NAG Library Function Document

nag_rand_arma (g05phc)

1 Purpose

nag_rand_arma (g05phc) generates a realization of a univariate time series from an autoregressive moving average (ARMA) model. The realization may be continued or a new realization generated at subsequent calls to nag_rand_arma (g05phc).

2 Specification

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

```
void nag_rand_arma (Nag_ModeRNG mode, Integer n, double xmean, Integer ip,
     const double phi[], Integer iq, const double theta[], double avar,
     double r[], Integer lr, Integer state[], double *var, double x[],
     NagError *fail)
```

3 Description

Let the vector x_t, denote a time series which is assumed to follow an autoregressive moving average (ARMA) model of the form:

$$\begin{aligned} x_t - \mu = \quad & \phi_1(x_{t-1} - \mu) + \phi_2(x_{t-2} - \mu) + \cdots + \phi_p(x_{t-p} - \mu) + \\ & \epsilon_t - \theta_1\epsilon_{t-1} - \theta_2\epsilon_{t-2} - \cdots - \theta_q\epsilon_{t-q} \end{aligned}$$

where ϵ_t, is a residual series of independent random perturbations assumed to be Normally distributed with zero mean and variance σ^2. The parameters $\{\phi_i\}$, for $i = 1, 2, \ldots, p$, are called the autoregressive (AR) parameters, and $\{\theta_j\}$, for $j = 1, 2, \ldots, q$, the moving average (MA) parameters. The parameters in the model are thus the p ϕ values, the q θ values, the mean μ and the residual variance σ^2.

nag_rand_arma (g05phc) sets up a reference vector containing initial values corresponding to a stationary position using the method described in Tunnicliffe–Wilson (1979). The function can then return a realization of $x_1, x_2, \ldots, x_n$. On a successful exit, the recent history is updated and saved in the reference vector **r** so that nag_rand_arma (g05phc) may be called again to generate a realization of $x_{n+1}, x_{n+2}, \ldots$, etc. See the description of the argument **mode** in Section 5 for details.

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_arma (g05phc).

4 References

Knuth D E (1981) *The Art of Computer Programming (Volume 2)* (2nd Edition) Addison–Wesley

Tunnicliffe–Wilson G (1979) Some efficient computational procedures for high order ARMA models *J. Statist. Comput. Simulation* **8** 301–309

5 Arguments

1: **mode** – Nag_ModeRNG *Input*

On entry: a code for selecting the operation to be performed by the function.

mode = Nag_InitializeReference
Set up reference vector only.

mode = Nag_GenerateFromReference
Generate terms in the time series using reference vector set up in a prior call to nag_rand_arma (g05phc).

mode = Nag_InitializeAndGenerate
Set up reference vector and generate terms in the time series.

Constraint: **mode** = Nag_InitializeReference, Nag_GenerateFromReference or Nag_InitializeAndGenerate.

2: **n** – Integer *Input*

On entry: n, the number of observations to be generated.

Constraint: $\mathbf{n} \geq 0$.

3: **xmean** – double *Input*

On entry: the mean of the time series.

4: **ip** – Integer *Input*

On entry: p, the number of autoregressive coefficients supplied.

Constraint: $\mathbf{ip} \geq 0$.

5: **phi**[**ip**] – const double *Input*

On entry: the autoregressive coefficients of the model, $\phi_1, \phi_2, \ldots, \phi_p$.

6: **iq** – Integer *Input*

On entry: q, the number of moving average coefficients supplied.

Constraint: $\mathbf{iq} \geq 0$.

7: **theta**[**iq**] – const double *Input*

On entry: the moving average coefficients of the model, $\theta_1, \theta_2, \ldots, \theta_q$.

8: **avar** – double *Input*

On entry: σ^2, the variance of the Normal perturbations.

Constraint: $\mathbf{avar} \geq 0.0$.

9: **r**[**lr**] – double *Communication Array*

On entry: if **mode** = Nag_GenerateFromReference, the reference vector from the previous call to nag_rand_arma (g05phc).

On exit: the reference vector.

10: **lr** – Integer *Input*

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

Constraint: $\mathbf{lr} \geq \mathbf{ip} + \mathbf{iq} + 6 + \max(\mathbf{ip}, \mathbf{iq} + 1)$.

11: **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.

12: **var** – double * *Output*

On exit: the proportion of the variance of a term in the series that is due to the moving-average (error) terms in the model. The smaller this is, the nearer is the model to non-stationarity.

