NAG Library Function Document

nag_tsa_inhom_iema_all (g13mfc)

1 Purpose

nag_tsa_inhom_iema_all (g13mfc) calculates the iterated exponential moving average for an inhomogeneous time series, returning the intermediate results.

2 Specification

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

void nag_tsa_inhom_iema_all (Nag_OrderType order, Integer nb,
    const double *z, double iema[], Integer pdiema, const double *t,
    double tau, Integer m1, Integer m2, const double *sinit,
    const Nag_TS_Interpolation inter[], Nag_TS_Transform ftype, double *p,
    const double *x, Integer *pn, double rcomm[], NagError *fail)
```

3 Description

nag_tsa_inhom_iema_all (g13mfc) calculates the iterated exponential moving average for an inhomogeneous time series. The time series is represented by two vectors of length \(n\): a vector of times, \(t\); and a vector of values, \(z\). Each element of the time series is therefore composed of the pair of scalar values \((t_i, z_i)\), for \(i = 1, 2, \ldots, n\). Time can be measured in any arbitrary units, as long as all elements of \(t\) use the same units.

The exponential moving average (EMA), with parameter \(\tau\), is an average operator, with the exponentially decaying kernel given by

\[
e^{-t_i/\tau}
\]

The exponential form of this kernel gives rise to the following iterative formula (Zumbach and Müller (2001)) for the EMA operator:

\[
EMA[\tau; y](t) = \mu EMA[\tau; y](t-1) + (\nu - \mu)y_{i-1} + (1 - \nu)y_i
\]

where

\[
\mu = e^{-\alpha} \quad \text{and} \quad \alpha = \frac{t_i - t_{i-1}}{\tau}.
\]

The value of \(\nu\) depends on the method of interpolation chosen and the relationship between \(y\) and the input series \(z\) depends on the transformation function chosen. nag_tsa_inhom_iema_all (g13mfc) gives the option of three interpolation methods:

1. Previous point: \(\nu = 1\);
2. Linear: \(\nu = (1 - \mu)/\alpha\);
3. Next point: \(\nu = \mu\).

and three transformation functions:

1. Identity: \(y_i = z_i^{[p]}\);
2. Absolute value: \(y_i = |z_i|^{[p]}\);
3. Absolute difference: \(y_i = |z_i - x_i|^{[p]}\);
where the notation \( \lfloor p \rfloor \) is used to denote the integer nearest to \( p \). In the case of the absolute difference \( x \) is a user-supplied vector of length \( n \) and therefore each element of the time series is composed of the triplet of scalar values, \((t_i, z_i, x_i)\).

The \( m \)-iterated exponential moving average, \( \text{EMA}[\tau, m; y](t_i) \), is defined using the recursive formula:

\[
\text{EMA}[\tau, m; y](t_i) = \text{EMA}[\tau; \text{EMA}[\tau, m-1; y](t_i)](t_i)
\]

with

\[
\text{EMA}[\tau, 1; y](t_i) = \text{EMA}[\tau; y](t_i).
\]

For large datasets or where all the data is not available at the same time, \( z, t \) and, where required, \( x \) can be split into arbitrary sized blocks and \text{nag_tsa_inhom_iema_all} (g13mfc) called multiple times.

4 References


5 Arguments

1: \textbf{order} – Nag_OrderType \hspace{1cm} \textit{Input}

\textit{On entry:} the \textbf{order} argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by \textbf{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

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

2: \textbf{nb} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \( b \), the number of observations in the current block of data. At each call the size of the block of data supplied in \( z, t \) and \( x \) can vary; therefore \textbf{nb} can change between calls to \text{nag_tsa_inhom_iema_all} (g13mfc).

\textit{Constraint:} \( \textbf{nb} \geq 0 \).

3: \textbf{z[nb]} – const double \hspace{1cm} \textit{Input}

\textit{On entry:} \( z_i \), the current block of observations, for \( i = k + 1, \ldots, k + b \), where \( k \) is the number of observations processed so far, i.e., the value supplied in \textbf{pn} on entry.

