NAG Library Function Document

nag_tsa_varma_forecast (g13djc)

1 Purpose

nag_tsa_varma_forecast (g13djc) computes forecasts of a multivariate time series. It is assumed that a vector ARMA model has already been fitted to the appropriately differenced/transformed time series using nag_tsa_varma_estimate (g13ddc). The standard deviations of the forecast errors are also returned. A reference vector is set up so that, should future series values become available, the forecasts and their standard errors may be updated by calling nag_tsa_varma_update (g13dkc).

2 Specification

```
#include <nag.h>
#include <nagg13.h>
void nag_tsa_varma_forecast (Integer k, Integer n, const double z[],
     Integer kmax, const Integer tr[], const Integer id[],
     const double delta[], Integer ip, Integer iq, Nag_IncludeMean mean,
     const double par[], Integer lpar, double qq[], const double v[],
     Integer lmax, double predz[], double sefz[], double ref[], Integer lref,
     NagError *fail)
```

3 Description

Let the vector $Z_t = (z_{1t}, z_{2t}, \ldots, z_{kt})^{\mathrm{T}}$, for $t = 1, 2, \ldots, n$, denote a k-dimensional time series for which forecasts of $Z_{n+1}, Z_{n+2}, \ldots, Z_{n+l_{\max}}$ are required. Let $W_t = (w_{1t}, w_{2t}, \ldots, w_{kt})^{\mathrm{T}}$ be defined as follows:

$$w_{it} = \delta_i(B) z^*_{it}, \quad i = 1, 2, \ldots, k,$$

where $\delta_i(B)$ is the differencing operator applied to the ith series and where z^*_{it} is equal to either z_{it}, $\sqrt{z_{it}}$ or $\log_{\mathrm{e}}(z_{it})$ depending on whether or not a transformation was required to stabilize the variance before fitting the model.

If the order of differencing required for the ith series is d_i, then the differencing operator for the ith series is defined by $\delta_i(B) = 1 - \delta_{i1}B - \delta_{i2}B^2 - \cdots - \delta_{id_i}B^{d_i}$ where B is the backward shift operator; that is, $BZ_t = Z_{t-1}$. The differencing parameters δ_{ij}, for $i = 1, 2, \ldots, k$ and $j = 1, 2, \ldots, d_i$, must be supplied by you. If the ith series does not require differencing, then $d_i = 0$.

W_t is assumed to follow a multivariate ARMA model of the form:

$$W_t - \mu = \phi_1(W_{t-1} - \mu) + \phi_2(W_{t-2} - \mu) + \cdots + \phi_p\big(W_{t-p} - \mu\big) + \epsilon_t - \theta_1\epsilon_{t-1} - \cdots - \theta_q\epsilon_{t-q}, \quad (1)$$

where $\epsilon_t = (\epsilon_{1t}, \epsilon_{2t}, \ldots, \epsilon_{kt})^{\mathrm{T}}$, for $t = 1, 2, \ldots, n$, 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 and θ_j are k by k matrices of parameters. The matrices ϕ_i, for $i = 1, 2, \ldots, p$, are the autoregressive (AR) parameter matrices, and the matrices θ_i, for $i = 1, 2, \ldots, q$, the moving average (MA) parameter matrices. The parameters 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. The ARMA model (1) must be both stationary and invertible; see nag_tsa_arma_roots (g13dxc) for a method of checking these conditions.

The ARMA model (1) may be rewritten as

$$\phi(B)\big(\delta(B)Z^*_t - \mu\big) = \theta(B)\epsilon_t,$$

where $\phi(B)$ and $\theta(B)$ are the autoregressive and moving average polynomials and $\delta(B)$ denotes the k by k diagonal matrix whose ith diagonal elements is $\delta_i(B)$ and $Z^*_t = \big(z^*_{1t}, z^*_{2t} \ldots z^*_{kt}\big)^{\mathrm{T}}$.

This may be rewritten as

$$\phi(B)\delta(B)Z_t^* = \phi(B)\mu + \theta(B)\epsilon_t$$

or

$$Z_t^* = \tau + \psi(B)\epsilon_t = \tau + \epsilon_t + \psi_1\epsilon_{t-1} + \psi_2\epsilon_{t-2} + \cdots$$

where $\psi(B) = \delta^{-1}(B)\phi^{-1}(B)\theta(B)$ and $\tau = \delta^{-1}(B)\mu$ is a vector of length k.

