NAG Library Function Document

nag_tsa_arma_filter (g13bac)

1 Purpose

nag_tsa_arma_filter (g13bac) filters a time series by an ARIMA model.

2 Specification

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

```
void nag_tsa_arma_filter (const double y[], Integer ny,
     Nag_ArimaOrder *arimaf, Nag_ArimaOrder *arimav, const double par[],
     Integer npar, double cy, double b[], Integer nb, NagError *fail)
```

3 Description

From a given series $y_1, y_2, \ldots, y_n$, a new series $b_1, b_2, \ldots, b_n$ is calculated using a supplied (filtering) ARIMA model. This model will be one which has previously been fitted to a series x_t with residuals a_t. The equations defining b_t in terms of y_t are very similar to those by which a_t is obtained from x_t. The only dissimilarity is that no constant correction is applied after differencing. This is because the series y_t is generally distinct from the series x_t with which the model is associated, though y_t may be related to x_t. Whilst it is appropriate to apply the ARIMA model to y_t so as to preserve the same relationship between b_t and a_t as exists between y_t and x_t, the constant term in the ARIMA model is inappropriate for y_t. The consequence is that b_t will not necessarily have zero mean.

The equations are precisely:

$$w_t = \nabla^d \nabla_s^D y_t, \tag{1}$$

the appropriate differencing of y_t; both the seasonal and non-seasonal inverted autoregressive operations are then applied,

$$u_t = w_t - \Phi_1 w_{t-s} - \cdots - \Phi_P w_{t-s\times P} \tag{2}$$

$$v_t = u_t - \phi_1 u_{t-1} - \cdots - \phi_p u_{t-p} \tag{3}$$

followed by the inverted moving average operations

$$z_t = v_t + \Theta_1 z_{t-s} + \cdots + \Theta_Q z_{t-s\times Q} \tag{4}$$

$$b_t = z_t + \theta_1 b_{t-1} + \cdots + \theta_q b_{t-q}. \tag{5}$$

Because the filtered series value b_t depends on present and past values $y_t, y_{t-1}, \ldots$, there is a problem arising from ignorance of $y_0, y_{-1}, \ldots$ which particularly affects calculation of the early values $b_1, b_2, \ldots$, causing 'transient errors'. The function allows two possibilities.

(i) The equations (1), (2) and (3) are applied from successively later time points so that all terms on their right-hand sides are known, with v_t being defined for $t = (1 + d + s \times D + s \times P), \ldots, n$. Equations (4) and (5) are then applied over the same range, taking any values on the right-hand side associated with previous time points to be zero.

This procedure may still however result in unacceptably large transient errors in early values of b_t.

(ii) The unknown values $y_0, y_{-1}, \ldots$ are estimated by backforecasting. This requires that an ARIMA model distinct from that which has been supplied for filtering, should have been previously fitted to y_t.

For efficiency, you are asked to supply both this ARIMA model for y_t and a limited number of backforecasts which are prefixed to the known values of y_t. Within the function further backforecasts of y_t, and the series w_t, u_t, v_t in (1), (2) and (3) are then easily calculated, and a set of linear equations solved for backforecasts of z_t, b_t for use in (4) and (5) in the case that $q + Q > 0$.

Even if the best model for y_t is not available, a very approximate guess such as

$$y_t = c + e_t$$

or

$$\nabla y_t = e_t$$

can help to reduce the transients substantially.

The backforecasts which need to be prefixed to y_t are of length $Q'_y = q_y + s_y \times Q_y$, where q_y and Q_y are the non-seasonal and seasonal moving average orders and s_y the seasonal period for the ARIMA model of y_t. Thus you need not carry out the backforecasting exercise if $Q'_y = 0$. Otherwise, the series $y_1, y_2, \ldots, y_n$ should be reversed to obtain $y_n, y_{n-1}, \ldots, y_1$ and nag_tsa_multi_inp_model_forecast (g13bjc) should be used to forecast Q'_y values, $\hat{y}_0, \ldots, \hat{y}_{1-Q'_y}$. The ARIMA model used is that fitted to y_t (as a forward series) except that, if $d_y + D_y$ is odd, the constant should be changed in sign (to allow, for example, for the fact that a forward upward trend is a reversed downward trend). The ARIMA model for y_t supplied to the filtering function must however have the appropriate constant for the forward series.

