NAG Library Function Document

nag_tsa_inhom_ma (g13mgc)

1 Purpose

nag_tsa_inhom_ma (g13mgc) provides a moving average, moving norm, moving variance and moving standard deviation operator for an inhomogeneous time series.

2 Specification

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

void nag_tsa_inhom_ma (Integer nb, double ma[], const double t[], double tau,
                      Integer m1, Integer m2, const double sinit[],
                      const Nag_TS_Interpolation inter[], Nag_TS_Transform ftype, double *p,
                      Integer *pn, double wma[], double rcomm[], NagError *fail)
```

3 Description

nag_tsa_inhom_ma (g13mgc) provides a number of operators 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 \( i = 1, 2, \ldots, n \).

The main operator available, the moving average (MA), with parameter \( \tau \) is defined as

\[
MA(\tau, m_1, m_2; y)(t_i) = \frac{1}{m_2 - m_1 + 1} \sum_{j=m_1}^{m_2} EMA(\tilde{\tau}, j; y)(t_i)
\]

where \( \tilde{\tau} = \frac{2\tau}{m_2 + m_1} \), \( m_1 \) and \( m_2 \) are user-supplied integers controlling the amount of lag and smoothing respectively, with \( m_2 \geq m_1 \) and \( EMA[:;] \) is the iterated exponential moving average operator.

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

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

with

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

and

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

where

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

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_ma (g13mgc) 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: \( y_i = |z_i - \text{MA}[\tau, m_1, m_2; z(t_i)]^p \).

where the notation \([p]\) is used to denote the integer nearest to \(p\). In addition, if either the absolute value or absolute difference transformation are used then the resulting moving average can be scaled by \(p^{-1}\).

The various parameter options allow a number of different operators to be applied by nag_tsa_inhom_ma (g13mgc), a few of which are:

(i) **Moving Average (MA)**, as defined in (1) (obtained by setting \(\text{ftype} = \text{Nag Identity}\) and \(p = 1\)).

(ii) **Moving Norm (MNorm)**, defined as

\[
\text{MNorm}(\tau, m, p; z) = \text{MA}[\tau, 1, m; |z|^{[p]}]^{1/p}
\]

(obtained by setting \(\text{ftype} = \text{Nag AbsValScaled}\), \(m_1 = 1\) and \(m_2 = m\)).

(iii) **Moving Variance (MVar)**, defined as

\[
\text{MVar}(\tau, m, p; z) = \text{MA}[\tau, 1, m; |z - \text{MA}[\tau, 1, m; z]|^{[p]}]
\]

(obtained by setting \(\text{ftype} = \text{Nag AbsDiff}\), \(m_1 = 1\) and \(m_2 = m\)).

(iv) **Moving Standard Deviation (MSD)**, defined as

\[
\text{MSD}(\tau, m, p; z) = \text{MA}[\tau, 1, m; |z - \text{MA}[\tau, 1, m; z]|^{[p]}]^{1/p}
\]

(obtained by setting \(\text{ftype} = \text{Nag AbsDiffScaled}\), \(m_1 = 1\) and \(m_2 = m\)).

For large datasets or where not all the data is not available at the same time, \(z\) and \(t\) can be split into arbitrary sized blocks and nag_tsa_inhom_ma (g13mgc) called multiple times.

4 References


5 Arguments

1. **nb** – Integer

   *Input*

   *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 \(\text{ma}\) and \(\text{t}\) can vary; therefore \(\text{nb}\) can change between calls to nag_tsa_inhom_ma (g13mgc).

   *Constraint:* \(\text{nb} \geq 0\).

2. **\(\text{ma}[\text{nb}]\)** – double

   *Input/Output*

   *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 \(\text{pn}\) on entry.

   *On exit:* the moving average:

   if \(\text{ftype} = \text{Nag AbsValScaled}\) or \(\text{Nag AbsDiffScaled}\)

   \[
   \text{ma}[i - 1] = \text{MA}[\tau, m_1, m_2; y(t_i)]^{1/p},
   \]

   otherwise

   \[
   \text{ma}[i - 1] = \text{MA}[\tau, m_1, m_2; y](t_i).
   \]
3: \[ t[\text{nb}] \] – const double

*Input*

