NAG Library Function Document

nag_lookback_fls_greeks (s30bbc)

1   Purpose

nag_lookback_fls_greeks (s30bbc) computes the price of a floating-strike lookback option together with its sensitivities (Greeks).

2   Specification

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

```
void nag_lookback_fls_greeks (Nag_OrderType order, Nag_CallPut option,
     Integer m, Integer n, const double sm[], double s, const double t[],
     double sigma, double r, double q, double p[], double delta[],
     double gamma[], double vega[], double theta[], double rho[],
     double crho[], double vanna[], double charm[], double speed[],
     double colour[], double zomma[], double vomma[], NagError *fail)
```

3   Description

nag_lookback_fls_greeks (s30bbc) computes the price of a floating-strike lookback call or put option, together with the Greeks or sensitivities, which are the partial derivatives of the option price with respect to certain of the other input parameters. A call option of this type confers the right to buy the underlying asset at the lowest price, $S_{\min}$, observed during the lifetime of the contract. A put option gives the holder the right to sell the underlying asset at the maximum price, $S_{\max}$, observed during the lifetime of the contract. Thus, at expiry, the payoff for a call option is $S - S_{\min}$, and for a put, $S_{\max} - S$.

For a given minimum value the price of a floating-strike lookback call with underlying asset price, S, and time to expiry, T, is

$$P_{\text{call}} = S e^{-qT} \Phi(a_1) - S_{\min} e^{-rT} \Phi(a_2) + S e^{-rT} \frac{\sigma^2}{2b} \left[\left(\frac{S}{S_{\min}} \right)^{-2b/\sigma^2} \Phi\left(-a_1 + \frac{2b}{\sigma} \sqrt{T} \right) - e^{bT} \Phi(-a_1) \right],$$

where $b = r - q \neq 0$. The volatility, σ, risk-free interest rate, r, and annualised dividend yield, q, are constants.

The corresponding put price is

$$P_{\text{put}} = S_{\max} e^{-rT} \Phi(-a_2) - S e^{-qT} \Phi(-a_1) + S e^{-rT} \frac{\sigma^2}{2b} \left[-\left(\frac{S}{S_{\max}} \right)^{-2b/\sigma^2} \Phi\left(a_1 - \frac{2b}{\sigma} \sqrt{T} \right) + e^{bT} \Phi(a_1) \right].$$

In the above, Φ denotes the cumulative Normal distribution function,

$$\Phi(x) = \int_{-\infty}^{x} \phi(y) dy$$

where ϕ denotes the standard Normal probability density function

$$\phi(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-y^2/2 \right)$$

and

$$a_1 = \frac{\ln(S/S_{\text{m}}) + \left(b + \sigma^2/2 \right) T}{\sigma \sqrt{T}}$$

$$a_2 = a_1 - \sigma \sqrt{T}$$

where S_m is taken to be the minimum price attained by the underlying asset, $S_{\min}$, for a call and the maximum price, $S_{\max}$, for a put.

The option price $P_{ij} = P(X = X_i, T = T_j)$ is computed for each minimum or maximum observed price in a set $S_{\min}(i)$ or $S_{\max}(i)$, $i = 1, 2, \ldots, m$, and for each expiry time in a set T_j, $j = 1, 2, \ldots, n$.

4   References

Goldman B M, Sosin H B and Gatto M A (1979) Path dependent options: buy at the low, sell at the high *Journal of Finance* **34** 1111–1127

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:    **option** – Nag_CallPut   *Input*

On entry: determines whether the option is a call or a put.

option = Nag_Call
        A call; the holder has a right to buy.

option = Nag_Put
        A put; the holder has a right to sell.

Constraint: **option** = Nag_Call or Nag_Put.

3:    **m** – Integer   *Input*

On entry: the number of minimum or maximum prices to be used.

Constraint: **m** $\geq$ 1.

4:    **n** – Integer   *Input*

On entry: the number of times to expiry to be used.

Constraint: **n** $\geq$ 1.

5:    **sm**[**m**] – const double   *Input*

On entry: **sm**$[i - 1]$ must contain $S_{\min}(i)$, the ith minimum observed price of the underlying asset when **option** = Nag_Call, or $S_{\max}(i)$, the maximum observed price when **option** = Nag_Put, for $i = 1, 2, \ldots, \mathbf{m}$.