\textit{Constraint:} if \textbf{ftype} = Nag_Identity or Nag_AbsVal and \( p < 0.0 \), \( z[i-1] \neq 0 \), for \( i = 1, 2, \ldots, \textbf{nb} \).

4: \textbf{iema[dim]} – double \hspace{1cm} \textit{Output}

\textit{Note:} the dimension, \textit{dim}, of the array \textbf{iema} must be at least \( \text{pdiema} \times \).

Where \textit{IEMA}(i, j)\) appears in this document, it refers to the array element

\[
iema[(j-1) \times \text{pdiema} + i - 1] \text{ when order} = \text{Nag_ColMajor};\]
\[
iema[(i-1) \times \text{pdiema} + j - 1] \text{ when order} = \text{Nag_RowMajor}.
\]

\textit{On exit:} the iterated exponential moving average.

If \textbf{order} = Nag_ColMajor, \textit{IEMA}(i, j) = \text{EMA}[\tau, j + \text{m1} - 1; y](t_{i+k}).

If \textbf{order} = Nag_RowMajor, \textit{IEMA}(j, i) = \text{EMA}[\tau, j + \text{m1} - 1; y](t_{i+k}).

For \( i = 1, 2, \ldots, \textbf{nb} \), \( j = 1, 2, \ldots, \text{m2} - \text{m1} + 1 \) and \( k \) is the number of observations processed so far, i.e., the value supplied in \textbf{pn} on entry.
On entry: the stride separating row or column elements (depending on the value of order) in the array iema.

Constraints:
if order = Nag_ColMajor, pdiema \geq nb;
otherwise pdiema \geq m2 - m1 + 1.

On entry: \( t_i \) for \( i = k + 1, \ldots, k + b \), where \( k \) is the number of observations processed so far, i.e., the value supplied in \( pn \) on entry.

If \( t_i \leq t_{i-1} \), fail.code = NE_NOT_STRICTLY_INCREASING will be returned, but nag_tsa_inhom_ema_all will continue as if \( t \) was strictly increasing by using the absolute value.

On entry: \( \tau \), the parameter controlling the rate of decay. \( \tau \) must be sufficiently large that \( e^{-\alpha} \), \( \alpha = (t_i - t_{i-1})/\tau \) can be calculated without overflowing, for all \( i \).

Constraint: \( \tau > 0.0 \).

On entry: the minimum number of times the EMA operator is to be iterated.

Constraint: \( m1 \geq 1 \).

On entry: the maximum number of times the EMA operator is to be iterated. Therefore nag_tsa_inhom_ema_all returns \( \text{EMA}[\tau, m; y], \) for \( m = m1, m1 + 1, \ldots, m2 \).

Constraint: \( m2 \geq m1 \).

On entry: if \( pn = 0 \), the values used to start the iterative process, with

\( \text{sinit}[0] = t_0, \) 
\( \text{sinit}[1] = y_0, \) 
\( \text{sinit}[j+1] = \text{EMA}[\tau, j; y](t_0), \) for \( j = 1, 2, \ldots, m2 \).

If \( pn \neq 0 \) then \( \text{sinit} \) is not referenced and may be NULL.

Constraint: if \( ftype \neq \text{Nag_Identity} \), \( \text{sinit}[j-1] \geq 0 \), for \( j = 2, 3, \ldots, m2 + 2 \).

On entry: the type of interpolation used with \( \text{inter}[0] \) indicating the interpolation method to use when calculating \( \text{EMA}[\tau, 1; z] \) and \( \text{inter}[1] \) the interpolation method to use when calculating \( \text{EMA}[\tau, j; z], j > 1 \).

Three types of interpolation are possible:

\( \text{inter}[i] = \text{Nag_PreviousPoint} \)
Previous point, with \( \nu = 1 \).

\( \text{inter}[i] = \text{Nag_Linear} \)
Linear, with \( \nu = (1 - \mu)/\alpha \).

\( \text{inter}[i] = \text{Nag_NextPoint} \)
Next point, \( \nu = \mu \).
Zumbach and Müller (2001) recommend that linear interpolation is used in second and subsequent iterations, i.e., $\text{inter}[1] = \text{Nag Linear}$, irrespective of the interpolation method used at the first iteration, i.e., the value of $\text{inter}[0]$.

