NAG Library Function Document

nag_1d_withdraw_quad_gauss_1 (d01tac)

1 Purpose

nag_1d_withdraw_quad_gauss_1 (d01tac) computes an estimate of the definite integral of a function of known analytical form, using a Gaussian quadrature formula with a specified number of abscissae. Formulae are provided for a finite interval (Gauss–Legendre), a semi-infinite interval (Gauss–Laguerre, rational Gauss), and an infinite interval (Gauss–Hermite).

2 Specification

```
#include <nag.h>
#include <nagd01.h>
double nag_1d_withdraw_quad_gauss_1 (Nag_GaussFormulae quadrule,
       double (*f)(double x, Nag_User *comm),
       double a, double b, Integer npts, Nag_User *comm, NagError *fail)
```

3 Description

3.1 General

nag_1d_withdraw_quad_gauss_1 (d01tac) evaluates an estimate of the definite integral of a function $f(x)$, over a finite or infinite interval, by n-point Gaussian quadrature (see Davis and Rabinowitz (1967), FrÎberg (1970), Ward (1975) or Stroud and Secrest (1966)). The integral is approximated by a summation

$$\sum_{i=1}^{n} \omega_i f(x_i)$$

where ω_i are called the weights, and the x_i the abscissae. A selection of values of n is available. (See Section 5.)

3.2 Both Limits Finite

$$\int_a^b f(x)dx$$

The Gauss–Legendre weights and abscissae are used, and the formula is exact for any function of the form:

$$f(x) = \sum_{i=0}^{2n-1} c_i x^i.$$

The formula is appropriate for functions which can be well approximated by such a polynomial over $[a, b]$. It is inappropriate for functions with algebraic singularities at one or both ends of the interval, such as $\frac{1}{\sqrt{(1+x)}}$ on $[-1, 1]$.

3.3 One Limit Infinite

$$\int_a^\infty f(x)dx \quad \text{or} \quad \int_{-\infty}^a f(x)dx.$$

Two quadrature formulae are available for these integrals:

(a) The Gauss–Laguerre formula is exact for any function of the form:

$$f(x) = e^{-bx}\sum_{i=0}^{2n-1} c_i x^i.$$

(b) This formula is appropriate for functions decaying exponentially at infinity; the argument b should be chosen if possible to match the decay rate of the function.

(c) The rational Gauss formula is exact for any function of the form:

$$f(x) = \sum_{i=2}^{2n+1} \frac{c_i}{(x+b)^i} = \frac{\sum_{i=0}^{2n-1} c_{2n+1-i}(x+b)^i}{(x+b)^{2n+1}}$$

(d) This formula is likely to be more accurate for functions having only an inverse power rate of decay for large x. Here the choice of a suitable value of b may be more difficult; unfortunately a poor choice of b can make a large difference to the accuracy of the computed integral.

3.4 Both Limits Infinite

$$\int_{-\infty}^{+\infty} f(x)dx.$$

The Gauss–Hermite weights and abscissae are used, and the formula is exact for any function of the form:

$$f(x) = e^{-b(x-a)^2}\sum_{i=0}^{2n-1} c_i x^i.$$

Again, for general functions not of this exact form, the argument b should be chosen to match if possible the decay rate at $\pm\infty$.

4 References

Davis P J and Rabinowitz P (1967) *Numerical Integration* 33–52 Blaisdell Publishing Company

FrÎberg C E (1970) *Introduction to Numerical Analysis* Addison–Wesley

Stroud A H and Secrest D (1966) *Gaussian Quadrature Formulas* Prentice–Hall

Ward R C (1975) The combination shift QZ algorithm *SIAM J. Numer. Anal.* **12** 835–853

5 Arguments

1: **quadrule** – Nag_GaussFormulae *Input*

On entry: indicates the quadrature formula:

quadrule = Nag_Legendre, for Gauss–Legendre quadrature on a finite interval;

quadrule = Nag_Rational, for rational Gauss quadrature on a semi-infinite interval;

quadrule = Nag_Laguerre, for Gauss–Laguerre quadrature on a semi-infinite interval;

quadrule = Nag_Hermite, for Gauss–Hermite quadrature on an infinite interval.