*On entry:* \( t_i \), the times for 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 \( t_i \leq t_{i-1} \), \text{fail.code} = \text{NE_NOT_STRICTLY_INCREASING} \) will be returned, but nag_tsa_inhom_ma (g13mgc) will continue as if \( t \) was strictly increasing by using the absolute value. The lagged difference, \( t_i - t_{i-1} \) must be sufficiently small that \( e^{-\alpha} \), \( \alpha = (t_i - t_{i-1})/\bar{\tau} \) can be calculated without overflowing, for all \( i \).

4: \[ \text{tau} \] – double

*Input*

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

*Constraint:* \( \text{tau} > 0 \).

5: \[ m_1 \] – Integer

*Input*

*On entry:* \( m_1 \), the iteration of the EMA operator at which the sum is started.

*Constraint:* \( m_1 \geq 1 \).

6: \[ m_2 \] – Integer

*Input*

*On entry:* \( m_2 \), the iteration of the EMA operator at which the sum is ended.

*Constraint:* \( m_2 \geq m_1 \).

7: \[ \text{sinit}[\text{dim}] \] – const double

*Input*

*Note:* the dimension, \( \text{dim} \), of the array \( \text{sinit} \) must be at least

\[ 2 \times m_2 + 3 \text{ when } \text{ftype} = \text{Nag_AbsDiff} \text{ or } \text{Nag_AbsDiffScaled}; \\
2 \times m_2 + 2 \text{ when } \text{ftype} = \text{Nag_Identity, Nag_AbsVal or Nag_AbsValScaled}; \\
\text{sinit} \text{ may be } \text{NULL} \text{ when } \text{pn} \neq 0. \]

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

\[ \text{sinit}[0] = t_0, \\
\text{sinit}[1] = y_0, \\
\text{sinit}[j + 1] = \text{EMA}[\tau, j; y](t_0), \text{ for } i = 1, 2, \ldots, m_2. \]

In addition, if \( \text{ftype} = \text{Nag_AbsDiff} \text{ or } \text{Nag_AbsDiffScaled} \) then

\[ \text{sinit}[m_2 + 2] = z_0, \\
\text{sinit}[m_2 + j + 1] = \text{EMA}[\tau, j; z](t_0), \text{ for } j = 1, 2, \ldots, m_2. \]

i.e., initial values based on the original data \( z \) as opposed to the transformed data \( y \).

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

*Constraint:* if \( \text{ftype} \neq \text{Nag_Identity} \), \( \text{sinit}[j - 1] \geq 0 \), for \( j = 2, 3, \ldots, m_2 + 2 \).

8: \[ \text{inter}[2] \] – const Nag_TS_Interpolation

*Input*

*On entry:* the type of interpolation used with \( \text{inter}[0] \) indicating the interpolation method to use when calculating \( \text{EMA}[\tau, j; z] \) 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 \( \nu = (1 - \mu)/\alpha \).
inter[n] = Nag_NextPoint
Next point, \( \nu = \mu \).

Zumbach and Müller (2001) recommend that linear interpolation is used in second and subsequent iterations, i.e., inter[1] = Nag_Linear, irrespective of the interpolation method used at the first iteration, i.e., the value of inter[0].

Constraint: inter[i - 1] = Nag_PreviousPoint, Nag_Linear or Nag_NextPoint, for \( i = 1, 2 \).

9: ftype – Nag_TS_Transform

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

ftype = Nag_Identity
The identity function, with \( y_i = z_i^p \).

ftype = Nag_AbsVal or Nag_AbsValScaled
The absolute value, with \( y_i = |z_i|^p \).

ftype = Nag_AbsDiff or Nag_AbsDiffScaled
The absolute difference, with \( y_i = |z_i - MA[\tau, m; y](t_i)|^p \).

If ftype = Nag_AbsValScaled or Nag_AbsDiffScaled then the resulting vector of averages is scaled by \( p^{-1} \) as described in ma.

Constraint: ftype = Nag_Identity, Nag_AbsVal, Nag_AbsDiff, Nag_AbsValScaled or Nag_AbsDiffScaled.

10: p – double *

Input/Output

On entry: \( p \), the power used in the transformation function.

On exit: if ftype = Nag_Identity, then \([p]\), the actual power used in the transformation function is returned, otherwise \( p \) is unchanged.

Constraint: \( p \neq 0 \).

11: pn – Integer *

Input/Output

On entry: \( k \), the number of observations processed so far. On the first call to nag_tsa_inhom_ma (g13mgc), or when starting to summarise a new dataset, \( pn \) must be set to 0. On subsequent calls it must be the same value as returned by the last call to nag_tsa_inhom_ma (g13mgc).

