NAG Library Function Document

nag_barrier_std_price (s30fac)

1 Purpose

nag_barrier_std_price (s30fac) computes the price of a standard barrier option.

2 Specification

```
#include <nag.h>
#include <nags.h>
void nag_barrier_std_price (Nag_OrderType order, Nag_CallPut option,
     Nag_Barrier type, Integer m, Integer n, const double x[], double s,
     double h, double k, const double t[], double sigma, double r, double q,
     double p[], NagError *fail)
```

3 Description

nag_barrier_std_price (s30fac) computes the price of a standard barrier option, where the exercise, for a given strike price, X, depends on the underlying asset price, S, reaching or crossing a specified barrier level, H. Barrier options of type **In** only become active (are knocked in) if the underlying asset price attains the pre-determined barrier level during the lifetime of the contract. Those of type **Out** start active and are knocked out if the underlying asset price attains the barrier level during the lifetime of the contract. A cash rebate, K, may be paid if the option is inactive at expiration. The option may also be described as **Up** (the underlying price starts below the barrier level) or **Down** (the underlying price starts above the barrier level). This gives the following options which can be specified as put or call contracts.

Down-and-In: the option starts inactive with the underlying asset price above the barrier level. It is knocked in if the underlying price moves down to hit the barrier level before expiration.

Down-and-Out: the option starts active with the underlying asset price above the barrier level. It is knocked out if the underlying price moves down to hit the barrier level before expiration.

Up-and-In: the option starts inactive with the underlying asset price below the barrier level. It is knocked in if the underlying price moves up to hit the barrier level before expiration.

Up-and-Out: the option starts active with the underlying asset price below the barrier level. It is knocked out if the underlying price moves up to hit the barrier level before expiration.

The payoff is $\max(S - X, 0)$ for a call or $\max(X - S, 0)$ for a put, if the option is active at expiration, otherwise it may pay a pre-specified cash rebate, K. Following Haug (2007), the prices of the various standard barrier options can be written as shown below. The volatility, σ, risk-free interest rate, r, and annualised dividend yield, q, are constants. The integer parameters, j and k, take the values ± 1, depending on the type of barrier.

$$
\begin{aligned}
A &= jSe^{-qT}\Phi(jx_1) - jXe^{-rT}\Phi\big(j\big[x_1 - \sigma\sqrt{T}\big]\big)\\
B &= jSe^{-qT}\Phi(jx_2) - jXe^{-rT}\Phi\big(j\big[x_2 - \sigma\sqrt{T}\big]\big)\\
C &= jSe^{-qT}\big(\tfrac{H}{S}\big)^{2(\mu+1)}\Phi(ky_1) - jXe^{-rT}\big(\tfrac{H}{S}\big)^{2\mu}\Phi\big(k\big[y_1 - \sigma\sqrt{T}\big]\big)\\
D &= jSe^{-qT}\big(\tfrac{H}{S}\big)^{2(\mu+1)}\Phi(ky_2) - jXe^{-rT}\big(\tfrac{H}{S}\big)^{2\mu}\Phi\big(k\big[y_2 - \sigma\sqrt{T}\big]\big)\\
E &= Ke^{-rT}\Big\{\Phi\big(k\big[x_2 - \sigma\sqrt{T}\big]\big) - \big(\tfrac{H}{S}\big)^{2\mu}\Phi\big(k\big[y_2 - \sigma\sqrt{T}\big]\big)\Big\}\\
F &= K\Big\{\big(\tfrac{H}{S}\big)^{\mu+\lambda}\Phi(kz) + \big(\tfrac{H}{S}\big)^{\mu-\lambda}\Phi\big(k\big[z - \sigma\sqrt{T}\big]\big)\Big\}
\end{aligned}
$$

with

$$
\begin{aligned}
x_1 &= \frac{\ln(S/X)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T} \\
x_2 &= \frac{\ln(S/H)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T} \\
y_1 &= \frac{\ln\left(H^2/(SX)\right)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T} \\
y_2 &= \frac{\ln(H/S)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T} \\
z &= \frac{\ln(H/S)}{\sigma\sqrt{T}} + \lambda\sigma\sqrt{T} \\
\mu &= \frac{r-q-\sigma^2/2}{\sigma^2} \\
\lambda &= \sqrt{\mu^2 + \frac{2r}{\sigma^2}}
\end{aligned}
$$

and where Φ denotes the cumulative Normal distribution function,

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

Down-and-In ($S > H$):