Constraint: **quadrule** = Nag_Legendre, Nag_Rational, Nag_Laguerre or Nag_Hermite.

2: **f** – function, supplied by the user *External Function*

f must return the value of the integrand f at a given point.

The specification of **f** is:

```
double f (double x, Nag_User *comm)
```

1: **x** – double *Input*

On entry: the point at which the integrand f must be evaluated.

2: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **comm**→**p** should be cast to the required type, e.g., `struct user *s = (struct user *)comm → p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

Some points to bear in mind when coding **f** are mentioned in Section 9.

3: **a** – double *Input*

4: **b** – double *Input*

On entry: the arguments a and b which occur in the integration formulae:

Gauss–Legendre: a is the lower limit and b is the upper limit of the integral. It is not necessary that $a < b$.

Rational Gauss: b must be chosen so as to make the integrand match as closely as possible the exact form given in Section 3. The interval of integration is $[a, \infty)$ if $a + b > 0$, and $(-\infty, a]$ if $a + b < 0$.

Gauss–Laguerre: b must be chosen so as to make the integrand match as closely as possible the exact form given in Section 3. The interval of integration is $[a, \infty)$ if $b > 0$, and $(-\infty, a]$ if $b < 0$.

Gauss–Hermite: a and b must be chosen so as to make the integrand match as closely as possible the exact form given in Section 3.

Constraints:

Rational Gauss: $\mathbf{a} + \mathbf{b} \neq 0.0$;
Gauss–Laguerre: $\mathbf{b} \neq 0.0$;
Gauss–Hermite: $\mathbf{b} > 0.0$.

5: **npts** – Integer *Input*

On entry: the number of abscissae to be used, n.

Constraint: **npts** = 1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 16, 20, 24, 32, 48 or 64.

6: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **comm**→**p**, of type Pointer, allows you to communicate information to and from **f**(). An object of the required type should be declared, e.g., a structure, and its address assigned to the pointer **comm**→**p** by means of a cast to Pointer in the calling program, e.g., `comm.p = (Pointer)&s`. The type pointer will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

7: **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_BAD_PARAM

On entry, argument **quadrule** had an illegal value.

NE_QUAD_GAUSS_CONS

Gauss–Hermite input is invalid with $\mathbf{b} \leq 0.0$.
Constraint: $\mathbf{b} > 0.0$.

Gauss–Laguerre input is invalid with $\mathbf{b} = 0.0$.
Constraint: $\mathbf{b} \neq 0.0$.

Rational Gauss input is invalid with $\mathbf{a} + \mathbf{b} = 0.0$.
Constraint: $\mathbf{a} + \mathbf{b} \neq 0.0$.

The answer is returned as zero.

NE_QUAD_GAUSS_NPTS_RULE

The N-point rule is not among those stored.
The answer is evaluated for $\langle value \rangle$, the largest possible value of **npts** less than the requested value, $\langle value \rangle$.

7 Accuracy

The accuracy depends on the behaviour of the integrand, and on the number of abscissae used. No tests are carried out in nag_1d_withdraw_quad_gauss_1 (d01tac) to estimate the accuracy of the result. If such an estimate is required, the function may be called more than once, with a different number of abscissae each time, and the answers compared. It is to be expected that for sufficiently smooth functions a larger number of abscissae will give improved accuracy.

Alternatively, the interval of integration may be subdivided, the integral estimated separately for each sub-interval, and the sum of these estimates compared with the estimate over the whole interval.