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

Constraint: \( pn \geq 0 \).

12: wma[2nb] – double

Output

On exit: either the moving average or exponential moving average, depending on the value of ftype.

if ftype = Nag_AbsDiff or Nag_AbsDiffScaled
\[ wma[i - 1] = MA[\tau; y](t_i) \]

otherwise
\[ wma[i - 1] = EMA[\hat{\tau}; y](t_i). \]

13: rcomm[2 \times m2 + 20] – double

Communication Array

On entry: communication array, used to store information between calls to nag_tsa_inhom_ma (g13mgc). If rcomm is NULL then \( pn \) must be set to zero and all the data must be supplied in one go.
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_BAD_PARAM**
On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

**NE_ILLEGAL_COMM**
\( \text{rcomm} \) has been corrupted between calls.

**NE_INT**
On entry, \( m1 = \langle \text{value} \rangle \).
Constraint: \( m1 \geq 1 \).

On entry, \( nb = \langle \text{value} \rangle \).
Constraint: \( nb \geq 0 \).

On entry, \( pn = \langle \text{value} \rangle \).
Constraint: \( pn \geq 0 \).

**NE_INT_2**
On entry, \( m1 = \langle \text{value} \rangle \) and \( m2 = \langle \text{value} \rangle \).
Constraint: \( m2 \geq m1 \).

**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, \( i = \langle \text{value} \rangle \), \( t[i-2] = \langle \text{value} \rangle \) and \( t[i-1] = \langle \text{value} \rangle \).
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, \( m1 = \langle \text{value} \rangle \).
On entry at previous call, \( m1 = \langle \text{value} \rangle \).
Constraint: if \( pn > 0 \) then \( m1 \) must be unchanged since previous call.

On entry, \( m2 = \langle \text{value} \rangle \).
On entry at previous call, \( m2 = \langle \text{value} \rangle \).
Constraint: if \( pn > 0 \) then \( m2 \) must be unchanged since previous call.
On entry, $p = \langle\text{value}\rangle$.
On exit from previous call, $p = \langle\text{value}\rangle$.
Constraint: if $pn > 0$ then $p$ must be unchanged since previous call.

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

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

On entry, $i = \langle\text{value}\rangle$, $ma[i-1] = \langle\text{value}\rangle$ and $p = \langle\text{value}\rangle$.
Constraint: if $ftype \neq \text{Nag\_Identity}$, $Nag\_AbsVal$ or $Nag\_AbsValScaled$ and $ma[i-1] = 0$ for any $i$ then $p > 0.0$.

On entry, $i = \langle\text{value}\rangle$, $ma[i-1] = \langle\text{value}\rangle$, $wma[i-1] = \langle\text{value}\rangle$ and $p = \langle\text{value}\rangle$.
Constraint: if $p < 0.0$, $ma[i-1] - wma[i-1] \neq 0.0$, for any $i$.

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

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

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

NE_REAL

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

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

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

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

7 Accuracy
Not applicable.

8 Parallelism and Performance

`nag_tsa_inhom_ma` (g13mgc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

`nag_tsa_inhom_ma` (g13mgc) 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 $4m_2$ real elements are internally allocated by nag_tsa_inhom_ma (g13mgc). If ftype = Nag_AbsDiff or Nag_AbsDiffScaled then a further $nb$ real elements are also allocated.

The more data you supply to nag_tsa_inhom_ma (g13mgc) in one call, i.e., the larger $nb$ is, the more efficient the function will be.

Checks are made during the calculation of $y_i$ 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 ma, t or tau are supplied.

10 Example

The example reads in a simulated time series, $(t, z(t))$ and calculates the moving average. The data is supplied in three blocks of differing sizes.

10.1 Program Text

