NAG Library Function Document

nag_quad_1d_gauss_vec (d01uac)

1 Purpose

nag_quad_1d_gauss_vec (d01uac) 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>
void nag_quad_1d_gauss_vec (Nag_QuadType quad_type, double a, double b,
     Integer n,
     void (*f)(const double x[], Integer nx, double fv[], Integer *iflag,
          Nag_Comm *comm),
     double *dinest, Nag_Comm *comm, NagError *fail)
```

3 Description

3.1 General

nag_quad_1d_gauss_vec (d01uac) evaluates an estimate of the definite integral of a function $f(x)$, over a finite or infinite range, by n-point Gaussian quadrature (see Davis and Rabinowitz (1975), FrÎberg (1970), Ralston (1965) or Stroud and Secrest (1966)). The integral is approximated by a summation of the product of a set of weights and a set of function evaluations at a corresponding set of abscissae x_i. For adjusted weights, the function values correspond to the values of the integrand f, and hence the sum will be

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

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

Where applicable, normal weights may instead be used, in which case the corresponding weight function ω is factored out of the integrand as $f(x) = \omega(x)g(x)$ and hence the sum will be

$$\sum_{i=-1}^{n} \bar{w}_i g(x),$$

where the normal weights $\bar{w}_i = w_i\omega(x_i)$ are computed internally.

nag_quad_1d_gauss_vec (d01uac) uses a vectorized **f** to evaluate the integrand or normalized integrand at a set of abscissae, x_i, for $i = 1, 2, \ldots, n_x$. If adjusted weights are used, the integrand $f(x_i)$ must be evaluated otherwise the normalized integrand $g(x_i)$ must be evaluated.

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 $(1 + x)^{-1/2}$ 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 argument b should be chosen if possible to match the decay rate of the function.

If the adjusted weights are selected, the complete integrand $f(x)$ should be provided through **f**.

If the normal form is selected, the contribution of e^{-bx} is accounted for internally, and **f** should only return $g(x)$, where $f(x) = e^{-bx} g(x)$.

If $b < 0$ is supplied, the interval of integration will be $[a, \infty)$. Otherwise if $b > 0$ is supplied, the interval of integration will be $(-\infty, a]$.

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

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

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.

Only the adjusted form of the rational Gauss formula is available, and as such, the complete integrand $f(x)$ must be supplied in **f**.

If $a + b < 0$, the interval of integration will be $[a, \infty)$. Otherwise if $a + b > 0$, the interval of integration will be $(-\infty, a]$.

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:

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

where $b > 0$. Again, for general functions not of this exact form, the argument b should be chosen to match if possible the decay rate at $\pm\infty$.

If the adjusted weights are selected, the complete integrand $f(x)$ should be provided through **f**.

If the normal form is selected, the contribution of $e^{-b(x-a)^2}$ is accounted for internally, and **f** should only return $g(x)$, where $f(x) = e^{-b(x-a)^2}g(x)$.

4 References

Davis P J and Rabinowitz P (1975) *Methods of Numerical Integration* Academic Press

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

Ralston A (1965) *A First Course in Numerical Analysis* pp. 87–90 McGraw–Hill

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

5 Arguments

1: **quad_type** – Nag_QuadType *Input*

On entry: indicates the quadrature formula.

quad_type = Nag_Quad_Gauss_Legendre
Gauss–Legendre quadrature on a finite interval, using normal weights.

quad_type = Nag_Quad_Gauss_Laguerre
Gauss–Laguerre quadrature on a semi-infinite interval, using normal weights.

quad_type = Nag_Quad_Gauss_Laguerre_Adjusted
Gauss–Laguerre quadrature on a semi-infinite interval, using adjusted weights.

quad_type = Nag_Quad_Gauss_Hermite
Gauss–Hermite quadrature on an infinite interval, using normal weights.

quad_type = Nag_Quad_Gauss_Hermite_Adjusted
Gauss–Hermite quadrature on an infinite interval, using adjusted weights.

quad_type = Nag_Quad_Gauss_Rational_Adjusted
Rational Gauss quadrature on a semi-infinite interval, using adjusted weights.

Constraint: **quad_type** = Nag_Quad_Gauss_Legendre, Nag_Quad_Gauss_Laguerre, Nag_Quad_Gauss_Laguerre_Adjusted, Nag_Quad_Gauss_Hermite, Nag_Quad_Gauss_Hermite_Adjusted or Nag_Quad_Gauss_Rational_Adjusted.

2: **a** – double *Input*

3: **b** – double *Input*

On entry: the quantities a and b as described in the appropriate subsection of Section 3.

Constraints:

Rational Gauss: $\mathbf{a} + \mathbf{b} \neq 0.0$;
Gauss–Laguerre: $\mathbf{b} \neq 0.0$;
Gauss–Hermite: $\mathbf{b} > 0$.

4: **n** – Integer *Input*

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

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

If the soft fail option is used, the answer is evaluated for the largest valid value of **n** less than the requested value.

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

f must return the value of the integrand f, or the normalized integrand g, at a specified point.

The specification of **f** is:

```
void f (const double x[], Integer nx, double fv[], Integer *iflag,
     Nag_Comm *comm)
```

1: **x**[**nx**] – const double *Input*

On entry: the abscissae, x_i, for $i = 1, 2, \ldots, n_x$ at which function values are required.

2: **nx** – Integer *Input*

On entry: n_x, the number of abscissae.

3: **fv**[**nx**] – double *Output*

On exit: if adjusted weights are used, the values of the integrand f. **fv**$[i-1] = f(x_i)$, for $i = 1, 2, \ldots, n_x$.

Otherwise the values of the normalized integrand g. **fv**$[i-1] = g(x_i)$, for $i = 1, 2, \ldots, n_x$.

4: **iflag** – Integer * *Input/Output*

On entry: **iflag** $= 0$.

On exit: set **iflag** < 0 if you wish to force an immediate exit from nag_quad_1d_gauss_vec (d01uac) with **fail.code** = NE_USER_STOP.

5: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **f**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_quad_1d_gauss_vec (d01uac) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag_quad_1d_gauss_vec (d01uac) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

Some points to bear in mind when coding **f** are mentioned in Section 7.

6: **dinest** – double * *Output*

On exit: the estimate of the definite integral.

7: **comm** – Nag_Comm *

The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

8: **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

The value of **a** and/or **b** is invalid for the chosen **quad_type**. Either:

On entry, argument ⟨*value*⟩ had an illegal value.

The value of **a** and/or **b** is invalid.
On entry, **quad_type** = ⟨*value*⟩.
On entry, **a** = ⟨*value*⟩ and **b** = ⟨*value*⟩.
Constraint: $|\mathbf{a} + \mathbf{b}| > 0.0$.

The value of **a** and/or **b** is invalid.
On entry, **quad_type** = ⟨*value*⟩.
On entry, **a** = ⟨*value*⟩ and **b** = ⟨*value*⟩.
Constraint: $|\mathbf{b}| > 0.0$.

The value of **a** and/or **b** is invalid.
On entry, **quad_type** = ⟨*value*⟩.
On entry, **a** = ⟨*value*⟩ and **b** = ⟨*value*⟩.
Constraint: $\mathbf{b} > 0.0$.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} > 0$.

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_QUAD_GAUSS_NPTS_RULE

The n-point rule is not among those stored.
On entry: **n** = ⟨*value*⟩.
n-point rule used: **n** = ⟨*value*⟩.

NE_TOO_SMALL

Underflow occurred in calculation of normal weights.
Reduce **n** or use adjusted weights: **n** = ⟨*value*⟩.

NE_USER_STOP

Exit requested from **f** with **iflag** = ⟨*value*⟩.

NE_WEIGHT_ZERO

No nonzero weights were generated for the provided parameters.

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_quad_1d_gauss_vec (d01uac) 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 range 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 range.

The coding of **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 with

$$f(x) = \text{sgn}(g(x)) \times \exp(-bx + \ln|g(x)|)$$

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, when $m = 63$, 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. 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_quad_1d_gauss_vec (d01uac) is currently neither directly nor indirectly threaded. In particular, the user-supplied argument **f** is not called from within a parallel region initialized inside nag_quad_1d_gauss_vec (d01uac).

The user-supplied argument **f** uses a vectorized interface, allowing for the required vector of function values to be evaluated in parallel; for example by placing appropriate OpenMP directives in the code for the user-supplied argument **f**.

9 Further Comments

The time taken by nag_quad_1d_gauss_vec (d01uac) 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 $n = 2, 4, 8, 16, 32$ and 64 are used in each case. Both adjusted and normal weights are used for Gauss–Laguerre and Gauss–Hermite quadrature.

10.1 Program Text

```
/* nag_quad_1d_gauss_vec (d01uac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd01.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static void NAG_CALL f(const double x[], const Integer nx, double fv[],
                         Integer *iflag, Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  /* Scalars */
  Integer exit_status = 0;
  double a, b, dinest;
  Integer funid, i, nstor;
  /* Arrays */
  Integer iuser[1];
  /* Nag Types */
  Nag_Comm comm;
  NagError fail;
  Nag_QuadType quad_type;

  printf("nag_quad_1d_gauss_vec (d01uac) Example Program Results\n");

  INIT_FAIL(fail);

  /* Use comm to pass information to f. */
  comm.iuser = iuser;

  for (funid = 1; funid <= 6; funid++) {
    switch (funid) {
    case 1:
      {
        printf("\nGauss-Legendre example\n");
        a = 0.0;
        b = 1.0;
        quad_type = Nag_Quad_Gauss_Legendre;
        break;
```