The series $\hat{y}_{1-Q'_y}, \ldots, \hat{y}_0, y_1, \ldots, y_n$ is then supplied to the function, and a corresponding set of values returned for b_t.

4 References

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

5 Arguments

1: **y**[**ny**] – const double *Input*

On entry: the Q'_y backforecasts, starting with backforecast at time $1 - Q'_y$ to backforecast at time 0, followed by the time series starting at time 1, where $Q'_y =$ **arimav**.**q** + **arimav**.**bigq** × **arimav**.**s**. If there are no backforecasts, either because the ARIMA model for the time series is not known, or because it is known but has no moving average terms, then the time series starts at the beginning of **y**.

2: **ny** – Integer *Input*

On entry: the total number of backforecasts and time series data points in array **y**.

Constraint: **ny** $\geq \max\left(1 + Q'_y, \mathbf{npar}\right)$.

3: **arimaf** – Nag_ArimaOrder * *Input*

On entry: the orders for the filtering ARIMA model as a pointer to structure of type Nag_ArimaOrder with the following members:

p – Integer
d – Integer *Input*
q – Integer *Input*
bigp – Integer *Input*
bigd – Integer *Input*
bigq – Integer *Input*
s – Integer *Input*

On entry: these seven members of **arimaf** must specify the orders vector (p, d, q, P, D, Q, s), respectively, of the ARIMA model for the output noise component.

p, q, P and Q refer, respectively, to the number of autoregressive (ϕ), moving average (θ), seasonal autoregressive (Φ) and seasonal moving average (Θ) parameters.

d, D and s refer, respectively, to the order of non-seasonal differencing, the order of seasonal differencing and the seasonal period.

Constraints:

arimaf.**p** ≥ 0;
arimaf.**d** ≥ 0;
arimaf.**q** ≥ 0;
arimaf.**bigp** ≥ 0;
arimaf.**bigd** ≥ 0;
arimaf.**bigq** ≥ 0;
arimaf.**s** ≥ 0;
arimaf.**s** $\neq 1$;
arimaf.**p** $+$ **arimaf**.**q** $+$ **arimaf**.**bigp** $+$ **arimaf**.**bigq** > 0;
if **arimaf**.**s** $= 0$, **arimaf**.**bigp** $+$ **arimaf**.**bigd** $+$ **arimaf**.**bigq** $= 0$;
if **arimaf**.**s** $\neq 0$, **arimaf**.**bigp** $+$ **arimaf**.**bigd** $+$ **arimaf**.**bigq** $\neq 0$.

4: **arimav** – Nag_ArimaOrder * *Input*

On entry: if available, the orders for the ARIMA model for the series as a pointer to structure of type Nag_ArimaOrder with the following members:

p – Integer
d – Integer *Input*
q – Integer *Input*
bigp – Integer *Input*
bigd – Integer *Input*
bigq – Integer *Input*
s – Integer *Input*

On entry: these seven members of **arimav** must specify the orders vector (p, d, q, P, D, Q, s), respectively, of the ARIMA model for the output noise component.

p, q, P and Q refer, respectively, to the number of autoregressive (ϕ), moving average (θ), seasonal autoregressive (Φ) and seasonal moving average (Θ) parameters.

d, D and s refer, respectively, to the order of non-seasonal differencing, the order of seasonal differencing and the seasonal period.

If no ARIMA model for the series is to be supplied **arimav** should be set to a **NULL** pointer.

Constraints:

arimav.**p** ≥ 0;
arimav.**d** ≥ 0;
arimav.**q** ≥ 0;
arimav.**bigp** ≥ 0;
arimav.**bigd** ≥ 0;
arimav.**bigq** ≥ 0;
arimav.**s** ≥ 0;
arimav.**s** $\neq 1$;

> if **arimav**.**s** $= 0$, **arimav**.**bigp** $+$ **arimav**.**bigd** $+$ **arimav**.**bigq** $= 0$;
> if **arimav**.**s** $\neq 0$, **arimav**.**bigp** $+$ **arimav**.**bigd** $+$ **arimav**.**bigq** $\neq 0$.

5: **par**[**npar**] – const double *Input*

On entry: the parameters of the filtering model, followed by the parameters of the ARIMA model for the time series, if supplied. Within each model the parameters are in the standard order of non-seasonal AR and MA followed by seasonal AR and MA.