Forecasts are computed using a multivariate version of the procedure described in Box and Jenkins (1976). If $\hat{Z}_n^*(l)$ denotes the forecast of Z_{n+l}^*, then $\hat{Z}_n^*(l)$ is taken to be that linear function of $Z_n^*, Z_{n-1}^*, \ldots$ which minimizes the elements of $E\{e_n(l)e_n'(l)\}$ where $e_n(l) = Z_{n+l}^* - \hat{Z}_n^*(l)$ is the forecast error. $\hat{Z}_n^*(l)$ is referred to as the linear minimum mean square error forecast of Z_{n+l}^*.

The linear predictor which minimizes the mean square error may be expressed as

$$\hat{Z}_n^*(l) = \tau + \psi_l\epsilon_n + \psi_{l+1}\epsilon_{n-1} + \psi_{l+2}\epsilon_{n-2} + \cdots.$$

The forecast error at t for lead l is then

$$e_n(l) = Z_{n+l}^* - \hat{Z}_n^*(l) = \epsilon_{n+l} + \psi_1\epsilon_{n+l-1} + \psi_2\epsilon_{n+l-2} + \cdots + \psi_{l-1}\epsilon_{n+1}.$$

Let $d = \max(d_i)$, for $i = 1, 2, \ldots, k$. Unless $q = 0$ the function requires estimates of ϵ_t, for $t = d+1, \ldots, n$, which are obtainable from nag_tsa_varma_estimate (g13ddc). The terms ϵ_t are assumed to be zero, for $t = n+1, \ldots, n+l_{\max}$. You may use nag_tsa_varma_update (g13dkc) to update these $l_{\max}$ forecasts should further observations, $Z_{n+1}, Z_{n+2}, \ldots$, become available. Note that when $l_{\max}$ or more further observations are available then nag_tsa_varma_forecast (g13djc) must be used to produce new forecasts for $Z_{n+l_{\max}+1}, Z_{n+l_{\max}+2}, \ldots$, should they be required.

When a transformation has been used the forecasts and their standard errors are suitably modified to give results in terms of the original series, Z_t; see Granger and Newbold (1976).

4 References

Box G E P and Jenkins G M (1976) *Time Series Analysis: Forecasting and Control* (Revised Edition) Holden–Day

Granger C W J and Newbold P (1976) Forecasting transformed series *J. Roy. Statist. Soc. Ser. B* **38** 189–203

Wei W W S (1990) *Time Series Analysis: Univariate and Multivariate Methods* Addison–Wesley

5 Arguments

The quantities **k**, **n**, **kmax**, **ip**, **iq**, **par**, **npar**, **qq** and **v** from nag_tsa_varma_estimate (g13ddc) are suitable for input to nag_tsa_varma_forecast (g13djc).

1: **k** – Integer *Input*

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

Constraint: $\mathbf{k} \geq 1$.

2: **n** – Integer *Input*

On entry: n, the number of observations in the series, Z_t, prior to differencing.

Constraint: $\mathbf{n} \geq 3$.

The total number of observations must exceed the total number of parameters in the model; that is

if **mean** = Nag_MeanZero, $\mathbf{n} \times \mathbf{k} > (\mathbf{ip} + \mathbf{iq}) \times \mathbf{k} \times \mathbf{k} + \mathbf{k} \times (\mathbf{k}+1)/2$;

if **mean** = Nag_MeanInclude, $\mathbf{n} \times \mathbf{k} > (\mathbf{ip} + \mathbf{iq}) \times \mathbf{k} \times \mathbf{k} + \mathbf{k} + \mathbf{k} \times (\mathbf{k}+1)/2$,

(see the arguments **ip**, **iq** and **mean**).

3: **z**[**kmax** × **n**] – const double *Input*

On entry: **z**$[(t-1) \times$ **kmax** $+ i - 1]$ must contain the ith series at time t, for $t = 1, 2, \ldots, n$ and $i = 1, 2, \ldots, k$.

