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

```c
#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:

\[
\begin{align*}
  w_t &= \nabla^d \nabla_s^D y_t, \\
  u_t &= w_t - \phi_1 u_{t-s} - \cdots - \phi_p w_{t-s\times P} \\
  v_t &= u_t - \phi_1 u_{t-1} - \cdots - \phi_p u_{t-p} \\
  z_t &= v_t + \Theta_1 z_{t-s} + \cdots + \Theta_q z_{t-s\times Q} \\
  b_t &= z_t + \theta_1 b_{t-1} + \cdots + \theta_q b_{t-q}.
\end{align*}
\]

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.

Mark 25
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

\[
y_t = c + \varepsilon_t
\]

or

\[
\nabla y_t = \varepsilon_t
\]

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


5 Arguments

1: \( \text{y}[^{\text{ny}}] \) – const double
   \[ \text{Input} \]
   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 \( Q_y' = \text{arimaf}.q + \text{arimaf}.bigq + \text{arimaf}.s \). 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: \( \text{ny} \) – Integer
   \[ \text{Input} \]
   On entry: the total number of backforecasts and time series data points in array \( y \).
   Constraint: \( \text{ny} \geq \max\left(1 + Q_y', \text{npar}\right) \).

3: \( \text{arimaf} \) – Nag_ArimaOrder *
   \[ \text{Input} \]
   On entry: the orders for the filtering ARIMA model as a pointer to structure of type Nag_ArimaOrder with the following members:
On entry: these seven members of \text{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 \((\phi)\), moving average \((\theta)\), seasonal autoregressive \((\Phi)\) and seasonal moving average \((\Theta)\) parameters.

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

**Constraints:**
\[
\begin{align*}
\text{arimaf.p} & \geq 0; \\
\text{arimaf.d} & \geq 0; \\
\text{arimaf.q} & \geq 0; \\
\text{arimaf.bigp} & \geq 0; \\
\text{arimaf.bigd} & \geq 0; \\
\text{arimaf.bigq} & \geq 0; \\
\text{arimaf.s} & \geq 0; \\
\text{arimaf.s} & \neq 1; \\
\text{arimaf.p + arimaf.q + arimaf.bigp + arimaf.bigd + arimaf.bigq} & > 0; \\
\text{if arimaf.s = 0, arimaf.bigp + arimaf.bigq = 0;} \\
\text{if arimaf.s \neq 0, arimaf.bigp + arimaf.bigq \neq 0.}
\end{align*}
\]

4: \text{arimav} – Nag_ArimaOrder * 

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:

\[
\begin{align*}
p & \quad \text{Integer} \\
d & \quad \text{Integer} \\
q & \quad \text{Integer} \\
\text{bigp} & \quad \text{Integer} \\
\text{bigd} & \quad \text{Integer} \\
\text{bigq} & \quad \text{Integer} \\
s & \quad \text{Integer}
\end{align*}
\]

On entry: these seven members of \text{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 \((\phi)\), moving average \((\theta)\), seasonal autoregressive \((\Phi)\) and seasonal moving average \((\Theta)\) 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 \text{arimav} should be set to a NULL pointer.

**Constraints:**
\[
\begin{align*}
\text{arimav.p} & \geq 0; \\
\text{arimav.d} & \geq 0; \\
\text{arimav.q} & \geq 0; \\
\text{arimav.bigp} & \geq 0; \\
\text{arimav.bigd} & \geq 0; \\
\text{arimav.bigq} & \geq 0; \\
\text{arimav.s} & \geq 0; \\
\text{arimav.s} & \neq 1;
\end{align*}
\]
if \( \text{arimav.s} = 0 \), \( \text{arimav.bigp} + \text{arimav.bigd} + \text{arimav.bigq} = 0 \);
if \( \text{arimav.s} \neq 0 \), \( \text{arimav.bigp} + \text{arimav.bigd} + \text{arimav.bigq} \neq 0 \).

5: \( \text{par}[\text{npar}] \) – 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: \( \text{npar} \) – Integer

*Input*

On entry: the total number of parameters held in array \( \text{par} \).

Constraints:

if \( \text{arimav} \) is not NULL, \( \text{npar} = \text{arimaf.p} + \text{arimaf.q} + \text{arimaf.bigp} + \text{arimaf.bigq} \);
if \( \text{arimav} \) is NULL, \( \text{npar} = \text{arimaf.p} + \text{arimaf.q} + \text{arimaf.bigp} + \text{arimaf.bigq} + \text{arimav.p} + \text{arimav.q} + \text{arimav.bigp} + \text{arimav.bigq} \).