6: **npar** – Integer *Input*

On entry: the total number of parameters held in array **par**.

Constraints:

> if **arimav** is not **NULL**, **npar** $=$ **arimaf**.**p** $+$ **arimaf**.**q** $+$ **arimaf**.**bigp** $+$ **arimaf**.**bigq**;
> if **arimav** is **NULL**, **npar** $=$ **arimaf**.**p** $+$ **arimaf**.**q** $+$ **arimaf**.**bigp** $+$ **arimaf**.**bigq** $+$ **arimav**.**p** $+$ **arimav**.**q** $+$ **arimav**.**bigp** $+$ **arimav**.**bigq**.

Note: the first constraint (i.e., **arimaf**.**p** $+$ **arimaf**.**q** $+$ **arimaf**.**bigp** $+$ **arimaf**.**bigq** > 0) on the orders of the filtering model, in argument **arimav**, ensures that **npar** > 0.

7: **cy** – double *Input*

On entry: if the ARIMA model is known , **cy** must specify the constant term of the ARIMA model for the time series. If this model is not known , then **cy** is not used.

8: **b**[**nb**] – double *Output*

On exit: the filtered output series. If the ARIMA model for the time series was known, and hence Q_y' backforecasts were supplied in **y**, then **b** contains Q_y' 'filtered' backforecasts followed by the filtered series. Otherwise, the filtered series begins at the start of **b** just as the original series began at the start of **y**. In either case, if the value of the series at time t is held in **y**$[t-1]$, then the filtered value at time t is held in **b**$[t-1]$.

9: **nb** – Integer *Input*

On entry: the dimension of the array **b**. In addition to holding the returned filtered series, **b** is also used as an intermediate work array if the ARIMA model for the time series was known.

Constraints:

> if **arimav** is **NULL**, **nb** $\geq$ **ny** $+ \max(K_3, K_1 + K_2)$;
> if **arimav** is not **NULL**, **nb** $\geq$ **ny**.

Where

> $K_1 =$ **arimaf**.**p** $+$ **arimaf**.**bigp** $\times$ **arimaf**.**s**;
> $K_2 =$ **arimaf**.**d** $+$ **arimaf**.**bigd** $\times$ **arimaf**.**s**;
> $K_3 =$ **arimaf**.**q** $+$ **arimaf**.**bigq** $\times$ **arimaf**.**s**.

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

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_ARRAY_SIZE

The array **b** is too small. Minimum required size: $\langle$*value*$\rangle$.

NE_BAD_PARAM

On entry, argument $\langle$*value*$\rangle$ had an illegal value.

NE_CONSTRAINT

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

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

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

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

On entry, **arimaf** $= \langle$*value*$\rangle$.
Constraint: **arimaf**.**p** $+$ **arimaf**.**q** $+$ **arimaf**.**bigp** $+$ **arimaf**.**bigq** > 0.

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

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

On entry, **arimaf** $= \langle$*value*$\rangle$.
Constraint: **arimaf**.**s** $\neq 1$.

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

On entry, **arimaf** $= \langle$*value*$\rangle$.
Constraint: if **arimaf**.**s** $= 0$, **arimaf**.**bigp** $+$ **arimaf**.**bigd** $+$ **arimaf**.**bigq** $= 0$.

On entry, **arimaf** $= \langle$*value*$\rangle$.
Constraint: if **arimaf**.**s** $\neq 0$, **arimaf**.**bigp** $+$ **arimaf**.**bigd** $+$ **arimaf**.**bigq** $\neq 0$.

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

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

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

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

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

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

On entry, **arimav** $= \langle$*value*$\rangle$.
Constraint: **arimav**.**s** $\neq 1$.

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

On entry, **arimav** $= \langle$*value*$\rangle$.
Constraint: if **arimav**.**s** $= 0$, **arimav**.**bigp** $+$ **arimav**.**bigd** $+$ **arimav**.**bigq** $= 0$.

On entry, **arimav** $= \langle$*value*$\rangle$.
Constraint: if **arimav**.**s** $\neq 0$, **arimav**.**bigp** $+$ **arimav**.**bigd** $+$ **arimav**.**bigq** $\neq 0$.