4: **kmax** – Integer *Input*

On entry: the stride separating row elements in the two-dimensional data stored in the arrays **z**, **delta**, **qq**, **v**, **predz**, **sefz**.

Constraint: **kmax** $\geq$ **k**.

5: **tr**[**k**] – const Integer *Input*

On entry: **tr**$[i-1]$ indicates whether the ith series is to be transformed, for $i = 1, 2, \ldots, k$.

tr$[i-1] = -1$
A square root transformation is used.

tr$[i-1] = 0$
No transformation is used.

tr$[i-1] = 1$
A log transformation is used.

Constraint: **tr**$[i-1] = -1$, 0 or 1, for $i = 1, 2, \ldots, k$.

6: **id**[**k**] – const Integer *Input*

On entry: **id**$[i-1]$ must specify, d_i, the order of differencing required for the ith series.

Constraint: $0 \leq$ **id**$[i-1] <$ **n** $-$ max(**ip**, **iq**), for $i = 1, 2, \ldots, k$.

7: **delta**$[dim]$ – const double *Input*

Note: the dimension, dim, of the array **delta** must be at least **kmax** $\times\, d$, where $d =$ max(**id**$[i-1]$).

On entry: if **id**$[i-1] > 0$, then **delta**$[(j-1) \times$ **kmax** $+ i - 1]$ must be set equal to δ_{ij}, for $j = 1, 2, \ldots, d_i$ and $i = 1, 2, \ldots, k$.

If $d = 0$, **delta** is not referenced.

8: **ip** – Integer *Input*

On entry: p, the number of AR parameter matrices.

Constraint: **ip** ≥ 0.

9: **iq** – Integer *Input*

On entry: q, the number of MA parameter matrices.

Constraint: **iq** ≥ 0.

10: **mean** – Nag_IncludeMean *Input*

On entry: **mean** = Nag_MeanInclude, if components of μ have been estimated and **mean** = Nag_MeanZero, if all elements of μ are to be taken as zero.

Constraint: **mean** = Nag_MeanInclude or Nag_MeanZero.

11: **par**[**lpar**] – const double *Input*

On entry: must contain the parameter estimates read in row by row in the order $\phi_1, \phi_2, \ldots, \phi_p$, $\theta_1, \theta_2, \ldots, \theta_q$, μ.

Thus,

if **ip** > 0, **par**$[(l-1)\times k\times k+(i-1)\times k+j-1]$ must be set equal to an estimate of the (i,j)th element of ϕ_l, for $l=1,2,\ldots,p$, $i=1,2,\ldots,k$ and $j=1,2,\ldots,k$;

if **iq** > 0, **par**$[p\times k\times k+(l-1)\times k\times k+(i-1)\times k+j-1]$ must be set equal to an estimate of the (i,j)th element of θ_l, for $l=1,2,\ldots,q$, $i=1,2,\ldots,k$ and $j=1,2,\ldots,k$;

if **mean** $=$ Nag_MeanInclude, **par**$[(p+q)\times k\times k+i-1]$ must be set equal to an estimate of the ith component of μ, for $i=1,2,\ldots,k$.

Constraint: the first **ip** $\times$ **k** $\times$ **k** elements of **par** must satisfy the stationarity condition and the next **iq** $\times$ **k** $\times$ **k** elements of **par** must satisfy the invertibility condition.

12: **lpar** – Integer *Input*

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

Constraints:

if **mean** $=$ Nag_MeanZero, **lpar** $\geq \max(1,(\mathbf{ip}+\mathbf{iq})\times\mathbf{k}\times\mathbf{k})$;
if **mean** $=$ Nag_MeanInclude, **lpar** $\geq (\mathbf{ip}+\mathbf{iq})\times\mathbf{k}\times\mathbf{k}+\mathbf{k}$.

13: **qq**[**kmax** $\times$ **k**] – double *Input/Output*

On entry: **qq**$[(j-1)\times$ **kmax** $+i-1]$ must contain an estimate of the (i,j)th element of Σ. The lower triangle only is needed.

Constraint: **qq** must be positive definite.