13: **x**[**n**] – double *Output*

On exit: contains the next n observations from the time series.

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 $\langle value \rangle$ had an illegal value.

NE_INT

On entry, **ip** $= \langle value \rangle$.
Constraint: **ip** ≥ 0.

On entry, **iq** $= \langle value \rangle$.
Constraint: **iq** ≥ 0.

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

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 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** $= \langle value \rangle$ and **ip** $= \langle value \rangle$.
Previous value of **iq** $= \langle value \rangle$ and **iq** $= \langle value \rangle$.

NE_REAL

On entry, **avar** $= \langle value \rangle$.
Constraint: **avar** ≥ 0.0.

NE_REF_VEC

Reference vector **r** has been corrupted or not initialized correctly.

NE_STATIONARY_AR

On entry, the AR parameters are outside the stationarity region.

7 Accuracy

Any errors in the reference vector's initial values should be very much smaller than the error term; see Tunnicliffe–Wilson (1979).

8 Parallelism and Performance

nag_rand_arma (g05phc) 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

The time taken by nag_rand_arma (g05phc) is essentially of order $(\mathbf{ip})^2$.

Note: The reference vector, **r**, contains a copy of the recent history of the series. If attempting to re-initialize the series by calling nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc) a call to nag_rand_arma (g05phc) with **mode** $=$ Nag_InitializeReference must also be made. In the repeatable case the calls to nag_rand_arma (g05phc) should be performed in the same order (at the same point(s) in simulation) every time nag_rand_init_repeatable (g05kfc) is used. When the generator state is saved and restored using the argument **state**, the time series reference vector must be saved and restored as well.

The ARMA model for a time series can also be written as:

$$(x_n - E) = A_1(x_{n-1} - E) + \cdots + A_{NA}(x_{n-NA} - E) + B_1 a_n + \cdots + B_{NB} a_{n-NB+1}$$

where

x_n is the observed value of the time series at time n,

NA is the number of autoregressive parameters, A_i,

NB is the number of moving average parameters, B_i,

E is the mean of the time series,

and

a_t is a series of independent random Standard Normal perturbations.

This is related to the form given in Section 3 by:

$B_1^2 = \sigma^2$,

$B_{i+1} = -\theta_i \sigma = -\theta_i B_1, \quad i = 1, 2, \ldots, q$,

$NB = q + 1$,

$E = \mu$,

$A_i = \phi_i, \quad i = 1, 2, \ldots, p$,

$NA = p$.

10 Example

This example generates values for an autoregressive model given by

$$x_t = 0.4x_{t-1} + 0.2x_{t-2} + \epsilon_t$$

where ϵ_t is a series of independent random Normal perturbations with variance 1.0. The random number generators are initialized by nag_rand_init_repeatable (g05kfc) and then nag_rand_arma (g05phc) is called to initialize a reference vector and generate a sample of ten observations.

10.1 Program Text

```
/* nag_rand_arma (g05phc) 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_ModeRNG mode;

  /* Double scalar and array declarations */
  double      var;
  double      *r = 0, *x = 0;

  /* Set the number of observations to generate */
  Integer     n = 10;

  /* Set up the parameters for the series being generated */
  Integer     ip = 2;
  Integer     iq = 0;
  double      phi[] = { 0.4e0, 0.2e0 };
  double      xmean = 0.0e0;
  double      avar = 1.0e0;
  /* Need a dummy, non-null theta, even if we are not using it */
  double      theta[] = { 0.0e0 };

  /* 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_arma (g05phc) Example Program Results\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 = (ip > iq + 1)?ip:iq + 1;
  lr += ip+iq+6;

  /* Allocate arrays */
  if (!(r = NAG_ALLOC(lr, double)) ||
      !(x = NAG_ALLOC(n, 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;
    }

  /* Set up the reference vector and generate the N realizations */
  mode = Nag_InitializeAndGenerate;
  nag_rand_arma(mode, n, xmean, ip, phi, iq, theta, avar, r, lr, state,
                &var, x, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_arma (g05phc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Display the variates */
  for (i = 0; i < n; i++)
    printf(" %12.4f\n", x[i]);

 END:
  NAG_FREE(r);
  NAG_FREE(x);
  NAG_FREE(state);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_rand_arma (g05phc) Example Program Results

      -1.7103
      -0.4042
      -0.1845
      -1.5004
      -1.1946
```

```
 -1.8184
 -1.0895
  1.6408
  1.3555
  1.1908
```