1 Purpose
nag_jumpdiff_merton_price (s30jac) computes the European option price using the Merton jump-diffusion model.

2 Specification

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

void nag_jumpdiff_merton_price (Nag_OrderType order, Nag_CallPut option,
    Integer m, Integer n, const double x[], double s, const double t[],
    double sigma, double r, double lambda, double jvol, double p[],
    NagError *fail)
```

3 Description

nag_jumpdiff_merton_price (s30jac) uses Merton’s jump-diffusion model (Merton (1976)) to compute the price of a European option. This assumes that the asset price is described by a Brownian motion with drift, as in the Black–Scholes–Merton case, together with a compound Poisson process to model the jumps. The corresponding stochastic differential equation is,

\[
dS = (\alpha - \lambda k)dt + \sigma dW_t + d\eta_t.
\]

Here \( \alpha \) is the instantaneous expected return on the asset price, \( S \); \( \sigma^2 \) is the instantaneous variance of the return when the Poisson event does not occur; \( dW_t \) is a standard Brownian motion; \( \eta_t \) is the independent Poisson process and \( k = E[Y - 1] \) where \( Y - 1 \) is the random variable change in the stock price if the Poisson event occurs and \( E \) is the expectation operator over the random variable \( Y \).

This leads to the following price for a European option (see Haug (2007))

\[
P_{\text{call}} = \sum_{j=0}^{\infty} e^{-\lambda T} \left( \frac{\lambda T}{j!} \right) C_j(S, X, T, r, \sigma_j'),
\]

where \( T \) is the time to expiry; \( X \) is the strike price; \( r \) is the annual risk-free interest rate; \( C_j(S, X, T, r, \sigma_j') \) is the Black–Scholes–Merton option pricing formula for a European call (see nag_bsm_price (s30aac)).

\[
\sigma_j' = \sqrt{z^2 + \delta^2 \left( \frac{j}{T} \right)},
\]

\[
z^2 = \sigma^2 - \lambda \delta^2,
\]

\[
\delta^2 = \frac{\gamma^2}{X},
\]

where \( \sigma \) is the total volatility including jumps; \( \lambda \) is the expected number of jumps given as an average per year; \( \gamma \) is the proportion of the total volatility due to jumps.

The value of a put is obtained by substituting the Black–Scholes–Merton put price for \( C_j(S, X, T, r, \sigma_j') \).

The option price \( P_j = P(X = X_i, T = T_j) \) 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


Merton R C (1976) Option pricing when underlying stock returns are discontinuous Journal of Financial Economics 3 125–144

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: 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, . . . , m.
   Constraint: x[i - 1] ≥ z and x[i - 1] ≤ 1/z, where z = nag_real_safe_small_number, the safe range parameter, for i = 1, 2, . . . , m.

6: s – double  
   Input
   On entry: S, the price of the underlying asset.
   Constraint: s ≥ z and s ≤ 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 i_th time, in years, to expiry, for i = 1, 2, . . . , n.
   Constraint: t[i - 1] ≥ z, where z = nag_real_safe_small_number, the safe range parameter, for i = 1, 2, . . . , n.

8: sigma – double  
   Input
   On entry: σ, the annual total volatility, including jumps.
   Constraint: sigma > 0.0.
9: \textbf{r} – double
Input

\textit{On entry}: r, the annual risk-free interest rate, continuously compounded. Note that a rate of 5\% should be entered as 0.05.
\textit{Constraint}: r \geq 0.0.

10: \textbf{lambda} – double
Input

\textit{On entry}: \lambda, the number of expected jumps per year.
\textit{Constraint}: \lambda > 0.0.

11: \textbf{jvol} – double
Input

\textit{On entry}: the proportion of the total volatility associated with jumps.
\textit{Constraint}: 0.0 \leq jvol < 1.0.

12: \textbf{p[m x n]} – double
Output

\textbf{Note}: where \( P(i,j) \) appears in this document, it refers to the array element
- \( p[(j-1) \times m + i - 1] \) when \texttt{order} = \texttt{Nag\_ColMajor};
- \( p[(i-1) \times n + j - 1] \) when \texttt{order} = \texttt{Nag\_RowMajor}.

\textit{On exit}: \( P(i,j) \) contains \( P_{ij} \), the option price evaluated for the strike price \( x_i \) at expiry \( t_j \) for \( i = 1,2,\ldots,m \) and \( j = 1,2,\ldots,n \).

13: \textbf{fail} – NagError *
Input/Output

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

6 Error Indicators and Warnings

\textbf{NE\_ALLOC\_FAIL}
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE\_BAD\_PARAM}
On entry, argument \langle value \rangle had an illegal value.

\textbf{NE\_INT}
On entry, \texttt{m} = \langle value \rangle.
\textit{Constraint}: \texttt{m} \geq 1.

On entry, \texttt{n} = \langle value \rangle.
\textit{Constraint}: \texttt{n} \geq 1.

\textbf{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.

\textbf{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.
On entry, \( jvol = \langle \text{value} \rangle \).
Constraint: \( jvol \geq 0.0 \) and \( jvol < 1.0 \).

On entry, \( \lambda = \langle \text{value} \rangle \).
Constraint: \( \lambda > 0.0 \).

