

## 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

```
#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  $i = 1, 2, \dots, n$ . 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

$$\frac{e^{-t_i/\tau}}{\tau}.$$

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

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

where

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

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  $[p]$  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 `nag_tsa_inhom_iema_all` (g13mfc) called multiple times.

## 4 References

Dacorogna M M, Gencay R, Müller U, Olsen R B and Pictet O V (2001) *An Introduction to High-frequency Finance* Academic Press

Zumbach G O and Müller U A (2001) Operators on inhomogeneous time series *International Journal of Theoretical and Applied Finance* **4(1)** 147–178

## 5 Arguments

1: **order** – Nag\_OrderType *Input*

*On entry:* the **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 **order** = Nag\_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

*Constraint:* **order** = Nag\_RowMajor or Nag\_ColMajor.

2: **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 **z**, **t** and **x** can vary; therefore **nb** can change between calls to `nag_tsa_inhom_iema_all` (g13mfc).

*Constraint:* **nb**  $\geq 0$ .

3: **z[nb]** – const double *Input*

*On entry:*  $z_i$ , the current block of observations, for  $i = k + 1, \dots, k + b$ , where  $k$  is the number of observations processed so far, i.e., the value supplied in **pn** on entry.

*Constraint:* if **ftype** = Nag\_Identity or Nag\_AbsVal and **p**  $< 0.0$ , **z** $[i - 1] \neq 0$ , for  $i = 1, 2, \dots, \mathbf{nb}$ .

4: **iema** $[dim]$  – double *Output*

**Note:** the dimension,  $dim$ , of the array **iema** must be at least **pdiema**  $\times$  .

Where **IEMA** $(i, j)$  appears in this document, it refers to the array element

$$\begin{aligned} &\mathbf{iema}[(j - 1) \times \mathbf{pdiema} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ &\mathbf{iema}[(i - 1) \times \mathbf{pdiema} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

*On exit:* the iterated exponential moving average.

If **order** = Nag\_ColMajor, **IEMA** $(i, j) = \text{EMA}[\tau, j + \mathbf{m1} - 1; y](t_{i+k})$ .

If **order** = Nag\_RowMajor, **IEMA** $(j, i) = \text{EMA}[\tau, j + \mathbf{m1} - 1; y](t_{i+k})$ .

For  $i = 1, 2, \dots, \mathbf{nb}$ ,  $j = 1, 2, \dots, \mathbf{m2} - \mathbf{m1} + 1$  and  $k$  is the number of observations processed so far, i.e., the value supplied in **pn** on entry.

- 5: **pdiema** – Integer *Input*  
*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.
- 6: **t[nb]** – const double *Input*  
*On entry:*  $t_i$ , the times for the current block of observations, for  $i = k + 1, \dots, k + b$ , 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\_iema\_all** (g13mfc) will continue as if  $t$  was strictly increasing by using the absolute value.
- 7: **tau** – double *Input*  
*On entry:*  $\tau$ , the parameter controlling the rate of decay.  $\tau$  must be sufficiently large that  $e^{-\alpha}$ ,  $\alpha = (t_i - t_{i-1})/\tau$  can be calculated without overflowing, for all  $i$ .  
*Constraint:* **tau** > 0.0.
- 8: **m1** – Integer *Input*  
*On entry:* the minimum number of times the EMA operator is to be iterated.  
*Constraint:* **m1**  $\geq$  1.
- 9: **m2** – Integer *Input*  
*On entry:* the maximum number of times the EMA operator is to be iterated. Therefore **nag\_tsa\_inhom\_iema\_all** (g13mfc) returns  $\text{EMA}[\tau, m; y]$ , for  $m = \mathbf{m1}, \mathbf{m1} + 1, \dots, \mathbf{m2}$ .  
*Constraint:* **m2**  $\geq$  **m1**.
- 10: **sinit[m2 + 2]** – const double *Input*  
*On entry:* if **pn** = 0, the values used to start the iterative process, with  
 $\mathbf{sinit}[0] = t_0$ ,  
 $\mathbf{sinit}[1] = y_0$ ,  
 $\mathbf{sinit}[j + 1] = \text{EMA}[\tau, j; y](t_0)$ ,  $j = 1, 2, \dots, \mathbf{m2}$ .  
 If **pn**  $\neq$  0 then **sinit** is not referenced and may be **NULL**.  
*Constraint:* if **ftype**  $\neq$  Nag\_Identity,  $\mathbf{sinit}[j - 1] \geq 0$ , for  $j = 2, 3, \dots, \mathbf{m2} + 2$ .
- 11: **inter[2]** – const Nag\_TS\_Interpolation *Input*  
*On entry:* the type of interpolation used with **inter**[0] indicating the interpolation method to use when calculating  $\text{EMA}[\tau, 1; z]$  and **inter**[1] the interpolation method to use when calculating  $\text{EMA}[\tau, j; z]$ ,  $j > 1$ .  
 Three types of interpolation are possible:  
**inter**[ $i$ ] = Nag\_PreviousPoint  
 Previous point, with  $\nu = 1$ .  
