

# NAG Library Function Document

## nag\_tsa\_exp\_smooth (g13amc)

### 1 Purpose

nag\_tsa\_exp\_smooth (g13amc) performs exponential smoothing using either single exponential, double exponential or a Holt–Winters method.

### 2 Specification

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

void nag_tsa_exp_smooth (Nag_InitialValues mode, Nag_ExpSmoothType itype,
    Integer p, const double param[], Integer n, const double y[], Integer k,
    double init[], Integer nf, double fv[], double fse[], double yhat[],
    double res[], double *dv, double *ad, double r[], NagError *fail)
```

### 3 Description

Exponential smoothing is a relatively simple method of short term forecasting for a time series. nag\_tsa\_exp\_smooth (g13amc) provides five types of exponential smoothing; single exponential, Brown's double exponential, linear Holt (also called double exponential smoothing in some references), additive Holt–Winters and multiplicative Holt–Winters. The choice of smoothing method used depends on the characteristics of the time series. If the mean of the series is only slowly changing then single exponential smoothing may be suitable. If there is a trend in the time series, which itself may be slowly changing, then double exponential smoothing may be suitable. If there is a seasonal component to the time series, e.g., daily or monthly data, then one of the two Holt–Winters methods may be suitable.

For a time series  $y_t$ , for  $t = 1, 2, \dots, n$ , the five smoothing functions are defined by the following:

Single Exponential Smoothing

$$\begin{aligned} m_t &= \alpha y_t + (1 - \alpha)m_{t-1} \\ \hat{y}_{t+f} &= m_t \\ \text{var}(\hat{y}_{t+f}) &= \text{var}(\epsilon_t)(1 + (f - 1)\alpha^2) \end{aligned}$$

Brown Double Exponential Smoothing

$$\begin{aligned} m_t &= \alpha y_t + (1 - \alpha)m_{t-1} \\ r_t &= \alpha(m_t - m_{t-1}) + (1 - \alpha)r_{t-1} \\ \hat{y}_{t+f} &= m_t + ((f - 1) + 1/\alpha)r_t \\ \text{var}(\hat{y}_{t+f}) &= \text{var}(\epsilon_t) \left( 1 + \sum_{i=0}^{f-1} (2\alpha + (i - 1)\alpha^2)^2 \right) \end{aligned}$$

Linear Holt Smoothing

$$\begin{aligned} m_t &= \alpha y_t + (1 - \alpha)(m_{t-1} + \phi r_{t-1}) \\ r_t &= \gamma(m_t - m_{t-1}) + (1 - \gamma)\phi r_{t-1} \\ \hat{y}_{t+f} &= m_t + \sum_{i=1}^f \phi^i r_t \\ \text{var}(\hat{y}_{t+f}) &= \text{var}(\epsilon_t) \left( 1 + \sum_{i=1}^{f-1} \left( \alpha + \frac{\alpha\gamma\phi(\phi^i - 1)}{(\phi - 1)} \right)^2 \right) \end{aligned}$$

## Additive Holt–Winters Smoothing

$$\begin{aligned}
m_t &= \alpha(y_t - s_{t-p}) + (1 - \alpha)(m_{t-1} + \phi r_{t-1}) \\
r_t &= \gamma(m_t - m_{t-1}) + (1 - \gamma)\phi r_{t-1} \\
s_t &= \beta(y_t - m_t) + (1 - \beta)s_{t-p} \\
\hat{y}_{t+f} &= m_t + \left( \sum_{i=1}^f \phi^i r_t \right) + s_{t-p} \\
\text{var}(\hat{y}_{t+f}) &= \text{var}(\epsilon_t) \left( 1 + \sum_{i=1}^{f-1} \psi_i^2 \right) \\
\psi_i &= \begin{cases} 0 & \text{if } i \geq f \\ \alpha + \frac{\alpha\gamma\phi(\phi^i - 1)}{(\phi - 1)} & \text{if } i \bmod p \neq 0 \\ \alpha + \frac{\alpha\gamma\phi(\phi^i - 1)}{(\phi - 1)} + \beta(1 - \alpha) & \text{otherwise} \end{cases}
\end{aligned}$$

## Multiplicative Holt–Winters Smoothing

$$\begin{aligned}
m_t &= \alpha y_t / s_{t-p} + (1 - \alpha)(m_{t-1} + \phi r_{t-1}) \\
r_t &= \gamma(m_t - m_{t-1}) + (1 - \gamma)\phi r_{t-1} \\
s_t &= \beta y_t / m_t + (1 - \beta)s_{t-p} \\
