

NAG Library Function Document

nag_binary_con_greeks (s30cbc)

1 Purpose

nag_binary_con_greeks (s30cbc) computes the price of a binary or digital cash-or-nothing option together with its sensitivities (Greeks).

2 Specification

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

void nag_binary_con_greeks (Nag_OrderType order, Nag_CallPut option,
    Integer m, Integer n, const double x[], double s, double k,
    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_binary_con_greeks (s30cbc) computes the price of a binary or digital cash-or-nothing 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. This option pays a fixed amount, K , at expiration if the option is in-the-money (see Section 2.4 in the s Chapter Introduction). For a strike price, X , underlying asset price, S , and time to expiry, T , the payoff is therefore K , if $S > X$ for a call or $S < X$ for a put. Nothing is paid out when this condition is not met.

The price of a call with volatility, σ , risk-free interest rate, r , and annualised dividend yield, q , is

$$P_{\text{call}} = Ke^{-rT}\Phi(d_2)$$

and for a put,

$$P_{\text{put}} = Ke^{-rT}\Phi(-d_2)$$

where Φ is the cumulative Normal distribution function,

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp(-y^2/2) dy,$$

and

$$d_2 = \frac{\ln(S/X) + (r - q - \sigma^2/2)T}{\sigma\sqrt{T}}.$$

The option price $P_{ij} = P(X = X_i, T = T_j)$ is computed for each strike price in a set X_i , $i = 1, 2, \dots, m$, and for each expiry time in a set T_j , $j = 1, 2, \dots, n$.

4 References

Reiner E and Rubinstein M (1991) Unscrambling the binary code *Risk* 4

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 2.3.1.3 in How to Use the NAG Library and its Documentation 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 strike prices to be used.
Constraint: **m** ≥ 1 .

- 4: **n** – Integer *Input*
On entry: the number of times to expiry to be used.
Constraint: **n** ≥ 1 .

- 5: **x[m]** – const double *Input*
On entry: **x**[$i - 1$] must contain X_i , the i th strike price, for $i = 1, 2, \dots, \mathbf{m}$.
Constraint: **x**[$i - 1$] $\geq z$ and **x**[$i - 1$] $\leq 1/z$, where $z = \text{nag_real_safe_small_number}$, the safe range parameter, for $i = 1, 2, \dots, \mathbf{m}$.

- 6: **s** – double *Input*
On entry: S , the price of the underlying asset.
Constraint: **s** $\geq z$ and **s** $\leq 1.0/z$, where $z = \text{nag_real_safe_small_number}$, the safe range parameter.

- 7: **k** – double *Input*
On entry: the amount, K , to be paid at expiration if the option is in-the-money, i.e., if **s** $>$ **x**[$i - 1$] when **option** = Nag_Call, or if **s** $<$ **x**[$i - 1$] when **option** = Nag_Put, for $i = 1, 2, \dots, \mathbf{m}$.
Constraint: **k** ≥ 0.0 .

- 8: **t[n]** – const double *Input*
On entry: **t**[$i - 1$] must contain T_i , the i th time, in years, to expiry, for $i = 1, 2, \dots, \mathbf{n}$.
Constraint: **t**[$i - 1$] $\geq z$, where $z = \text{nag_real_safe_small_number}$, the safe range parameter, for $i = 1, 2, \dots, \mathbf{n}$.

- 9: **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.
- 10: **r** – double *Input*
On entry: r , the annual risk-free interest rate, continuously compounded. Note that a rate of 5% should be entered as 0.05.
Constraint: **r** ≥ 0.0.
- 11: **q** – double *Input*
On entry: q , the annual continuous yield rate. Note that a rate of 8% should be entered as 0.08.
Constraint: **q** ≥ 0.0.
- 12: **p**[**m** × **n**] – double *Output*
Note: where **P**(i, j) appears in this document, it refers to the array element

$$\mathbf{p}[(j-1) \times \mathbf{m} + i - 1] \text{ when } \mathbf{order} = \text{Nag_ColMajor};$$

$$\mathbf{p}[(i-1) \times \mathbf{n} + j - 1] \text{ when } \mathbf{order} = \text{Nag_RowMajor}.$$
On exit: **P**(i, j) contains P_{ij} , the option price evaluated for the strike price \mathbf{x}_i at expiry \mathbf{t}_j for $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$.
- 13: **delta**[**m** × **n**] – double *Output*
Note: the (i, j)th element of the matrix is stored in

$$\mathbf{delta}[(j-1) \times \mathbf{m} + i - 1] \text{ when } \mathbf{order} = \text{Nag_ColMajor};$$

$$\mathbf{delta}[(i-1) \times \mathbf{n} + j - 1] \text{ when } \mathbf{order} = \text{Nag_RowMajor}.$$
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.
- 14: **gamma**[**m** × **n**] – double *Output*
Note: the (i, j)th element of the matrix is stored in

$$\mathbf{gamma}[(j-1) \times \mathbf{m} + i - 1] \text{ when } \mathbf{order} = \text{Nag_ColMajor};$$

$$\mathbf{gamma}[(i-1) \times \mathbf{n} + j - 1] \text{ when } \mathbf{order} = \text{Nag_RowMajor}.$$
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.
- 15: **vega**[**m** × **n**] – double *Output*
Note: where **VEGA**(i, j) appears in this document, it refers to the array element

$$\mathbf{vega}[(j-1) \times \mathbf{m} + i - 1] \text{ when } \mathbf{order} = \text{Nag_ColMajor};$$

$$\mathbf{vega}[(i-1) \times \mathbf{n} + j - 1] \text{ when } \mathbf{order} = \text{Nag_RowMajor}.$$
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 $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$.
- 16: **theta**[**m** × **n**] – double *Output*
Note: where **THETA**(i, j) appears in this document, it refers to the array element

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

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

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 $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$, where $b = r - q$.

