NAG Library Function Document

nag_tsa_varma_update (g13dkc)

1 Purpose

nag_tsa_varma_update (g13dkc) accepts a sequence of new observations in a multivariate time series and updates both the forecasts and the standard deviations of the forecast errors. A call to nag_tsa_varma_forecast (g13djc) must be made prior to calling this function in order to calculate the elements of a reference vector together with a set of forecasts and their standard errors. On a successful exit from nag_tsa_varma_update (g13dkc) the reference vector is updated so that should future series values become available these forecasts may be updated by recalling nag_tsa_varma_update (g13dkc).

2 Specification

```
#include <nag.h>
#include <nagg13.h>
```

```
void nag_tsa_varma_update (Integer k, Integer lmax, Integer m,
     Integer *mlast, const double z[], Integer kmax, double ref[],
     Integer lref, double v[], double predz[], double sefz[], NagError *fail)
```

3 Description

Let $Z_t = (z_{1t}, z_{2t}, \ldots, z_{kt})^{\mathrm{T}}$, for $t = 1, 2, \ldots, n$, denote a k-dimensional time series for which forecasts of $\hat{Z}_{n+1}, \hat{Z}_{n+2}, \ldots, \hat{Z}_{n+l_{\max}}$ have been computed using nag_tsa_varma_forecast (g13djc). Given m further observations $Z_{n+1}, Z_{n+2}, \ldots, Z_{n+m}$, where $m < l_{\max}$, nag_tsa_varma_update (g13dkc) updates the forecasts of $Z_{n+m+1}, Z_{n+m+2}, \ldots, Z_{n+l_{\max}}$ and their corresponding standard errors.

nag_tsa_varma_update (g13dkc) uses a multivariate version of the procedure described in Box and Jenkins (1976). The forecasts are updated using the ψ weights, computed in nag_tsa_varma_forecast (g13djc). If Z_t^* denotes the transformed value of Z_t and $\hat{Z}_t^*(l)$ denotes the forecast of Z_{t+l}^* from time t with a lead of l (that is the forecast of Z_{t+l}^* given observations $Z_t^*, Z_{t-1}^*, \ldots$), then

$$\hat{Z}_{t+1}^*(l) = \tau + \psi_l \epsilon_{t+1} + \psi_{l+1} \epsilon_t + \psi_{l+2} \epsilon_{t-1} + \cdots$$

and

$$\hat{Z}_t^*(l+1) = \tau + \psi_{l+1} \epsilon_t + \psi_{l+2} \epsilon_{t-1} + \cdots$$

where τ is a constant vector of length k involving the differencing parameters and the mean vector μ. By subtraction we obtain

$$\hat{Z}_{t+1}^*(l) = \hat{Z}_t^*(l+1) + \psi_l \epsilon_{t+1}.$$

Estimates of the residuals corresponding to the new observations are also computed as $\epsilon_{n+l} = Z_{n+l}^* - \hat{Z}_n^*(l)$, for $l = 1, 2, \ldots, m$. These may be of use in checking that the new observations conform to the previously fitted model.

On a successful exit, the reference array is updated so that nag_tsa_varma_update (g13dkc) may be called again should future series values become available, see Section 9.

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**, **lmax**, **kmax**, **ref** and **lref** from nag_tsa_varma_forecast (g13djc) are suitable for input to nag_tsa_varma_update (g13dkc).

1: **k** – Integer *Input*

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

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

2: **lmax** – Integer *Input*

On entry: the number, $l_{\max}$, of forecasts requested in the call to nag_tsa_varma_forecast (g13djc).

Constraint: $\mathbf{lmax} \geq 2$.

3: **m** – Integer *Input*

On entry: m, the number of new observations available since the last call to either nag_tsa_varma_forecast (g13djc) or nag_tsa_varma_update (g13dkc). The number of new observations since the last call to nag_tsa_varma_forecast (g13djc) is then $\mathbf{m} + \mathbf{mlast}$.

Constraint: $0 < \mathbf{m} < \mathbf{lmax} - \mathbf{mlast}$.

4: **mlast** – Integer * *Input/Output*

On entry: on the first call to nag_tsa_varma_update (g13dkc), since calling nag_tsa_varma_forecast (g13djc), **mlast** must be set to 0 to indicate that no new observations have yet been used to update the forecasts; on subsequent calls **mlast** must contain the value of **mlast** as output on the previous call to nag_tsa_varma_update (g13dkc).

On exit: is incremented by m to indicate that $\mathbf{mlast} + \mathbf{m}$ observations have now been used to update the forecasts since the last call to nag_tsa_varma_forecast (g13djc).

mlast must not be changed between calls to nag_tsa_varma_update (g13dkc), unless a call to nag_tsa_varma_forecast (g13djc) has been made between the calls in which case **mlast** should be reset to 0.

Constraint: $0 \leq \mathbf{mlast} < \mathbf{lmax} - \mathbf{m}$.