On entry, **npar** is inconsistent with **arimaf** and **arimav**: **npar** $= \langle$*value*$\rangle$.

NE_INIT_FILTER

The initial values of the filtered series are indeterminate for the given models.

NE_INT

On entry, **ny** is too small to carry out requested filtering: **ny** $= \langle value \rangle$.

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_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_ORDERS_ARIMA

On entry, **arimaf** or **arimav** is invalid.

The orders vector for the ARIMA model is invalid.

NE_ORDERS_FILTER

The orders vector for the filtering model is invalid.

7 Accuracy

Accuracy and stability are high except when the MA parameters are close to the invertibility boundary.

8 Parallelism and Performance

nag_tsa_arma_filter (g13bac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_tsa_arma_filter (g13bac) 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 an ARIMA model is supplied, local workspace arrays of fixed lengths are allocated internally by nag_tsa_arma_filter (g13bac). The total size of these arrays amounts to K Integer elements and $K \times (K + 2)$ double elements, where $K = \mathbf{arimaf.q} + \mathbf{arimaf.bigq} \times \mathbf{arimaf.s} + \mathbf{arimav.p} + \mathbf{arimav.d} + (\mathbf{arimav.bigp} + \mathbf{arimav.bigd}) \times \mathbf{arimav.s}$.

The time taken by nag_tsa_arma_filter (g13bac) is approximately proportional to

$$\mathbf{ny} \times (\mathbf{arimaf.p} + \mathbf{arimaf.q} + \mathbf{arimaf.bigp} + \mathbf{arimaf.bigq}),$$

with an appreciable fixed increase if an ARIMA model is supplied for the time series.

10 Example

This example reads a time series of length 296. It reads the univariate ARIMA $(4,0,2,0,0,0,0)$ model and the ARIMA filtering $(3,0,0,0,0,0,0)$ model for the series. Two initial backforecasts are required and these are calculated by a call to nag_tsa_multi_inp_model_forecast (g13bjc). The backforecasts are inserted at the start of the series and nag_tsa_arma_filter (g13bac) is called to perform the calculations.

