nag_1d_quad_gauss_1 (d01tac)

1. Purpose

nag_1d_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, Gauss-Rational), and an infinite interval (Gauss-Hermite).

2. Specification

```
#include <nag.h>
#include <nagd01.h>

double nag_1d_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

This function 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), Froberg (1965), Ralston (1965) 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 4.)

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:

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

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

(b) The Gauss-Rational formula is exact for any function of the form:

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

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 parameter b should be chosen to match if possible the decay rate at $\pm\infty$.

4. Parameters

quadrule

Members of this enum type indicate the quadrature formula:

Nag_Legendre, for Gauss-Legendre quadrature on a finite interval;
Nag_Rational, for Gauss-Rational quadrature on a semi-infinite interval;
Nag_Laguerre, for Gauss-Laguerre quadrature on a semi-infinite interval;
Nag_Hermite, for Gauss-Hermite quadrature on an infinite interval.

f

The function **f**, supplied by the user, must return the value of the integrand f at a given point.
The specification of **f** is:

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

x

Input: the point at which the integrand f must be evaluated.

comm

Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer

Input/Output: 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 6.

a
b

Input: the parameters 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 $a < b$.

Gauss-Rational:
b must be chosen so as to make the integrand match as closely as possible the exact form given in Section 3.3(b). 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 3.3(a). 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 3.4.

Constraints: Gauss-Rational: **a** + **b** $\neq 0$
Gauss-Laguerre: **b** $\neq 0$
Gauss-Hermite: **b** > 0.

npts

Input: 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.

comm

Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer

Input/Output: the pointer **p**, of type Pointer, allows the user to communicate information to and from the user-defined function **f()**. An object of the required type should be declared by the user, e.g. a structure, and its address assigned to the pointer **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.

fail

The NAG error parameter, see the Essential Introduction to the NAG C Library.
Users are recommended to declare and initialize **fail** and set **fail.print** = TRUE for this function.

5. Error Indications and Warnings

NE_BAD_PARAM

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

NE_QUAD_GAUSS_NPTS_RULE

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

NE_QUAD_GAUSS_CONS

Gauss-Rational input is invalid with **a** + **b** = 0.
Constraint: **a** + **b** $\neq 0$.
Gauss-Laguerre input is invalid with **b** = 0.
Constraint: **b** $\neq 0$.
Gauss-Hermite input is invalid with **b** ≤ 0.
Constraint: **b** > 0.
The answer is returned as zero.

6. Further Comments

The time taken by the routine depends on the complexity of the expression for the integrand and on the number of abscissae required.

6.1. Accuracy

The accuracy depends on the behaviour of the integrand, and on the number of abscissae used. No tests are carried out in the routine to estimate the accuracy of the result. If such an estimate

is required, the routine 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, the user should be aware of the order of magnitude of the integrand, and should judge the answer in that light.

6.2. References

Davis P J and Rabinowitz P (1967) *Numerical Integration* Blaisdell Publishing Company pp. 33–52.
Froberg C E (1965) *Introduction to Numerical Analysis* Addison-Wesley pp. 181–187.
Ralston A (1965) *A First Course in Numerical Analysis* McGraw-Hill pp. 87–90.
Stroud A H and Secrest D (1966) *Gaussian Quadrature Formulas* Prentice-Hall.

7. See Also

None.

8. Example

This example program 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 Gauss-Rational 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.

8.1. Program Text

```
/* nag_1d_quad_guass_1(d01tac) Example Program
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 */

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

#ifdef NAG_PROTO
static double fun1(double x, Nag_User *comm);
#else
static double fun1();
#endif

#ifdef NAG_PROTO
static double fun2(double x, Nag_User *comm);
#else
static double fun2();
#endif

#ifdef NAG_PROTO
static double fun3(double x, Nag_User *comm);
#else
static double fun3();
#endif

#ifdef NAG_PROTO
static double fun4(double x, Nag_User *comm);
#else
static double fun4();
#endif

main()
{
  static Integer nstor[3] = {4, 8, 16};

  double a, b;
  Integer i;
  static NagError fail;
  double ans;
  Nag_GaussFormulae gaussformula;
  Boolean success = TRUE;
  Nag_User comm;

  fail.print = TRUE;