```
    }
  case 2:
    {
      printf("\nRational Gauss example\n");
      a = 2.0;
      b = 0.0;
      quad_type = Nag_Quad_Gauss_Rational_Adjusted;
      break;
    }
  case 3:
    {
      printf("\nGauss-Laguerre example (adjusted weights)\n");
      a = 2.0;
      b = 1.0;
      quad_type = Nag_Quad_Gauss_Laguerre_Adjusted;
      break;
    }
  case 4:
    {
      printf("\nGauss-Laguerre example (normal weights)\n");
      a = 2.0;
      b = 1.0;
      quad_type = Nag_Quad_Gauss_Laguerre;
      break;
    }
  case 5:
    {
      printf("\nGauss-Hermite example (adjusted weights)\n");
      a = -1.0;
      b = 3.0;
      quad_type = Nag_Quad_Gauss_Hermite_Adjusted;
      break;
    }
  case 6:
    {
      printf("\nGauss-Hermite example (normal weights)\n");
      a = -1.0;
      b = 3.0;
      quad_type = Nag_Quad_Gauss_Hermite;
      break;
    }
  }

  iuser[0] = funid;
  for (i = 0; i < 6; i++) {
    nstor = pow(2, i + 1);
    /* Compute the one-dimensional integral employing Gaussian quadrature,
     * with quadrature type and weights specified in quad_type, using
     * nag_quad_1d_gauss_vec (d01uac).
     */
    nag_quad_1d_gauss_vec(quad_type, a, b, nstor, f, &dinest, &comm, &fail);
    switch (fail.code) {
    case NE_NOERROR:
    case NE_QUAD_GAUSS_NPTS_RULE:
    case NE_UNDERFLOW:
    case NE_WEIGHT_ZERO:
      {
        /* The definite integral has been estimated. */
        printf("%5" NAG_IFMT "   Points  Answer = %10.5f\n", nstor, dinest);
        break;
      }
    default:
      {
        /* A solution could not be calculated due to an illegal parameter
         * or a requested exit.
         */
        printf("%s\n", fail.message);
        exit_status++;
      }
    }
  }
```

```
  }
  return exit_status;
}

static void NAG_CALL f(const double x[], const Integer nx, double fv[],
                       Integer *iflag, Nag_Comm *comm)
{
  Integer i, funid;

  funid = comm->iuser[0];
  switch (funid) {
  case 1:
    for (i = 0; i < nx; i++)
      fv[i] = 4.0 / (1.0 + x[i] * x[i]);
    break;
  case 2:
    for (i = 0; i < nx; i++)
      fv[i] = 1.0 / (x[i] * x[i] * log(x[i]));
    break;
  case 3:
    for (i = 0; i < nx; i++)
      fv[i] = exp(-x[i]) / x[i];
    break;
  case 4:
    for (i = 0; i < nx; i++)
      fv[i] = 1.0 / x[i];
    break;
  case 5:
    for (i = 0; i < nx; i++)
      fv[i] = exp(-3.0 * x[i] * x[i] - 4.0 * x[i] - 1.0);
    break;
  case 6:
    for (i = 0; i < nx; i++)
      fv[i] = exp(2.0 * x[i] + 2.0);
    break;
  default:
    *iflag = -1;
  }
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_quad_1d_gauss_vec (d01uac) Example Program Results

Gauss-Legendre example
    2   Points  Answer =    3.14754
    4   Points  Answer =    3.14161
    8   Points  Answer =    3.14159
   16   Points  Answer =    3.14159
   32   Points  Answer =    3.14159
   64   Points  Answer =    3.14159

Rational Gauss example
    2   Points  Answer =    0.37989
    4   Points  Answer =    0.37910
    8   Points  Answer =    0.37876
   16   Points  Answer =    0.37869
   32   Points  Answer =    0.37867
   64   Points  Answer =    0.37867

Gauss-Laguerre example (adjusted weights)
    2   Points  Answer =    0.04833
    4   Points  Answer =    0.04887
    8   Points  Answer =    0.04890
   16   Points  Answer =    0.04890
   32   Points  Answer =    0.04890
```

```
   64   Points  Answer =    0.04890

Gauss-Laguerre example (normal weights)
    2   Points  Answer =    0.04833
    4   Points  Answer =    0.04887
    8   Points  Answer =    0.04890
   16   Points  Answer =    0.04890
   32   Points  Answer =    0.04890
   64   Points  Answer =    0.04890

Gauss-Hermite example (adjusted weights)
    2   Points  Answer =    1.38381
    4   Points  Answer =    1.42803
    8   Points  Answer =    1.42817
   16   Points  Answer =    1.42817
   32   Points  Answer =    1.42817
   64   Points  Answer =    1.42817

Gauss-Hermite example (normal weights)
    2   Points  Answer =    1.38381
    4   Points  Answer =    1.42803
    8   Points  Answer =    1.42817
   16   Points  Answer =    1.42817
   32   Points  Answer =    1.42817
   64   Points  Answer =    1.42817
```