On exit: if **fail.code** $=$ NE_EIGENVALUES, NE_G13D_AR, NE_G13D_MA, NE_NEARLY_POS_DEF, NE_NOT_POS_DEF, NE_OVERFLOW_LIKELY or NE_TRANSFORMATION, then the upper triangle is set equal to the lower triangle.

14: **v**[dim] – const double *Input*

Note: the dimension, dim, of the array **v** must be at least **kmax** $\times$ (**n** $- d$), where $d=\max(\mathbf{id}[i-1])$.

On entry: **v**$[(t-1)\times$ **kmax** $+i-1]$ must contain an estimate of the ith component of ϵ_{t+d}, for $i=1,2,\ldots,k$ and $t=1,2,\ldots,n-d$.

If $q=0$, **v** is not used.

15: **lmax** – Integer *Input*

On entry: the number, $l_{\max}$, of forecasts required.

Constraint: **lmax** ≥ 1.

16: **predz**[**kmax** $\times$ **lmax**] – double *Output*

On exit: **predz**$[(l-1)\times$ **kmax** $+i-1]$ contains the forecast of $z_{i,n+l}$, for $i=1,2,\ldots,k$ and $l=1,2,\ldots,l_{\max}$.

17: **sefz**[**kmax** $\times$ **lmax**] – double *Output*

On exit: **sefz**$[(l-1)\times$ **kmax** $+i-1]$ contains an estimate of the standard error of the forecast of $z_{i,n+l}$, for $i=1,2,\ldots,k$ and $l=1,2,\ldots,l_{\max}$.

18: **ref**[**lref**] – double *Output*

On exit: the reference vector which may be used to update forecasts using nag_tsa_varma_update (g13dkc). The first (**lmax** $-1)\times$ **k** $\times$ **k** elements contain the ψ weight matrices, $\psi_1,\psi_2,\ldots,\psi_{l_{\max}-1}$. The next **k** $\times$ **lmax** elements contain the forecasts of the transformed series $\hat{Z}^*_{n+1},\hat{Z}^*_{n+2},\ldots,\hat{Z}^*_{n+l_{\max}}$ and the next **k** $\times$ **lmax** contain the variances of the forecasts of the transformed variables. The last **k** elements are used to store the transformations for the series.

19: **lref** – Integer *Input*

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

Constraint: **lref** $\geq (\mathbf{lmax} - 1) \times \mathbf{k} \times \mathbf{k} + 2 \times \mathbf{k} \times \mathbf{lmax} + \mathbf{k}$.

20: **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 3.2.1.2 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_EIGENVALUES

An excessive number of iterations were needed by nag_tsa_arma_roots (g13dxc) to evaluate the eigenvalues of the matrices used to test for stationarity and invertibility.

NE_G13D_AR

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

NE_G13D_MA

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

NE_INT

On entry, **ip** = ⟨*value*⟩.
Constraint: **ip** ≥ 0.

On entry, **iq** = ⟨*value*⟩.
Constraint: **iq** ≥ 0.

On entry, **k** = ⟨*value*⟩.
Constraint: **k** ≥ 1.

On entry, **lmax** = ⟨*value*⟩.
Constraint: **lmax** ≥ 1.

On entry, **lpar** is too small: **lpar** = ⟨*value*⟩ but must be at least ⟨*value*⟩.

On entry, **lref** is too small: **lref** = ⟨*value*⟩ but must be at least ⟨*value*⟩.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 3.

NE_INT_2

On entry, **kmax** = ⟨*value*⟩ and **k** = ⟨*value*⟩.
Constraint: **kmax** $\geq$ **k**.

NE_INT_ARRAY

On entry, **id**$[\langle value \rangle] = \langle value \rangle$ and $\mathbf{n} - \max(\mathbf{ip}, \mathbf{iq}) = \langle value \rangle$.
Constraint: $0 \leq \mathbf{id}[i-1] < \mathbf{n} - \max(\mathbf{ip}, \mathbf{iq})$.

On entry, **tr**[$\langle value \rangle$] = $\langle value \rangle$.
Constraint: **tr**[i] = -1, 0 or 1.

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 How to Use the NAG Library and its Documentation for further information.