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

7: \( \text{cy} \) – double

*Input*

On entry: if the ARIMA model is known, \( \text{cy} \) must specify the constant term of the ARIMA model for the time series. If this model is not known, then \( \text{cy} \) is not used.

8: \( \text{b}[\text{nb}] \) – 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: \( \text{nb} \) – 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 \( \text{arimav} \) is NULL, \( \text{nb} \geq ny + \max(K_3, K_1 + K_2) \);
if \( \text{arimav} \) is not NULL, \( \text{nb} \geq ny \).

Where

\[
K_1 = \text{arimaf.p} + \text{arimaf.bigp} \times \text{arimaf.s} ; \\
K_2 = \text{arimaf.d} + \text{arimaf.bigd} \times \text{arimaf.s} ; \\
K_3 = \text{arimaf.q} + \text{arimaf.bigq} \times \text{arimaf.s} .
\]

10: \( \text{fail} \) – 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: \( \langle \text{value} \rangle \).
NE_BAD_PARAM

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

NE_CONSTRAINT

On entry, \texttt{arimaf} = \langle value \rangle.
Constraint: \texttt{arimaf.bigd} \geq 0.

On entry, \texttt{arimaf} = \langle value \rangle.
Constraint: \texttt{arimaf.bigp} \geq 0.

On entry, \texttt{arimaf} = \langle value \rangle.
Constraint: \texttt{arimaf.bigq} \geq 0.

On entry, \texttt{arimaf} = \langle value \rangle.
Constraint: \texttt{arimaf.d} \geq 0.

On entry, \texttt{arimaf} = \langle value \rangle.
Constraint: \texttt{arimaf.p + arimaf.q + arimaf.bigp + arimaf.bigq} > 0.

On entry, \texttt{arimaf} = \langle value \rangle.
Constraint: \texttt{arimaf.p} \geq 0.

On entry, \texttt{arimaf} = \langle value \rangle.
Constraint: \texttt{arimaf.q} \geq 0.

On entry, \texttt{arimaf} = \langle value \rangle.
Constraint: \texttt{arimaf.s} \neq 1.

On entry, \texttt{arimaf} = \langle value \rangle.
Constraint: \texttt{arimaf.s} \geq 0.

On entry, \texttt{arimaf} = \langle value \rangle.
Constraint: if \texttt{arimaf.s} = 0, \texttt{arimaf.bigp + arimaf.bigd + arimaf.bigq} = 0.

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

On entry, \texttt{arimav} = \langle value \rangle.
Constraint: \texttt{arimav.bigd} \geq 0.

On entry, \texttt{arimav} = \langle value \rangle.
Constraint: \texttt{arimav.bigp} \geq 0.

On entry, \texttt{arimav} = \langle value \rangle.
Constraint: \texttt{arimav.bigq} \geq 0.

On entry, \texttt{arimav} = \langle value \rangle.
Constraint: \texttt{arimav.d} \geq 0.

On entry, \texttt{arimav} = \langle value \rangle.
Constraint: \texttt{arimav.p} \geq 0.

On entry, \texttt{arimav} = \langle value \rangle.
Constraint: \texttt{arimav.q} \geq 0.

On entry, \texttt{arimav} = \langle value \rangle.
Constraint: \texttt{arimav.s} \neq 1.

On entry, \texttt{arimav} = \langle value \rangle.
Constraint: \texttt{arimav.s} \geq 0.

On entry, \texttt{arimav} = \langle value \rangle.
Constraint: if \texttt{arimav.s} = 0, \texttt{arimav.bigp + arimav.bigd + arimav.bigq} = 0.

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

On entry, \texttt{npar} is inconsistent with \texttt{arimaf} and \texttt{arimav}: \texttt{npar} = \langle value \rangle.
The initial values of the filtered series are indeterminate for the given models.

On entry, ny is too small to carry out requested filtering: ny = ⟨value⟩.

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.

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.

On entry, arimaf or arimav is invalid.
The orders vector for the ARIMA model is invalid.

The orders vector for the filtering model is invalid.

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

nag_tsa_arma_filter (g13bac) is threaded by NAG for parallel execution in multithreaded implementa-
tions 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.

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 × (K + 2) double elements, where

K = arimaf.q + arimaf.biqq × arimaf.s + arimav.p +
arimav.d + (arimav.bigp + arimav.bigd) × arimav.s.