10.1 Program Text

```
/* nag_tsa_arma_filter (g13bac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2002.
 * Mark 7b revised, 2004.
 */

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

int main(void)
{
  /* Scalars */
  double          a1, a2, cx, cy;
  Integer         i, idd, ii, ij, iqxd,
                  j, k, n, nb, ni, npar, nparx, nx, ny,
                  nser, npara, tdxxy, tdmrx, ldparx, tdparx;
  Integer         exit_status = 0;

  /* Arrays */
  double          *b = 0, *fsd = 0, *fva = 0, *par = 0, *parx = 0,
  *x = 0, *y = 0, *rms = 0, *parxx = 0;
  Integer         mr[14], mrx[7], *mrxx = 0;
  Nag_ArimaOrder  arimaj, arimaf, arimav;
  Nag_TransfOrder transfj;
  Nag_G13_Opt     options;
  NagError        fail;

  INIT_FAIL(fail);

  exit_status = 0;

  /* Initialise the options structure used by nag_tsa_multi_inp_model_forecast
   * (g13bjc) */
  /* nag_tsa_options_init (g13bxc).
   * Initialization function for option setting
   */
  nag_tsa_options_init(&options);

  printf("nag_tsa_arma_filter (g13bac) Example Program Results\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n] ", &nx);
#else
  scanf("%"NAG_IFMT"%*[^\n] ", &nx);
#endif

  if (nx > 0)
    {
      /* Allocate array x */
      if (!(x = NAG_ALLOC(nx+2, double)))
```

```
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }

      for (i = 1; i <= nx; ++i)
#ifdef _WIN32
        scanf_s("%lf", &x[i-1]);
#else
        scanf("%lf", &x[i-1]);
#endif
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif

      /* Read univariate Arima for series */
      for (i = 1; i <= 7; ++i)
#ifdef _WIN32
        scanf_s("%"NAG_IFMT"", &mrx[i-1]);
#else
        scanf("%"NAG_IFMT"", &mrx[i-1]);
#endif
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif

#ifdef _WIN32
      scanf_s("%lf%*[^\n] ", &cx);
#else
      scanf("%lf%*[^\n] ", &cx);
#endif

      nparx = mrx[0] + mrx[2] + mrx[3] + mrx[5];

      arimaj.p = mrx[0];
      arimaj.d = mrx[1];
      arimaj.q = mrx[2];
      arimaj.bigp = mrx[3];
      arimaj.bigd = mrx[4];
      arimaj.bigq = mrx[5];
      arimaj.s = mrx[6];

      nser = 1;

      if (nparx > 0)
        {
          /* Allocate array parx */
          if (!(parx = NAG_ALLOC(nparx+nser, double)))
            {
              printf("Allocation failure\n");
              exit_status = -1;
              goto END;
            }

          for (i = 1; i <= nparx; ++i)
#ifdef _WIN32
            scanf_s("%lf", &parx[i-1]);
#else
            scanf("%lf", &parx[i-1]);
#endif
#ifdef _WIN32
          scanf_s("%*[^\n] ");
#else
          scanf("%*[^\n] ");
#endif
```

```
          /* Read model by which to filter series */
          for (i = 1; i <= 7; ++i)
#ifdef _WIN32
            scanf_s("%"NAG_IFMT"", &mr[i-1]);
#else
            scanf("%"NAG_IFMT"", &mr[i-1]);
#endif
#ifdef _WIN32
          scanf_s("%*[^\n] ");
#else
          scanf("%*[^\n] ");
#endif

          arimaf.p = mr[0];
          arimaf.d = mr[1];
          arimaf.q = mr[2];
          arimaf.bigp = mr[3];
          arimaf.bigd = mr[4];
          arimaf.bigq = mr[5];
          arimaf.s = mr[6];

          npar = mr[0] + mr[2] + mr[3] + mr[5];
          if (npar > 0)
            {
              /* Allocate array par */
              if (!(par = NAG_ALLOC(npar + nparx, double)))
                {
                  printf("Allocation failure\n");
                  exit_status = -1;
                  goto END;
                }
              for (i = 1; i <= npar; ++i)
#ifdef _WIN32
                scanf_s("%lf", &par[i-1]);
#else
                scanf("%lf", &par[i-1]);
#endif
#ifdef _WIN32
              scanf_s("%*[^\n] ");
#else
              scanf("%*[^\n] ");
#endif

              /* Initially backforecast QY values */
              /* (1) Reverse series in situ */
              n = nx / 2;
              ni = nx;
              for (i = 1; i <= n; ++i)
                {
                  a1 = x[i-1];
                  a2 = x[ni-1];
                  x[i-1] = a2;
                  x[ni-1] = a1;
                  --ni;
                }
              idd = mrx[1] + mrx[4];
              /* (2) Possible sign reversal for ARIMA constant */
              if (idd % 2 != 0)
                cx = -cx;

              /* (3) Calculate number of backforecasts required */
              iqxd = mrx[2] + mrx[5] * mrx[6];

              /* Calculate series length */
              ny = nx + iqxd;

              /* Allocate array y */
              if (!(y = NAG_ALLOC(ny, double)))
                {
                  printf("Allocation failure\n");
                  exit_status = -1;
```

```
      goto END;
    }

  if (iqxd != 0)
    {
      /* Allocate arrays fsd, fva and st. */
      if (!(fsd = NAG_ALLOC(iqxd, double)) ||