Constraint: $\text{inter}[i - 1] = \text{Nag PreviousPoint, Nag Linear or Nag NextPoint}$, for $i = 1, 2$.

12: $\text{ftype} = \text{Nag TS Transform}$

On entry: the function type used to define the relationship between $y$ and $z$ when calculating $\text{EMA}[\tau, 1; y]$. Three functions are provided:

- $\text{ftype} = \text{Nag Identity}$
  The identity function, with $y_i = z_i$. ($\text{ftype} = \text{Nag Identity}$)

- $\text{ftype} = \text{Nag AbsVal}$
  The absolute value, with $y_i = |z_i|^p$. ($\text{ftype} = \text{Nag AbsVal}$)

- $\text{ftype} = \text{Nag AbsDiff}$
  The absolute difference, with $y_i = |z_i - x_i|^p$, where the vector $x$ is supplied in $x$. ($\text{ftype} = \text{Nag AbsDiff}$)

Constraint: $\text{ftype} = \text{Nag Identity, Nag AbsVal or Nag AbsDiff}$. ($\text{ftype} = \text{Nag Identity, Nag AbsVal or Nag AbsDiff}$)

13: $p$ – double *

On entry: $p$, the power used in the transformation function.

On exit: if $\text{ftype} = \text{Nag Identity}$, then $[p]$, the actual power used in the transformation function is returned, otherwise $p$ is unchanged.

Constraint: $p \neq 0$. ($p \neq 0$)

14: $x[nb]$ – const double

On entry: if $\text{ftype} = \text{Nag AbsDiff}$, $x_i$, the vector used to shift the current block of observations, for $i = k + 1, \ldots, k + b$, where $k$ is the number of observations processed so far, i.e., the value supplied in $\text{pn}$ on entry.

If $\text{ftype} \neq \text{Nag AbsDiff}$ then $x$ is not referenced and may be NULL.

Constraint: if $\text{ftype} = \text{Nag AbsDiff}$ and $p < 0$, $x[i - 1] \neq z[i - 1]$, for $i = 1, 2, \ldots, nb$. ($x[nb]$)

15: $\text{pn}$ – Integer *

On entry: $k$, the number of observations processed so far. On the first call to $\text{nag_tsa_inhom_iema_all (g13mfc)}$, or when starting to summarise a new dataset, $\text{pn}$ must be set to 0. On subsequent calls it must be the same value as returned by the last call to $\text{nag_tsa_inhom_iema_all (g13mfc)}$.

On exit: $k + b$, the updated number of observations processed so far.

Constraint: $\text{pn} \geq 0$. ($\text{pn} \geq 0$)

16: $\text{rcomm[m2 + 20]}$ – double

On entry: communication array, used to store information between calls to $\text{nag_tsa_inhom_iema_all (g13mfc)}$. If $\text{rcomm}$ is NULL then $\text{pn}$ must be set to zero and all the data must be supplied in one go.

17: $\text{fail}$ – NagError *

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**
On entry, order = Nag_ColMajor, pdiema = 〈value〉 and nb = 〈value〉. Constraint: pdiema ≥ nb.
On entry, order = Nag_RowMajor, pdiema = 〈value〉 and m2 − m1 + 1 = 〈value〉. Constraint: pdiema ≥ m2 − m1 + 1.

**NE_BAD_PARAM**
On entry, argument 〈value〉 had an illegal value.

**NE_ILLEGAL_COMM**
rcomm has been corrupted between calls.

**NE_INT**
On entry, m1 = 〈value〉. Constraint: m1 ≥ 1.
On entry, nb = 〈value〉. Constraint: nb ≥ 0.
On entry, pn = 〈value〉. Constraint: pn ≥ 0.

**NE_INT_2**
On entry, m1 = 〈value〉 and m2 = 〈value〉. Constraint: m2 ≥ m1.

**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_NOT STRICTLY_INCREASING**
On entry, i = 〈value〉, t[i − 2] = 〈value〉 and t[i − 1] = 〈value〉. Constraint: t should be strictly increasing.

**NE_PREV_CALL**
If pn > 0 then ftype must be unchanged since previous call.