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

ny × (arimaf.p + arimaf.q + arimaf.bigp + 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 (g13bje). The backforecasts are inserted at the start of the series and nag_tsa_arima_filter (g13bac) is called to perform the calculations.

10.1 Program Text

/* nag_tsa_arima_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
   * (g13bje) */
  /* nag_tsa_options_init (g13bxc).
   * Initialization function for option setting
   */
  nag_tsa_options_init(&options);

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

  /* Skip heading in data file */
  #ifdef _WIN32
    scanf_s("%*[\n] ");
  #else
    scanf("%*[\n] ");
  #endif
  #ifdef _WIN32
    scanf("%"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)
#else _WIN32
    scanf("%lf", &x[i-1]);
#else
    scanf("%lf", &x[i-1]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#else _WIN32
    scanf_s("%lf%*[\n ] ", &cx);
#else
    scanf("%lf%*[\n ] ", &cx);
#endif


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)
#else _WIN32
        scanf_s("%lf", &parx[i-1]);
#else
        scanf("%lf", &parx[i-1]);
#else _WIN32
        scanf("%*[\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];
if (npar > 0)
{
  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];
  if (npar > 0)
  {
    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];
    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 */
      /* Calculate series length */
      ny = nx + iqxd;
      /* Allocate array y */
      if (!(y = NAG_ALLOC(ny, double)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
      }
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 gl3bjc */
    npara = nparx+nser;
    parx[npara-1] = cx;
    tdxxy = nser;
    tdmrx = nser-1;
    ldparx = nser-1;
    tdpaxr = nser-1;
    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 (gl3bjc) not to
     * print parameters on entry */
    options.list = Nag_FALSE;

    /* nag_tsa_multi_inp_model_forecast (gl3bjc).
     * Forecasting function */
    nag_tsa_multi_inp_model_forecast(&arimaj, nser, &transfj, parx, npara, nx, iqxd, x,
                                     tdxxy, rms, mrxx, tdmrx,
                                     parxx, ldparx, tdpaxr, fva, fsd, &options, &fail);
    if (fail.code != NE_NOERROR)
    { printf("Error from nag_tsa_multi_inp_model_forecast "
             "(gl3bjc)\n", fail.message);
      exit_status = 1;
      goto END;
    }
}

/* nag_tsa_transf_orders (gl3byc).
 * 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 (gl3byc)" 
        ".\n\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;
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_arima_filter (g13bac) */

/* 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_arima_filter (g13bac) */
    nag_tsa_arima_filter(y, ny, &arimaf, &arimav, par, npar, cy, b, nb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_tsa_arima_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\n");
printf(" Filtered Filtered 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 (g13zxc). */
/* 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

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 50.8 50.0 49.2 48.3 47.0 46.3 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
54.1 53.2 50.2 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 54.4 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 55.6 56.0 56.5 57.1 57.3 56.8 55.6 55.0 54.1 53.5 53.6
57.8 58.3 58.6 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 51.1 50.4 50.0 50.0 50.2 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 54.0
