NAG Library Function Document

nag_jumpdiff_merton_greeks (s30jbc)

1 Purpose

nag_jumpdiff_merton_greeks (s30jbc) computes the European option price together with its sensitivities (Greeks) using the Merton jump-diffusion model.

2 Specification

```
#include <nag.h>
#include <nags.h>
void nag_jumpdiff_merton_greeks (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[],
     double delta[], double gamma[], double vega[], double theta[],
     double rho[], double vanna[], double charm[], double speed[],
     double colour[], double zomma[], double vomma[], NagError *fail)
```

3 Description

nag_jumpdiff_merton_greeks (s30jbc) uses Merton's jump-diffusion model (Merton (1976)) to compute the price of a European 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. Merton's model 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,

$$\frac{dS}{S} = (\alpha - \lambda k)dt + \hat{\sigma}dW_t + dq_t.$$

Here α is the instantaneous expected return on the asset price, S; $\hat{\sigma}^2$ is the instantaneous variance of the return when the Poisson event does not occur; dW_t is a standard Brownian motion; q_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))

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

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

$$\sigma'_j = \sqrt{z^2 + \delta^2\left(\frac{j}{T}\right)},$$

$$z^2 = \sigma^2 - \lambda\delta^2,$$

$$\delta^2 = \frac{\gamma\sigma^2}{\lambda},$$

where σ is the total volatility including jumps; λ is the expected number of jumps given as an average per year; γ 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 $C_j\left(S, X, T, r, \sigma'_j\right)$.

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

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

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

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

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 annual total volatility, including jumps.

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: **lambda** – double *Input*

On entry: λ, the number of expected jumps per year.

Constraint: **lambda** > 0.0.

11: **jvol** – double *Input*

On entry: the proportion of the total volatility associated with jumps.

Constraint: $0.0 \leq$ **jvol** < 1.0.

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

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

p$[(j-1) \times$ **m** $+ i - 1]$ when **order** = Nag_ColMajor;
p$[(i-1) \times$ **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**.

13: **delta**[**m** $\times$ **n**] – double *Output*

Note: the (i,j)th element of the matrix is stored in

delta$[(j-1) \times$ **m** $+ i - 1]$ when **order** = Nag_ColMajor;
delta$[(i-1) \times$ **n** $+ j - 1]$ when **order** = 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** $\times$ **n**] – double *Output*

Note: the (i,j)th element of the matrix is stored in

gamma$[(j-1) \times$ **m** $+ i - 1]$ when **order** = Nag_ColMajor;
gamma$[(i-1) \times$ **n** $+ j - 1]$ when **order** = 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** $\times$ **n**] – double *Output*

Note: where **VEGA**(i,j) appears in this document, it refers to the array element

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

16: **theta**[**m** × **n**] – double *Output*

Note: where **THETA**(i,j) appears in this document, it refers to the array element

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

17: **rho**[**m** × **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 $i = 1,2,\ldots,\mathbf{m}$ and $j = 1,2,\ldots,\mathbf{n}$.

18: **vanna**[**m** × **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 $i = 1,2,\ldots,\mathbf{m}$ and $j = 1,2,\ldots,\mathbf{n}$.

19: **charm**[**m** × **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 $i = 1,2,\ldots,\mathbf{m}$ and $j = 1,2,\ldots,\mathbf{n}$.

20: **speed**[**m** × **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 $i = 1,2,\ldots,\mathbf{m}$ and $j = 1,2,\ldots,\mathbf{n}$.

21: **colour**[**m** × **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 $i = 1,2,\ldots,\mathbf{m}$ and $j = 1,2,\ldots,\mathbf{n}$.

22: **zomma**[**m** × **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 $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

23: **vomma**[**m** × **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 $i = 1, 2, \ldots, \mathbf{m}$ and $j = 1, 2, \ldots, \mathbf{n}$.

24: **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 ⟨*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.

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, **jvol** = ⟨*value*⟩.
Constraint: **jvol** ≥ 0.0 and **jvol** < 1.0.

On entry, **lambda** $= \langle value \rangle$.
Constraint: **lambda** > 0.0.

On entry, **r** $= \langle value \rangle$.
Constraint: **r** ≥ 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, occurring in C_j. 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_greeks (s30jbc) 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 two European calls with jumps. The time to expiry is 6 months, the stock price is 100 and strike prices are 80 and 90 respectively. The number of jumps per year is 5 and the percentage of the total volatility due to jumps is 25%. The risk-free interest rate is 8% per year while the total volatility is 25% per year.

10.1 Program Text

```
/* nag_jumpdiff_merton_greeks (s30jbc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#include <stdio.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 jvol, lambda, r, s, sigma;
  double *charm = 0, *colour = 0, *delta = 0, *gamma = 0, *p = 0;
  double *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_jumpdiff_merton_greeks (s30jbc) 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, (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 lambda, 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 CHARM(I, J)  charm[(J-1)*m + I-1]
#define COLOUR(I, J) colour[(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 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)) ||
      !(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
#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_greeks (s30jbc)
   * Jump-diffusion, Merton's model, option pricing formula with Greeks
   */
  nag_jumpdiff_merton_greeks(order, putnum, m, n, x, s, t, sigma, r,
                             lambda, jvol, p, delta, gamma, vega,
                             theta, rho, vanna, charm, speed, colour,
                             zomma, vomma, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_jumpdiff_merton_greeks (s30jbc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
  if (putnum == Nag_Call)
    printf("%s\n\n", "European Call :");
  else if (putnum == Nag_Put)
    printf("%s\n\n", "European Put :");
  printf("  Spot       = %8.4f\n", s);
  printf("  Volatility = %8.4f\n", sigma);
  printf("  Rate       = %8.4f\n", r);
```

```
  printf("  Jumps      = %8.4f\n", lambda);
  printf("  Jump vol   = %8.4f\n", jvol);
  printf("\n");
  for (j = 1; j <= n; j++) {
    printf("\n  Time to Expiry :    %8.4f\n", t[j - 1]);
    printf("  Strike    Price    Delta    Gamma     Vega    "
           "Theta     Rho\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));
    printf("                      Vanna    Charm    Speed   "
           "Colour    Zomma    Vomma\n");
    for (i = 1; i <= m; i++)
      printf("%26.4f %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(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_jumpdiff_merton_greeks (s30jbc) Example Program Data
 Nag_Call                 : Nag_Call or Nag_Put
 5.0 100.0 0.25 0.08 0.25 : lambda (jumps), s, sigma, r, jvol
 2 1                      : m, n
 80.0
 90.0                     : X(I), I = 1,2,...m
 0.5                      : T(I), I = 1,2,...n
```

10.3 Program Results

```
nag_jumpdiff_merton_greeks (s30jbc) Example Program Results
Merton Jump-Diffusion Model

European Call :

  Spot       = 100.0000
  Volatility =   0.2500
  Rate       =   0.0800
  Jumps      =   5.0000
  Jump vol   =   0.2500


  Time to Expiry :      0.5000
  Strike    Price    Delta    Gamma     Vega    Theta     Rho
```

```
 80.0000  23.6090   0.9431   0.0064   8.1206  -7.6718  35.3480
 90.0000  15.4193   0.8203   0.0149  18.5256  -9.9695  33.3037
                     Vanna    Charm    Speed   Colour    Zomma    Vomma
                   -0.6334   0.1080  -0.0006  -0.0035   0.0315  70.6824
                   -0.7726   0.0770  -0.0009   0.0109  -0.0186  49.7161
```