NAG C Library Function Document

nag_rngs_varma_time_series (g05pcc)

1 Purpose

nag_rngs_varma_time_series (g05pcc) generates a realisation of a multivariate time series from a vector autoregressive moving average (VARMA) model. The realisation may be continued or a new realisation generated at subsequent calls to this function.

2 Specification

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

void nag_rngs_varma_time_series (Nag_OrderType order, Integer mode, Integer k,
     const double xmean[], Integer p, const double phi[], Integer q,
     const double theta[], const double var[], Integer pdv, Integer n, double x[],
     Integer pdx, Integer igen, Integer iseed[], double r[], NagError *fail)
```

3 Description

Let the vector $X_t = (x_{1t}, x_{2t}, \ldots, x_{kt})^{\mathrm{T}}$, denote a k dimensional time series which is assumed to follow a vector autoregressive moving average (VARMA) model of the form:

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

where $\epsilon_t = (\epsilon_{1t}, \epsilon_{2t}, \ldots, \epsilon_{kt})^{\mathrm{T}}$, is a vector of k residual series assumed to be Normally distributed with zero mean and positive-definite covariance matrix Σ. The components of ϵ_t are assumed to be uncorrelated at non-simultaneous lags. The ϕ_i's and θ_j's are k by k matrices of arguments. $\{\phi_i\}$, for $i = 1, 2, \ldots, p$, are called the autoregressive (AR) argument matrices, and $\{\theta_j\}$, for $j = 1, 2, \ldots, q$, the moving average (MA) argument matrices. The arguments in the model are thus the p k by k ϕ-matrices, the q k by k θ-matrices, the mean vector μ and the residual error covariance matrix Σ. Let

$$A(\phi) = \begin{bmatrix} \phi_1 & I & 0 & . & . & . & 0 \\ \phi_2 & 0 & I & 0 & . & . & 0 \\ . & & & . & & & \\ . & & & & . & & \\ . & & & & & . & \\ \phi_{p-1} & 0 & . & . & . & 0 & I \\ \phi_p & 0 & . & . & . & 0 & 0 \end{bmatrix}_{pk \times pk} \quad \text{and} \quad B(\theta) = \begin{bmatrix} \theta_1 & I & 0 & . & . & . & 0 \\ \theta_2 & 0 & I & 0 & . & . & 0 \\ . & & & . & & & \\ . & & & & . & & \\ . & & & & & . & \\ \theta_{q-1} & 0 & . & . & . & 0 & I \\ \theta_q & 0 & . & . & . & 0 & 0 \end{bmatrix}_{qk \times qk}$$

where I denotes the k by k identity matrix.

The model (1) must be both stationary and invertible. The model is said to be stationary if the eigenvalues of $A(\phi)$ lie inside the unit circle and invertible if the eigenvalues of $B(\theta)$ lie inside the unit circle.

For $k \geq 6$ the VARMA model (1) is recast into state space form and a realisation of the state vector at time zero computed. For all other cases the function computes a realisation of the pre-observed vectors $X_0, X_{-1}, \ldots, X_{1-p}$, $\epsilon_0, \epsilon_{-1}, \ldots, \epsilon_{1-q}$, from equation (1), see Shea (1988). This realisation is then used to generate a sequence of successive time series observations. Note that special action is taken for pure MA models, that is for $p = 0$.

At your request a new realisation of the time series may be generated with less computation using only the information saved in a reference vector from a previous call to nag_rngs_varma_time_series (g05pcc). See the description of the argument **mode** in Section 5 for details.

The function returns a realisation of $X_1, X_2, \ldots, X_n$. On a successful exit, the recent history is updated and saved in the array **r** so that nag_rngs_varma_time_series (g05pcc) may be called again to generate a realisation of $X_{n+1}, X_{n+2}, \ldots$,etc. See the description of the argument **mode** in Section 5 for details.

Further computational details are given in Shea (1988). Note however that this function uses a spectral decomposition rather than a Cholesky factorization to generate the multivariate Normals. Although this method involves more multiplications than the Cholesky factorization method and is thus slightly slower it is more stable when faced with ill-conditioned covariance matrices. A method of assigning the AR and MA coefficient matrices so that the stationarity and invertibility conditions are satisfied is described in Barone (1987).

One of the initialization functions nag_rngs_init_repeatable (g05kbc) (for a repeatable sequence if computed sequentially) or nag_rngs_init_nonrepeatable (g05kcc) (for a non-repeatable sequence) must be called prior to the first call to nag_rngs_varma_time_series (g05pcc).

4 References

Barone P (1987) A method for generating independent realisations of a multivariate normal stationary and invertible ARMA(p, q) process *J. Time Ser. Anal.* **8** 125–130

Shea B L (1988) A note on the generation of independent realisations of a vector autoregressive moving average process *J. Time Ser. Anal.* **9** 403–410