  Vprintf("d01tac Example Program Results\n");
  Vprintf("\nGauss-Legendre example\n\n");
  for (i=0; i<3; ++i)
    {
      a = 0.0;
      b = 1.0;
      gaussformula = Nag_Legendre;
      ans = d01tac(gaussformula, fun1, a, b, nstor[i], &comm, &fail);
      if (fail.code == NE_NOERROR || fail.code == NE_QUAD_GAUSS_NPTS_RULE)
        Vprintf("%ld Points    Answer = %10.5f\n\n", nstor[i], ans);
      else
        {
          Vprintf("%s\n", fail.message);
          success = FALSE;
        }
    }
  Vprintf("\nGauss-Rational example\n\n");
```

```
  for (i=0; i<3; ++i)
    {
      a = 2.0;
      b = 0.0;
      gaussformula = Nag_Rational;
      ans = d01tac(gaussformula, fun2, a, b, nstor[i], &comm, &fail);
      if (fail.code == NE_NOERROR || fail.code == NE_QUAD_GAUSS_NPTS_RULE)
        Vprintf("%ld Points    Answer = %10.5f\n\n", nstor[i], ans);
      else
        {
          Vprintf("%s\n", fail.message);
          success = FALSE;
        }
    }
  Vprintf("\nGauss-Laguerre example\n\n");
  for (i=0; i<3; ++i)
    {
      a = 2.0;
      b = 1.0;
      gaussformula = Nag_Laguerre;
      ans = d01tac(gaussformula, fun3, a, b, nstor[i], &comm, &fail);
      if (fail.code == NE_NOERROR || fail.code == NE_QUAD_GAUSS_NPTS_RULE)
        Vprintf("%ld Points    Answer = %10.5f\n\n", nstor[i], ans);
      else
        {
          Vprintf("%s\n", fail.message);
          success = FALSE;
        }
    }
  Vprintf("\nGauss-Hermite  example\n\n");
  for (i=0; i<3; ++i)
    {
      a = -1.0;
      b = 3.0;
      gaussformula = Nag_Hermite;
      ans = d01tac(gaussformula, fun4, a, b, nstor[i], &comm, &fail);
      if (fail.code == NE_NOERROR || fail.code == NE_QUAD_GAUSS_NPTS_RULE)
        Vprintf("%ld Points    Answer = %10.5f\n\n", nstor[i], ans);
      else
        {
          Vprintf("%s\n", fail.message);
          success = FALSE;
        }
    }
  if (success)
    exit(EXIT_SUCCESS);
  else
    exit(EXIT_FAILURE);
}                               /* main */


#ifdef NAG_PROTO
static double fun1(double x, Nag_User *comm)
#else
     static double fun1(x, comm)
     double x;
     Nag_User *comm;
#endif
{
  return 4.0/(x*x+1.0);
}

#ifdef NAG_PROTO
static double fun2(double x, Nag_User *comm)
#else
     static double fun2(x, comm)
     double x;
     Nag_User *comm;
#endif
{
```

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

#ifdef NAG_PROTO
static double fun3(double x, Nag_User *comm)
#else
     static double fun3(x, comm)
     double x;
     Nag_User *comm;
#endif
{
  return  exp(-x)/x;
}

#ifdef NAG_PROTO
static double fun4(double x, Nag_User *comm)
#else
     static double fun4(x, comm)
     double x;
     Nag_User *comm;
#endif
{
  return  exp(x*(-3.0)*x-x*4.0-1.0);
}
```

8.2. Program Data

None.

8.3. Program Results

```
d01tac Example Program Results

Gauss-Legendre example

4 Points    Answer =    3.14161

8 Points    Answer =    3.14159

16 Points    Answer =    3.14159


Gauss-Rational example

4 Points    Answer =    0.37910

8 Points    Answer =    0.37876

16 Points    Answer =    0.37869


Gauss-Laguerre example

4 Points    Answer =    0.04887

8 Points    Answer =    0.04890

16 Points    Answer =    0.04890


Gauss-Hermite  example

4 Points    Answer =    1.42803

8 Points    Answer =    1.42817

16 Points    Answer =    1.42817
```