NE_NEARLY_POS_DEF

The covariance matrix may be nearly non positive definite.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_NOT_POS_DEF

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

NE_OBSERV_LT_P

On entry, the total number of observations is less than the total number of parameters (including the covariance matrix). Number of observations = $\langle value \rangle$ and number of parameters = $\langle value \rangle$.

NE_OVERFLOW_LIKELY

The forecasts will overflow if computed.

NE_TRANSFORMATION

On entry, one (or more) of the transformations requested is invalid.

7 Accuracy

The matrix computations are believed to be stable.

8 Parallelism and Performance

nag_tsa_varma_forecast (g13djc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_tsa_varma_forecast (g13djc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

The same differencing operator does not have to be applied to all the series. For example, suppose we have $k = 2$, and wish to apply the second order differencing operator ∇^2 to the first series and the first-order differencing operator ∇ to the second series:

$$w_{1t} = \nabla^2 z_{1t} = (1-B)^2 z_{1t} = \left(1 - 2B + B^2\right) Z_{1t}, \quad \text{and}$$
$$w_{2t} = \nabla z_{2t} = (1-B) z_{2t}.$$

Then $d_1 = 2, d_2 = 1,\ d = \max(d_1, d_2) = 2,$ and

$$\textbf{delta} = \begin{bmatrix} \delta_{11} & \delta_{12} \\ \delta_{21} & \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 1 & \end{bmatrix}.$$

Note: although differencing may already have been applied prior to the model fitting stage, the differencing parameters supplied in **delta** are part of the model definition and are still required by this function to produce the forecasts.

nag_tsa_varma_forecast (g13djc) should not be used when the moving average parameters lie close to the boundary of the invertibility region. The function does test for both invertibility and stationarity but if in doubt, you may use nag_tsa_arma_roots (g13dxc), before calling this function, to check that the VARMA model being used is invertible.

On a successful exit, the quantities **k**, **lmax**, **kmax**, **ref** and **lref** will be suitable for input to nag_tsa_varma_update (g13dkc).

10 Example

This example computes forecasts of the next five values in two series each of length 48. No transformation is to be used and no differencing is to be applied to either of the series. nag_tsa_varma_estimate (g13ddc) is first called to fit an AR(1) model to the series. The mean vector μ is to be estimated and $\phi_1(2,1)$ constrained to be zero.