\hat{y}_{t+f} &= \left( m_t + \sum_{i=1}^f \phi^i r_t \right) \times s_{t-p} \\
\text{var}(\hat{y}_{t+f}) &= \text{var}(\epsilon_t) \left( \sum_{i=0}^{\infty} \sum_{j=0}^{p-1} \left( \psi_{j+ip} \frac{s_{t+f}}{s_{t+f-j}} \right)^2 \right)
\end{aligned}$$

and  $\psi$  is defined as in the additive Holt–Winters smoothing,

where  $m_t$  is the mean,  $r_t$  is the trend and  $s_t$  is the seasonal component at time  $t$  with  $p$  being the seasonal order. The  $f$ -step ahead forecasts are given by  $\hat{y}_{t+f}$  and their variances by  $\text{var}(\hat{y}_{t+f})$ . The term  $\text{var}(\epsilon_t)$  is estimated as the mean deviation.

The parameters,  $\alpha$ ,  $\beta$  and  $\gamma$  control the amount of smoothing. The nearer these parameters are to one, the greater the emphasis on the current data point. Generally these parameters take values in the range 0.1 to 0.3. The linear Holt and two Holt–Winters smoothers include an additional parameter,  $\phi$ , which acts as a trend dampener. For  $0.0 < \phi < 1.0$  the trend is dampened and for  $\phi > 1.0$  the forecast function has an exponential trend,  $\phi = 0.0$  removes the trend term from the forecast function and  $\phi = 1.0$  does not dampen the trend.

For all methods, values for  $\alpha$ ,  $\beta$ ,  $\gamma$  and  $\psi$  can be chosen by trying different values and then visually comparing the results by plotting the fitted values along side the original data. Alternatively, for single exponential smoothing a suitable value for  $\alpha$  can be obtained by fitting an ARIMA(0, 1, 1) model (see `nag_tsa_multi_inp_model_estim` (g13bec)). For Brown's double exponential smoothing and linear Holt smoothing with no dampening, (i.e.,  $\phi = 1.0$ ), suitable values for  $\alpha$  and  $\gamma$  can be obtained by fitting an ARIMA(0, 2, 2) model. Similarly, the linear Holt method, with  $\phi \neq 1.0$ , can be expressed as an ARIMA(1, 2, 2) model and the additive Holt–Winters, with no dampening, ( $\phi = 1.0$ ), can be expressed as a seasonal ARIMA model with order  $p$  of the form ARIMA(0, 1,  $p + 1$ )(0, 1, 0). There is no similar procedure for obtaining parameter values for the multiplicative Holt–Winters method, or the additive Holt–Winters method with  $\phi \neq 1.0$ . In these cases parameters could be selected by minimizing a measure of fit using one of the nonlinear optimization functions in Chapter e04.

In addition to values for  $\alpha$ ,  $\beta$ ,  $\gamma$  and  $\psi$ , initial values,  $m_0$ ,  $r_0$  and  $s_{-j}$ , for  $j = 0, 1, \dots, p - 1$ , are required to start the smoothing process. You can either supply these or they can be calculated by `nag_tsa_exp_smooth` (g13amc) from the first  $k$  observations. For single exponential smoothing the mean of the observations is used to estimate  $m_0$ . For Brown double exponential smoothing and linear Holt smoothing, a simple linear regression is carried out with the series as the dependent variable and the sequence  $1, 2, \dots, k$  as the independent variable. The intercept is then used to estimate  $m_0$  and the slope to estimate  $r_0$ . In the case of the additive Holt–Winters method, the same regression is carried out, but a separate intercept is used for each of the  $p$  seasonal groupings. The slope gives an estimate for  $r_0$  and the mean of the  $p$  intercepts is used as the estimate of  $m_0$ . The seasonal parameters  $s_{-j}$ , for  $j = 0, 1, \dots, p - 1$ , are estimated as the  $p$  intercepts  $- m_0$ . A similar approach is adopted for the multiplicative Holt–Winter's method.