          !(fva = NAG_ALLOC(iqxd, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      /* (4) Set up parameter list for call to forecast
       * routine g13bjc
       */
      npara = nparx+nser;
      parx[npara-1] = cx;
      tdxxy = nser;
      tdmrx = nser-1;
      ldparx = nser-1;
      tdparx = nser-1;
      if (!(rms = NAG_ALLOC(nser, double)) ||
          !(parxx = NAG_ALLOC(nser, double)) ||
          !(mrxx = NAG_ALLOC(7*nser, Integer)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }

      /* nag_tsa_transf_orders (g13byc).
       * Allocates memory to transfer function model orders
       */
      nag_tsa_transf_orders(nser, &transfj, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_tsa_transf_orders (g13byc)"
            ".\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }

      rms[0] = 0;
      transfj.nag_b = 0;
      transfj.nag_q = 0;
      transfj.nag_p = 0;
      transfj.nag_r = 1;
      for (i = 1; i <= 7; ++i)
        mrxx[i-1] = 0;
      parxx[0] = 0;

      /* Tell nag_tsa_multi_inp_model_forecast (g13bjc) not to
       * print parameters on entry */
      options.list = Nag_FALSE;

      /* nag_tsa_multi_inp_model_forecast (g13bjc).
       * Forecasting function
       */
      nag_tsa_multi_inp_model_forecast(&arimaj, nser, &transfj,
                                       parx, npara, nx, iqxd, x,
                                       tdxxy, rms, mrxx, tdmrx,
                                       parxx, ldparx, tdparx,
                                       fva, fsd, &options, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf(
                 "Error from nag_tsa_multi_inp_model_forecast "
                 "(g13bjc).\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }

      j = iqxd;
      for (i = 1; i <= iqxd; ++i)
        {
          y[i-1] = fva[j-1];
          --j;
        }

      /* Move series into y */
      j = iqxd + 1;
      k = nx;
      for (i = 1; i <= nx; ++i)
        {
          if (j > 305)
            goto END;
          y[j-1] = x[k-1];
          ++j;
          --k;
        }
    }

  /* Move ARIMA for series into mr */
  for (i = 1; i <= 7; ++i)
    mr[i+6] = mrx[i-1];

  arimav.p = mr[7];
  arimav.d = mr[8];
  arimav.q = mr[9];
  arimav.bigp = mr[10];
  arimav.bigd = mr[11];
  arimav.bigq = mr[12];
  arimav.s = mr[13];

  /* Move parameters of ARIMA for y into par */
  for (i = 1; i <= nparx; ++i)
    par[npar+i-1] = parx[i-1];
  npar += nparx;

  /* Move constant and reset sign reversal */
  cy = cx;
  if (idd % 2 != 0)
    cy = -cy;

  /* Set parameters for call to filter routine
   * nag_tsa_arma_filter (g13bac) */
  nb = ny + MAX(mr[2] + mr[5] * mr[6],
                mr[0] + mr[1] + (mr[3] + mr[4]) * mr[6]);

  /* Allocate array b */
  if (!(b = NAG_ALLOC(nb, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Filter series by call to nag_tsa_arma_filter (g13bac) */
  /* nag_tsa_arma_filter (g13bac).
   * Multivariate time series, filtering (pre-whitening) by an
   * ARIMA model
   */
  nag_tsa_arma_filter(y, ny, &arimaf, &arimav, par, npar, cy, b,
                      nb, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf(
              "Error from nag_tsa_arma_filter (g13bac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
```

```
                }

              printf("\n");
              printf("                Original        Filtered\n");
              printf("Backforecasts    y-series         series\n");
              if (iqxd != 0)
                {
                  ij = -iqxd;
                  for (i = 1; i <= iqxd; ++i)
                    {
                      printf("%8"NAG_IFMT"%17.4f%15.4f\n", ij, y[i-1],
                              b[i-1]);
                      ++ij;
                    }
                }

              printf("\n");
              printf("       Filtered        Filtered        "
                      "Filtered        Filtered\n");
              printf("        series          series  "
                      "        series           series\n");
              for (i = iqxd + 1; i <= ny; i += 4)
                {
                  for (ii = i; ii <= MIN(ny, i+3); ++ii)
                    {
                      printf("%5"NAG_IFMT"", ii-iqxd);
                      printf("%9.4f  ", b[ii-1]);
                    }
                  printf("\n");
                }
            }
        }
    }

 END:

  /* Free the options structure used by nag_tsa_multi_inp_model_forecast
   * (g13bjc) */
  /* nag_tsa_free (g13xzc).
   * Freeing function for use with g13 option setting
   */
  nag_tsa_free(&options);

  NAG_FREE(b);
  NAG_FREE(fsd);
  NAG_FREE(fva);
  NAG_FREE(par);
  NAG_FREE(parx);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(rms);
  NAG_FREE(parxx);
  NAG_FREE(mrxx);

  return exit_status;
}
```