57.0 58.0 58.6 58.5 58.3 58.2 57.0
4020000
0.000
-0.230 0.310 -0.470
3000000
1.970 -1.370 0.340

10.3 Program Results

nag_tsa arma_filter (g13bac) Example Program Results

<table>
<thead>
<tr>
<th>Filtered</th>
<th>Original</th>
<th>Filtered</th>
<th>Backforecasts</th>
<th>Filtered</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>49.9807</td>
<td>3.4222</td>
<td>3.4222</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>52.6714</td>
<td>3.0809</td>
<td>3.0809</td>
<td></td>
</tr>
</tbody>
</table>

Mark 25
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>173</td>
<td>3.3520</td>
<td>174</td>
<td>3.9180</td>
<td>175</td>
<td>2.9190</td>
</tr>
<tr>
<td>177</td>
<td>2.2580</td>
<td>178</td>
<td>3.5150</td>
<td>179</td>
<td>2.8010</td>
</tr>
<tr>
<td>181</td>
<td>3.2610</td>
<td>182</td>
<td>3.5300</td>
<td>183</td>
<td>3.3270</td>
</tr>
<tr>
<td>185</td>
<td>3.5240</td>
<td>186</td>
<td>3.2720</td>
<td>187</td>
<td>3.1110</td>
</tr>
<tr>
<td>189</td>
<td>3.2330</td>
<td>190</td>
<td>3.1500</td>
<td>191</td>
<td>3.5710</td>
</tr>
<tr>
<td>193</td>
<td>2.7820</td>
<td>194</td>
<td>2.9040</td>
<td>195</td>
<td>3.2350</td>
</tr>
<tr>
<td>197</td>
<td>3.1320</td>
<td>198</td>
<td>3.1680</td>
<td>199</td>
<td>4.5210</td>
</tr>
<tr>
<td>201</td>
<td>4.6870</td>
<td>202</td>
<td>3.9470</td>
<td>203</td>
<td>3.2220</td>
</tr>
<tr>
<td>205</td>
<td>3.9950</td>
<td>206</td>
<td>3.4820</td>
<td>207</td>
<td>3.3630</td>
</tr>
<tr>
<td>209</td>
<td>3.2950</td>
<td>210</td>
<td>2.6910</td>
<td>211</td>
<td>3.4600</td>
</tr>
<tr>
<td>213</td>
<td>3.4400</td>
<td>214</td>
<td>3.1830</td>
<td>215</td>
<td>3.4200</td>
</tr>
<tr>
<td>217</td>
<td>4.0550</td>
<td>218</td>
<td>2.9990</td>
<td>219</td>
<td>3.8250</td>
</tr>
<tr>
<td>221</td>
<td>3.5010</td>
<td>222</td>
<td>3.0430</td>
<td>223</td>
<td>3.2660</td>
</tr>
<tr>
<td>225</td>
<td>3.2650</td>
<td>226</td>
<td>3.3720</td>
<td>227</td>
<td>3.2880</td>
</tr>
<tr>
<td>229</td>
<td>3.6840</td>
<td>230</td>
<td>3.3100</td>
<td>231</td>
<td>3.6790</td>
</tr>
<tr>
<td>233</td>
<td>2.9360</td>
<td>234</td>
<td>2.7910</td>
<td>235</td>
<td>3.8020</td>
</tr>
<tr>
<td>237</td>
<td>4.1690</td>
<td>238</td>
<td>3.7460</td>
<td>239</td>
<td>3.4560</td>
</tr>
<tr>
<td>241</td>
<td>3.5820</td>
<td>242</td>
<td>3.6220</td>
<td>243</td>
<td>3.4870</td>
</tr>
<tr>
<td>245</td>
<td>3.4240</td>
<td>246</td>
<td>3.3960</td>
<td>247</td>
<td>3.1220</td>
</tr>
<tr>
<td>249</td>
<td>3.4580</td>
<td>250</td>
<td>3.0280</td>
<td>251</td>
<td>3.7660</td>
</tr>
<tr>
<td>253</td>
<td>3.2470</td>
<td>254</td>
<td>3.0180</td>
<td>255</td>
<td>2.9720</td>
</tr>
<tr>
<td>257</td>
<td>3.2040</td>
<td>258</td>
<td>2.8020</td>
<td>259</td>
<td>3.4100</td>
</tr>
<tr>
<td>261</td>
<td>2.4600</td>
<td>262</td>
<td>2.8810</td>
<td>263</td>
<td>3.1750</td>
</tr>
<tr>
<td>265</td>
<td>4.8640</td>
<td>266</td>
<td>3.0600</td>
<td>267</td>
<td>2.9600</td>
</tr>
<tr>
<td>269</td>
<td>2.5620</td>
<td>270</td>
<td>3.3150</td>
<td>271</td>
<td>3.3480</td>
</tr>
<tr>
<td>273</td>
<td>3.2560</td>
<td>274</td>
<td>3.2320</td>
<td>275</td>
<td>3.6160</td>
</tr>
<tr>
<td>277</td>
<td>3.2890</td>
<td>278</td>
<td>3.1200</td>
<td>279</td>
<td>3.3300</td>
</tr>
<tr>
<td>281</td>
<td>2.9420</td>
<td>282</td>
<td>3.4070</td>
<td>283</td>
<td>2.8720</td>
</tr>
<tr>
<td>285</td>
<td>3.1920</td>
<td>286</td>
<td>3.4880</td>
<td>287</td>
<td>4.0680</td>
</tr>
<tr>
<td>289</td>
<td>3.0510</td>
<td>290</td>
<td>3.9680</td>
<td>291</td>
<td>3.3900</td>
</tr>
<tr>
<td>293</td>
<td>3.6170</td>
<td>294</td>
<td>3.1700</td>
<td>295</td>
<td>3.4150</td>
</tr>
</tbody>
</table>