One step ahead forecasts,  $\hat{y}_{t+1}$  are supplied along with the residuals computed as  $(y_{t+1} - \hat{y}_{t+1})$ . In addition, two measures of fit are provided. The mean absolute deviation,

$$\frac{1}{n} \sum_{t=1}^n |y_t - \hat{y}_t|$$

and the square root of the mean deviation

$$\sqrt{\frac{1}{n} \sum_{t=1}^n (y_t - \hat{y}_t)^2}.$$

## 4 References

Chatfield C (1980) *The Analysis of Time Series* Chapman and Hall

## 5 Arguments

1: **mode** – Nag\_InitialValues *Input*

*On entry:* indicates if nag\_tsa\_exp\_smooth (g13amc) is continuing from a previous call or, if not, how the initial values are computed.

**mode** = Nag\_InitialValuesSupplied

Required values for  $m_0$ ,  $r_0$  and  $s_{-j}$ , for  $j = 0, 1, \dots, p-1$ , are supplied in **init**.

**mode** = Nag\_ContinueAndUpdate

nag\_tsa\_exp\_smooth (g13amc) continues from a previous call using values that are supplied in **r**.

**mode** = Nag\_EstimateInitialValues

Required values for  $m_0$ ,  $r_0$  and  $s_{-j}$ , for  $j = 0, 1, \dots, p-1$ , are estimated using the first  $k$  observations.

*Constraint:* **mode** = Nag\_InitialValuesSupplied, Nag\_ContinueAndUpdate or Nag\_EstimateInitialValues.

2: **itype** – Nag\_ExpSmoothType *Input*

*On entry:* the smoothing function.

**itype** = Nag\_SingleExponential

Single exponential.

**itype** = Nag\_BrownsExponential

Brown double exponential.

**itype** = Nag\_LinearHolt

Linear Holt.

**itype** = Nag\_AdditiveHoltWinters

Additive Holt–Winters.

**itype** = Nag\_MultiplicativeHoltWinters

Multiplicative Holt–Winters.

*Constraint:* **itype** = Nag\_SingleExponential, Nag\_BrownsExponential, Nag\_LinearHolt, Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters.

3: **p** – Integer *Input*

*On entry:* if **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters, the seasonal order,  $p$ , otherwise **p** is not referenced.

*Constraint:* if **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters, **p** > 1.

4: **param** $[dim]$  – const double *Input*

**Note:** the dimension,  $dim$ , of the array **param** must be at least

- 1 when **itype** = Nag\_SingleExponential or Nag\_BrownsExponential;
- 3 when **itype** = Nag\_LinearHolt;
- 4 when **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters.

*On entry:* the smoothing parameters.

If **itype** = Nag\_SingleExponential or Nag\_BrownsExponential, **param** $[0] = \alpha$  and any remaining elements of **param** are not referenced.

If **itype** = Nag\_LinearHolt, **param** $[0] = \alpha$ , **param** $[1] = \gamma$ , **param** $[2] = \phi$  and any remaining elements of **param** are not referenced.

If **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters, **param** $[0] = \alpha$ , **param** $[1] = \gamma$ , **param** $[2] = \beta$  and **param** $[3] = \phi$ .

*Constraints:*

- if **itype** = Nag\_SingleExponential,  $0.0 \leq \alpha \leq 1.0$ ;
- if **itype** = Nag\_BrownsExponential,  $0.0 < \alpha \leq 1.0$ ;
- if **itype** = Nag\_LinearHolt,  $0.0 \leq \alpha \leq 1.0$  and  $0.0 \leq \gamma \leq 1.0$  and  $\phi \geq 0.0$ ;