If pn > 0 then inter must be unchanged since previous call.
On entry, m1 = 〈value〉.
On entry at previous call, m1 = 〈value〉. Constraint: if pn > 0 then m1 must be unchanged since previous call.
On entry, $m2 = \langle value \rangle$.
On entry at previous call, $m2 = \langle value \rangle$.
Constraint: if $pn > 0$ then $m2$ must be unchanged since previous call.

On entry, $p = \langle value \rangle$.
On exit from previous call, $p = \langle value \rangle$.
Constraint: if $pn > 0$ then $p$ must be unchanged since previous call.

On entry, $pn = \langle value \rangle$.
On exit from previous call, $pn = \langle value \rangle$.
Constraint: if $pn > 0$ then $pn$ must be unchanged since previous call.

On entry, $\tau = \langle value \rangle$.
On exit from previous call, $\tau = \langle value \rangle$.
Constraint: if $pn > 0$ then $\tau$ must be unchanged since previous call.

On entry, $i = \langle value \rangle$, $z[i-1] = \langle value \rangle$ and $p = \langle value \rangle$.
Constraint: if $\text{ftype} = \text{Nag}_\text{Identity}$ or $\text{Nag}_\text{AbsVal}$ and $z[i] = 0$ for any $i$ then $p > 0.0$.

On entry, $i = \langle value \rangle$, $z[i-1] = \langle value \rangle$, $x[i-1] = \langle value \rangle$ and $p = \langle value \rangle$.
Constraint: if $\text{ftype} = \text{Nag}_\text{AbsDiff}$ and $z[i] = x[i]$ for any $i$ then $p > 0.0$.

On entry, $p = \langle value \rangle$.
Constraint: absolute value of $p$ must be representable as an integer.

On entry, $p = \langle value \rangle$.
Constraint: if $\text{ftype} \neq \text{Nag}_\text{Identity}$, $p \neq 0.0$. If $\text{ftype} = \text{Nag}_\text{Identity}$, the nearest integer to $p$ must not be 0.

On entry, $\tau = \langle value \rangle$.
Constraint: $\tau > 0.0$.

On entry, $\text{ftype} \neq \text{Nag}_\text{Identity}$, $j = \langle value \rangle$ and $\text{sinit}[j-1] = \langle value \rangle$.
Constraint: if $\text{ftype} \neq \text{Nag}_\text{Identity}$, $\text{sinit}[j-1] \geq 0.0$, for $j = 2, 3, \ldots, m2 + 2$.

On entry, $i = \langle value \rangle$, $t[i-2] = \langle value \rangle$ and $t[i-1] = \langle value \rangle$.
Constraint: $t[i-1] \neq t[i-2]$ if linear interpolation is being used.

Truncation occurred to avoid overflow, check for extreme values in $t$, $z$, $x$ or for $\tau$. Results are returned using the truncated values.

7 Accuracy

Not applicable.

8 Parallelism and Performance

\texttt{nag_tsa_inhom_iema_all} (g13mfc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\texttt{nag_tsa_inhom_iema_all} (g13mfc) 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

Approximately $4 \times m_2$ real elements are internally allocated by nag_tsa_inhom_iema_all (g13mfc).

The more data you supply to nag_tsa_inhom_iema_all (g13mfc) in one call, i.e., the larger $nb$ is, the more efficient the routine will be.

Checks are made during the calculation of $\alpha$ and $y_i$ to avoid overflow. If a potential overflow is detected the offending value is replaced with a large positive or negative value, as appropriate, and the calculations performed based on the replacement values. In such cases fail.code = NW_OVERFLOW_WARN is returned. This should not occur in standard usage and will only occur if extreme values of $z$, $t$, $x$ or $\tau$ are supplied.

10 Example

This example reads in three blocks of simulated data from an inhomogeneous time series, then calculates and prints the iterated EMA for $m$ between 2 and 6.