On entry, \( r = \langle \text{value} \rangle \).
Constraint: \( r \geq 0.0 \).

On entry, \( s = \langle \text{value} \rangle \).
Constraint: \( s \geq \langle \text{value} \rangle \) and \( s \leq \langle \text{value} \rangle \).

On entry, \( \sigma = \langle \text{value} \rangle \).
Constraint: \( \sigma > 0.0 \).

On entry, \( t[i] = \langle \text{value} \rangle \).
Constraint: \( t[i] \geq \langle \text{value} \rangle \).

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

7 Accuracy

The accuracy of the output is dependent on the accuracy of the cumulative Normal distribution function, \( \Phi \), occurring in \( C_i \). 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_jumpdiff_merton_price (s30jac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_jumpdiff_merton_price (s30jac) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

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 European call with jumps. The time to expiry is 3 months, the stock price is 45 and the strike price is 55. The number of jumps per year is 3 and the percentage of the total volatility due to jumps is 40%. The risk-free interest rate is 10% per year and the total volatility is 25% per year.
10.1 Program Text

/* nag_jumpdiff_merton_price (s30jac) 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;
  /* Double scalar and array declarations */
  double jvol, lambda, r, s, sigma;
  double *p = 0, *t = 0, *x = 0;
  /* Character scalar and array declarations */
  char put[8+1];
  Nag_OrderType order;
  NagError fail;
  Nag_CallPut putnum;

  INIT_FAIL(fail);

  printf("nag_jumpdiff_merton_price (s30jac) Example Program Results\n");
  printf("Merton Jump-Diffusion Model\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 lambda, s, sigma, r, jvol */
  #ifdef _WIN32
  scanf_s("%lf%lf%lf%lf%lf%*[\n ] ", &lambda, &s, &sigma, &r, &jvol);
  #else
  scanf("%lf%lf%lf%lf%lf%*[\n ] ", &lambda, &s, &sigma, &r, &jvol);
  #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 = 0; 
  /* Character scalar and array declarations */
  char put[8+1];
  Nag_OrderType order;
  NagError fail;
  Nag_CallPut putnum;

  INIT_FAIL(fail);

  printf("nag_jumpdiff_merton_price (s30jac) Example Program Results\n");
  printf("Merton Jump-Diffusion Model\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 lambda, s, sigma, r, jvol */
  #ifdef _WIN32
  scanf_s("%lf%lf%lf%lf%lf%*[\n ] ", &lambda, &s, &sigma, &r, &jvol);
  #else
  scanf("%lf%lf%lf%lf%lf%*[\n ] ", &lambda, &s, &sigma, &r, &jvol);
  #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 = 0; 
  /* Character scalar and array declarations */
  char put[8+1];
  Nag_OrderType order;
  NagError fail;
  Nag_CallPut putnum;

  INIT_FAIL(fail);

  printf("nag_jumpdiff_merton_price (s30jac) Example Program Results\n");
  printf("Merton Jump-Diffusion Model\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 lambda, s, sigma, r, jvol */
  #ifdef _WIN32
  scanf_s("%lf%lf%lf%lf%lf%*[\n ] ", &lambda, &s, &sigma, &r, &jvol);
  #else
  scanf("%lf%lf%lf%lf%lf%*[\n ] ", &lambda, &s, &sigma, &r, &jvol);
  #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 = 0; 
  /* Character scalar and array declarations */
  char put[8+1];
  Nag_OrderType order;
  NagError fail;
  Nag_CallPut putnum;

  INIT_FAIL(fail);

  printf("nag_jumpdiff_merton_price (s30jac) Example Program Results\n");
  printf("Merton Jump-Diffusion Model\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 lambda, s, sigma, r, jvol */
  #ifdef _WIN32
  scanf_s("%lf%lf%lf%lf%lf%*[\n ] ", &lambda, &s, &sigma, &r, &jvol);
  #else
  scanf("%lf%lf%lf%lf%lf%*[\n ] ", &lambda, &s, &sigma, &r, &jvol);
  #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

/* 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_jumpdiff_merton_price (s30jac) */
    nag_jumpdiff_merton_price(order, putnum, m, n, x, s, t, sigma, r, lambda,
    jvol, p, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_jumpdiff_merton_price (s30jac).
        %s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    if (putnum == Nag_Call)
        printf("%s\n", "European Call :");
    else if (putnum == Nag_Put)
        printf("%s\n", "European Put :");
    printf("%8.4f\n", " Spot = ", s);
    printf("%8.4f\n", " Volatility = ", sigma);
    printf("%8.4f\n", " Rate = ", r);
    printf("%8.4f\n", " Jumps = ", lambda);
    printf("%8.4f\n", " Jump vol = ", jvol);

    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_jumpdiff_merton_price (s30jac) Example Program Data
Nag_Call : Nag_Call or Nag_Put
3.0 45.0 0.25 0.1 0.4 : lambda (jumps), s, sigma, r, jvol
1 1 : m, n
55.0 : X(I), I = 1,2,...m
0.25 : T(I), I = 1,2,...n

10.3 Program Results

nag_jumpdiff_merton_price (s30jac) Example Program Results
Merton Jump-Diffusion Model

European Call :

<table>
<thead>
<tr>
<th>Strike</th>
<th>Expiry</th>
<th>Option Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>55.0000</td>
<td>0.2500</td>
<td>0.2417</td>
</tr>
</tbody>
</table>