The coding of the function **f** may also have a bearing on the accuracy. For example, if a high-order Gauss–Laguerre formula is used, and the integrand is of the form

$$f(x) = e^{-bx} g(x)$$

it is possible that the exponential term may underflow for some large abscissae. Depending on the machine, this may produce an error, or simply be assumed to be zero. In any case, it would be better to evaluate the expression as:

$$f(x) = \exp(-bx + \ln g(x)).$$

Another situation requiring care is exemplified by

$$\int_{-\infty}^{+\infty} e^{-x^2} x^m dx = 0, \quad m \text{ odd}.$$

The integrand here assumes very large values; for example, for $m = 63$, the peak value exceeds 3×10^{33}. Now, if the machine holds floating-point numbers to an accuracy of k significant decimal digits, we could not expect such terms to cancel in the summation leaving an answer of much less than 10^{33-k} (the weights being of order unity); that is instead of zero, we obtain a rather large answer through rounding error. Fortunately, such situations are characterised by great variability in the answers returned by formulae with different values of n. In general, you should be aware of the order of magnitude of the integrand, and should judge the answer in that light.

8 Parallelism and Performance

nag_1d_withdraw_quad_gauss_1 (d01tac) is not threaded in any implementation.

9 Further Comments

The time taken by nag_1d_withdraw_quad_gauss_1 (d01tac) depends on the complexity of the expression for the integrand and on the number of abscissae required.

10 Example

This example evaluates the integrals

$$\int_0^1 \frac{4}{1+x^2}dx = \pi$$

by Gauss–Legendre quadrature;

$$\int_2^\infty \frac{1}{x^2 \ln x}dx = 0.378671$$

by rational Gauss quadrature with $b = 0$;

$$\int_2^\infty \frac{e^{-x}}{x}dx = 0.048901$$

by Gauss–Laguerre quadrature with $b = 1$; and

$$\int_{-\infty}^{+\infty} e^{-3x^2-4x-1}dx = \int_{-\infty}^{+\infty} e^{-3(x+1)^2}e^{2x+2}dx = 1.428167$$

by Gauss–Hermite quadrature with $a = -1$ and $b = 3$.

The formulae with $n = 4, 8, 16$ are used in each case.