- if **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters,  $0.0 \leq \alpha \leq 1.0$  and  $0.0 \leq \gamma \leq 1.0$  and  $0.0 \leq \beta \leq 1.0$  and  $\phi \geq 0.0$ .

5: **n** – Integer *Input*

*On entry:* the number of observations in the series.

*Constraint:*  $n \geq 0$ .

6: **y** $[n]$  – const double *Input*

*On entry:* the time series.

7: **k** – Integer *Input*

*On entry:* if **mode** = Nag\_EstimateInitialValues, the number of observations used to initialize the smoothing.

If **mode**  $\neq$  Nag\_EstimateInitialValues, **k** is not referenced.

*Constraints:*

- if **mode** = Nag\_EstimateInitialValues and **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters,  $2 \times p \leq k \leq n$ ;
- if **mode** = Nag\_EstimateInitialValues and **itype** = Nag\_SingleExponential, Nag\_BrownsExponential or Nag\_LinearHolt,  $1 \leq k \leq n$ .

8: **init** $[dim]$  – double *Input/Output*

**Note:** the dimension,  $dim$ , of the array **init** must be at least

- 1 when **itype** = Nag\_SingleExponential;
- 2 when **itype** = Nag\_BrownsExponential or Nag\_LinearHolt;
- $2 + p$  when **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters.

*On entry:* if **mode** = Nag\_InitialValuesSupplied, the initial values for  $m_0$ ,  $r_0$  and  $s_{-j}$ , for  $j = 0, 1, \dots, p-1$ , used to initialize the smoothing.

If **itype** = Nag\_SingleExponential, **init** $[0] = m_0$  and the remaining elements of **init** are not referenced.

If **itype** = Nag\_BrownsExponential or Nag\_LinearHolt, **init** $[0] = m_0$  and **init** $[1] = r_0$  and the remaining elements of **init** are not referenced.

If **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters, **init**[0] =  $m_0$ , **init**[1] =  $r_0$  and **init**[2] to **init**[ $p + 1$ ] hold the values for  $s_{-j}$ , for  $j = 0, 1, \dots, p - 1$ . The remaining elements of **init** are not referenced.

*On exit:* if **mode**  $\neq$  Nag\_ContinueAndUpdate, the values used to initialize the smoothing. These are in the same order as described above.

- 9: **nf** – Integer *Input*  
*On entry:* the number of forecasts required beyond the end of the series. Note, the one step ahead forecast is always produced.  
*Constraint:* **nf**  $\geq 0$ .
- 10: **fv[nf]** – double *Output*  
*On exit:*  $\hat{y}_{t+f}$ , for  $f = 1, 2, \dots, \mathbf{nf}$ , the next **nf** step forecasts. Where  $t = \mathbf{n}$ , if **mode**  $\neq$  Nag\_ContinueAndUpdate, else  $t$  is the total number of smoothed and forecast values already produced.
- 11: **fse[nf]** – double *Output*  
*On exit:* the forecast standard errors for the values given in **fv**.
- 12: **yhat[n]** – double *Output*  
*On exit:*  $\hat{y}_{t+1}$ , for  $t = 1, 2, \dots, \mathbf{n}$ , the one step ahead forecast values, with **yhat**[ $i - 1$ ] being the one step ahead forecast of **y**[ $i - 2$ ].
- 13: **res[n]** – double *Output*  
*On exit:* the residuals,  $(y_{t+1} - \hat{y}_{t+1})$ , for  $t = 1, 2, \dots, \mathbf{n}$ .
- 14: **dv** – double \* *Output*  
*On exit:* the square root of the mean deviation.
- 15: **ad** – double \* *Output*  
*On exit:* the mean absolute deviation.
- 16: **r[dim]** – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **r** must be at least  
13 when **itype** = Nag\_SingleExponential, Nag\_BrownsExponential or Nag\_LinearHolt;  
13 + **p** when **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters.  