Constraints:

        sm$[i - 1] \geq z$ and **sm**$[i - 1] \leq 1/z$, where $z = $ nag_real_safe_small_number, the safe range parameter, for $i = 1, 2, \ldots, \mathbf{m}$;
        if **option** = Nag_Call, **sm**$[i - 1] \leq S$, for $i = 1, 2, \ldots, \mathbf{m}$;
        if **option** = Nag_Put, **sm**$[i - 1] \geq S$, for $i = 1, 2, \ldots, \mathbf{m}$.

6:    **s** – double   *Input*

On entry: S, the price of the underlying asset.

Constraint: **s** $\geq z$ and **s** $\leq 1.0/z$, where $z = $ nag_real_safe_small_number, the safe range parameter.

7: **t**[**n**] – const double *Input*

On entry: **t**$[i-1]$ must contain T_i, the ith time, in years, to expiry, for $i = 1, 2, \ldots, \mathbf{n}$.

Constraint: **t**$[i-1] \geq z$, where $z = \text{nag\_real\_safe\_small\_number}$, the safe range parameter, for $i = 1, 2, \ldots, \mathbf{n}$.

8: **sigma** – double *Input*

On entry: σ, the volatility of the underlying asset. Note that a rate of 15% should be entered as 0.15.

Constraint: **sigma** > 0.0.

9: **r** – double *Input*

On entry: the annual risk-free interest rate, r, continuously compounded. Note that a rate of 5% should be entered as 0.05.

Constraint: $\mathbf{r} \geq 0.0$ and $\text{abs}(\mathbf{r} - \mathbf{q}) > 10 \times \text{eps} \times \max(\text{abs}(\mathbf{r}), 1)$, where eps = nag_machine_precision, the ***machine precision***.

10: **q** – double *Input*

On entry: the annual continuous yield rate. Note that a rate of 8% should be entered as 0.08.

Constraint: $\mathbf{q} \geq 0.0$ and $\text{abs}(\mathbf{r} - \mathbf{q}) > 10 \times \text{eps} \times \max(\text{abs}(\mathbf{r}), 1)$, where eps = nag_machine_precision, the ***machine precision***.

11: **p**[**m** × **n**] – double *Output*

Note: where $\mathbf{P}(i, j)$ appears in this document, it refers to the array element

$\quad$ **p**$[(j-1) \times \mathbf{m} + i - 1]$ when **order** = Nag_ColMajor;
$\quad$ **p**$[(i-1) \times \mathbf{n} + j - 1]$ when **order** = Nag_RowMajor.

On exit: $\mathbf{P}(i, j)$ contains P_{ij}, the option price evaluated for the minimum or maximum observed price $S_{\min}(i)$ or $S_{\max}(i)$ at expiry $\mathbf{t}_j$ for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

12: **delta**[**m** × **n**] – double *Output*

Note: the (i, j)th element of the matrix is stored in

$\quad$ **delta**$[(j-1) \times \mathbf{m} + i - 1]$ when **order** = Nag_ColMajor;
$\quad$ **delta**$[(i-1) \times \mathbf{n} + j - 1]$ when **order** = Nag_RowMajor.

On exit: the $m \times n$ array **delta** contains the sensitivity, $\frac{\partial P}{\partial S}$, of the option price to change in the price of the underlying asset.

13: **gamma**[**m** × **n**] – double *Output*

Note: the (i, j)th element of the matrix is stored in

$\quad$ **gamma**$[(j-1) \times \mathbf{m} + i - 1]$ when **order** = Nag_ColMajor;
$\quad$ **gamma**$[(i-1) \times \mathbf{n} + j - 1]$ when **order** = Nag_RowMajor.

On exit: the $m \times n$ array **gamma** contains the sensitivity, $\frac{\partial^2 P}{\partial S^2}$, of **delta** to change in the price of the underlying asset.

14: **vega**[**m** × **n**] – double *Output*

Note: where $\mathbf{VEGA}(i, j)$ appears in this document, it refers to the array element

$\quad$ **vega**$[(j-1) \times \mathbf{m} + i - 1]$ when **order** = Nag_ColMajor;
$\quad$ **vega**$[(i-1) \times \mathbf{n} + j - 1]$ when **order** = Nag_RowMajor.