10.2 Program Data

```
nag_tsa_arma_filter (g13bac) Example Program Data
   296
 53.8 53.6 53.5 53.5 53.4 53.1 52.7 52.4 52.2 52.0 52.0 52.4 53.0 54.0 54.9 56.0
 56.8 56.8 56.4 55.7 55.0 54.3 53.2 52.3 51.6 51.2 50.8 50.5 50.0 49.2 48.4 47.9
 47.6 47.5 47.5 47.6 48.1 49.0 50.0 51.1 51.8 51.9 51.7 51.2 50.0 48.3 47.0 45.8
 45.6 46.0 46.9 47.8 48.2 48.3 47.9 47.2 47.2 48.1 49.4 50.6 51.5 51.6 51.2 50.5
 50.1 49.8 49.6 49.4 49.3 49.2 49.3 49.7 50.3 51.3 52.8 54.4 56.0 56.9 57.5 57.3
 56.6 56.0 55.4 55.4 56.4 57.2 58.0 58.4 58.4 58.1 57.7 57.0 56.0 54.7 53.2 52.1
 51.6 51.0 50.5 50.4 51.0 51.8 52.4 53.0 53.4 53.6 53.7 53.8 53.8 53.8 53.3 53.0
 52.9 53.4 54.6 56.4 58.0 59.4 60.2 60.0 59.4 58.4 57.6 56.9 56.4 56.0 55.7 55.3
 55.0 54.4 53.7 52.8 51.6 50.6 49.4 48.8 48.5 48.7 49.2 49.8 50.4 50.7 50.9 50.7
 50.5 50.4 50.2 50.4 51.2 52.3 53.2 53.9 54.1 54.0 53.6 53.2 53.0 52.8 52.3 51.9
 51.6 51.6 51.4 51.2 50.7 50.0 49.4 49.3 49.7 50.6 51.8 53.0 54.0 55.3 55.9 55.9
 54.6 53.5 52.4 52.1 52.3 53.0 53.8 54.6 55.4 55.9 55.9 55.2 54.4 53.7 53.6 53.6
 53.2 52.5 52.0 51.4 51.0 50.9 52.4 53.5 55.6 58.0 59.5 60.0 60.4 60.5 60.2 59.7
 59.0 57.6 56.4 55.2 54.5 54.1 54.1 54.4 55.5 56.2 57.0 57.3 57.4 57.0 56.4 55.9
 55.5 55.3 55.2 55.4 56.0 56.5 57.1 57.3 56.8 55.6 55.0 54.1 54.3 55.3 56.4 57.2
 57.8 58.3 58.6 58.8 58.8 58.6 58.0 57.4 57.0 56.4 56.3 56.4 56.4 56.0 55.2 54.0
 53.0 52.0 51.6 51.6 51.1 50.4 50.0 50.0 52.0 54.0 55.1 54.5 52.8 51.4 50.8 51.2
 52.0 52.8 53.8 54.5 54.9 54.9 54.8 54.4 53.7 53.3 52.8 52.6 52.6 53.0 54.3 56.0
 57.0 58.0 58.6 58.5 58.3 57.8 57.3 57.0
     4     0     2     0     0     0     0
     0.000
     2.420    -2.380     1.160    -0.230     0.310    -0.470
     3     0     0     0     0     0     0
     1.970    -1.370     0.340
```

