NAG Library Function Document

nag_rand_varma (g05pjc)

1 Purpose

nag_rand_varma (g05pjc) generates a realization of a multivariate time series from a vector autoregressive moving average (VARMA) model. The realization may be continued or a new realization generated at subsequent calls to nag_rand_varma (g05pjc).

2 Specification

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

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

3 Description

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

$$
X_t - \mu = \phi_1(X_{t-1} - \mu) + \phi_2(X_{t-2} - \mu) + \cdots + \phi_p(X_{t-p} - \mu) +
\epsilon_t - \theta_1\epsilon_{t-1} - \theta_2\epsilon_{t-2} - \cdots - \theta_q\epsilon_{t-q},
$$

(1)

where $\epsilon_t = (\epsilon_{1t}, \epsilon_{2t}, ..., \epsilon_{kt})^T$, is a vector of $k$ residual series assumed to be Normally distributed with zero mean and covariance matrix $\Sigma$. The components of $\epsilon_t$ are assumed to be uncorrelated at non-simultaneous lags. The $\phi_i$’s and $\theta_j$’s are $k$ by $k$ matrices of parameters. $\{\phi_i\}$, for $i = 1, 2, ..., p$, are called the autoregressive (AR) parameter matrices, and $\{\theta_j\}$, for $j = 1, 2, ..., q$, the moving average (MA) parameter matrices. The parameters in the model are thus the $pk$ by $k$ $\phi$-matrices, the $qk$ by $k$ $\theta$-matrices, the mean vector $\mu$ and the residual error covariance matrix $\Sigma$. Let

$$
A(\phi) = \begin{bmatrix}
\phi_1 & I & 0 & \cdots & 0 \\
\phi_2 & 0 & I & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\phi_{p-1} & 0 & \cdots & 0 & I \\
\phi_p & 0 & \cdots & 0 & 0
\end{bmatrix}_{pk \times pk}
$$

and

$$
B(\theta) = \begin{bmatrix}
\theta_1 & I & 0 & \cdots & 0 \\
\theta_2 & 0 & I & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\theta_{q-1} & 0 & \cdots & 0 & I \\
\theta_q & 0 & \cdots & 0 & 0
\end{bmatrix}_{qk \times qk}
$$

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 realization of the state vector at time zero computed. For all other cases the function computes a realization of the pre-observed vectors $X_0, X_{-1}, ..., X_{1-p}, \epsilon_0, \epsilon_{-1}, ..., \epsilon_{1-q}$, from (1), see Shea (1988). This realization 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 realization of the time series may be generated more efficiently using the information in a reference vector created during a previous call to nag_rand_varma (g05pjc). See the description of the argument `mode` in Section 5 for details.
The function returns a realization 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_rand_varma (g05pjc) may be called again to generate a realization of \(X_{n+1}, X_{n+2}, \ldots\), etc. See the description of the argument \texttt{mode} in Section 5 for details.

Further computational details are given in Shea (1988). Note, however, that nag_rand_varma (g05pjc) 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_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_varma (g05pjc).

### 4 References


### 5 Arguments

1. \texttt{order} – Nag_OrderType \hspace{1cm} \textit{Input}

   \textit{On entry}: the \texttt{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 \texttt{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

   \textit{Constraint}: \texttt{order} = Nag_RowMajor or Nag_ColMajor.

2. \texttt{mode} – Nag_ModeRNG \hspace{1cm} \textit{Input}

   \textit{On entry}: a code for selecting the operation to be performed by the function.

   \texttt{mode} = Nag_InitializeReference
   
   Set up reference vector and compute a realization of the recent history.

   \texttt{mode} = Nag_GenerateFromReference
   
   Generate terms in the time series using reference vector set up in a prior call to nag_rand_varma (g05pjc).

   \texttt{mode} = Nag_InitializeAndGenerate
   
   \textit{Combine the operations of} \texttt{mode} = Nag_InitializeReference and Nag_GenerateFromReference.

   \texttt{mode} = Nag_ReGenerateFromReference
   
   A new realization of the recent history is computed using information stored in the reference vector, and the following sequence of time series values are generated.

   If \texttt{mode} = Nag_GenerateFromReference or Nag_ReGenerateFromReference, then you must ensure that the reference vector \(r\) and the values of \(k, p, q, xmean, \phi, \theta, \var, \text{ and } pdv\) have not been changed between calls to nag_rand_varma (g05pjc).