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = **Nag_RowMajor**. See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: **order** = **Nag_RowMajor** or **Nag_ColMajor**.

2: **mode** – Integer *Input*

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

mode = 0 (start)

Set up reference vector and compute a realisation of the recent history.

mode = 1 (continue)

Generate terms in the time series using reference vector set up in a prior call to nag_rngs_varma_time_series (g05pcc).

mode = 2 (start and generate)

Combine the operations of **mode** = 0 and **mode** = 1.

mode = 3 (restart and generate)

A new realisation of the recent history is computed using information stored in the reference vector, and the following sequence of time series values are generated.

If **mode** = 1 or 3, then you must ensure that the reference vector **r** and the values of **k**, **p**, **q**, **xmean**, **phi**, **theta**, **var** and **pdv** have not been changed between calls to nag_rngs_varma_time_series (g05pcc).

Constraint: $0 \leq$ **mode** ≤ 3.

3: **k** – Integer *Input*

On entry: k, the dimension of the multivariate time series.

Constraint: **k** ≥ 1.

4: **xmean**[**k**] – const double *Input*

On entry: μ, the vector of means of the multivariate time series.

5: **p** – Integer *Input*

On entry: p, the number of autoregressive argument matrices.

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

6: **phi**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **phi** must be at least $\max(1, \mathbf{p} \times \mathbf{k} \times \mathbf{k})$.

On entry: contains the elements of the $\mathbf{pk} \times \mathbf{k}$ autoregressive argument matrices of the model, $\phi_1, \phi_2, \ldots, \phi_p$. The (i,j)th element of ϕ_l is stored in **phi**$[(l-1) \times k \times k + (j-1) \times k + i - 1]$, for $l = 1, 2, \ldots, p$; $i,j = 1, 2, \ldots, k$.

Constraint: the first $\mathbf{p} \times \mathbf{k} \times \mathbf{k}$ elements of **phi** must satisfy the stationarity condition.

7: **q** – Integer *Input*

On entry: q, the number of moving average argument matrices.

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

8: **theta**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **theta** must be at least $\max(1, \mathbf{q} \times \mathbf{k} \times \mathbf{k})$.

On entry: contains the elements of the $\mathbf{qk} \times \mathbf{k}$ moving average argument matrices of the model, $\theta_1, \theta_2, \ldots, \theta_q$. The (i,j)th element of θ_l is stored in **theta**$[(l-1) \times k \times k + (j-1) \times k + i - 1]$, for $l = 1, 2, \ldots, q$; $i,j = 1, 2, \ldots, k$.

9: **var**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **var** must be at least $\mathbf{pdv} \times \mathbf{k}$.

Where **VAR**(i,j) appears in this document, it refers to the array element

if **order** = **Nag_ColMajor**, **var**$[(j-1) \times \mathbf{pdv} + i - 1]$;
if **order** = **Nag_RowMajor**, **var**$[(i-1) \times \mathbf{pdv} + j - 1]$.

On entry: **VAR**(i,j) must contain the (i,j)th element of Σ. Only the lower triangle is required.

Constraint: the elements of **var** must be such that Σ is positive-definite.

10: **pdv** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in the array **var**.

Constraint: $\mathbf{pdv} \geq \mathbf{k}$.

11: **n** – Integer *Input*

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

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

12: **x**[dim] – double *Output*

Note: the dimension, *dim*, of the array **x** must be at least

$\max(1, \mathbf{pdx} \times \mathbf{n})$ when **order** = **Nag_ColMajor**;
$\max(1, \mathbf{pdx} \times \mathbf{k})$ when **order** = **Nag_RowMajor**.

Where $\mathbf{X}(i,j)$ appears in this document, it refers to the array element

if **order** = **Nag_ColMajor**, $\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$;
if **order** = **Nag_RowMajor**, $\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$.

On exit: $\mathbf{X}(i,t)$ will contain a realisation of the ith component of $\mathbf{x}_t$, for $i = 1, 2, \ldots, k$; $t = 1, 2, \ldots, n$.

13: **pdx** – Integer *Input*

On entry: the stride separating matrix row or column elements (depending on the value of **order**) in the array **x**.

Constraints:

if **order** = **Nag_ColMajor**, $\mathbf{pdx} \geq \mathbf{k}$;
if **order** = **Nag_RowMajor**, $\mathbf{pdx} \geq \max(1, \mathbf{n})$.

14: **igen** – Integer *Input*

On entry: must contain the identification number for the generator to be used to return a pseudo-random number and should remain unchanged following initialization by a prior call to one of the functions nag_rngs_init_repeatable (g05kbc) or nag_rngs_init_nonrepeatable (g05kcc).