10.3 Program Results

```
nag_tsa_arma_filter (g13bac) Example Program Results

                Original        Filtered
Backforecasts   y-series         series
      -2         49.9807         3.4222
      -1         52.6714         3.0809

       Filtered         Filtered         Filtered         Filtered
        series           series           series           series
     1   2.9813     2   2.7803     3   3.7057     4   3.2450
     5   3.0760     6   3.0070     7   3.0610     8   3.1720
     9   3.1170    10   3.0360    11   3.2580    12   3.4520
    13   3.3320    14   3.6980    15   3.3140    16   3.8070
    17   3.3330    18   2.9580    19   3.2800    20   3.0960
    21   3.2270    22   3.0830    23   2.6410    24   3.1870
    25   2.9910    26   3.1110    27   2.8460    28   3.0240
    29   2.7030    30   2.6130    31   2.8060    32   2.9560
    33   2.8170    34   2.8950    35   2.8510    36   2.9160
    37   3.2530    38   3.3050    39   3.1830    40   3.3760
    41   2.9730    42   2.8610    43   3.0490    44   2.8420
    45   2.3190    46   2.3660    47   2.9410    48   2.3810
    49   3.3420    50   2.9340    51   3.1800    52   2.9230
    53   2.6470    54   2.8860    55   2.5310    56   2.6200
    57   3.4170    58   3.4940    59   3.2590    60   3.1310
    61   3.1420    62   2.6710    63   2.8990    64   2.8180
    65   3.2150    66   2.8800    67   2.9610    68   2.8800
    69   3.0020    70   2.8930    71   3.1210    72   3.2210
    73   3.2040    74   3.5360    75   3.7520    76   3.5630
    77   3.7260    78   3.1560    79   3.6310    80   2.9380
    81   3.1480    82   3.4490    83   3.1400    84   3.7380
    85   4.1200    86   3.1540    87   3.7480    88   3.3280
    89   3.3640    90   3.3400    91   3.3950    92   3.0720
    93   3.0050    94   2.8520    95   2.7810    96   3.1950
    97   3.2490    98   2.6370    99   3.0080   100   3.2410
   101   3.5570   102   3.2080   103   3.0880   104   3.3980
   105   3.1660   106   3.1960   107   3.2460   108   3.2870
   109   3.1590   110   3.2620   111   2.7280   112   3.4130
   113   3.2190   114   3.6750   115   3.8550   116   4.0100
   117   3.5380   118   3.8440   119   3.4660   120   3.0640
   121   3.4780   122   3.1140   123   3.5300   124   3.2400
   125   3.3630   126   3.2610   127   3.3020   128   3.1150
   129   3.3280   130   2.8730   131   3.0800   132   2.8390
   133   2.6570   134   3.0260   135   2.4580   136   3.2600
   137   2.8380   138   3.2150   139   3.1140   140   3.1050
   141   3.1400   142   2.9100   143   3.1370   144   2.7500
   145   3.1160   146   3.0680   147   2.8590   148   3.3840
   149   3.5500   150   3.4160   151   3.1770   152   3.3390
   153   3.0190   154   3.1780   155   3.0110   156   3.1940
   157   3.2680   158   3.0500   159   2.8060   160   3.1850
   161   3.0560   162   3.2690   163   2.7940   164   3.0900
   165   2.7100   166   2.7890   167   2.9510   168   3.2440
   169   3.2570   170   3.4360   171   3.4450   172   3.3780
```

```
  173    3.3520     174    3.9180     175    2.9190     176    3.1780
  177    2.2580     178    3.5150     179    2.8010     180    3.6030
  181    3.2610     182    3.5300     183    3.3270     184    3.4420
  185    3.5240     186    3.2720     187    3.1110     188    2.8240
  189    3.2330     190    3.1500     191    3.5710     192    3.0810
  193    2.7820     194    2.9040     195    3.2350     196    2.7970
  197    3.1320     198    3.1680     199    4.5210     200    2.6650
  201    4.6870     202    3.9470     203    3.2220     204    3.3410
  205    3.9950     206    3.4820     207    3.3630     208    3.4550
  209    3.2950     210    2.6910     211    3.4600     212    2.9440
  213    3.4400     214    3.1830     215    3.4200     216    3.4100
  217    4.0550     218    2.9990     219    3.8250     220    3.1340
  221    3.5010     222    3.0430     223    3.2660     224    3.3660
  225    3.2650     226    3.3720     227    3.2880     228    3.5470
  229    3.6840     230    3.3100     231    3.6790     232    3.1780
  233    2.9360     234    2.7910     235    3.8020     236    2.6100
  237    4.1690     238    3.7460     239    3.4560     240    3.3910
  241    3.5820     242    3.6220     243    3.4870     244    3.5770
  245    3.4240     246    3.3960     247    3.1220     248    3.4300
  249    3.4580     250    3.0280     251    3.7660     252    3.3770
  253    3.2470     254    3.0180     255    2.9720     256    2.8000
  257    3.2040     258    2.8020     259    3.4100     260    3.1680
  261    2.4600     262    2.8810     263    3.1750     264    3.1740
  265    4.8640     266    3.0600     267    2.9600     268    2.2530
  269    2.5620     270    3.3150     271    3.3480     272    3.5900
  273    3.2560     274    3.2320     275    3.6160     276    3.1700
  277    3.2890     278    3.1200     279    3.3300     280    2.9910
  281    2.9420     282    3.4070     283    2.8720     284    3.3470
  285    3.1920     286    3.4880     287    4.0680     288    3.7550
  289    3.0510     290    3.9680     291    3.3900     292    3.1380
  293    3.6170     294    3.1700     295    3.4150     296    3.4830
```