   \textit{Constraint}: \texttt{mode} = Nag_InitializeReference, Nag_GenerateFromReference, Nag_InitializeAndGenerate or Nag_ReGenerateFromReference.

3. \texttt{n} – Integer \hspace{1cm} \textit{Input}

   \textit{On entry}: \(n\), the number of observations to be generated.

   \textit{Constraint}: \(n \geq 0\).
4: \( k \) – Integer  
On entry: \( k \), the dimension of the multivariate time series.  
Constraint: \( k \geq 1 \).

5: \( xmean[k] \) – const double  
On entry: \( \mu \), the vector of means of the multivariate time series.

6: \( p \) – Integer  
On entry: \( p \), the number of autoregressive parameter matrices.  
Constraint: \( p \geq 0 \).

7: \( \text{phi}[k \times k \times p] \) – const double  
On entry: must contain the elements of the \( p \times k \times k \) autoregressive parameter matrices of the model, \( \phi_1, \phi_2, \ldots, \phi_p \). The \((i,j)\)th element of \( \phi_l \) is stored in \( \text{phi}[(l-1) \times k \times k + (j-1) \times k + i + 1] \), for \( l = 1,2,\ldots,p \), \( i = 1,2,\ldots,k \) and \( j = 1,2,\ldots,k \).  
Constraint: the elements of \( \text{phi} \) must satisfy the stationarity condition.

8: \( q \) – Integer  
On entry: \( q \), the number of moving average parameter matrices.  
Constraint: \( q \geq 0 \).

9: \( \text{theta}[k \times k \times q] \) – const double  
On entry: must contain the elements of the \( q \times k \times k \) moving average parameter matrices of the model, \( \theta_1, \theta_2, \ldots, \theta_q \). The \((i,j)\)th element of \( \theta_l \) is stored in \( \text{theta}[(l-1) \times k \times k + (j-1) \times k + i + 1] \), for \( l = 1,2,\ldots,q \), \( i = 1,2,\ldots,k \) and \( j = 1,2,\ldots,k \).  
Constraint: the elements of \( \text{theta} \) must be within the invertibility region.

10: \( \text{var}[\text{dim}] \) – const double  
Note: the dimension, \( \text{dim} \), of the array \( \text{var} \) must be at least \( \text{pdv} \times k \).  
Where \( \text{VAR}(i,j) \) appears in this document, it refers to the array element  
\[
\text{VAR}[(j-1) \times \text{pdv} + i - 1] \quad \text{when order = Nag\_ColMajor}; \\
\text{VAR}[(i-1) \times \text{pdv} + j - 1] \quad \text{when order = Nag\_RowMajor}.
\]  
On entry: \( \text{VAR}(i,j) \) must contain the \((i,j)\)th element of \( \Sigma \), for \( i = 1,2,\ldots,k \) and \( j = 1,2,\ldots,k \).  
Only the lower triangle is required.  
Constraint: the elements of \( \text{var} \) must be such that \( \Sigma \) is positive semidefinite.

11: \( \text{pdv} \) – Integer  
On entry: the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( \text{var} \).  
Constraint: \( \text{pdv} \geq k \).

12: \( r[lr] \) – double  
Communication Array  
On entry: if \( \text{mode} = \text{Nag\_GenerateFromReference} \) or \( \text{Nag\_ReGenerateFromReference} \), the array \( r \) as output from the previous call to \( \text{nag\_rand\_varma} \) (g05pjc) must be input without any change.  
If \( \text{mode} = \text{Nag\_InitializeReference} \) or \( \text{Nag\_InitializeAndGenerate} \), the contents of \( r \) need not be set.
On exit: information required for any subsequent calls to the function with \texttt{mode = Nag\_GenerateFromReference} or \texttt{Nag\_ReGenerateFromReference}. See Section 9.

13: \texttt{lr} – Integer

\textit{Input}

\textit{On entry:} the dimension of the array \texttt{r}.

\textit{Constraints:}

\begin{align*}
\text{if } k \geq 6, \quad lr &\geq (5r^2 + 1) \times k^2 + (4r + 3) \times k + 4; \\
\text{if } k < 6, \quad lr &\geq \left( (p + q)^2 + 1 \right) \times k^2 + \\
&\quad (4 \times (p + q) + 3) \times k + \max\left(k^2(k^2 + 2), k^2(p + q)^2 + l(l + 3) + k^2(q + 1)\right) + 4.
\end{align*}

Where \( r = \max(p, q) \) and if \( p = 0, \quad l = k(k + 1)/2, \) or if \( p \geq 1, \quad l = k(k + 1)/2 + (p - 1)k^2. \)

See Section 9 for some examples of the required size of the array \texttt{r}.