*On entry:* if **mode** = Nag\_ContinueAndUpdate, **r** must contain the values as returned by a previous call to nag\_rand\_exp\_smooth (g05pmc) or nag\_tsa\_exp\_smooth (g13amc), **r** need not be set otherwise.  
If **itype** = Nag\_SingleExponential, Nag\_BrownsExponential or Nag\_LinearHolt, only the first 13 elements of **r** are referenced, otherwise the first 13 +  $p$  elements are referenced.  
*On exit:* the information on the current state of the smoothing.  
*Constraint:* if **mode** = Nag\_ContinueAndUpdate, **r** must have been initialized by at least one previous call to nag\_rand\_exp\_smooth (g05pmc) or nag\_tsa\_exp\_smooth (g13amc) with **mode**  $\neq$  Nag\_ContinueAndUpdate, and **r** should not have been changed since the last call to nag\_rand\_exp\_smooth (g05pmc) or nag\_tsa\_exp\_smooth (g13amc).
- 17: **fail** – NagError \* *Input/Output*  
The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

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

### NE\_ENUM\_INT

On entry, **itype** =  $\langle value \rangle$  and **p** =  $\langle value \rangle$ .

Constraint: if **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters, **p** > 1.

On entry, **p** =  $\langle value \rangle$ .

Constraint: if **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters, **p** > 1.

### NE\_INT

On entry, **n** =  $\langle value \rangle$ .

Constraint: **n** ≥ 0.

On entry, **nf** =  $\langle value \rangle$ .

Constraint: **nf** ≥ 0.

### NE\_INT\_2

On entry, **k** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: if **mode** = Nag\_EstimateInitialValues and **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters,  $1 \leq k \leq n$ .

### NE\_INT\_3

On entry, **k** =  $\langle value \rangle$ ,  $2 \times \mathbf{p} = \langle value \rangle$ .

Constraint: if **mode** = Nag\_EstimateInitialValues and **itype** = Nag\_AdditiveHoltWinters or Nag\_MultiplicativeHoltWinters,  $2 \times \mathbf{p} \leq k$ .

### 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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

### NE\_MODEL\_PARAMS

A multiplicative Holt–Winters model cannot be used with the supplied data.

### NE\_NO\_LICENCE

Your licence key may have expired or may not have been installed correctly.

See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

### NE\_REAL\_ARRAY

On entry, **param**[ $\langle value \rangle$ ] =  $\langle value \rangle$ .

Constraint:  $0.0 \leq \mathbf{param}[\langle value \rangle] \leq 1.0$ .

On entry, **param**[ $\langle value \rangle$ ] =  $\langle value \rangle$ .

Constraint: if **itype** = Nag\_BrownsExponential,  $0.0 < \mathbf{param}[\langle value \rangle] \leq 1.0$ .

On entry, **param**[*value*] = *value*.

Constraint: **param**[*value*]  $\geq 0.0$ .

On entry, the array **r** has not been initialized correctly.

## 7 Accuracy

Not applicable.

## 8 Parallelism and Performance

nag\_tsa\_exp\_smooth (g13amc) is not threaded in any implementation.

## 9 Further Comments

Single exponential, Brown's double exponential and linear Holt smoothing methods are stable, whereas the two Holt–Winters methods can be affected by poor initial values for the seasonal components.

See also the function document for nag\_rand\_exp\_smooth (g05pmc).

## 10 Example

This example smooths a time series relating to the rate of the earth's rotation about its polar axis.

### 10.1 Program Text