10.1 Program Text

```
/* nag_tsa_varma_forecast (g13djc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg13.h>

int main(void)
{
  /* Scalars */
  double cgetol, rlogl;
  Integer exit_status = 0, i, i2, idmax, idmin, ip, iprint, iq, ishow;
  Integer j, l2, lref, lmax, loop, maxcal, n, nd, niter, k, l, npar;
  Integer kmax, icm;
  /* Arrays */
  double *cm = 0, *delta = 0, *g = 0, *par = 0, *predz = 0, *qq = 0;
  double *ref = 0, *sefz = 0, *v = 0, *w = 0, *z = 0;
  Integer *id = 0, *tr = 0;
  char nag_enum_arg[40];
  /* Nag types */
  Nag_Boolean *parhld = 0;
  Nag_Boolean exact;
  Nag_IncludeMean mean;
  NagError fail;

#define DELTA(I, J) delta[(J - 1) * kmax + I - 1]
#define PREDZ(I, J) predz[(J - 1) * kmax + I - 1]
#define QQ(I, J)    qq[(J - 1) * kmax + I - 1]
#define SEFZ(I, J)  sefz[(J - 1) * kmax + I - 1]
#define Z(I, J)     z[(J - 1) * kmax + I - 1]

  INIT_FAIL(fail);

  printf("nag_tsa_varma_forecast (g13djc) Example Program Results\n\n");
```

```
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT " %39s %"
          NAG_IFMT "%*[^\n]", &k, &n, &ip, &iq, nag_enum_arg,
          (unsigned)_countof(nag_enum_arg), &lmax);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT " %39s %" NAG_IFMT
        "%*[^\n]", &k, &n, &ip, &iq, nag_enum_arg, &lmax);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  mean = (Nag_IncludeMean) nag_enum_name_to_value(nag_enum_arg);

  npar = (ip + iq) * k * k;
  if (mean == Nag_MeanInclude) {
    npar += k;
  }
  if (k > 0 && n >= 1 && npar >= 1 && lmax >= 1) {
    kmax = k;
    icm = npar;
    lref = (lmax - 1) * k * k + 2 * k * lmax + k;
    /* Allocate memory */
    if (!(tr = NAG_ALLOC(k, Integer)) ||
        !(cm = NAG_ALLOC(npar * icm, double)) ||
        !(g = NAG_ALLOC(npar, double)) ||
        !(par = NAG_ALLOC(npar, double)) ||
        !(predz = NAG_ALLOC(lmax * kmax, double)) ||
        !(qq = NAG_ALLOC(k * kmax, double)) ||
        !(ref = NAG_ALLOC(lref, double)) ||
        !(sefz = NAG_ALLOC(lmax * kmax, double)) ||
        !(v = NAG_ALLOC(n * kmax, double)) ||
        !(w = NAG_ALLOC(n * kmax, double)) ||
        !(z = NAG_ALLOC(n * kmax, double)) ||
        !(id = NAG_ALLOC(k, Integer)) ||
        !(parhld = NAG_ALLOC(npar, Nag_Boolean)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    printf("Invalid parameters\n");
    exit_status = -1;
    goto END;
  }

  for (i = 1; i <= k; ++i) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &id[i - 1]);
#else
    scanf("%" NAG_IFMT "", &id[i - 1]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  idmin = 0;
  idmax = 0;
  for (i = 1; i <= k; ++i) {
    idmin = MIN(id[i - 1], idmin);
```

```
    idmax = MAX(id[i - 1], idmax);
  }

  if (idmin >= 0) {
    if (!(delta = NAG_ALLOC(k * idmax, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
    for (i = 1; i <= k; ++i) {
      for (j = 1; j <= n; ++j) {
#ifdef _WIN32
        scanf_s("%lf ", &Z(i, j));
#else
        scanf("%lf ", &Z(i, j));
#endif
      }
    }
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif

    for (i = 1; i <= k; ++i) {
#ifdef _WIN32
      scanf_s("%" NAG_IFMT " ", &tr[i - 1]);
#else
      scanf("%" NAG_IFMT " ", &tr[i - 1]);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif

    if (idmax > 0) {
      for (i = 1; i <= k; ++i) {
        for (j = 1; j <= id[i - 1]; ++j) {
#ifdef _WIN32
          scanf_s("%lf", &DELTA(i, j));
#else
          scanf("%lf", &DELTA(i, j));
#endif
        }
#ifdef _WIN32
        scanf_s("%*[^\n] ");
#else
        scanf("%*[^\n] ");
#endif
      }
    }

    /* nag_tsa_multi_diff (g13dlc).
     * Multivariate time series, differences and/or transforms
     */
    nag_tsa_multi_diff(k, n, z, tr, id, delta, w, &nd, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_tsa_multi_diff (g13dlc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    for (i = 1; i <= npar; ++i) {
      par[i - 1] = 0.0;
      parhld[i - 1] = Nag_FALSE;
    }
    for (i = 1; i <= k; ++i) {
      for (j = 1; j <= i; ++j) {
```

```
      QQ(i, j) = 0.0;
    }
  }
  parhld[2] = Nag_TRUE;
  exact = Nag_TRUE;
  /* ** Set iprint < 0 for no monitoring */
  iprint = -1;
  cgetol = 1.0e-4;
  maxcal = npar * 40 * (npar + 5);
  ishow = 0;
  /* nag_tsa_varma_estimate (g13ddc).
   * Multivariate time series, estimation of VARMA model
   */
  nag_tsa_varma_estimate(k, nd, ip, iq, mean, par, npar, qq, kmax, w,
                         parhld, exact, iprint, cgetol, maxcal, ishow,
                         0, &niter, &rlogl, v, g, cm, icm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("\n nag_tsa_varma_estimate (g13ddc) message: %s\n\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  if (fail.code == NE_NOERROR || fail.code == NE_G13D_MAXCAL ||
      fail.code == NE_MAX_LOGLIK || fail.code == NE_G13D_BOUND ||
      fail.code == NE_G13D_DERIV || fail.code == NE_HESS_NOT_POS_DEF) {
    /* nag_tsa_varma_forecast (g13djc).
     * Multivariate time series, forecasts and their standard
     * errors
     */
    nag_tsa_varma_forecast(k, n, z, kmax, tr, id, delta, ip, iq, mean,
                           par, npar, qq, v, lmax, predz, sefz, ref,
                           lref, &fail);
    if (fail.code != NE_NOERROR) {
      printf("\n nag_tsa_varma_forecast (g13djc) message: %s\n\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

    printf("\n");
    printf("Forecast summary table\n");
    printf("----------------------\n\n");
    printf("Forecast origin is set at t = %4" NAG_IFMT "\n\n", n);

    loop = lmax / 5;
    if (lmax % 5 != 0) {
      ++loop;
    }

    for (j = 1; j <= loop; ++j) {
      i2 = (j - 1) * 5;
      l2 = MIN(i2 + 5, lmax);
      printf("%s%13s", "Lead Time", "");
      for (i = i2 + 1; i <= l2; ++i) {
        printf("%10" NAG_IFMT "%s", i,
               (i % 5 == 0 || i == l2 ? "\n" : " "));
      }
      printf("\n");

      for (i = 1; i <= k; ++i) {
        printf("%-7s%2" NAG_IFMT "%-15s", "Series", i, ": Forecast");
        for (l = i2 + 1; l <= l2; ++l) {
          printf("%10.2f%s", PREDZ(i, l),
                 (l % 5 == 0 || l == l2 ? "\n" : " "));
        }

        printf("%9s%-18s", "", ": Standard Error ");
        for (l = i2 + 1; l <= l2; ++l) {
          printf("%7.2f%s", SEFZ(i, l),
                 (l % 5 == 0 || l == l2 ? "\n" : "    "));
```

```
          }
          printf("\n");
        }
      }
    }
  }

END:
  NAG_FREE(tr);
  NAG_FREE(cm);
  NAG_FREE(delta);
  NAG_FREE(g);
  NAG_FREE(par);
  NAG_FREE(predz);
  NAG_FREE(qq);
  NAG_FREE(ref);
  NAG_FREE(sefz);
  NAG_FREE(v);
  NAG_FREE(w);
  NAG_FREE(z);
  NAG_FREE(id);
  NAG_FREE(parhld);

  return exit_status;
}
```