14: \texttt{state[dim]} – Integer

\textit{Communication Array}

\textit{Note:} the dimension, \texttt{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 \texttt{state} in the previous call to \texttt{nag\_rand\_init\_repeatable} (g05kfc) or \texttt{nag\_rand\_init\_nonrepeatable} (g05kge).

\textit{On entry:} contains information on the selected base generator and its current state.

\textit{On exit:} contains updated information on the state of the generator.

15: \texttt{x[dim]} – double

\textit{Output}

\textit{Note:} the dimension, \texttt{dim}, of the array \texttt{x} must be at least

\begin{align*}
\max(1, \texttt{pdx} \times \texttt{n}) \quad &\text{when } \texttt{order} = \text{Nag\_ColMajor}; \\
\max(1, \texttt{k} \times \texttt{pdx}) \quad &\text{when } \texttt{order} = \text{Nag\_RowMajor}.
\end{align*}

Where \( X(i, t) \) appears in this document, it refers to the array element

\begin{align*}
x[(t - 1) \times \texttt{pdx} + i - 1] \quad &\text{when } \texttt{order} = \text{Nag\_ColMajor}; \\
x[(i - 1) \times \texttt{pdx} + t - 1] \quad &\text{when } \texttt{order} = \text{Nag\_RowMajor}.
\end{align*}

\textit{On exit:} \( X(i, t) \) will contain a realization of the \( i \)th component of \( X_t \), for \( i = 1, 2, \ldots, k \) and \( t = 1, 2, \ldots, n. \)

16: \texttt{pdx} – Integer

\textit{Input}

\textit{On entry:} the stride separating row or column elements (depending on the value of \texttt{order}) in the array \texttt{x}.

\textit{Constraints:}

\begin{align*}
\text{if } \texttt{order} = \text{Nag\_ColMajor}, \quad \texttt{pdx} &\geq \texttt{k}; \\
\text{if } \texttt{order} = \text{Nag\_RowMajor}, \quad \texttt{pdx} &\geq \texttt{n}.
\end{align*}

17: \texttt{fail} – NagError *

\textit{Input/Output}

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 \hspace{1em} \textbf{Error Indicators and Warnings}

\textbf{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 \textit{value} had an illegal value.

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

NE_INT
On entry, \( k = \langle \text{value} \rangle \).
Constraint: \( k \geq 1 \).

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

On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).

On entry, \( p = \langle \text{value} \rangle \).
Constraint: \( p \geq 0 \).

On entry, \( pdv = \langle \text{value} \rangle \).
Constraint: \( pdv > 0 \).

On entry, \( pdx = \langle \text{value} \rangle \).
Constraint: \( pdx > 0 \).

On entry, \( q = \langle \text{value} \rangle \).
Constraint: \( q \geq 0 \).

NE_INT_2
On entry, \( pdv = \langle \text{value} \rangle \) and \( k = \langle \text{value} \rangle \).
Constraint: \( pdv \geq k \).

On entry, \( pdx = \langle \text{value} \rangle \) and \( k = \langle \text{value} \rangle \).
Constraint: \( pdx \geq k \).

On entry, \( pdx = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( pdx \geq n \).

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 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, \textit{state} vector has been corrupted or not initialized.

NE_INVERTIBILITY
On entry, the moving average parameter matrices are such that the model is non-invertible.

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_POS_DEF
On entry, the covariance matrix \textit{var} is not positive semidefinite to \textit{machine precision}. 

\textit{Mark 25}
**NE_PREV_CALL**

- \( k \) is not the same as when \( r \) was set up in a previous call.
- Previous value of \( k = \text{value} \) and \( k = \text{value} \).

**NE_STATIONARY_AR**

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

**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 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 Parallelism and Performance**

`nag_rand_varma (g05pjc)` is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

`nag_rand_varma (g05pjc)` makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

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**

Note that, in reference to `fail.code = NE_INVERTIBILITY`, `nag_rand_varma (g05pjc)` will permit moving average parameters 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 \( lr \), for \( k = 1, 2 \) or \( 3 \), and for various values of \( p \) and \( q \).
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 \( k \).