10.1 Program Text

```c
/* nag_tsa_inhom_iema_all (g13mfc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */
/* Pre-processor includes */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg13.h>
#define IEMA(i, j) iema[(order == Nag_RowMajor)?(i*pdiema + j):(j*pdiema + i)]

int main(void)
{
    /* Integer scalar and array declarations */
    Integer i, j, pdiema, m1, m2, miema, nb, pn, ierr;
    Integer exit_status = 0;
    /* NAG structures and types */
    NagError fail;
    Nag_OrderType order;
    Nag_TS_Interpolation inter[2];
    Nag_TS_Transform ftype;
    /* Double scalar and array declarations */
    double p, tau;
    double *iema = 0, *rcomm = 0, *sinit = 0, *t = 0, *x = 0, *z = 0;
    /* Character scalar and array declarations */
    char corder[40], cinter[40], cftype[40];
    /* Initialise the error structure */
    INIT_FAIL(fail);
    printf("nag_tsa_inhom_iema_all (g13mfc) Example Program Results\n\n");
    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#endif
    /* Read in the required order for the output matrix */
#ifdef _WIN32
    scanf_s("%39s%[^\n ]",corder, _countof(corder));
#endif
```
#else
    scanf("%39s", corder);
#endif

    order = (Nag_OrderType) nag_enum_name_to_value(corder);

/* Read in the problem size */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%"NAG_IFMT"%[\n] ", &m1, &m2);
#else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%[\n] ", &m1, &m2);
#endif

/* Read in the transformation function and its parameter */
#ifdef _WIN32
    scanf_s("%39s", cftype, _countof(cftype));
#else
    scanf("%39s", cftype);
#endif

    ftype = (Nag_TS_Transform) nag_enum_name_to_value(cftype);
#ifdef _WIN32
    scanf_s("%lf", &p);
#else
    scanf("%lf", &p);
#endif

/* Read in the interpolation method to use */
#ifdef _WIN32
    scanf_s("%39s", cinter, _countof(cinter));
#else
    scanf("%39s", cinter);
#endif

    inter[0] = (Nag_TS_Interpolation) nag_enum_name_to_value(cinter);
#ifdef _WIN32
    scanf_s("%39s", cinter, _countof(cinter));
#else
    scanf("%39s", cinter);
#endif

    inter[1] = (Nag_TS_Interpolation) nag_enum_name_to_value(cinter);

/* Read in the decay parameter */
#ifdef _WIN32
    scanf_s("%lf%*[\n] ", &tau);
#else
    scanf("%lf%*[\n] ", &tau);
#endif

/* Read in the initial values */
if (!(sinit = NAG_ALLOC(m2 + 2, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    for (i = 0; i < m2 + 2; i++)
    {
        #ifdef _WIN32
            scanf_s("%lf", &sinit[i]);
        #else
            scanf("%lf", &sinit[i]);
        #endif
    }

    miema = m2 - m1 + 1;

#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

    /* Print some titles */
    for (i = 0; i < 20 + 5 * miema; i++)
        printf(" ");
printf("Iteration\n");
printf(" Time ");
for (i = m1; i <= m2; i++) printf("%2"NAG_IFMT" ", i);
printf("\n ");
for (i = 0; i < 21 + 10 * miema; i++) printf("-");
printf("\n")

if (!(rcomm = NAG_ALLOC(m2 + 20, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
for (pn = 0;;)
{
    /* Read in the number of observations in this block */
    #ifdef _WIN32
    ierr = scanf_s("%"NAG_IFMT, &nb);