```
/* nag_tsa_exp_smooth (g13amc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg13.h>

int main(void)
{
    /* Integer scalar and array declarations */
    Integer exit_status = 0;
    Integer i, ival, k, n, nf, p;
    /* Double scalar and array declarations */
    double ad, dv;
    double *fse = 0, *fv = 0, *init = 0, *param = 0, *r = 0, *res = 0;
    double *y = 0, *yhat = 0;
    /* Character scalar and array declarations */
    char smode[40], sitype[40];
    /* NAG structures */
    Nag_InitialValues mode;
    Nag_ExpSmoothType itype;
    NagError fail;

    /* Initialize the error structure */
    INIT_FAIL(fail);

    printf("nag_tsa_exp_smooth (g13amc) Example Program Results\n");

    /* Skip headings in data file */
#ifdef _WIN32
    scanf_s("%*[^\\n] ");
#endif
}
```

```

#else
    scanf("%*[^\\n] ");
#endif
/* Read in the initial arguments */
#ifdef _WIN32
    scanf_s("%39s%39s" NAG_IFMT "%" NAG_IFMT "%*[^\\n] ", smode,
            (unsigned)_countof(smode), sitype, (unsigned)_countof(sitype),
            &n, &nf);
#else
    scanf("%39s%39s" NAG_IFMT "%" NAG_IFMT "%*[^\\n] ", smode, sitype, &n, &nf);
#endif
/*
 * nag_enum_name_to_value (x04nac).
 * Converts NAG enum member name to value
 */
mode = (Nag_InitialValues) nag_enum_name_to_value(smode);
itype = (Nag_ExpSmoothType) nag_enum_name_to_value(sitype);

if (!(fse = NAG_ALLOC(nf, double)) ||
    !(fv = NAG_ALLOC(nf, double)) ||
    !(res = NAG_ALLOC(n, double)) ||
    !(y = NAG_ALLOC(n, double)) || !(yhat = NAG_ALLOC(n, double)))
{
    printf("Allocation failure\\n");
    exit_status = -1;
    goto END;
}

/* Read in the observed data */
for (i = 0; i < n; i++)
#ifdef _WIN32
    scanf_s("%lf ", &y[i]);
#else
    scanf("%lf ", &y[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[^\\n] ");
#else
    scanf("%*[^\\n] ");
#endif
/* Read in the itype dependent arguments (skipping headings) */
#ifdef _WIN32
    scanf_s("%*[^\\n] ");
#else
    scanf("%*[^\\n] ");
#endif
if (itype == Nag_SingleExponential) {
    /* Single exponential smoothing required */
    if (!(param = NAG_ALLOC(1, double)))
    {
        printf("Allocation failure\\n");
        exit_status = -1;
        goto END;
    }
}
#ifdef _WIN32
    scanf_s("%lf%*[^\\n] ", &param[0]);
#else
    scanf("%lf%*[^\\n] ", &param[0]);
#endif
p = 0;
ival = 1;
}
else if (itype == Nag_BrownsExponential) {
    /* Browns exponential smoothing required */
    if (!(param = NAG_ALLOC(2, double)))
    {
        printf("Allocation failure\\n");
        exit_status = -1;
        goto END;
    }
}
#ifdef _WIN32

```



```

        scanf_s("%lf %lf%*[\n] ", &param[0], &param[1]);
#else
        scanf("%lf %lf%*[\n] ", &param[0], &param[1]);
#endif
        p = 0;
        ival = 2;
    }
    else if (itype == Nag_LinearHolt) {
        /* Linear Holt smoothing required */
        if (!(param = NAG_ALLOC(3, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
#ifdef _WIN32
        scanf_s("%lf %lf %lf%*[\n] ", &param[0], &param[1], &param[2]);
#else
        scanf("%lf %lf %lf%*[\n] ", &param[0], &param[1], &param[2]);
#endif
        p = 0;
        ival = 2;
    }
    else if (itype == Nag_AdditiveHoltWinters) {
        /* Additive Holt Winters smoothing required */
        if (!(param = NAG_ALLOC(4, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
#ifdef _WIN32
        scanf_s("%lf %lf %lf %lf %" NAG_IFMT "%*[\n] ", &param[0], &param[1],
            &param[2], &param[3], &p);
#else
        scanf("%lf %lf %lf %lf %" NAG_IFMT "%*[\n] ", &param[0], &param[1],
            &param[2], &param[3], &p);
#endif
        ival = p + 2;
    }
    else if (itype == Nag_MultiplicativeHoltWinters) {
        /* Multiplicative Holt Winters smoothing required */
        if (!(param = NAG_ALLOC(4, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
#ifdef _WIN32
        scanf_s("%lf %lf %lf %lf %" NAG_IFMT "%*[\n] ", &param[0], &param[1],
            &param[2], &param[3], &p);
#else
        scanf("%lf %lf %lf %lf %" NAG_IFMT "%*[\n] ", &param[0], &param[1],
            &param[2], &param[3], &p);
#endif
        ival = p + 2;
    }
    else {
        printf("%s is an unknown type\n", sitype);
        exit_status = -1;
        goto END;
    }