On exit: **VEGA**(i, j), contains the first-order Greek measuring the sensitivity of the option price P_{ij} to change in the volatility of the underlying asset, i.e., $\frac{\partial P_{ij}}{\partial \sigma}$, for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

15:   **theta**$[\mathbf{m} \times \mathbf{n}]$ – double   *Output*

Note: where **THETA**(i, j) appears in this document, it refers to the array element

      theta$[(j - 1) \times \mathbf{m} + i - 1]$ when **order** $=$ Nag_ColMajor;
      theta$[(i - 1) \times \mathbf{n} + j - 1]$ when **order** $=$ Nag_RowMajor.

On exit: **THETA**(i, j), contains the first-order Greek measuring the sensitivity of the option price P_{ij} to change in time, i.e., $-\frac{\partial P_{ij}}{\partial T}$, for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$, where $b = r - q$.

16:   **rho**$[\mathbf{m} \times \mathbf{n}]$ – double   *Output*

Note: where **RHO**(i, j) appears in this document, it refers to the array element

      rho$[(j - 1) \times \mathbf{m} + i - 1]$ when **order** $=$ Nag_ColMajor;
      rho$[(i - 1) \times \mathbf{n} + j - 1]$ when **order** $=$ Nag_RowMajor.

On exit: **RHO**(i, j), contains the first-order Greek measuring the sensitivity of the option price P_{ij} to change in the annual risk-free interest rate, i.e., $-\frac{\partial P_{ij}}{\partial r}$, for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

17:   **crho**$[\mathbf{m} \times \mathbf{n}]$ – double   *Output*

Note: where **CRHO**(i, j) appears in this document, it refers to the array element

      crho$[(j - 1) \times \mathbf{m} + i - 1]$ when **order** $=$ Nag_ColMajor;
      crho$[(i - 1) \times \mathbf{n} + j - 1]$ when **order** $=$ Nag_RowMajor.

On exit: **CRHO**(i, j), contains the first-order Greek measuring the sensitivity of the option price P_{ij} to change in the annual cost of carry rate, i.e., $-\frac{\partial P_{ij}}{\partial b}$, for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$, where $b = r - q$.

18:   **vanna**$[\mathbf{m} \times \mathbf{n}]$ – double   *Output*

Note: where **VANNA**(i, j) appears in this document, it refers to the array element

      vanna$[(j - 1) \times \mathbf{m} + i - 1]$ when **order** $=$ Nag_ColMajor;
      vanna$[(i - 1) \times \mathbf{n} + j - 1]$ when **order** $=$ Nag_RowMajor.

On exit: **VANNA**(i, j), contains the second-order Greek measuring the sensitivity of the first-order Greek Δ_{ij} to change in the volatility of the asset price, i.e., $-\frac{\partial \Delta_{ij}}{\partial T} = -\frac{\partial^2 P_{ij}}{\partial S \partial \sigma}$, for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

19:   **charm**$[\mathbf{m} \times \mathbf{n}]$ – double   *Output*

Note: where **CHARM**(i, j) appears in this document, it refers to the array element

      charm$[(j - 1) \times \mathbf{m} + i - 1]$ when **order** $=$ Nag_ColMajor;
      charm$[(i - 1) \times \mathbf{n} + j - 1]$ when **order** $=$ Nag_RowMajor.

On exit: **CHARM**(i, j), contains the second-order Greek measuring the sensitivity of the first-order Greek Δ_{ij} to change in the time, i.e., $-\frac{\partial \Delta_{ij}}{\partial T} = -\frac{\partial^2 P_{ij}}{\partial S \partial T}$, for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

20:   **speed**$[\mathbf{m} \times \mathbf{n}]$ – double   *Output*

Note: where **SPEED**(i, j) appears in this document, it refers to the array element

      speed$[(j - 1) \times \mathbf{m} + i - 1]$ when **order** $=$ Nag_ColMajor;
      speed$[(i - 1) \times \mathbf{n} + j - 1]$ when **order** $=$ Nag_RowMajor.

On exit: **SPEED**(i, j), contains the third-order Greek measuring the sensitivity of the second-order Greek Γ_{ij} to change in the price of the underlying asset, i.e., $-\frac{\partial \Gamma_{ij}}{\partial S} = -\frac{\partial^3 P_{ij}}{\partial S^3}$, for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

21:   **colour**$[\mathbf{m} \times \mathbf{n}]$ – double   *Output*

Note: where **COLOUR**(i, j) appears in this document, it refers to the array element

    colour$[(j - 1) \times \mathbf{m} + i - 1]$ when **order** = Nag_ColMajor;
    colour$[(i - 1) \times \mathbf{n} + j - 1]$ when **order** = Nag_RowMajor.