10.2 Program Data

```
nag_tsa_varma_forecast (g13djc) Example Program Data
2 48 1 0 Nag_MeanInclude 5  :  k, n, ip, iq, mean, lmax
0  0                        :  id[i-1], i=1,k
-1.490 -1.620  5.200  6.230  6.210  5.860  4.090  3.180
 2.620  1.490  1.170  0.850 -0.350  0.240  2.440  2.580
 2.040  0.400  2.260  3.340  5.090  5.000  4.780  4.110
 3.450  1.650  1.290  4.090  6.320  7.500  3.890  1.580
 5.210  5.250  4.930  7.380  5.870  5.810  9.680  9.070
 7.290  7.840  7.550  7.320  7.970  7.760  7.000  8.350
 7.340  6.350  6.960  8.540  6.620  4.970  4.550  4.810
 4.750  4.760 10.880 10.010 11.620 10.360  6.400  6.240
 7.930  4.040  3.730  5.600  5.350  6.810  8.270  7.680
 6.650  6.080 10.250  9.140 17.750 13.300  9.630  6.800
 4.080  5.060  4.940  6.650  7.940 10.760 11.890  5.850
 9.010  7.500 10.020 10.380  8.150  8.370 10.730 12.140 : End of time series
 0  0       :  tr[i-1], i=1,k
```

10.3 Program Results

```
nag_tsa_varma_forecast (g13djc) Example Program Results


Forecast summary table
----------------------

Forecast origin is set at t =   48

Lead Time                      1         2         3         4         5

Series  1: Forecast          7.82      7.28      6.77      6.33      5.95
         : Standard Error    1.72      2.23      2.51      2.68      2.79

Series  2: Forecast         10.31      9.25      8.65      8.30      8.10
         : Standard Error    2.32      2.68      2.78      2.82      2.83
```