    #else
    ierr = scanf("%"NAG_IFMT, &nb);
    #endif
    if (ierr == EOF || ierr < 1) break;
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n] ");
    #endif
    /* Reallocate Z and T to the required size */
    NAG_FREE(z);
    NAG_FREE(t);
    if (!(z = NAG_ALLOC(nb, double))
    || !(t = NAG_ALLOC(nb, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Read in the data for this block */
    if (ftype != 3)
    {
        for (i = 0; i < nb; i++)
        {
            #ifdef _WIN32
            scanf_s("%lf%lf", &t[i], &z[i]);
            #else
            scanf("%lf%lf", &t[i], &z[i]);
            #endif
        }
    }
    else
    {
        /* Reallocate X to the required size */
        NAG_FREE(x);
        if (!(x = NAG_ALLOC(nb, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
        for (i = 0; i < nb; i++)
        {
            #ifdef _WIN32
            scanf_s("%lf%lf%lf", &t[i], &z[i], &x[i]);
            #else
            scanf("%lf%lf%lf", &t[i], &z[i], &x[i]);
            #endif
        }
    }
}
#ifdef _WIN32
  scanf_s("%*[\n] ");
#else
  scanf("%*[\n] ");
#endif

if (order == Nag_ColMajor)
{
  pdiema = nb;
}
else
{
  pdiema = miema;
}

/* Reallocate the output array */
NAG_FREE(iema);
if (!((iema = NAG_ALLOC(nb*miema, double))))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* Call nag_tsa_inhom_iema_all (g13mfc) to update the iterated EMA for
this block of data */
nag_tsa_inhom_iema_all(order,nb,z,iema,pdiema,t,tau,m1,m2,sinit,inter,
ftype,&p,x,&pn,rcomm,&fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_tsa_inhom_iema_all (g13mfc).\n%s\n", fail.message);
  exit_status = -1;
  goto END;
}

/* Display the results for this block of data */
for (i = 0; i < nb; i++)
{
  printf(" %3"NAG_IFMT" %10.1f ", pn - nb + i + 1, t[i]);
  for (j = 0; j < miema; j++)
  {
    printf(" %8.3f", IEMA(i,j));
  }
  printf("\n");
}

END:
NAG_FREE(iema);
NAG_FREE(t);
NAG_FREE(z);
NAG_FREE(x);
NAG_FREE(sinit);
NAG_FREE(rcomm);
return(exit_status);
10.2 Program Data

nag_tsa_inhom_iema_all (g13mfc) Example Program Data

Nag_RowMajor :: sorder
2 6 :: m1,m2
Nag_Identity 1.0
Nag_NextPoint Nag_Linear 2.0 :: ftype, p, inter[0:1], tau
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 :: sinit
5 :: nb
7.5 0.6
8.2 0.6
18.1 0.8
22.8 0.1
25.8 0.2 :: t and z for first block
10 :: nb
26.8 0.2
31.1 0.5
38.4 0.7
45.9 0.1
48.2 0.4
48.9 0.7
57.9 0.8
58.5 0.3
63.9 0.2
65.2 0.5 :: t and z for second block
15 :: nb
66.6 0.2
67.4 0.3
69.3 0.8
69.9 0.6
73.0 0.1
75.6 0.7
77.0 0.9
84.7 0.6
86.8 0.3
88.0 0.1
88.5 0.1
91.0 0.4
93.0 1.0
93.7 1.0
94.0 0.1 :: t and z for third block

10.3 Program Results

nag_tsa_inhom_iema_all (g13mfc) Example Program Results

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Time</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.5</td>
<td>0.433</td>
<td>0.320</td>
<td>0.237</td>
<td>0.175</td>
<td>0.130</td>
</tr>
<tr>
<td>2</td>
<td>8.2</td>
<td>0.479</td>
<td>0.361</td>
<td>0.268</td>
<td>0.198</td>
<td>0.147</td>
</tr>
<tr>
<td>3</td>
<td>18.1</td>
<td>0.756</td>
<td>0.700</td>
<td>0.631</td>