On exit: **COLOUR**(i, j), contains the third-order Greek measuring the sensitivity of the second-order Greek Γ_{ij} to change in the time, i.e., $-\frac{\partial \Gamma_{ij}}{\partial T} = -\frac{\partial^3 P_{ij}}{\partial S \partial T}$, for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

22:   **zomma**$[\mathbf{m} \times \mathbf{n}]$ – double   *Output*

Note: where **ZOMMA**(i, j) appears in this document, it refers to the array element

    zomma$[(j - 1) \times \mathbf{m} + i - 1]$ when **order** = Nag_ColMajor;
    zomma$[(i - 1) \times \mathbf{n} + j - 1]$ when **order** = Nag_RowMajor.

On exit: **ZOMMA**(i, j), contains the third-order Greek measuring the sensitivity of the second-order Greek Γ_{ij} to change in the volatility of the underlying asset, i.e., $-\frac{\partial \Gamma_{ij}}{\partial \sigma} = -\frac{\partial^3 P_{ij}}{\partial S^2 \partial \sigma}$, for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

23:   **vomma**$[\mathbf{m} \times \mathbf{n}]$ – double   *Output*

Note: where **VOMMA**(i, j) appears in this document, it refers to the array element

    vomma$[(j - 1) \times \mathbf{m} + i - 1]$ when **order** = Nag_ColMajor;
    vomma$[(i - 1) \times \mathbf{n} + j - 1]$ when **order** = Nag_RowMajor.

On exit: **VOMMA**(i, j), contains the second-order Greek measuring the sensitivity of the first-order Greek Δ_{ij} to change in the volatility of the underlying asset, i.e., $-\frac{\partial \Delta_{ij}}{\partial \sigma} = -\frac{\partial^2 P_{ij}}{\partial \sigma^2}$, for $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

24:   **fail** – NagError *   *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6   Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, $\mathbf{m} = $ ⟨*value*⟩.
Constraint: $\mathbf{m} \geq 1$.

On entry, $\mathbf{n} = $ ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 1$.

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_REAL

On entry, $\mathbf{q} = \langle value \rangle$.
Constraint: $\mathbf{q} \geq 0.0$.

On entry, $\mathbf{r} = \langle value \rangle$.
Constraint: $\mathbf{r} \geq 0.0$.

On entry, $\mathbf{s} = \langle value \rangle$.
Constraint: $\mathbf{s} \geq \langle value \rangle$ and $\mathbf{s} \leq \langle value \rangle$.

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

NE_REAL_2

On entry, $\mathbf{r} = \langle value \rangle$ and $\mathbf{q} = \langle value \rangle$.
Constraint: $|\mathbf{r} - \mathbf{q}| > 10 \times \text{eps} \times \max(|\mathbf{r}|, 1)$, where eps is the ***machine precision***.

NE_REAL_ARRAY

On entry, $\mathbf{sm}[\langle value \rangle] = \langle value \rangle$.
Constraint: $\langle value \rangle \leq \mathbf{sm}[i] \leq \langle value \rangle$ for all i.

On entry, $\mathbf{t}[\langle value \rangle] = \langle value \rangle$.
Constraint: $\mathbf{t}[i] \geq \langle value \rangle$ for all i.

On entry with a call option, $\mathbf{sm}[\langle value \rangle] = \langle value \rangle$.
Constraint: for call options, $\mathbf{sm}[i] \leq \langle value \rangle$ for all i.

On entry with a put option, $\mathbf{sm}[\langle value \rangle] = \langle value \rangle$.
Constraint: for put options, $\mathbf{sm}[i] \geq \langle value \rangle$ for all i.

7　Accuracy

The accuracy of the output is dependent on the accuracy of the cumulative Normal distribution function, Φ. This is evaluated using a rational Chebyshev expansion, chosen so that the maximum relative error in the expansion is of the order of the ***machine precision*** (see nag_cumul_normal (s15abc) and nag_erfc (s15adc)). An accuracy close to ***machine precision*** can generally be expected.

8　Parallelism and Performance

nag_lookback_fls_greeks (s30bbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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

None.