15: **iseed**[**4**] – Integer *Input/Output*

On entry: contains values which define the current state of the selected generator.

On exit: contains updated values defining the new state of the selected generator.

16: **r**[] – double *Input/Output*

On entry: if $\mathbf{mode} = 1$, then the array **r** as output from the previous call to nag_rngs_varma_time_series (g05pcc) must be input without any change to the first $m + (k+1)(k+2) + (m+1)(m+2)$ elements where $m = k \times \max(p,q)$ if $k \geq 6$ and $k(p+q)$ if $k < 6$.

If **mode** = 0 or 2, then the contents of **r** need not be set.

On exit: the first $m + (k+1)(k+2) + (m+1)(m+2)$ elements of the array **r** contain information required for any subsequent calls to the function with **mode** = 1 or 3; the rest of the array is used as workspace. See Section 8.

17: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_CLOSE_TO_STATIONARITY

The reference vector cannot be computed because the AR arguments are too close to the boundary of the stationarity region.

NE_INT

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

On entry, **mode** $= \langle value \rangle$.
Constraint: $0 \leq$ **mode** ≤ 3.

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

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

On entry, **pdv** $= \langle value \rangle$.
Constraint: **pdv** > 0.

On entry, **pdx** $= \langle value \rangle$.
Constraint: **pdx** > 0.

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

NE_INT_2

On entry, **pdv** $= \langle value \rangle$, **k** $= \langle value \rangle$.
Constraint: **pdv** $\geq$ **k**.

On entry, **pdx** $= \langle value \rangle$, **n** $= \langle value \rangle$.
Constraint: **pdx** $\geq \max(1,$ **n**$)$.

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 consult NAG for assistance.

NE_INVERTIBILITY

On entry, the MA argument matrices are outside the invertibility region.

NE_OUTSIDE_STATIONARITY

On entry, the AR argument matrices are outside the stationarity region.

NE_POS_DEF

On entry, the covariance matrix **var** is not positive-definite.

NE_TOO_MANY_ITER

An excessive number of iterations were required by the NAG function used to evaluate the eigenvalues of the covariance matrix.

An excessive number of iterations were required by the NAG function used to evaluate the eigenvalues of the matrices used to test for stationarity or invertibility.

An excessive number of iterations were required by the NAG function used to evaluate the eigenvalues to be stored in the reference vector.

7 Accuracy

The accuracy is limited by the matrix computations performed, and this is dependent on the condition of the argument and covariance matrices.

8 Further Comments

Note that, in reference to **fail.code** = **NE_INVERTIBILITY**, nag_rngs_varma_time_series (g05pcc) will permit moving average arguments on the boundary of the invertibility region.

The elements of **r** contain amongst other information details of the spectral decompositions which are used to generate future multivariate Normals. Note that these eigenvectors may not be unique on different machines. For example the eigenvectors corresponding to multiple eigenvalues may be permuted. Although an effort is made to ensure that the eigenvectors have the same sign on all machines, differences in the signs may theoretically still occur.

The following table gives some examples of the required size of the array **r**, specified by the argument , for $k = 1, 2, 3$, and for various values of p and q.

p	q = 0	1	2	3
0	13	20	31	46
	36	56	92	144
	85	124	199	310
1	19	30	45	64
	52	88	140	208
	115	190	301	448
2	35	50	69	92
	136	188	256	340
	397	508	655	838
3	57	76	99	126
	268	336	420	520
	877	1024	1207	1426

Note that nag_tsa_arma_roots (g13dxc) may be used to check whether a VARMA model is stationary and invertible.

The time taken depends on the values of p, q and especially n and k.

9 Example

This program generates two realisations, each of length 48, from the bivariate AR(1) model

$$X_t - \mu = \phi_1(X_{t-1} - \mu) + \epsilon_t$$

with

$$\phi_1 = \begin{bmatrix} 0.80 & 0.07 \\ 0.00 & 0.58 \end{bmatrix},$$

$$\mu = \begin{bmatrix} 5.00 \\ 9.00 \end{bmatrix},$$

and

$$\Sigma = \begin{bmatrix} 2.97 & 0 \\ 0.64 & 5.38 \end{bmatrix}.$$

The pseudo-random number generator is initialized by a call to nag_rngs_init_repeatable (g05kbc). Then, in the first call to nag_rngs_varma_time_series (g05pcc), **mode** is set to 2 in order to set up the reference vector before generating the first realisation. In the subsequent call **mode** is set to 3 and a new recent history is generated and used to generate the second realisation.

9.1 Program Text