When $X \geq H$, with $j = k = 1$,

$$P_{\text{call}} = C + E$$

and with $j = -1,\ k = 1$

$$P_{\text{put}} = B - C + D + E$$

When $X < H$, with $j = k = 1$

$$P_{\text{call}} = A - B + D + E$$

and with $j = -1,\ k = 1$

$$P_{\text{put}} = A + E.$$

Down-and-Out ($S > H$):

When $X \geq H$, with $j = k = 1$,

$$P_{\text{call}} = A - C + F$$

and with $j = -1,\ k = 1$

$$P_{\text{put}} = A - B + C - D + F$$

When $X < H$, with $j = k = 1$,

$$P_{\text{call}} = B - D + F$$

and with $j = -1,\ k = 1$

$$P_{\text{put}} = F.$$

Up-and-In ($S < H$):

When $X \geq H$, with $j = 1,\ k = -1$,

$$P_{\text{call}} = A + E$$

and with $j = k = -1$,

$$P_{\text{put}} = A - B + D + E$$

When $X < H$, with $j = 1$, $k = -1$,

$$P_{\text{call}} = B - C + D + E$$

and with $j = k = -1$,

$$P_{\text{put}} = C + E.$$

Up-and-Out ($S < H$):

When $X \geq H$, with $j = 1$, $k = -1$,

$$P_{\text{call}} = F$$

and with $j = k = -1$,

$$P_{\text{put}} = B - D + F$$

When $X < H$, with $j = 1$, $k = -1$,

$$P_{\text{call}} = A - B + C - D + F$$

and with $j = k = -1$,

$$P_{\text{put}} = A - C + F.$$

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

4 References

Haug E G (2007) *The Complete Guide to Option Pricing Formulas* (2nd Edition) McGraw-Hill

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: **type** – Nag_Barrier *Input*

On entry: indicates the barrier type as **In** or **Out** and its relation to the price of the underlying asset as **Up** or **Down**.

type = Nag_DownandIn
Down-and-In.

type = Nag_DownandOut
Down-and-Out.

type = Nag_UpandIn
Up-and-In.

type = Nag_UpandOut
Up-and-Out.

Constraint: **type** = Nag_DownandIn, Nag_DownandOut, Nag_UpandIn or Nag_UpandOut.

4: **m** – Integer *Input*

On entry: the number of strike prices to be used.

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

5: **n** – Integer *Input*

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

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

6: **x**[**m**] – const double *Input*

On entry: $\mathbf{x}[i-1]$ must contain X_i, the ith strike price, for $i = 1, 2, \ldots, \mathbf{m}$.

Constraint: $\mathbf{x}[i-1] \geq z$ and $\mathbf{x}[i-1] \leq 1/z$, where $z =$ nag_real_safe_small_number, the safe range parameter, for $i = 1, 2, \ldots, \mathbf{m}$.

7: **s** – double *Input*

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

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

8: **h** – double *Input*

On entry: the barrier price.

Constraint: $\mathbf{h} \geq z$ and $\mathbf{h} \leq 1/z$, where $z =$ nag_real_safe_small_number, the safe range parameter.

9: **k** – double *Input*

On entry: the value of a possible cash rebate to be paid if the option has not been knocked in (or out) before expiration.

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

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

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

Constraint: $\mathbf{t}[i-1] \geq z$, where $z =$ nag_real_safe_small_number, the safe range parameter, for $i = 1, 2, \ldots, \mathbf{n}$.

11: **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.

12: **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: $\mathbf{r} \geq 0.0$.

13: **q** – double *Input*

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

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

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

15: **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_BAD_PARAM

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

NE_ENUM_REAL_2

On entry, **s** and **h** are inconsistent with **type**: **s** = ⟨*value*⟩ and **h** = ⟨*value*⟩.

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 1.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 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.

NE_REAL

On entry, **h** = ⟨*value*⟩.
Constraint: **h** ≥ ⟨*value*⟩ and **h** ≤ ⟨*value*⟩.

On entry, **k** = ⟨*value*⟩.
Constraint: **k** ≥ 0.0.