**inter**[ $i$ ] = Nag\_Linear  
 Linear, with  $\nu = (1 - \mu)/\alpha$ .  
**inter**[ $i$ ] = 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  $i = 1, 2$ .

12: **f**type – Nag\_TS\_Transform *Input*

*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:

**f**type = Nag\_Identity

The identity function, with  $y_i = z_i^{[p]}$ .

**f**type = Nag\_AbsVal

The absolute value, with  $y_i = |z_i|^p$ .

**f**type = Nag\_AbsDiff

The absolute difference, with  $y_i = |z_i - x_i|^p$ , where the vector  $x$  is supplied in **x**.

*Constraint:* **f**type = Nag\_Identity, Nag\_AbsVal or Nag\_AbsDiff.

13: **p** – double \* *Input/Output*

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

*On exit:* if **f**type = Nag\_Identity, then  $[p]$ , the actual power used in the transformation function is returned, otherwise **p** is unchanged.

*Constraint:* **p**  $\neq 0$ .

14: **x**[**nb**] – const double *Input*

*On entry:* if **f**type = Nag\_AbsDiff,  $x_i$ , the vector used to shift the current block of observations, for  $i = k + 1, \dots, k + b$ , where  $k$  is the number of observations processed so far, i.e., the value supplied in **pn** on entry.

If **f**type  $\neq$  Nag\_AbsDiff then **x** is not referenced and may be **NULL**.

*Constraint:* if **f**type = Nag\_AbsDiff and **p**  $< 0$ , **x**[ $i - 1$ ]  $\neq$  **z**[ $i - 1$ ], for  $i = 1, 2, \dots, \mathbf{nb}$ .

15: **pn** – Integer \* *Input/Output*

*On entry:*  $k$ , the number of observations processed so far. On the first call to nag\_tsa\_inhom\_iema\_all (g13mfc), 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\_iema\_all (g13mfc).

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

*Constraint:* **pn**  $\geq 0$ .

16: **rcomm**[**m2** + 20] – double *Communication Array*

*On entry:* communication array, used to store information between calls to nag\_tsa\_inhom\_iema\_all (g13mfc). If **rcomm** is **NULL** then **pn** must be set to zero and all the data must be supplied in one go.

17: **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.

### NE\_ARRAY\_SIZE

On entry, **order** = Nag\_ColMajor, **pdiema** =  $\langle value \rangle$  and **nb** =  $\langle value \rangle$ .  
Constraint: **pdiema**  $\geq$  **nb**.

On entry, **order** = Nag\_RowMajor, **pdiema** =  $\langle value \rangle$  and **m2** – **m1** + 1 =  $\langle value \rangle$ .  
Constraint: **pdiema**  $\geq$  **m2** – **m1** + 1.

### NE\_BAD\_PARAM

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

### NE\_ILLEGAL\_COMM

**rcomm** has been corrupted between calls.

### NE\_INT

On entry, **m1** =  $\langle value \rangle$ .  
Constraint: **m1**  $\geq$  1.

On entry, **nb** =  $\langle value \rangle$ .  
Constraint: **nb**  $\geq$  0.

On entry, **pn** =  $\langle value \rangle$ .  
Constraint: **pn**  $\geq$  0.

### NE\_INT\_2

On entry, **m1** =  $\langle value \rangle$  and **m2** =  $\langle value \rangle$ .  
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.

### NE\_NOT\_STRICTLY\_INCREASING

On entry,  $i = \langle value \rangle$ ,  $t[i - 2] = \langle value \rangle$  and  $t[i - 1] = \langle value \rangle$ .  
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** =  $\langle value \rangle$ .

On entry at previous call, **m1** =  $\langle value \rangle$ .

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

**NE\_REAL**

On entry,  $i = \langle value \rangle$ ,  $z[i - 1] = \langle value \rangle$  and **p** =  $\langle value \rangle$ .  
 Constraint: if **fctype** = Nag\_Identity or Nag\_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 **fctype** = Nag\_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 **fctype**  $\neq$  Nag\_Identity, **p**  $\neq$  0.0. If **fctype** = Nag\_Identity, the nearest integer to **p** must not be 0.

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

**NE\_REAL\_ARRAY**

On entry, **fctype**  $\neq$  Nag\_Identity,  $j = \langle value \rangle$  and **sinit**[ $j - 1$ ] =  $\langle value \rangle$ .  
 Constraint: if **fctype**  $\neq$  Nag\_Identity, **sinit**[ $j - 1$ ]  $\geq$  0.0, for  $j = 2, 3, \dots, m2 + 2$ .

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.

**NW\_OVERFLOW\_WARN**

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**

nag\_tsa\_inhom\_iema\_all (g13mfc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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 Users' Note for your implementation for any additional implementation-specific information.

**9 Further Comments**

Approximately  $4 \times m2$  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