```
/* nag_rngs_varma_time_series (g05pcc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>

int main(void)
{
  /* Scalars */
  Integer  i, igen, ii, ip, iq, j, k, l, n, nr;
  Integer  exit_status=0;
  NagError fail;
  Integer  pdx, pdvar;
  Nag_OrderType order;

  /* Arrays */
  double   *phi=0, *r=0, *theta=0, *var=0, *x=0, *xmean=0;
  Integer  iseed[4];

#ifdef NAG_COLUMN_MAJOR
#define X(I,J) x[(J-1)*pdx + I - 1]
#define VAR(I,J) var[(J-1)*pdvar + I - 1]
  order = Nag_ColMajor;
#else
#define X(I,J) x[(I-1)*pdx + J - 1]
#define VAR(I,J) var[(I-1)*pdvar + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);
  Vprintf("nag_rngs_varma_time_series (g05pcc) Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] %ld%ld%ld%ld%*[^\n] ", &k,
         &ip, &iq, &n);
  nr = 600;
  /* Allocate memory */
  if ( !(phi = NAG_ALLOC(k*k*ip, double)) ||
       !(r = NAG_ALLOC(nr, double)) ||
       !(theta = NAG_ALLOC(MAX(1,k*k*iq), double)) ||
       !(var = NAG_ALLOC(k * k, double)) ||
       !(x = NAG_ALLOC(k * n, double)) ||
       !(xmean = NAG_ALLOC(k, double)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

#ifdef NAG_COLUMN_MAJOR
  pdx = k;
  pdvar = k ;
#else
  pdx = n;
  pdvar = k ;
#endif


  if (n > 0 && n <= 100)
    {
      for (l = 0; l < ip; ++l)
        {
          for (i = 0; i < k; ++i)
```

```
        {
          ii = l * k * k + i;

          for (j = 0; j < k; ++j)
            {
              Vscanf("%lf", &phi[ii + k * j]);

            }
          Vscanf("%*[^\n] ");
        }
    }
  for (l = 0; l < iq; ++l)
    {
      for (i = 0; i < k; ++i)
        {
          ii = l * k * k + i;

          for (j = 0; j < k; ++j)
            Vscanf("%lf", &theta[ii + k * j]);
          Vscanf("%*[^\n] ");
        }
    }

  for (i = 0; i < k; ++i)
    {
      Vscanf("%lf", &xmean[i]);

    }

  Vscanf("%*[^\n] ");
  for (i = 1; i <= k; ++i)
    {
      for (j = 1; j <= i; ++j)
        Vscanf("%lf", &VAR(i,j));

      Vscanf("%*[^\n] ");
    }
  /* Initialise the seed to a repeatable sequence */
  iseed[0] = 1762543;
  iseed[1] = 9324783;
  iseed[2] = 4234401;
  iseed[3] = 742355;
  /* igen identifies the stream. */
  igen = 1;
  /* nag_rngs_init_repeatable (g05kbc).
   * Initialize seeds of a given generator for random number
   * generating functions (that pass seeds explicitly) to give
   * a repeatable sequence
   */
  nag_rngs_init_repeatable(&igen, iseed);