10 Example

This example computes the price of a floating-strike lookback put with a time to expiry of 6 months and a stock price of 87. The maximum price observed so far is 100. The risk-free interest rate is 6% per year and the volatility is 30% per year with an annual dividend return of 4%.

10.1 Program Text

```
/* nag_lookback_fls_greeks (s30bbc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nags.h>

int main(void)
{
  /* Integer scalar and array declarations */
  Integer      exit_status = 0;
  Integer      i, j, m, n;
  NagError      fail;
  Nag_CallPut   putnum;
  /* Double scalar and array declarations */
  double       q, r, s, sigma;
  double       *charm = 0, *colour = 0, *crho = 0, *delta = 0, *gamma = 0;
  double       *p = 0, *rho = 0, *sm = 0, *speed = 0, *t = 0, *theta = 0;
  double       *vanna = 0, *vega = 0, *vomma = 0, *zomma = 0;
  /* Character scalar and array declarations */
  char         put[8+1];
  Nag_OrderType order;

  INIT_FAIL(fail);

  printf("nag_lookback_fls_greeks (s30bbc) Example Program Results\n");
  printf("Floating-Strike Lookback\n\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /* Read put */
#ifdef _WIN32
  scanf_s("%8s%*[^\n] ", put, _countof(put));
#else
  scanf("%8s%*[^\n] ", put);
#endif
  /*
   * nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  putnum = (Nag_CallPut) nag_enum_name_to_value(put);
  /* Read sigma, r, jvol */
#ifdef _WIN32
  scanf_s("%lf%lf%lf%lf%*[^\n] ", &s, &sigma, &r, &q);
#else
  scanf("%lf%lf%lf%lf%*[^\n] ", &s, &sigma, &r, &q);
#endif
  /* Read m, n */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &m, &n);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &m, &n);
```

```
#endif
    #ifdef NAG_COLUMN_MAJOR
    #define CHARM(I, J)   charm[(J-1)*m + I-1]
    #define COLOUR(I, J)  colour[(J-1)*m + I-1]
    #define CRHO(I, J)    crho[(J-1)*m + I-1]
    #define DELTA(I, J)   delta[(J-1)*m + I-1]
    #define GAMMA(I, J)   gamma[(J-1)*m + I-1]
    #define P(I, J)       p[(J-1)*m + I-1]
    #define RHO(I, J)     rho[(J-1)*m + I-1]
    #define SPEED(I, J)   speed[(J-1)*m + I-1]
    #define THETA(I, J)   theta[(J-1)*m + I-1]
    #define VANNA(I, J)   vanna[(J-1)*m + I-1]
    #define VEGA(I, J)    vega[(J-1)*m + I-1]
    #define VOMMA(I, J)   vomma[(J-1)*m + I-1]
    #define ZOMMA(I, J)   zomma[(J-1)*m + I-1]
  order = Nag_ColMajor;
    #else
    #define CHARM(I, J)   charm[(I-1)*n + J-1]
    #define COLOUR(I, J)  colour[(I-1)*n + J-1]
    #define CRHO(I, J)    crho[(I-1)*n + J-1]
    #define DELTA(I, J)   delta[(I-1)*n + J-1]
    #define GAMMA(I, J)   gamma[(I-1)*n + J-1]
    #define P(I, J)       p[(I-1)*n + J-1]
    #define RHO(I, J)     rho[(I-1)*n + J-1]
    #define SPEED(I, J)   speed[(I-1)*n + J-1]
    #define THETA(I, J)   theta[(I-1)*n + J-1]
    #define VANNA(I, J)   vanna[(I-1)*n + J-1]
    #define VEGA(I, J)    vega[(I-1)*n + J-1]
    #define VOMMA(I, J)   vomma[(I-1)*n + J-1]
    #define ZOMMA(I, J)   zomma[(I-1)*n + J-1]
  order = Nag_RowMajor;
    #endif
  if (!(charm = NAG_ALLOC(m*n, double)) ||
      !(colour = NAG_ALLOC(m*n, double)) ||
      !(crho = NAG_ALLOC(m*n, double)) ||
      !(delta = NAG_ALLOC(m*n, double)) ||
      !(gamma = NAG_ALLOC(m*n, double)) ||
      !(p = NAG_ALLOC(m*n, double)) ||
      !(rho = NAG_ALLOC(m*n, double)) ||
      !(sm = NAG_ALLOC(m, double)) ||
      !(speed = NAG_ALLOC(m*n, double)) ||
      !(t = NAG_ALLOC(n, double)) ||