17: **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 $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$.

18: **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 $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$, where $b = r - q$.

19: **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 $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$.

20: **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 $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$.

21: **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 $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$.

22: **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 $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$.

23: **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 $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$.

24: **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 $i = 1, 2, \dots, \mathbf{m}$ and $j = 1, 2, \dots, \mathbf{n}$.

25: **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_INT

On entry, $\mathbf{m} = \langle value \rangle$.

Constraint: $\mathbf{m} \geq 1$.

On entry, $\mathbf{n} = \langle value \rangle$.

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

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

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

Constraint: **k** \geq 0.0.

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

Constraint: **q** \geq 0.0.

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

Constraint: **r** \geq 0.0.

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

Constraint: **s** $\geq \langle value \rangle$ and **s** $\leq \langle value \rangle$.

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

Constraint: **sigma** $>$ 0.0.

NE_REAL_ARRAY

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

Constraint: **t**[i] $\geq \langle value \rangle$.

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

Constraint: **x**[i] $\geq \langle value \rangle$ and **x**[i] $\leq \langle value \rangle$.

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_binary_con_greeks (s30cbc) 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 cash-or-nothing call with a time to expiry of 0.75 years, a stock price of 110 and a strike price of 87. The risk-free interest rate is 5% per year, there is an annual dividend return of 4% and the volatility is 35% per year. If the option is in-the-money at expiration, i.e., if $S > X$, the payoff is 5.

10.1 Program Text

```
/* nag_binary_con_greeks (s30cbc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#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 k, q, r, s, sigma;
    double *charm = 0, *colour = 0, *crho = 0, *delta = 0, *gamma = 0;
    double *p = 0, *rho = 0, *speed = 0, *t = 0, *theta = 0, *vanna = 0;
    double *vega = 0, *vomma = 0, *x = 0, *zomma = 0;
    /* Character scalar and array declarations */
    char put[8 + 1];
    Nag_OrderType order;

    INIT_FAIL(fail);

    printf("nag_binary_con_greeks (s30cbc) Example Program Results\n");
    printf("Binary (Digital): Cash-or-Nothing\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, (unsigned)_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 s, k, sigma, r, q */
#ifdef _WIN32
    scanf_s("%lf%lf%lf%lf%lf%*[\n] ", &s, &k, &sigma, &r, &q);
#else
    scanf("%lf%lf%lf%lf%lf%*[\n] ", &s, &k, &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)) ||
        !(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)) ||
        !(x = NAG_ALLOC(m, double)) || !(zomma = NAG_ALLOC(m * n, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Read array of strike/exercise prices, X */
    for (i = 0; i < m; i++)
#ifdef _WIN32
        scanf_s("%lf ", &x[i]);
#else
        scanf("%lf ", &x[i]);
#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
    /* nag_binary_con_greeks (s30cbc)
       * Binary option: Cash-or-nothing pricing formula with Greeks
       */
    nag_binary_con_greeks(order, putnum, m, n, x, s, k, 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_binary_con_greeks (s30cbc).\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", " Payout    = ", k);
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("%8.4f%8.4f%8.4f%8.4f%8.4f%8.4f%8.4f \n",
               x[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("%24.4f%8.4f%8.4f%8.4f%8.4f%8.4f \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(speed);
NAG_FREE(t);
NAG_FREE(theta);
NAG_FREE(vanna);
NAG_FREE(vega);
NAG_FREE(vomma);
NAG_FREE(x);
NAG_FREE(zomma);

return exit_status;
}

```

10.2 Program Data

```

nag_binary_con_greeks (s30cbc) Example Program Data
Nag_Call                : Nag_Call or Nag_Put
110.0 5.0 0.35 0.05 0.04 : s, k, sigma, r, q
1 1                      : m, n
87.0                    : X(I), I = 1,2,...m
0.75                    : T(I), I = 1,2,...n

```

10.3 Program Results

nag_binary_con_greeks (s30cbc) Example Program Results
 Binary (Digital): Cash-or-Nothing

European Call :

Spot = 110.0000
 Payout = 5.0000
 Volatility = 0.3500
 Rate = 0.0500
 Dividend = 0.0400

Time to Expiry :	0.7500						
Strike	Price	Delta	Gamma	Vega	Theta	Rho	CRho
87.0000	3.5696	0.0467	-0.0013	-4.2307	1.1142	1.1788	3.8560
		Vanna	Charm	Speed	Colour	Zomma	Vomma
		-0.0514	0.0153	0.0000	-0.0019	0.0079	12.8874