<td>0.558</td>
<td>0.485</td>
</tr>
<tr>
<td>4</td>
<td>22.8</td>
<td>0.406</td>
<td>0.535</td>
<td>0.592</td>
<td>0.600</td>
<td>0.577</td>
</tr>
<tr>
<td>5</td>
<td>25.8</td>
<td>0.232</td>
<td>0.351</td>
<td>0.459</td>
<td>0.530</td>
<td>0.561</td>
</tr>
<tr>
<td>6</td>
<td>26.8</td>
<td>0.217</td>
<td>0.301</td>
<td>0.406</td>
<td>0.491</td>
<td>0.540</td>
</tr>
<tr>
<td>7</td>
<td>31.1</td>
<td>0.357</td>
<td>0.309</td>
<td>0.318</td>
<td>0.364</td>
<td>0.422</td>
</tr>
<tr>
<td>8</td>
<td>38.4</td>
<td>0.630</td>
<td>0.556</td>
<td>0.490</td>
<td>0.445</td>
<td>0.425</td>
</tr>
<tr>
<td>9</td>
<td>45.9</td>
<td>0.263</td>
<td>0.357</td>
<td>0.407</td>
<td>0.428</td>
<td>0.432</td>
</tr>
<tr>
<td>10</td>
<td>48.2</td>
<td>0.241</td>
<td>0.284</td>
<td>0.343</td>
<td>0.388</td>
<td>0.413</td>
</tr>
<tr>
<td>11</td>
<td>48.9</td>
<td>0.279</td>
<td>0.277</td>
<td>0.325</td>
<td>0.372</td>
<td>0.403</td>
</tr>
<tr>
<td>12</td>
<td>57.9</td>
<td>0.713</td>
<td>0.617</td>
<td>0.543</td>
<td>0.496</td>
<td>0.469</td>
</tr>
<tr>
<td>13</td>
<td>58.5</td>
<td>0.717</td>
<td>0.643</td>
<td>0.566</td>
<td>0.511</td>
<td>0.478</td>
</tr>
<tr>
<td>14</td>
<td>63.9</td>
<td>0.385</td>
<td>0.495</td>
<td>0.541</td>
<td>0.546</td>
<td>0.531</td>
</tr>
<tr>
<td>15</td>
<td>65.2</td>
<td>0.346</td>
<td>0.432</td>
<td>0.502</td>
<td>0.533</td>
<td>0.535</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>16</td>
<td>66.6</td>
<td>0.330</td>
<td>0.384</td>
<td>0.453</td>
<td>0.504</td>
<td>0.526</td>
</tr>
<tr>
<td>17</td>
<td>67.4</td>
<td>0.315</td>
<td>0.364</td>
<td>0.427</td>
<td>0.483</td>
<td>0.515</td>
</tr>
<tr>
<td>18</td>
<td>69.3</td>
<td>0.409</td>
<td>0.367</td>
<td>0.389</td>
<td>0.435</td>
<td>0.478</td>
</tr>
<tr>
<td>19</td>
<td>69.9</td>
<td>0.459</td>
<td>0.385</td>
<td>0.386</td>
<td>0.423</td>
<td>0.465</td>
</tr>
<tr>
<td>20</td>
<td>73.0</td>
<td>0.377</td>
<td>0.403</td>
<td>0.394</td>
<td>0.398</td>
<td>0.419</td>
</tr>
<tr>
<td>21</td>
<td>75.6</td>
<td>0.411</td>
<td>0.399</td>
<td>0.399</td>
<td>0.397</td>
<td>0.403</td>
</tr>
<tr>
<td>22</td>
<td>77.0</td>
<td>0.536</td>
<td>0.440</td>
<td>0.410</td>
<td>0.401</td>
<td>0.401</td>
</tr>
<tr>
<td>23</td>
<td>84.7</td>
<td>0.632</td>
<td>0.606</td>
<td>0.563</td>
<td>0.524</td>
<td>0.493</td>
</tr>
<tr>
<td>24</td>
<td>86.8</td>
<td>0.538</td>
<td>0.587</td>
<td>0.583</td>
<td>0.557</td>
<td>0.526</td>
</tr>
<tr>
<td>25</td>
<td>88.0</td>
<td>0.444</td>
<td>0.542</td>
<td>0.574</td>
<td>0.567</td>
<td>0.542</td>
</tr>
<tr>
<td>26</td>
<td>88.5</td>
<td>0.401</td>
<td>0.515</td>
<td>0.564</td>
<td>0.567</td>
<td>0.548</td>
</tr>
<tr>
<td>27</td>
<td>91.0</td>
<td>0.331</td>
<td>0.404</td>
<td>0.481</td>
<td>0.529</td>
<td>0.545</td>
</tr>
<tr>
<td>28</td>
<td>93.0</td>
<td>0.495</td>
<td>0.418</td>
<td>0.438</td>
<td>0.483</td>
<td>0.518</td>
</tr>
<tr>
<td>29</td>
<td>93.7</td>
<td>0.585</td>
<td>0.455</td>
<td>0.438</td>
<td>0.469</td>
<td>0.506</td>
</tr>
<tr>
<td>30</td>
<td>94.0</td>
<td>0.612</td>
<td>0.475</td>
<td>0.441</td>
<td>0.465</td>
<td>0.500</td>
</tr>
</tbody>
</table>