: gamma – double

*Input*

*On entry:* the asymmetry parameter $\gamma$ 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: h[t] – double

*Output*

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

9: et[t] – double

*Output*

*On exit:* the observations $\epsilon_t$, 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_agarchII (g05pec), with fcall = Nag_FALSE.
**g05pec**

12:  **lr** – Integer  

*Input*

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

*Constraint:* \( \text{lr} \geq 2 \times (\text{ip} + \text{iq} + 2) \).

13:  **state[dim]** – Integer  

*Communication Array*

*Note:* the dimension, \( \text{dim} \), 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_ALLOC_FAIL**

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

**NE_BAD_PARAM**

On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

**NE_INT**

On entry, \( \text{df} = \langle \text{value} \rangle \).

*Constraint:* \( \text{df} \geq 3 \).

On entry, \( \text{ip} = \langle \text{value} \rangle \).

*Constraint:* \( \text{ip} \geq 0 \).

On entry, \( \text{iq} = \langle \text{value} \rangle \).

*Constraint:* \( \text{iq} \geq 1 \).

On entry, \( \text{lr} \) is not large enough, \( \text{lr} = \langle \text{value} \rangle \): minimum length required = \( \langle \text{value} \rangle \).

On entry, \( \text{num} = \langle \text{value} \rangle \).

*Constraint:* \( \text{num} \geq 0 \).

**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 3.6.6 in the Essential Introduction 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 3.6.5 in the Essential Introduction 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 = (value) and ip = (value).
Previous value of iq = (value) and iq = (value).

NE_REAL_ARRAY

On entry, sum of \( \theta[i - 1] \), for \( i = 2, 3, \ldots, ip + iq + 1 \) is \( \geq 1.0: \) sum = (value).
On entry, \( \theta[value] \) = (value).
Constraint: \( \theta[i - 1] \geq 0.0.\)

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_rand_agarchII (g05pec) 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_agarchII (g05pec) to generate two realizations, each consisting of ten observations, from an asymmetric GARCH(1,1) model.

10.1 Program Text

```c
/* nag_rand_agarchII (g05pec) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 9, 2009. */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nagg05.h>
#include <nag_stdlib.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[3] = { 0.08e0, 0.2e0, 0.7e0 };
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_agarchII (g05pec) 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_agarchII(dist, num, ip, iq, theta, gamma, df, ht, et, fcall, r,
                   lr, state, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_agarchII (g05pec).\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;

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_agarchII (g05pec) Example Program Results

Realization Number 1

<table>
<thead>
<tr>
<th>I</th>
<th>HT(I)</th>
<th>ET(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6400</td>
<td>0.2790</td>
</tr>
<tr>
<td>2</td>
<td>0.5336</td>
<td>-0.9098</td>
</tr>
<tr>
<td>3</td>
<td>0.7780</td>
<td>0.5840</td>
</tr>
<tr>
<td>4</td>
<td>0.6491</td>
<td>0.6731</td>
</tr>
<tr>
<td>5</td>
<td>0.5670</td>
<td>-0.9456</td>
</tr>
<tr>
<td>6</td>
<td>0.8275</td>
<td>-0.0172</td>
</tr>
<tr>
<td>7</td>
<td>0.6593</td>
<td>-0.2390</td>
</tr>
<tr>
<td>8</td>
<td>0.5639</td>
<td>0.5980</td>
</tr>
<tr>
<td>9</td>
<td>0.5005</td>
<td>-0.0032</td>
</tr>
<tr>
<td>10</td>
<td>0.4303</td>
<td>0.2917</td>
</tr>
</tbody>
</table>

Realization Number 2

<table>
<thead>
<tr>
<th>I</th>
<th>HT(I)</th>
<th>ET(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3874</td>
<td>-1.0205</td>
</tr>
<tr>
<td>2</td>
<td>0.7594</td>
<td>-0.5659</td>
</tr>
<tr>
<td>3</td>
<td>0.7371</td>
<td>0.2709</td>
</tr>
<tr>
<td>4</td>
<td>0.6013</td>
<td>-1.2499</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>5</td>
<td>1.1133</td>
<td>0.2505</td>
</tr>
<tr>
<td>6</td>
<td>0.8638</td>
<td>-0.5457</td>
</tr>
<tr>
<td>7</td>
<td>0.8014</td>
<td>-0.6395</td>
</tr>
<tr>
<td>8</td>
<td>0.8013</td>
<td>2.2341</td>
</tr>
<tr>
<td>9</td>
<td>1.0003</td>
<td>1.2908</td>
</tr>
<tr>
<td>10</td>
<td>0.9002</td>
<td>0.0727</td>
</tr>
</tbody>
</table>