```c
/* nag_tsa_inhom_ma (g13mgc) 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>

int main(void)
{
    /* Integer scalar and array declarations */
    Integer i, m1, m2, nb, pn, ierr, lsinit;
    Integer exit_status = 0;
    /* NAG structures and types */
    NagError fail;
    Nag_TS_Interpolation inter[2];
    Nag_TS_Transform ftype;
    /* Double scalar and array declarations */
    double p, tau;
    double *ma = 0, *rcomm = 0, *sinit = 0, *t = 0, *wma = 0;
    /* Character scalar and array declarations */
    char cinter[40], cftype[40];
    /* Initialise the error structure */
    INIT_FAIL(fail);
    printf("nag_tsa_inhom_ma (g13mgc) Example Program Results\n\n");
    /* Skip heading in data file */
    #ifdef _WIN32
        scanf_s("%*[`\n] ");
    #else
        scanf("%*[`\n] ");
    #endif
    /* 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);
```
/* 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 (ftype == Nag_AbsDiff || ftype == Nag_AbsDiffScaled) {
    lsinit = 2 * m2 + 3;
} else {
    lsinit = m2 + 2;
}
if (!lsinit) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
for (i = 0; i < lsinit; i++) {
    if (ftype == Nag_AbsDiff || ftype == Nag_AbsDiffScaled) {
        #ifdef _WIN32
            scanf_s("%lf", &sinit[i]);
        #else
            scanf("%lf", &sinit[i]);
        #endif
    } else {
        scanf("%lf", &sinit[i]);
    }
}

/* Print some titles */
printf(" Time MA\n");
printf("--------------------------------\n");
if (!rcomm) {
    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
    /* Allocate MA, T and WMA to the required size */
    NAG_FREE(ma);
    NAG_FREE(t);
    NAG_FREE(wma);
    if (!(ma = NAG_ALLOC(nb, double)) ||
        !(t = NAG_ALLOC(nb,double)) ||
        !(wma = NAG_ALLOC(nb,double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Read in the data for this block */
    for (i = 0; i < nb; i++)
    {
#ifdef _WIN32
        scanf_s("%lf%lf", &t[i], &ma[i]);
#else
        scanf("%lf%lf", &t[i], &ma[i]);
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#endif
    /* Call nag_tsa_inhom_ma (g13mgc) to update the moving average
    operator for this block of data. The routine overwrites the
    input data */
    nag_tsa_inhom_ma(nb, ma, t, tau, m1, m2, sinit, inter, ftype, &p, &pn, wma, rcomm,
    &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_tsa_inhom_ma (g13mgc).\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 %10.3f\n", pn-nb+i+1, t[i], ma[i]);
    }
    printf("\n");
}

END:
NAG_FREE(ma);
NAG_FREE(wma);
NAG_FREE(t);
NAG_FREE(sinit);
NAG_FREE(rcomm);

return(exit_status);
}

10.2 Program Data

nag_tsa_inhom_ma (g13mgc) Example Program Data

1 2 : ml,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 : sinit

5 : nb

7.5 0.6
8.2 0.6
18.1 0.8
22.8 0.1
25.8 0.2

: End of t, z 1st 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

: End of t, z 2nd 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

: End of t, z 3rd block

10.3 Program Results

nag_tsa_inhom_ma (g13mgc) Example Program Results

<table>
<thead>
<tr>
<th>Time</th>
<th>MA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.5</td>
</tr>
<tr>
<td>2</td>
<td>8.2</td>
</tr>
<tr>
<td>3</td>
<td>18.1</td>
</tr>
<tr>
<td>4</td>
<td>22.8</td>
</tr>
<tr>
<td>5</td>
<td>25.8</td>
</tr>
<tr>
<td>6</td>
<td>26.8</td>
</tr>
<tr>
<td>7</td>
<td>31.1</td>
</tr>
<tr>
<td>8</td>
<td>38.4</td>
</tr>
<tr>
<td>9</td>
<td>45.9</td>
</tr>
<tr>
<td>10</td>
<td>48.2</td>
</tr>
<tr>
<td>11</td>
<td>48.9</td>
</tr>
<tr>
<td>12</td>
<td>57.9</td>
</tr>
<tr>
<td>13</td>
<td>58.5</td>
</tr>
<tr>
<td>14</td>
<td>7.5</td>
</tr>
<tr>
<td>15</td>
<td>8.2</td>
</tr>
<tr>
<td>16</td>
<td>18.1</td>
</tr>
<tr>
<td>17</td>
<td>22.8</td>
</tr>
<tr>
<td>18</td>
<td>25.8</td>
</tr>
<tr>
<td>19</td>
<td>26.8</td>
</tr>
<tr>
<td>20</td>
<td>31.1</td>
</tr>
<tr>
<td>21</td>
<td>38.4</td>
</tr>
<tr>
<td>22</td>
<td>45.9</td>
</tr>
<tr>
<td>23</td>
<td>48.2</td>
</tr>
<tr>
<td>24</td>
<td>48.9</td>
</tr>
<tr>
<td>25</td>
<td>57.9</td>
</tr>
<tr>
<td>26</td>
<td>58.5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>14</td>
<td>63.9</td>
</tr>
<tr>
<td>15</td>
<td>65.2</td>
</tr>
<tr>
<td>16</td>
<td>66.6</td>
</tr>
<tr>
<td>17</td>
<td>67.4</td>
</tr>
<tr>
<td>18</td>
<td>69.3</td>
</tr>
<tr>
<td>19</td>
<td>69.9</td>
</tr>
<tr>
<td>20</td>
<td>73.0</td>
</tr>
<tr>
<td>21</td>
<td>75.6</td>
</tr>
<tr>
<td>22</td>
<td>77.0</td>
</tr>
<tr>
<td>23</td>
<td>84.7</td>
</tr>
<tr>
<td>24</td>
<td>86.8</td>
</tr>
<tr>
<td>25</td>
<td>88.0</td>
</tr>
<tr>
<td>26</td>
<td>88.5</td>
</tr>
<tr>
<td>27</td>
<td>91.0</td>
</tr>
<tr>
<td>28</td>
<td>93.0</td>
</tr>
<tr>
<td>29</td>
<td>93.7</td>
</tr>
<tr>
<td>30</td>
<td>94.0</td>
</tr>
</tbody>
</table>