

# NAG Library Function Document

## nag\_binary\_con\_price (s30cac)

### 1 Purpose

nag\_binary\_con\_price (s30cac) computes the price of a binary or digital cash-or-nothing option.

### 2 Specification

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

void nag_binary_con_price (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[],
    NagError *fail)
```

### 3 Description

nag\_binary\_con\_price (s30cac) computes the price of a binary or digital cash-or-nothing option which 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,  $\sigma$ , 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  $\Phi$  is the cumulative Normal distribution function,

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

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**  $\geq 1$ .
- 4: **n** – Integer *Input*  
*On entry:* the number of times to expiry to be used.  
*Constraint:* **n**  $\geq 1$ .
- 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**  $> \mathbf{x}[i - 1]$  when **option** = Nag\_Call, or if **s**  $< \mathbf{x}[i - 1]$  when **option** = Nag\_Put, for  $i = 1, 2, \dots, m$ .  
*Constraint:* **k**  $\geq 0.0$ .
- 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:*  $\sigma$ , 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**  $\geq 0.0$ .

- 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:*  $q \geq 0.0$ .
- 12: **p[m × n]** – double *Output*  
**Note:** where  $\mathbf{P}(i, j)$  appears in this document, it refers to the array element  
 $\mathbf{p}[(j-1) \times \mathbf{m} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{p}[(i-1) \times \mathbf{n} + j - 1]$  when **order** = Nag\_RowMajor.  
*On exit:*  $\mathbf{P}(i, j)$  contains  $P_{ij}$ , the option price evaluated for the 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: **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,  $\mathbf{k} = \langle value \rangle$ .

Constraint:  $\mathbf{k} \geq 0.0$ .

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,  $s = \langle value \rangle$ .  
 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,  $\Phi$ . 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\_price (s30cac) 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 put with a time to expiry of 0.75 years, a stock price of 100 and a strike price of 80. The risk-free interest rate is 6% per year and the volatility is 35% per year. If the option is in-the-money at expiration, i.e., if  $S > X$ , the payoff is 10.

### 10.1 Program Text

```
/* nag_binary_con_price (s30cac) 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 *p = 0, *t = 0, *x = 0;
/* Character scalar and array declarations */
char put[8 + 1];
Nag_OrderType order;

INIT_FAIL(fail);

printf("nag_binary_con_price (s30cac) 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 P(I, J) p[(J-1)*m + I-1]
order = Nag_ColMajor;
#else
#define P(I, J) p[(I-1)*n + J-1]
order = Nag_RowMajor;
#endif
if (!(p = NAG_ALLOC(m * n, double)) ||
    !(t = NAG_ALLOC(n, double)) || !(x = NAG_ALLOC(m, 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
#ifdef _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif
for (i = 0; i < n; i++)
#ifdef _WIN32
scanf_s("%lf ", &t[i]);
#else
scanf("%lf ", &t[i]);
#endif

```

```

        scanf("%lf ", &t[i]);
#endif
#ifdef _WIN32
        scanf_s("%*[^\\n] ");
#else
        scanf("%*[^\\n] ");
#endif
/*
 * nag_binary_con_price (s30cac)
 * Binary option: cash-or-nothing pricing formula
 */
nag_binary_con_price(order, putnum, m, n, x, s, k, t, sigma, r, q, p,
                    &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_binary_con_price (s30cac).\\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");
printf("%s\\n", " Strike Expiry Option Price");
for (i = 1; i <= m; i++)
    for (j = 1; j <= n; j++)
        printf("%9.4f %9.4f %12.4f\\n", x[i - 1], t[j - 1], P(i, j));

END:
NAG_FREE(p);
NAG_FREE(t);
NAG_FREE(x);

return exit_status;
}

```

## 10.2 Program Data

nag\_binary\_con\_price (s30cac) Example Program Data

Nag_Put	:	Nag_Call or Nag_Put
100.0 10.0 0.35 0.06 0.0	:	s, k, sigma, r, q
1 1	:	m, n
80.0	:	X(I), I = 1,2,...m
0.75	:	T(I), I = 1,2,...n

## 10.3 Program Results

nag\_binary\_con\_price (s30cac) Example Program Results  
Binary (Digital): Cash-or-Nothing

European Put :

Spot	=	100.0000
Payout	=	10.0000
Volatility	=	0.3500
Rate	=	0.0600
Dividend	=	0.0000
Strike	Expiry	Option Price
80.0000	0.7500	2.2155

---