NAG Library Function Document

nag_quad_1d_gauss_wrec (d01tdc)

1 Purpose

nag_quad_1d_gauss_wrec (d01tdc) computes the weights and abscissae of a Gaussian quadrature rule using the method of Golub and Welsch.

2 Specification

```
#include <nag.h>
#include <nagd01.h>
void nag_quad_1d_gauss_wrec (Integer n, const double a[], double b[],
     double c[], double muzero, double weight[], double abscis[],
     NagError *fail)
```

3 Description

A tri-diagonal system of equations is formed from the coefficients of an underlying three-term recurrence formula:

$$p(j)(x) = (a(j)x + b(j))p(j-1)(x) - c(j)p(j-2)(x)$$

for a set of othogonal polynomials $p(j)$ induced by the quadrature. This is described in greater detail in the d01 Chapter Introduction. The user is required to specify the three-term recurrence and the value of the integral of the chosen weight function over the chosen interval.

As described in Golub and Welsch (1969) the abscissae are computed from the eigenvalues of this matrix and the weights from the first component of the eigenvectors.

LAPACK functions are used for the linear algebra to speed up computation.

4 References

Golub G H and Welsch J H (1969) Calculation of Gauss quadrature rules *Math. Comput.* **23** 221–230

5 Arguments

1: **n** – Integer *Input*

On entry: n, the number of Gauss points required. The resulting quadrature rule will be exact for all polynomials of degree $2n - 1$.

Constraint: **n** > 0.

2: **a**[**n**] – const double *Input*

On entry: **a** contains the coefficients $a(j)$.

3: **b**[**n**] – double *Input/Output*

On entry: **b** contains the coefficients $b(j)$.

On exit: elements of **b** are altered to make the underlying eigenvalue problem symmetric.

4: **c**[**n**] – double *Input/Output*

On entry: **c** contains the coefficients $c(j)$.

On exit: elements of **c** are altered to make the underlying eigenvalue problem symmetric.

5: **muzero** – double *Input*

On entry: **muzero** contains the definite integral of the weight function for the interval of interest.

6: **weight**[**n**] – double *Output*

On exit: **weight**(j) contains the weight corresponding to the jth abscissa.

7: **abscis**[**n**] – double *Output*

On exit: **abscis**(j) the jth abscissa.

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 3.2.1.2 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, **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 3.6.6 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 3.6.5 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

In general the computed weights and abscissae are accurate to a reasonable multiple of ***machine precision***.

8 Parallelism and Performance

nag_quad_1d_gauss_wrec (d01tdc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_quad_1d_gauss_wrec (d01tdc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

The weight function must be non-negative to obtain sensible results. This and the validity of **muzero** are not something that the function can check, so please be particularly careful. If possible check the computed weights and abscissae by integrating a function with a function for which you already know the integral.

10 Example

This example program generates the weights and abscissae for the 4-point Gauss rules: Legendre, Chebyshev1, Chebyshev2, Jacobi, Laguerre and Hermite.