5: $\mathbf{z}[\mathbf{kmax} \times \mathbf{m}]$ – const double *Input*

On entry: $\mathbf{z}[\mathbf{kmax} \times (j-1) + i - 1]$ must contain the value of $z_{i,n+\mathbf{mlast}+j}$, for $i = 1,2,\ldots,k$ and $j = 1,2,\ldots,m$, and where n is the number of observations in the time series in the last call made to nag_tsa_varma_forecast (g13djc).

Constraint: if the transformation defined in **tr** in nag_tsa_varma_forecast (g13djc) for the ith series is the log transformation, then $\mathbf{z}[\mathbf{kmax} \times (j-1) + i - 1] > 0.0$, and if it is the square-root transformation, then $\mathbf{z}[\mathbf{kmax} \times (j-1) + i - 1] \geq 0.0$, for $j = 1,2,\ldots,m$ and $i = 1,2,\ldots,k$.

6: **kmax** – Integer *Input*

On entry: the first dimension of the arrays **Z**, **PREDZ**, **SEFZ** and **V**.

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

7: **ref**[**lref**] – double *Input/Output*

On entry: must contain the first $(\mathbf{lmax}-1)\times\mathbf{k}\times\mathbf{k}+2\times\mathbf{k}\times\mathbf{lmax}+\mathbf{k}$ elements of the reference vector as returned on a successful exit from nag_tsa_varma_forecast (g13djc) (or a previous call to nag_tsa_varma_update (g13dkc)).

On exit: the elements of **ref** are updated. The first $(\mathbf{lmax}-1)\times\mathbf{k}\times\mathbf{k}$ elements store the ψ weights $\psi_1,\psi_2,\ldots,\psi_{l_{\max}-1}$. The next $\mathbf{k}\times\mathbf{lmax}$ elements contain the forecasts of the transformed series and the next $\mathbf{k}\times\mathbf{lmax}$ elements contain the variances of the forecasts of the transformed variables; see nag_tsa_varma_forecast (g13djc). The last **k** elements are not updated.

8: **lref** – Integer *Input*

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

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

9: **v**[**kmax** × **m**] – double *Output*

On exit: $\mathbf{v}[\mathbf{kmax}\times(j-1)+i-1]$ contains an estimate of the ith component of $\epsilon_{n+\mathbf{mlast}+j}$, for $i=1,2,\ldots,k$ and $j=1,2,\ldots,m$.

10: **predz**[**kmax** × **lmax**] – double *Input/Output*

On entry: nonupdated values are kept intact.

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

The columns of **predz** corresponding to the new observations since the last call to either nag_tsa_varma_forecast (g13djc) or nag_tsa_varma_update (g13dkc) are set equal to the corresponding columns of **z**.

11: **sefz**[**kmax** × **lmax**] – double *Input/Output*

On entry: nonupdated values are kept intact.

On exit: $\mathbf{sefz}[\mathbf{kmax}\times(j-1)+i-1]$ contains an estimate of the standard error of the corresponding element of **predz**, for $i=1,2,\ldots,k$ and $j=\mathbf{mlast}+\mathbf{m}+1,\ldots,l_{\max}$.

The columns of **sefz** corresponding to the new observations since the last call to either nag_tsa_varma_forecast (g13djc) or nag_tsa_varma_update (g13dkc) are set equal to zero.

12: **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, $\mathbf{k}=\langle\mathit{value}\rangle$.
Constraint: $\mathbf{k}\geq 1$.

On entry, **lmax** $= \langle value \rangle$.
Constraint: **lmax** ≥ 2.

On entry, **lref** is too small: **lref** $= \langle value \rangle$ but must be at least $\langle value \rangle$.

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

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

NE_INT_2

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

NE_INT_3

On entry, **m** $= \langle value \rangle$, **lmax** $= \langle value \rangle$ and **mlast** $= \langle value \rangle$.
Constraint: **m** $<$ **lmax** $-$ **mlast**.

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_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_REF_VEC

On entry, some of the elements of the array **ref** have been corrupted.

NE_RESULT_OVERFLOW

The updated 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_update (g13dkc) 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

If a further m^* observations, $Z_{n+\mathbf{mlast}+1}, Z_{n+\mathbf{mlast}+2}, \ldots, Z_{n+\mathbf{mlast}+m^*}$, become available, then forecasts of $Z_{n+\mathbf{mlast}+m^*+1}, Z_{n+\mathbf{mlast}+m^*+2}, \ldots, Z_{n+l_{\max}}$ may be updated by recalling nag_tsa_varma_update (g13dkc) with $\mathbf{m} = m^*$. Note that **m** and the contents of the array **z** are the only quantities which need updating; **mlast** is updated on exit from the previous call. On a successful exit, **v** contains estimates of $\epsilon_{n+\mathbf{mlast}+1}, \epsilon_{n+\mathbf{mlast}+2}, \ldots, \epsilon_{n+\mathbf{mlast}+m^*}$; columns $\mathbf{mlast}+1, \mathbf{mlast}+2, \ldots, \mathbf{mlast}+m^*$ of **predz** contain the new observed values $Z_{n+\mathbf{mlast}+1}, Z_{n+\mathbf{mlast}+2}, \ldots, Z_{n+\mathbf{mlast}+m^*}$ and columns $\mathbf{mlast}+1, \mathbf{mlast}+2, \ldots, \mathbf{mlast}+m^*$ of **sefz** are set to zero.

