NAG Library Function Document

nag_amer_bs_price (s30qcc)

1 Purpose

nag_amer_bs_price (s30qcc) computes the Bjerksund and Stensland (2002) approximation to the price of an American option.

2 Specification

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

3 Description

nag_amer_bs_price (s30qcc) computes the price of an American option using the closed form approximation of Bjerksund and Stensland (2002). The time to maturity, T, is divided into two periods, each with a flat early exercise boundary, by choosing a time $t \in [0,T]$, such that $t = \frac{1}{2}\left(\sqrt{5}-1\right)T$. The two boundary values are defined as $\tilde{x} = \tilde{X}(t)$, $\tilde{X} = \tilde{X}(T)$ with

$$\tilde{X}(\tau) = B_0 + (B_\infty - B_0)(1 - \exp\{h(\tau)\}),$$

where

$$h(\tau) = -\left(b\tau + 2\sigma\sqrt{\tau}\right)\left(\frac{X^2}{(B_\infty - B_0)B_0}\right),$$

$$B_\infty \equiv \frac{\beta}{\beta - 1}X, \quad B_0 \equiv \max\left\{X, \left(\frac{r}{r-b}\right)X\right\},$$

$$\beta = \left(\frac{1}{2} - \frac{\mathrm{b}}{\sigma^2}\right) + \sqrt{\left(\frac{b}{\sigma^2} - \frac{1}{2}\right)^2 + 2\frac{r}{\sigma^2}}.$$

with $b = r - q$, the cost of carry, where r is the risk-free interest rate and q is the annual dividend rate. Here X is the strike price and σ is the annual volatility.

The price of an American call option is approximated as

$$\begin{aligned} P_{\text{call}} \quad = \quad & \alpha\left(\tilde{X}\right)S^\beta - \alpha\left(\tilde{X}\right)\phi\left(S,t|\beta,\tilde{X},\tilde{X}\right) + \\ & \phi\left(S,t|1,\tilde{X},\tilde{X}\right) - \phi\left(S,t|1,\tilde{x},\tilde{X}\right) - \\ & X\phi\left(S,t|0,\tilde{X},\tilde{X}\right) + X\phi\left(S,t|0,\tilde{x},\tilde{X}\right) + \\ & \alpha(\tilde{x})\phi\left(S,t|\beta,\tilde{x},\tilde{X}\right) - \alpha(\tilde{x})\Psi\left(S,T|\beta,\tilde{x},\tilde{X},\tilde{x},t\right) + \\ & \Psi\left(S,T|1,\tilde{x},\tilde{X},\tilde{x},t\right) - \Psi\left(S,T|1,X,\tilde{X},\tilde{x},t\right) - \\ & X\Psi\left(S,T|0,\tilde{x},\tilde{X},\tilde{x},t\right) + X\Psi\left(S,T|0,X,\tilde{X},\tilde{x},t\right), \end{aligned}$$

where α, ϕ and Ψ are as defined in Bjerksund and Stensland (2002).

The price of a put option is obtained by the put-call transformation,

$$P_{\text{put}}(X,S,T,\sigma,r,q) = P_{\text{call}}(S,X,T,\sigma,q,r).$$

The option price $P_{ij} = P\left(X = X_i, T = T_j\right)$ 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

Bjerksund P and Stensland G (2002) Closed form valuation of American options **Discussion Paper 2002/09** *NHH Bergen Norway* http://www.nhh.no/

Genz A (2004) Numerical computation of rectangular bivariate and trivariate Normal and t probabilities *Statistics and Computing* **14** 151–160

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 strike prices to be used.

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

4: **n** – Integer *Input*

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

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

5: **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 $i = 1, 2, \ldots, \mathbf{m}$.

6: **s** – double *Input*

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

Constraint: $\mathbf{s} \geq z$ and $\mathbf{s} \leq \frac{1}{z}$, where $z =$ nag_real_safe_small_number, the safe range parameter and $\mathbf{s}^{\beta} < \frac{1}{z}$ where β is as defined in Section 3.

7: **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 $i = 1, 2, \ldots, \mathbf{n}$.

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: 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.

10: **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.

11: **p**[**m** $\times$ **n**] – double *Output*

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

p$[(j-1) \times \mathbf{m} + i - 1]$ when **order** = Nag_ColMajor;
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 $i = 1, 2, \ldots,$ **m** and $j = 1, 2, \ldots,$ **n**.

12: **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_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, **q** = ⟨*value*⟩.
Constraint: **q** ≥ 0.0.

On entry, **r** = ⟨*value*⟩.
Constraint: **r** ≥ 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, $\mathbf{s} = \langle value \rangle$ and $\beta = \langle value \rangle$.
Constraint: $\mathbf{s}^{\beta} < \langle value \rangle$.

On entry, **sigma** $= \langle value \rangle$.
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 will be bounded by the accuracy of the cumulative bivariate Normal distribution function. The algorithm of Genz (2004) is used, as described in the document for nag_bivariate_normal_dist (g01hac), giving a maximum absolute error of less than 5×10^{-16}. The univariate cumulative Normal distribution function also forms part of the evaluation (see nag_cumul_normal (s15abc) and nag_erfc (s15adc)).

8 Parallelism and Performance

nag_amer_bs_price (s30qcc) 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 an American call with a time to expiry of 3 months, a stock price of 110 and a strike price of 100. The risk-free interest rate is 8% per year, there is an annual dividend return of 12% and the volatility is 20% per year.

10.1 Program Text

```
/* nag_amer_bs_price (s30qcc) 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;
  /* Double scalar and array declarations */
  double        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_amer_bs_price (s30qcc) Example Program Results\n");
  /* Skip heading in data file */
  scanf("%*[^\n] ");
  /* Read put */
  scanf("%8s%*[^\n] ", put);
  /*
   * 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 */
  scanf("%lf%lf%lf%lf%*[^\n] ", &s, &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] ");
  for (i = 0; i < n; i++)
    scanf("%lf ", &t[i]);
  scanf("%*[^\n] ");
  /*
   * nag_amer_bs_price (s30qcc)
   * American option: Bjerksund and Stensland pricing formula
   */
  nag_amer_bs_price(order, putnum, m, n, x, s, t, sigma, r, q, p,
                    &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_amer_bs_price (s30qcc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  if (putnum == Nag_Call)
    printf("%s\n\n", "American Call :");
  else if (putnum == Nag_Put)
    printf("%s\n\n", "American Put :");
  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");
  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_amer_bs_price (s30qcc) Example Program Data
 Nag_Call            : Nag_Call or Nag_Put
 110.0 0.2 0.08 0.12 : s, sigma, r, q
 1 1                 : m, n
 100.0               : X(I), I = 1,2,...m
 0.25                : T(I), I = 1,2,...n
```

10.3 Program Results

```
nag_amer_bs_price (s30qcc) Example Program Results
American Call :

  Spot       = 110.0000
  Volatility =   0.2000
  Rate       =   0.0800
  Dividend   =   0.1200

    Strike    Expiry   Option Price
  100.0000    0.2500     10.3340
```