    /* Allocate some more memory */
    if (!(init = NAG_ALLOC(p + 2, double)) || !(r = NAG_ALLOC(p + 13, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}

```

```

/* Read in the mode dependent arguments (skipping headings) */
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
if (mode == Nag_InitialValuesSupplied) {
    /* User supplied initial values */
    for (i = 0; i < ival; i++)
#ifdef _WIN32
        scanf_s("%lf ", &init[i]);
#else
        scanf("%lf ", &init[i]);
#endif
}
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
}
else if (mode == Nag_ContinueAndUpdate) {
    /* Continuing from a previously saved R */
    for (i = 0; i < p + 13; i++)
#ifdef _WIN32
        scanf_s("%lf ", &r[i]);
#else
        scanf("%lf ", &r[i]);
#endif
}
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
}
else if (mode == Nag_EstimateInitialValues) {
    /* Initial values calculated from first k observations */
#ifdef _WIN32
        scanf_s("%" NAG_IFMT "%*[\n] ", &k);
#else
        scanf("%" NAG_IFMT "%*[\n] ", &k);
#endif
}
else {
    printf("%s is an unknown mode\n", smode);
    exit_status = -1;
    goto END;
}

/* Call the library routine to smooth the series */
nag_tsa_exp_smooth(mode, itype, p, param, n, y, k, init, nf, fv, fse, yhat,
                  res, &dv, &ad, r, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_tsa_exp_smooth (g13amc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Display the output */
printf("Initial values used:\n");
for (i = 0; i < ival; i++)
    printf("%4" NAG_IFMT "    %12.3f  \n", i + 1, init[i]);
printf("\n");
printf("Mean Deviation      = %13.4e\n", dv);
printf("Absolute Deviation = %13.4e\n", ad);
printf("\n");
printf("      Observed      1-Step\n");
printf("Period  Values      Forecast      Residual\n");
for (i = 0; i < n; i++)
    printf("%4" NAG_IFMT "    %12.3f    %12.3f    %12.3f\n", i + 1, y[i],
          yhat[i], res[i]);
printf("\n");

```

```

printf("          Forecast      Standard\n");
printf(" Period   Values        Errors\n");
for (i = 0; i < nf; i++)
    printf("%4" NAG_IFMT "      %12.3f      %12.3f  \n", n + i + 1, fv[i], fse[i]);

END:
    NAG_FREE(fse);
    NAG_FREE(fv);
    NAG_FREE(init);
    NAG_FREE(param);
    NAG_FREE(r);
    NAG_FREE(res);
    NAG_FREE(y);
    NAG_FREE(yhat);

    return exit_status;
}

```

## 10.2 Program Data

```

nag_tsa_exp_smooth (g13amc) Example Program Data
Nag_EstimateInitialValues Nag_LinearHolt 11 5 : mode,itype,n,nf
180 135 213 181 148 204 228 225 198 200 187 : y
dependent arguments for itype = Nag_LinearHolt
0.01 1.0 1.0 : param[0],param[1],param[2]
dependent arguments for mode = Nag_ContinueAndUpdate
11 : k

```

## 10.3 Program Results

```

nag_tsa_exp_smooth (g13amc) Example Program Results
Initial values used:

```

```

    1      168.018
    2       3.800

```

```

Mean Deviation      =      2.5473e+01
Absolute Deviation =      2.1233e+01

```

Period	Observed Values	1-Step Forecast	Residual
1	180.000	171.818	8.182
2	135.000	175.782	-40.782
3	213.000	178.848	34.152
4	181.000	183.005	-2.005
5	148.000	186.780	-38.780
6	204.000	189.800	14.200
7	228.000	193.492	34.508
8	225.000	197.732	27.268
9	198.000	202.172	-4.172
10	200.000	206.256	-6.256
11	187.000	210.256	-23.256

Period	Forecast Values	Standard Errors
12	213.854	25.473
13	217.685	25.478
14	221.516	25.490
15	225.346	25.510
16	229.177	25.542

---