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

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

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

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_agarchII (g05pec).

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:

$$\sum_{i=2}^{\mathbf{iq}+\mathbf{ip}+1} \mathbf{theta}[i-1] < 1.0;$$
$$\mathbf{theta}[i-1] \geq 0.0, \text{ for } i = 2, 3, \ldots, \mathbf{ip} + \mathbf{iq} + 1.$$

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_agarchII (g05pec), 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 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, $\mathbf{df} = \langle value \rangle$.
Constraint: $\mathbf{df} \geq 3$.

On entry, $\mathbf{ip} = \langle value \rangle$.
Constraint: $\mathbf{ip} \geq 0$.

On entry, $\mathbf{iq} = \langle value \rangle$.
Constraint: $\mathbf{iq} \geq 1$.

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

On entry, $\mathbf{num} = \langle value \rangle$.
Constraint: $\mathbf{num} \geq 0$.

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** = ⟨*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_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

```
/* nag_rand_agarchII (g05pec) 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[] = { 0.08e0, 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;

  /* Initialize 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;
  }

  /* Initialize 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;
  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 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_agarchII (g05pec) Example Program Results


  Realization Number 1
    I            HT(I)            ET(I)
   ------------------------------------
      1          0.6400           0.2790
      2          0.5336          -0.9098
      3          0.7780           0.5840
      4          0.6491           0.6731
      5          0.5670          -0.9456
      6          0.8275          -0.0172
      7          0.6593          -0.2390
      8          0.5639           0.5980
      9          0.5005          -0.0032
     10          0.4303           0.2917

  Realization Number 2
    I            HT(I)            ET(I)
   ------------------------------------
      1          0.3874          -1.0205
      2          0.7594          -0.5659
      3          0.7371           0.2709
      4          0.6013          -1.2499
      5          1.1133           0.2505
```

```
     6          0.8638          -0.5457
     7          0.8014          -0.6395
     8          0.8013           2.2341
     9          1.0003           1.2908
    10          0.9002           0.0727
```