On entry, **q** = ⟨*value*⟩.
Constraint: **q** ≥ 0.0.

On entry, **r** = ⟨*value*⟩.
Constraint: **r** ≥ 0.0.

On entry, **s** = ⟨*value*⟩.
Constraint: **s** ≥ ⟨*value*⟩ and **s** ≤ ⟨*value*⟩.

On entry, **sigma** = ⟨*value*⟩.
Constraint: **sigma** > 0.0.

NE_REAL_ARRAY

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

On entry, $\mathbf{x}[\langle value \rangle] = \langle value \rangle$.
Constraint: $\mathbf{x}[i] \geq \langle value \rangle$ and $\mathbf{x}[i] \leq \langle value \rangle$.

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_barrier_std_price (s30fac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please 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 Down-and-In put with a time to expiry of 6 months, a stock price of 100 and a strike price of 100. The barrier value is 95 and there is a cash rebate of 3, payable on expiry if the option has not been knocked in. The risk-free interest rate is 8% per year, there is an annual dividend return of 4% and the volatility is 30% per year.

10.1 Program Text

```
/* nag_barrier_std_price (s30fac) Example Program.
 *
 * Copyright 2009, 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;
  Nag_Barrier   typenum;
  /* Double scalar and array declarations */
  double        h, k, q, r, s, sigma;
  double        *p = 0, *t = 0, *x = 0;
  /* Character scalar and array declarations */
  char          put[8+1];
  char          type[14+1];
  Nag_OrderType order;
```

```
  INIT_FAIL(fail);

  printf("nag_barrier_std_price (s30fac) Example Program Results\n");
  printf("Standard Barrier Option\n\n");
  /* Skip heading in data file */
  scanf("%*[^\n] ");
  /* Read put, type */
  scanf("%8s%14s%*[^\n] ", put, type);
  /*
   * nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  putnum = (Nag_CallPut) nag_enum_name_to_value(put);
  typenum = (Nag_Barrier) nag_enum_name_to_value(type);
  /* Read s, h, k, sigma, r, q */
  scanf("%lf%lf%lf%lf%lf%lf%*[^\n] ", &s, &h, &k, &sigma, &r, &q);
  /* Read m, n */
  scanf("%ld%ld%*[^\n] ", &m, &n);
    #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++)
    scanf("%lf ", &x[i]);
  scanf("%*[^\n] ");
  /* Read array of times to expiry */
  for (i = 0; i < n; i++)
    scanf("%lf ", &t[i]);
  scanf("%*[^\n] ");
  /*
   * nag_barrier_std_price (s30fac)
   * Standard Barrier option pricing formula
   */
  nag_barrier_std_price(order, putnum, typenum, m, n, x, s, h, k, t, sigma, r,
                        q, p, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_barrier_std_price (s30fac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  if (putnum == Nag_Call)
    printf("%s\n", "Call :");
  else if (putnum == Nag_Put)
    printf("%s\n", "Put :");
  if (typenum == Nag_DownandIn)
    printf("%s\n\n", "Down-and-In");
  else if (typenum == Nag_DownandOut)
    printf("%s\n\n", "Down-and-Out");
  else if (typenum == Nag_UpandIn)
    printf("%s\n\n", "Up-and-In");
  else if (typenum == Nag_UpandOut)
    printf("%s\n\n", "Up-and-Out");
  printf("%s%8.4f\n", "  Spot       = ", s);
  printf("%s%8.4f\n", "  Barrier    = ", h);
  printf("%s%8.4f\n", "  Rebate     = ", 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 %11.4f\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_barrier_std_price (s30fac) Example Program Data
 Nag_Put  Nag_DownandIn       : Put and Down-and-in barrier
 100.0 95.0 3.0 0.3 0.08 0.04 : s, h, k, sigma, r, q
 1 1                          : m, n
 100.0                        : X(I), I = 1,2,...m
 0.5                          : T(I), I = 1,2,...n
```

10.3 Program Results

```
nag_barrier_std_price (s30fac) Example Program Results
Standard Barrier Option

Put :
Down-and-In

  Spot       = 100.0000
  Barrier    =  95.0000
  Rebate     =   3.0000
  Volatility =   0.3000
  Rate       =   0.0800
  Dividend   =   0.0400

   Strike   Expiry   Option Price
 100.0000   0.5000      7.7988
```