```

/* nag_tsa_inhom_iema_all (g13mfc) Example Program.
 *
 * Copyright 2011, 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 */
    scanf("%*[\n] ");

    /* Read in the required order for the output matrix */
    scanf("%39s%*[\n] ", corder);
    order = (Nag_OrderType) nag_enum_name_to_value(corder);

    /* Read in the problem size */
    scanf("%ld%ld%*[\n] ", &m1, &m2);

    /* Read in the transformation function and its parameter */
    scanf("%39s", cftype);
    ftype = (Nag_TS_Transform) nag_enum_name_to_value(cftype);
    scanf("%lf", &p);

    /* Read in the interpolation method to use */
    scanf("%39s", cinter);
    inter[0] = (Nag_TS_Interpolation) nag_enum_name_to_value(cinter);
    scanf("%39s", cinter);

```

```

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

/* Read in the decay parameter */
scanf("%lf%*[\n] ", &tau);

/* 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++)
{
    scanf("%lf", &sinit[i]);
}
scanf("%*[\n] ");

miema = m2 - m1 + 1;

/* Print some titles */
for (i = 0; i < 20+5*miema; i++) printf(" ");
printf("Iteration\n");
printf("          Time          ");
for (i = m1; i <= m2; i++) printf("%2ld          ", 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 */
    ierr = scanf("%"NAG_IFMT, &nb);
    if (ierr == EOF || ierr < 1) break;
    scanf("%*[\n] ");

    /* 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++)
        {
            scanf("%lf%lf",&t[i], &z[i]);
        }
    }
    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++)
    {
        scanf("%lf%lf%lf",&t[i], &z[i], &x[i]);
    }
    scanf("%*[^\\n] ");

    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(" %3ld  %10.1f  ", pn - nb + i + 1, t[i]);
        for (j = 0; j < miema; j++)
        {
            printf(" %8.3f", IEMA(i,j));
        }
        printf("\\n");
    }
    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
  7.5  0.6                :: nb
  8.2  0.6
 18.1  0.8
 22.8  0.1
 25.8  0.2                :: t and z for first block

 10
 26.8  0.2                :: nb
 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
 66.6  0.2                :: nb
 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
```

	Time	Iteration				
		2	3	4	5	6
1	7.5	0.433	0.320	0.237	0.175	0.130
2	8.2	0.479	0.361	0.268	0.198	0.147
3	18.1	0.756	0.700	0.631	0.558	0.485
4	22.8	0.406	0.535	0.592	0.600	0.577
5	25.8	0.232	0.351	0.459	0.530	0.561
6	26.8	0.217	0.301	0.406	0.491	0.540
7	31.1	0.357	0.309	0.318	0.364	0.422
8	38.4	0.630	0.556	0.490	0.445	0.425
9	45.9	0.263	0.357	0.407	0.428	0.432
10	48.2	0.241	0.284	0.343	0.388	0.413
11	48.9	0.279	0.277	0.325	0.372	0.403
12	57.9	0.713	0.617	0.543	0.496	0.469
13	58.5	0.717	0.643	0.566	0.511	0.478
14	63.9	0.385	0.495	0.541	0.546	0.531
15	65.2	0.346	0.432	0.502	0.533	0.535

16	66.6	0.330	0.384	0.453	0.504	0.526
17	67.4	0.315	0.364	0.427	0.483	0.515
18	69.3	0.409	0.367	0.389	0.435	0.478
19	69.9	0.459	0.385	0.386	0.423	0.465
20	73.0	0.377	0.403	0.394	0.398	0.419
21	75.6	0.411	0.399	0.399	0.397	0.403
22	77.0	0.536	0.440	0.410	0.401	0.401
23	84.7	0.632	0.606	0.563	0.524	0.493
24	86.8	0.538	0.587	0.583	0.557	0.526
25	88.0	0.444	0.542	0.574	0.567	0.542
26	88.5	0.401	0.515	0.564	0.567	0.548
27	91.0	0.331	0.404	0.481	0.529	0.545
28	93.0	0.495	0.418	0.438	0.483	0.518
29	93.7	0.585	0.455	0.438	0.469	0.506
30	94.0	0.612	0.475	0.441	0.465	0.500

---