      !(theta = NAG_ALLOC(m*n, double)) ||
      !(vanna = NAG_ALLOC(m*n, double)) ||
      !(vega = NAG_ALLOC(m*n, double)) ||
      !(vomma = NAG_ALLOC(m*n, double)) ||
      !(zomma = NAG_ALLOC(m*n, double)))
    {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
    }
  /* Read array of min/max prices, SM */
  for (i = 0; i < m; i++)
#ifdef _WIN32
    scanf_s("%lf ", &sm[i]);
#else
    scanf("%lf ", &sm[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /* Read array of times to expiry */
  for (i = 0; i < n; i++)
#ifdef _WIN32
    scanf_s("%lf ", &t[i]);
#else
    scanf("%lf ", &t[i]);
#endif
```

```
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /*
   * nag_lookback_fls_greeks (s30bbc)
   * Floating-strike lookback option pricing formula with Greeks
   */
  nag_lookback_fls_greeks(order, putnum, m, n, sm, s, t, sigma, r, q, p,
                          delta, gamma, vega, theta, rho, crho, vanna, charm,
                          speed, colour, zomma, vomma, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_lookback_fls_greeks (s30bbc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
  if (putnum == Nag_Call)
    printf("European Call :\n\n");
  else if (putnum == Nag_Put)
    printf("European Put :\n\n");
  printf("%s%8.4f\n", "  Spot      = ", s);
  printf("%s%8.4f\n", "  Volatility = ", sigma);
  printf("%s%8.4f\n", "  Rate      = ", r);
  printf("%s%8.4f\n", "  Dividend  = ", q);
  printf("\n");
  for (j = 1; j <= n; j++)
    {
      printf("\n");
      printf("  Time to Expiry :   %8.4f\n", t[j-1]);
      printf("   Strike    Price    Delta     Gamma      Vega   "
             "  Theta      Rho     CRho\n");
      for (i = 1; i <= m; i++)
        printf("%9.4f %9.4f %9.4f %9.4f %9.4f %9.4f %9.4f %9.4f\n",
               sm[i-1], P(i, j), DELTA(i, j), GAMMA(i, j), VEGA(i, j),
               THETA(i, j), RHO(i, j), CRHO(i, j));
      printf("                            Vanna     Charm     Speed   "
             "Colour    Zomma     Vomma\n");
      for (i = 1; i <= m; i++)
        printf("%29.4f %9.4f %9.4f %9.4f %9.4f %9.4f\n", VANNA(i, j),
               CHARM(i, j), SPEED(i, j), COLOUR(i, j), ZOMMA(i, j),
               VOMMA(i, j));
    }

 END:
 NAG_FREE(charm);
 NAG_FREE(colour);
 NAG_FREE(crho);
 NAG_FREE(delta);
 NAG_FREE(gamma);
 NAG_FREE(p);
 NAG_FREE(rho);
 NAG_FREE(sm);
 NAG_FREE(speed);
 NAG_FREE(t);
 NAG_FREE(theta);
 NAG_FREE(vanna);
 NAG_FREE(vega);
 NAG_FREE(vomma);
 NAG_FREE(zomma);

  return exit_status;
}
```

10.2 Program Data

```
nag_lookback_fls_greeks (s30bbc) Example Program Data
 Nag_Put                : Nag_Call or Nag_Put
 87.0 0.3 0.06 0.04     : s, sigma, r, q
 1 1                    : m, n
 100.0                  : SM(I), I = 1,2,...m
 0.5                    : T(I), I = 1,2,...n
```

10.3 Program Results

```
nag_lookback_fls_greeks (s30bbc) Example Program Results
Floating-Strike Lookback

European Put :

  Spot       =   87.0000
  Volatility =    0.3000
  Rate       =    0.0600
  Dividend   =    0.0400


  Time to Expiry :      0.5000
   Strike      Price      Delta      Gamma       Vega      Theta        Rho       CRho
 100.0000    18.3530    -0.3560     0.0391    45.5353   -11.6139   -32.8139   -23.6374
                         Vanna      Charm      Speed     Colour      Zomma      Vomma
                        1.9141    -0.6199     0.0007     0.0221    -0.0648    76.1292
```