10.1 Program Text

```
/* nag_1d_withdraw_quad_gauss_1 (d01tac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 *
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <math.h>
#include <nagd01.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static double NAG_CALL fun1(double x, Nag_User *comm);
  static double NAG_CALL fun2(double x, Nag_User *comm);
  static double NAG_CALL fun3(double x, Nag_User *comm);
  static double NAG_CALL fun4(double x, Nag_User *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  static Integer use_comm[4] = { 1, 1, 1, 1 };
  Integer exit_status = 0;
  static Integer nstor[3] = { 4, 8, 16 };
  double a, b;
  Integer i;
```

```
  double ans;
  Nag_GaussFormulae gaussformula;
  Nag_User comm;
  NagError fail;

  fail.print = Nag_TRUE;

  INIT_FAIL(fail);

  printf("nag_1d_withdraw_quad_gauss_1 (d01tac) Example Program Results\n");

  /* For communication with user-supplied functions: */
  comm.p = (Pointer) &use_comm;

  printf("\nGauss-Legendre example\n\n");
  for (i = 0; i < 3; ++i) {
    a = 0.0;
    b = 1.0;
    gaussformula = Nag_Legendre;
    /* nag_1d_withdraw_quad_gauss_1 (d01tac).
     * One-dimensional Gaussian quadrature rule evaluation,
     * thread-safe
     */
    ans = nag_1d_withdraw_quad_gauss_1(gaussformula, fun1, a, b, nstor[i],
                                       &comm, &fail);
    if (fail.code == NE_NOERROR || fail.code == NE_QUAD_GAUSS_NPTS_RULE)
      printf("%" NAG_IFMT " Points    Answer = %10.5f\n\n", nstor[i], ans);
    else {
      printf("%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  }
  printf("\nGauss-Rational example\n\n");
  for (i = 0; i < 3; ++i) {
    a = 2.0;
    b = 0.0;
    gaussformula = Nag_Rational;
    /* nag_1d_withdraw_quad_gauss_1 (d01tac), see above. */
    ans = nag_1d_withdraw_quad_gauss_1(gaussformula, fun2, a, b, nstor[i],
                                       &comm, &fail);
    if (fail.code == NE_NOERROR || fail.code == NE_QUAD_GAUSS_NPTS_RULE)
      printf("%" NAG_IFMT " Points    Answer = %10.5f\n\n", nstor[i], ans);
    else {
      printf("%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  }
  printf("\nGauss-Laguerre example\n\n");
  for (i = 0; i < 3; ++i) {
    a = 2.0;
    b = 1.0;
    gaussformula = Nag_Laguerre;
    /* nag_1d_withdraw_quad_gauss_1 (d01tac), see above. */
    ans = nag_1d_withdraw_quad_gauss_1(gaussformula, fun3, a, b, nstor[i],
                                       &comm, &fail);
    if (fail.code == NE_NOERROR || fail.code == NE_QUAD_GAUSS_NPTS_RULE)
      printf("%" NAG_IFMT " Points    Answer = %10.5f\n\n", nstor[i], ans);
    else {
      printf("%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  }
  printf("\nGauss-Hermite  example\n\n");
  for (i = 0; i < 3; ++i) {
    a = -1.0;
    b = 3.0;
    gaussformula = Nag_Hermite;
    /* nag_1d_withdraw_quad_gauss_1 (d01tac), see above. */
```

```
    ans = nag_1d_withdraw_quad_gauss_1(gaussformula, fun4, a, b, nstor[i],
                                       &comm, &fail);
    if (fail.code == NE_NOERROR || fail.code == NE_QUAD_GAUSS_NPTS_RULE)
      printf("%" NAG_IFMT " Points    Answer = %10.5f\n\n", nstor[i], ans);
    else {
      printf("%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  }

END:
  return exit_status;
}

static double NAG_CALL fun1(double x, Nag_User *comm)
{
  Integer *use_comm = (Integer *) comm->p;

  if (use_comm[0]) {
    printf("(User-supplied callback fun1, first invocation.)\n");
    use_comm[0] = 0;
  }

  return 4.0 / (x * x + 1.0);
}

static double NAG_CALL fun2(double x, Nag_User *comm)
{
  Integer *use_comm = (Integer *) comm->p;

  if (use_comm[1]) {
    printf("(User-supplied callback fun2, first invocation.)\n");
    use_comm[1] = 0;
  }

  return 1.0 / (x * x * log(x));
}

static double NAG_CALL fun3(double x, Nag_User *comm)
{
  Integer *use_comm = (Integer *) comm->p;

  if (use_comm[2]) {
    printf("(User-supplied callback fun3, first invocation.)\n");
    use_comm[2] = 0;
  }

  return exp(-x) / x;
}

static double NAG_CALL fun4(double x, Nag_User *comm)
{
  Integer *use_comm = (Integer *) comm->p;

  if (use_comm[3]) {
    printf("(User-supplied callback fun4, first invocation.)\n");
    use_comm[3] = 0;
  }

  return exp(x * (-3.0) * x - x * 4.0 - 1.0);
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_1d_withdraw_quad_gauss_1 (d01tac) Example Program Results

Gauss-Legendre example

(User-supplied callback fun1, first invocation.)
4 Points    Answer =    3.14161

8 Points    Answer =    3.14159

16 Points    Answer =    3.14159


Gauss-Rational example

(User-supplied callback fun2, first invocation.)
4 Points    Answer =    0.37910

8 Points    Answer =    0.37876

16 Points    Answer =    0.37869


Gauss-Laguerre example

(User-supplied callback fun3, first invocation.)
4 Points    Answer =    0.04887

8 Points    Answer =    0.04890

16 Points    Answer =    0.04890


Gauss-Hermite  example

(User-supplied callback fun4, first invocation.)
4 Points    Answer =    1.42803

8 Points    Answer =    1.42817

16 Points    Answer =    1.42817
```