  /* nag_rngs_varma_time_series (g05pcc).
   * Generates a realisation of a multivariate time series
   * from a VARMA model
   */
  nag_rngs_varma_time_series(order, 2, k, xmean, ip, phi, iq, theta, var,
                             pdvar, n, x, pdx, igen, iseed, r, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from nag_rngs_varma_time_series (g05pcc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  Vprintf(" Realisation Number 1\n");
  Vprintf("\n");
  for (i = 1; i <= k; ++i)
    {
      Vprintf("  Series number %3ld\n", i);
      Vprintf("  -------------\n");
      Vprintf("\n");
```

```
          for (j = 1; j <= n; ++j)
            Vprintf("%8.3f%s", X(i,j), j%8 == 0 || j == n ?"\n":" ");
          Vprintf("\n");
        }
      /* nag_rngs_varma_time_series (g05pcc), see above. */
      nag_rngs_varma_time_series(order, 3, k, xmean, ip, phi, iq, theta, var,
                                 pdvar, n, x, pdx, igen, iseed, r, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from nag_rngs_varma_time_series (g05pcc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
      Vprintf("\n\n");

      Vprintf(" Realisation Number 2\n");
      Vprintf("\n");
      for (i = 1; i <= k; ++i)
        {

          Vprintf("  Series number %3ld\n", i);
          Vprintf("  -------------\n");
          Vprintf("\n");
          for (j = 1; j <= n; ++j)
            Vprintf("%8.3f%s", X(i,j), j%8 == 0 || j == n ?"\n":" ");
          Vprintf("\n");
        }
    }
 END:
  if (phi) NAG_FREE(phi);
  if (r) NAG_FREE(r);
  if (theta) NAG_FREE(theta);
  if (var) NAG_FREE(var);
  if (x) NAG_FREE(x);
  if (xmean) NAG_FREE(xmean);
  return exit_status;
}
```

9.2 Program Data

```
nag_rngs_varma_time_series (g05pcc) Example Program Data
 2 1 0 48                     : k, ip, iq, n
 0.80 0.07                    : phi(,,1)
 0.00 0.58                    :
 5.00 9.00                    : xmean
 2.97                         : var
 0.64 5.38
```

9.3 Program Results

```
nag_rngs_varma_time_series (g05pcc) Example Program Results

 Realisation Number 1

  Series number   1
  -------------

   5.765    3.983    0.496   -0.754   -1.718   -2.753   -0.568    1.719
   2.380    3.320    3.150    2.051    1.304    2.079    2.940    3.539
   3.876    3.179    7.230    5.791    6.297    6.508    7.993    7.041
   5.568    6.404    6.283    3.230    0.989   -0.397   -0.331    2.526
   1.629    2.693    3.093    2.967    5.162    6.331    8.169   10.801
   9.174    5.627    7.696    5.844    1.539    2.377    1.335   -0.144

  Series number   2
  -------------

  14.749   13.246    8.826    8.614    8.779    7.310    5.961    4.594
```

```
  5.312    6.354    6.843    7.136   10.653    9.416   10.083   11.012
 10.745    8.807    4.318    6.509    7.520    9.205    9.877    8.082
  8.408   11.218    9.648    9.260    6.631   10.407    7.784    6.509
  5.174    8.594    7.900    8.575    8.307    9.726    5.303    9.989
  9.924   13.597   16.134   15.464   11.816    8.428   11.221    7.869


Realisation Number 2

 Series number   1
 -------------

  6.253    9.454    8.619    7.403    6.684    7.998    6.669    7.011
  7.899    8.984    7.873    8.161    6.462    5.396    2.543    3.104
  4.285    3.837    1.109    2.372    5.804    1.929    6.479    5.964
  6.594    8.835    9.217    8.546    7.011    7.797    8.386    8.290
  8.227    4.163    4.232    5.161    6.794    7.188    6.552    7.101
  5.061    3.964    6.517    6.315    3.989    4.552    3.079    4.139

 Series number   2
 -------------

 11.114   13.425   10.316    9.063    6.324    3.673    5.568    7.103
  8.561    9.097   11.745   10.028   12.138    9.973    8.747   10.753
 15.455   10.861   14.161    9.624   12.976   10.141    7.931    8.289
  9.520   10.412    8.248    4.229    3.834    6.840    8.367   12.120
 13.373   14.411    9.508   12.724   11.858   12.463   14.742   12.301
 14.039   12.476   13.437   11.757   11.972    9.273    9.496    6.394
```