10.1 Program Text

```
/* nag_quad_1d_gauss_wrec (d01tdc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd01.h>
#include <nags.h>
#include <nagx01.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  Integer      exit_status = 0;
  Integer      n, i;
  double       muzero, alpha, beta, a1, b1, ab1, ab2, d, ri, fa1, fb1, fab2;
  /* Arrays */
  char         nag_enum_arg[40];
  double       *a = 0, *b = 0, *c = 0, *abscis = 0, *weight = 0;
  const char   *str_type;
  /* Nag Types */
  Nag_QuadType quadtype;
  NagError     fail, fail1, fail2;

  INIT_FAIL(fail);

  printf("nag_quad_1d_gauss_wrec (d01tdc) Example Program Results\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /* Input number of abscissae required, n */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &n);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &n);
#endif
  /* Input quadrature rule to simulate, Nag_QuadType */
```

```
#ifdef _WIN32
  scanf_s("%39s%*[^\n] ", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n] ", nag_enum_arg);
#endif
  quadtype = (Nag_QuadType) nag_enum_name_to_value(nag_enum_arg);

  /* Allocate coefficient, weight and abscissae arrays */
  if (!(a = NAG_ALLOC(n, double)) ||
      !(b = NAG_ALLOC(n, double)) ||
      !(c = NAG_ALLOC(n, double)) ||
      !(weight = NAG_ALLOC(n, double)) ||
      !(abscis = NAG_ALLOC(n, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Convert QuadType to string using
   * nag_enum_value_to_name (x04nbc).
   * Converts NAG enum member value to its name
   */
  str_type = nag_enum_value_to_name(quadtype);
  printf("\nQuadrature type = %s\n\n",str_type);

  /* Generate recursion coefficients a, b, c from quadtype */
  switch (quadtype) {
  case Nag_Quad_Gauss_Legendre:
    {
      muzero = 2.0;
      for (i = 0; i < n; ++i) {
 ri = (double) (i);
 a[i] = (2.0*ri + 1.0)/(ri + 1.0);
 b[i] = 0.0;
 c[i] = ri/(ri + 1.0);
      }
    }
    break;
  case Nag_Quad_Gauss_Chebyshev_first:
    {
      muzero = X01AAC;
      for (i = 0; i < n; ++i) {
 a[i] = 2.0;
 b[i] = 0.0;
 c[i] = 1.0;
      }
      a[0] = 1.0;
    }
    break;
  case Nag_Quad_Gauss_Chebyshev_second:
    {
      muzero = 0.5*X01AAC;
      for (i = 0; i < n; ++i) {
 a[i] = 2.0;
 b[i] = 0.0;
 c[i] = 1.0;
      }
    }
    break;
  case Nag_Quad_Gauss_Jacobi:
    {
      /* Input quadrature paramaters alpha and beta */
#ifdef _WIN32
      scanf_s("%lf%lf\n", &alpha, &beta);
#else
      scanf("%lf%lf\n", &alpha, &beta);
#endif
      printf("Using parameters alpha = %10.5f, beta = %10.5f\n\n", alpha, beta);
      a1   = alpha + 1.0;
      b1   = beta  + 1.0;
```

```
      ab1  = alpha + beta + 1.0;
      ab2  = a1    + b1;
      INIT_FAIL(fail1);
      INIT_FAIL(fail2);
      /* nag_gamma (s14aac).
       * Gamma function Gamma(x)
       */
      fa1  = nag_gamma(a1, &fail);
      fb1  = nag_gamma(b1, &fail1);
      fab2 = nag_gamma(ab2, &fail2);
      if (fail.code != NE_NOERROR || fail1.code != NE_NOERROR ||
  fail2.code != NE_NOERROR) {
 printf("Error from nag_gamma (s14aac).\n%s\n", fail.message);
 exit_status = 1;
 goto END;
      }
      muzero = pow(2.0,ab1)*fa1*fb1/fab2;
      a[0] = 0.5*ab2;
      b[0] = 0.5*(alpha-beta);
      c[0] = 0.0;
      for (i = 1; i < n; ++i) {
 ri = (double) i;
 d    = (2.0*ri + 2.0)*(ri + ab1);
 a[i] = (2.0*ri + ab1)*(2.0*ri + ab2)/d;
 d    = (2.0*ri + alpha + beta)*d;
 b[i] = (2.0*ri + ab1)*(alpha*alpha - beta*beta)/d;
 c[i] = 2.0*(ri + alpha)*(ri + beta)*(2.0*ri + ab2)/d;
      }
    }
    break;
  case Nag_Quad_Gauss_Laguerre:
    {
      /* Input quadrature paramaters alpha */
#ifdef _WIN32
      scanf_s("%lf\n", &alpha);
#else
      scanf("%lf\n", &alpha);
#endif
      printf("Using parameter alpha = %10.5f\n\n",alpha);
      a1   = alpha + 1.0;
      /* nag_gamma (s14aac).
       * gamma(x)
       */
      muzero  = nag_gamma(a1, &fail);
      if (fail.code != NE_NOERROR) {
 printf("Error from nag_gamma (s14aac).\n%s\n", fail.message);
 exit_status = 1;
 goto END;
      }
      for (i = 0; i < n; ++i) {
 ri = (double) i;
 a[i] = -1.0/(ri + 1.0);
 b[i] = (2.0*ri + a1)/(ri + 1.0);
 c[i] = (ri + alpha)/(ri + 1.0);
      }
    }
    break;
  case Nag_Quad_Gauss_Hermite:
    {
      muzero = sqrt(X01AAC);
      for (i = 0; i < n; ++i) {
 a[i] = 2.0;
 b[i] = 0.0;
 c[i] = (double) 2*i;
      }
    }
    break;
  default:
    {
      exit_status = 1;
```

```
      printf("The quadrature type %s is not handled here\n", str_type);
      goto END;
    }
  }

  /* nag_quad_1d_gauss_wrec (d01tdc).
   * Compute weights and abscissae for a Gaussian quadrature rule
   * governed by a three-term recurrence relation.
   */
  nag_quad_1d_gauss_wrec(n, a, b, c, muzero, weight, abscis, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_quad_1d_gauss_wrec (d01tdc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  printf(" %3" NAG_IFMT " points\n\n    Abscissae        Weights\n\n", n);
  for (i = 0; i < n; i++) {
    printf("%15.6e", abscis[i]);
    printf("%15.6e\n", weight[i]);
  }
  printf("\n");

END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(c);
  NAG_FREE(abscis);
  NAG_FREE(weight);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_quad_1d_gauss_wrec (d01tdc) Example Program Results

Quadrature type = Nag_Quad_Gauss_Jacobi

Using parameters alpha =    0.50000, beta =    0.50000

   6 points

    Abscissae        Weights

  -9.009689e-01   8.448869e-02
  -6.234898e-01   2.743331e-01
  -2.225209e-01   4.265764e-01
   2.225209e-01   4.265764e-01
   6.234898e-01   2.743331e-01
   9.009689e-01   8.448869e-02
```