10 Example

This example shows how to update the forecasts of two series each of length 48. No transformation has been used and no differencing applied to either of the series. nag_tsa_varma_estimate (g13ddc) is first called to fit an AR(1) model to the series. μ is to be estimated and $\phi_1(2,1)$ constrained to be zero. A call to nag_tsa_varma_forecast (g13djc) is then made in order to compute forecasts of the next five series values. After one new observation becomes available the four forecasts are updated. A further observation becomes available and the three forecasts are updated.

10.1 Program Text

```
/* nag_tsa_varma_update (g13dkc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

static void fprint(Integer k, Integer nm, Integer lmax, double predz[],
                   double sefz[], Integer kmax);

int main(void)
{
  /* Scalars */
  double cgetol, rlogl;
  Integer exit_status = 0, i, icm, idmax, idmin, kmax, ip, iprint, iq;
  Integer ishow, j, k, lmax, npar, lref, m, maxcal, mlast, n, nd;
  Integer niter, nm;
  /* 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 QQ(I, J)    qq[(J - 1) * kmax + I - 1]
#define Z(I, J)     z[(J - 1) * kmax + I - 1]

  INIT_FAIL(fail);

  printf("nag_tsa_varma_update (g13dkc) Example Program Results\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

  idmax = 0;
  idmin = 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 (j = 1; j <= k; ++j) {
      for (i = j; i <= k; ++i) {
        QQ(i, j) = 0.0;
      }
    }
    parhld[2] = Nag_TRUE;
    exact = Nag_TRUE;
    /* ** Set iprint > 0 for no intermediate monitoring */
    iprint = -1;
    cgetol = 1e-4;
    maxcal = npar * 40 * (npar + 5);
    /* ** Set ishow = 0 for no results from nag_tsa_varma_estimate (g13ddc) */
    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_G13D_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;
      }

      fprint(k, n, lmax, predz, sefz, kmax);

      m = 1;
      mlast = 0;
      Z(1, 1) = 8.1;
      Z(2, 1) = 10.2;

      /* nag_tsa_varma_update (g13dkc).
       * Multivariate time series, updates forecasts and their
       * standard errors
       */
      nag_tsa_varma_update(k, lmax, m, &mlast, z, k, ref, lref, v, predz,
                           sefz, &fail);
      if (fail.code != NE_NOERROR) {
        printf("\n nag_tsa_varma_update (g13dkc) message: %s\n\n",
               fail.message);
        exit_status = 1;
        goto END;
      }

      nm = n + mlast;
      fprint(k, nm, lmax, predz, sefz, kmax);
      m = 1;

      /* Leave mlast unchanged from last call */
      Z(1, 1) = 8.5;
      Z(2, 1) = 10.;

      /* nag_tsa_varma_update (g13dkc), see above. */
      nag_tsa_varma_update(k, lmax, m, &mlast, z, k, ref, lref, v, predz,
                           sefz, &fail);
      if (fail.code != NE_NOERROR) {
        printf("\n nag_tsa_varma_update (g13dkc) message: %s\n\n",
               fail.message);
        exit_status = 1;
        goto END;
      }

      nm = n + mlast;
      fprint(k, nm, lmax, predz, sefz, kmax);
    }
  }

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;
}

static void fprint(Integer k, Integer nm, Integer lmax, double predz[],
                   double sefz[], Integer kmax)
{
  /* Scalars */
  Integer i2, i, j, l, l2, loop;

#define SEFZ(I, J)  sefz[(J - 1) * kmax + I - 1]
#define PREDZ(I, J) predz[(J - 1) * kmax + I - 1]

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

  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");
    }
  }
}
```

10.2 Program Data

```
nag_tsa_varma_update (g13dkc) 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_update (g13dkc) 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


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

Forecast origin is set at t =   49

Lead Time                      1          2          3          4          5

Series  1: Forecast          8.10       7.49       6.94       6.46       6.06
         : Standard Error    0.00       1.72       2.23       2.51       2.68

Series  2: Forecast         10.20       9.19       8.61       8.28       8.08
         : Standard Error    0.00       2.32       2.68       2.78       2.82


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

Forecast origin is set at t =   50

Lead Time                      1          2          3          4          5

Series  1: Forecast          8.10       8.50       7.80       7.18       6.65
```

```
         : Standard Error      0.00        0.00        1.72        2.23        2.51

Series  2: Forecast           10.20       10.00        9.08        8.54        8.24
         : Standard Error      0.00        0.00        2.32        2.68        2.78
```