10 Example

This program generates two realizations, 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 pseudorandom number generator is initialized by a call to nag_rand_init_repeatable (g05kfc). Then, in the first call to nag_rand_varma (g05pjc), \texttt{mode} = Nag_InitializeAndGenerate in order to set up the reference vector before generating the first realization. In the subsequent call \texttt{mode} = Nag_ReGenerateFromReference and a new recent history is generated and used to generate the second realization.

10.1 Program Text

/* nag_rand_varma (g05pjc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 9, 2009.
*/
#include <stdio.h>
#include <math.h>

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
13 & 20 & 31 & 46 \\
36 & 56 & 92 & 144 \\
85 & 124 & 199 & 310 \\
19 & 30 & 45 & 64 \\
52 & 88 & 140 & 208 \\
115 & 190 & 301 & 448 \\
35 & 50 & 69 & 92 \\
136 & 188 & 256 & 340 \\
397 & 508 & 655 & 838 \\
57 & 76 & 99 & 126 \\
268 & 336 & 420 & 520 \\
877 & 1024 & 1207 & 1426 \\
\end{array}
\]
#include <nag.h>
#include <nag_stdlib.h>
#include <nagG05.h>

#define VAR(I, J) var[(order == Nag_RowMajor)?(I*pdvar+J):(J*pdvar+I)]
#define X(I, J) x[(order == Nag_RowMajor)?(I*pdx+J):(J*pdx+I)]
#define PHI(i, j, l) phi[l*k*k+j*k+i]
#define THETA(i, j, l) theta[l*k*k+j*k+i]

int main(void)
{
    /* Integer scalar and array declarations */
    Integer exit_status = 0;
    Integer lr, x_size, var_size, i, ip, iq, j, k, l, lstate, n, tmp1,
         tmp2, tmp3, tmp4, tmp5;
    Integer *state = 0;
    Integer pdx, pdvar;
    /* NAG structures */
    NagError fail;
    Nag_ModeRNG mode;
    /* Double scalar and array declarations */
    double *phi = 0, *r = 0, *theta = 0, *var = 0, *x = 0, *xmean = 0;
    /* Use column major order */
    Nag_OrderType order = Nag_ColMajor;
    /* 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);

    /* 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;
    }

    /* Read data from a file */
    /* Skip heading*/
    #ifdef _WIN32
        scanf_s("%*[\n ]");
    #else
        scanf("%*[\n ]");
    #endif
    /* Read in initial parameters */
    #ifdef _WIN32
        scanf("%d%d%d%d%*[\n ]", &k, &ip, &iq, &n);
    #else
        scanf("%d%d%d%d%*[\n ]", &k, &ip, &iq, &n);
    #endif

    /* Calculate the size of the reference vector */
    tmp1 = (ip > iq)?ip:iq;
    if (ip == 0)
    {
        tmp2 = k * (k + 1) / 2;
    }
    else
    {
        
    
}
tmp2 = k*(k+1)/2 + (ip-1)*k*k;
}
tmp3 = ip + iq;
if (k >= 6)
{
lr = (5*tmp1*tmp1+1)*k*k + (4*tmp1+3)*k + 4;
}
else
{
tmp4 = k*tmp1*(k*tmp1+2);
tmp5 = k*k*tmp3+tmp2*(tmp2+3)+k*k*(iq+1);
lr = (tmp3*tmp3+1)*k*k + (4*tmp3+3)*k + ((tmp4 > tmp5)?tmp4:tmp5) + 4;
}
lr = (5*tmp1*tmp1+1)*k*k + (4*tmp1+3)*k + 4;
}

/* Allocate arrays */
if (!(phi = NAG_ALLOC(ip*k*k, double)) ||
!(r = NAG_ALLOC(lr, double)) ||
!(theta = NAG_ALLOC(iq*k*k, double)) ||
!(var = NAG_ALLOC(var_size, double)) ||
!(x = NAG_ALLOC(x_size, double)) ||
!(xmean = NAG_ALLOC(k, double)) ||
!(state = NAG_ALLOC(lstate, Integer)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}

/* Read in the AR parameters */
for (l = 0; l < ip; l++)
{
for (i = 0; i < k; i++)
{
	for (j = 0; j < k; j++)
#ifdef _WIN32
	scanf_s("%lf ", &PHI(i, j, l));
#else
	scanf("%lf ", &PHI(i, j, l));
#endif
}
#ifdef _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif

/* Read in the MA parameters */
if (iq > 0)
{
for (l = 0; l < iq; l++)
{
	for (i = 0; i < k; i++)
	{
		for (j = 0; j < k; j++)
#ifdef _WIN32
	scanf_s("%lf ", &THETA(i, j, l));
#else
	scanf("%lf ", &THETA(i, j, l));
#endif
	}
}
```c
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

/* Read in the means */
for (i = 0; i < k; i++)
    {
#ifdef _WIN32
        scanf_s("%lf ", &xmean[i]);
#else
        scanf("%lf ", &xmean[i]);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

/* Read in the variance / covariance matrix*/
for (i = 0; i < k; i++)
    {
        for (j = 0; j <= i; j++)
            {
#ifdef _WIN32
                scanf_s("%lf ", &VAR(i, j));
#else
                scanf("%lf ", &VAR(i, j));
#endif
            }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    }
    /* 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).
            %s
", fail.message);
            exit_status = 1;
            goto END;
        }
    /* Generate the first realization */
    mode = Nag_InitializeAndGenerate;
    nag_rand_varma(order, mode, n, k, xmean, ip, phi, iq, theta,
                   var, pdvar, r, ir, state, x, pdx, &fail);
    if (fail.code != NE_NOERROR)
        {
            printf("Error from nag_rand_varma (g05pjc).
            %s
", fail.message);
            exit_status = 1;
            goto END;
        }
    /* Display the results */
    printf(" Realization Number 1\\n");
    for (i = 0; i < k; i++)
        {
            printf("\n Series number \%3”NAG_IFMT”\n”, i+1);
            printf( " \-------------------\n\n ");
            for (j = 0; j < n; j++)
                printf("%9.3f%*s", X(i, j), (j+1)%8? "":\n ");
        }
    printf("\\n");
    /* Generate a second realization */
    mode = Nag_ReGenerateFromReference;
```
nag_rand_varma(order, mode, n, k, xmean, ip, phi, iq, theta, var, pdvar, r, ir, state, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_varma (g05pjc).\n%sn", fail.message);
    exit_status = 1;
    goto END;
}
/* Display the results */
printf(" Realization Number 2\n");
for (i = 0; i < k; i++)
{
    printf("n Series number %3"NAG_IFMT"n", i+1);
    printf(" \n\n ");
    for (j = 0; j < n; j++)
        printf("%9.3f%s", X(i, j), (j+1)%8?" :"\n ");
}
printf("n");

END:
NAG_FREE(phi);
NAG_FREE(r);
NAG_FREE(theta);
NAG_FREE(var);
NAG_FREE(x);
NAG_FREE(xmean);
NAG_FREE(state);
return exit_status;
}

10.2 Program Data
nag_rand_varma (g05pjc) Example Program Data
2 1 0 48 : k, ip, iq, n
0.80 0.07
0.00 0.58 : phi(,,1)
5.00 9.00 : xmean 
2.97 
0.64 5.38 : var

10.3 Program Results
nag_rand_varma (g05pjc) Example Program Results

Realization Number 1

Series number 1

-------------

  4.833  2.813  3.224  3.825  1.023  1.415  2.184  3.005
  3.554  7.045  7.025  4.106  5.106  5.954  8.026  7.212

Series number 2

-------------

  8.458  9.140 10.866 10.975  9.245  5.054  5.023 12.486

Realization Number 2
<table>
<thead>
<tr>
<th>Series number 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.396</td>
<td>4.811</td>
</tr>
<tr>
<td>3.130</td>
<td>4.308</td>
</tr>
<tr>
<td>1.745</td>
<td>3.211</td>
</tr>
<tr>
<td>2.765</td>
<td>2.148</td>
</tr>
<tr>
<td>4.839</td>
<td>3.698</td>
</tr>
<tr>
<td>7.537</td>
<td>7.788</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Series number 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11.345</td>
<td>10.070</td>
</tr>
<tr>
<td>10.263</td>
<td>13.394</td>
</tr>
<tr>
<td>10.626</td>
<td>9.366</td>
</tr>
<tr>
<td>10.799</td>
<td>8.780</td>
</tr>
<tr>
<td>9.935</td>
<td>9.386</td>
</tr>
<tr>
<td>15.246</td>
<td>9.962</td>
</tr>
</tbody>
</table>

NAG Library Manual

g05pjc

Mark 25