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

```c
#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:

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

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


5 Arguments

1: dist – Nag_ErrorDistn  
   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  
   On entry: $T$, the number of terms in the sequence.
   
   Constraint: num > 0.

3: ip – Integer  
   On entry: the number of coefficients, $\beta_i$, for $i = 1, 2, \ldots, p$.
   
   Constraint: ip $\geq$ 0.

4: iq – Integer  
   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  
   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:
   
   \[
   \sum_{i=2}^{iq+ip+1} \text{theta}[i-1] < 1.0;
   \text{theta}[i-1] \geq 0.0, \text{ for } i = 1 \text{ and } i = iq+2, \ldots, iq+ip+1.
   \]

6: gamma – double  
   On entry: the asymmetry parameter $\gamma$ for the GARCH($p, q$) sequence.
   
   Constraint: gamma + theta[i - 1] $\geq$ 0.0, for $i = 2, 3, \ldots, iq + 1$.

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

9: et[num] – double  
   On exit: the observations $\epsilon_t$, for $t = 1, 2, \ldots, T$, for the GARCH($p, q$) sequence.
**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*.

**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.

**lr** – Integer

*Input*

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

Constraint: \( lr \geq 2 \times (ip + iq + 2) \).

**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.

**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 〈*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 3.6.6 in the Essential Introduction for further information.
On entry, \texttt{state} vector has been corrupted or not initialized.

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.

\texttt{ip} or \texttt{iq} is not the same as when \texttt{r} was set up in a previous call.
Previos value of \texttt{ip} = \langle value \rangle and \texttt{ip} = \langle value \rangle.
Previous value of \texttt{iq} = \langle value \rangle and \texttt{iq} = \langle value \rangle.

On entry, \texttt{theta} = \langle value \rangle and \gamma = \langle value \rangle.
Constraint: \( \alpha_i + \gamma \geq 0 \).

On entry, sum of \texttt{theta}[i] = \langle value \rangle.
Constraint: sum of \texttt{theta}[i], for \( i = 1, 2, \ldots, \texttt{ip} + \texttt{iq} \) is < 1.0.
On entry, \texttt{theta} = \langle value \rangle.
Constraint: \texttt{theta}[i] \geq 0.0.

7 Accuracy
Not applicable.

8 Parallelism and Performance
\texttt{nag_rand_garchGJR (g05pfc)} 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 \texttt{nag_rand_init_repeateable (g05kfc)} to initialize a base generator then calls \texttt{nag_rand_garchGJR (g05pfc)} to generate two realizations, each consisting of ten observations, from a GJR GARCH\((1,1)\) model.

10.1 Program Text
/* nag_rand_garchGJR (g05pfc) Example Program. */
/* Copyright 2014 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 = 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_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_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(" %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_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(" %5"NAG_IFMT" %16.4f %16.4f\n", i+1, ht[i], et[i];
  printf("\n");

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_garchGJR (g05pfc) Example Program Results

<table>
<thead>
<tr>
<th>Realization Number 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I  HT(I)  ET(I)</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
</tr>
<tr>
<td>1  1.8000  0.4679</td>
<td></td>
</tr>
<tr>
<td>2  1.6819  -1.6152</td>
<td></td>
</tr>
<tr>
<td>3  2.0991  0.9592</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>HT(I)</td>
</tr>
<tr>
<td>---</td>
<td>---------</td>
</tr>
<tr>
<td>4</td>
<td>1.9614</td>
</tr>
<tr>
<td>5</td>
<td>1.9099</td>
</tr>
<tr>
<td>6</td>
<td>2.3393</td>
</tr>
<tr>
<td>7</td>
<td>2.0377</td>
</tr>
<tr>
<td>8</td>
<td>1.8617</td>
</tr>
<tr>
<td>9</td>
<td>1.8212</td>
</tr>
<tr>
<td>10</td>
<td>1.6749</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>1.6055</td>
<td>-2.0776</td>
</tr>
<tr>
<td>2</td>
<td>2.3872</td>
<td>-1.0034</td>
</tr>
<tr>
<td>3</td>
<td>2.2724</td>
<td>0.4756</td>
</tr>
<tr>
<td>4</td>
<td>2.0133</td>
<td>-2.2871</td>
</tr>
<tr>
<td>5</td>
<td>2.8554</td>
<td>0.4012</td>
</tr>
<tr>
<td>6</td>
<td>2.4149</td>
<td>-0.9125</td>
</tr>
<tr>
<td>7</td>
<td>2.2570</td>
<td>-1.0732</td>
</tr>
<tr>
<td>8</td>
<td>2.2102</td>
<td>3.7105</td>
</tr>
<tr>
<td>9</td>
<td>3.3239</td>
<td>2.3530</td>
</tr>
<tr>
<td>10</td>
<td>3.2804</td>
<td>0.1388</td>
</tr>
</tbody>
</table>