

NAG Library Function Document

nag_quad_1d_gauss_recm (d01tec)

1 Purpose

Given the $2n + 1$ moments of the weight function, nag_quad_1d_gauss_recm (d01tec) generates the recursion coefficients needed by nag_quad_1d_gauss_wrec (d01tdc) to calculate a Gaussian quadrature rule.

2 Specification

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

void nag_quad_1d_gauss_recm (Integer n, const double mu[], double a[],
                             double b[], double c[], NagError *fail)
```

3 Description

nag_quad_1d_gauss_recm (d01tec) should only be used if the three-term recurrence cannot be determined analytically. A system of equations are formed, using the moments provided. This set of equations becomes ill-conditioned for moderate values of n , the number of abscissae and weights required. In most implementations quadruple precision calculation is used to maintain as much accuracy as possible.

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 weights and abscissae required.
Constraint: $n > 0$.
- 2: **mu[0 : 2 * n]** – const double *Input*
On entry: **mu**(i) must contain the value of the moment with respect to x^i i.e., $\int w(x)x^i dx$, for $i = 0, 1, \dots, 2n$.
- 3: **a[n]** – double *Output*
On exit: values helping define the three term recurrence used by nag_quad_1d_gauss_wrec (d01tdc).
- 4: **b[n]** – double *Output*
On exit: values helping define the three term recurrence used by nag_quad_1d_gauss_wrec (d01tdc).
- 5: **c[n]** – double *Output*
On exit: values helping define the three term recurrence used by nag_quad_1d_gauss_wrec (d01tdc).

- 6: **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 $\langle value \rangle$ had an illegal value.

NE_DATA_ILL_CONDITIONED

The problem is too ill conditioned, it breaks down at row $\langle value \rangle$.

NE_INT

The number of weights and abscissae requested (n) is less than 1: $n = \langle value \rangle$.

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.

7 Accuracy

Internally quadruple precision is used to minimize loss of accuracy as much as possible.

8 Parallelism and Performance

`nag_quad_1d_gauss_recm` (d01tec) is not threaded in any implementation.

9 Further Comments

Because the function cannot check the validity of all the data presented, the user is advised to independently check the result, perhaps by integrating a function whose integral is known, using `nag_quad_1d_gauss_recm` (d01tec) and subsequently `nag_quad_1d_gauss_wrec` (d01tdc), to compare answers.

10 Example

This example program uses `nag_quad_1d_gauss_recm` (d01tec) and moments to calculate a three-term recurrence relationship appropriate for Gauss–Legendre quadrature. It then uses the recurrence relationship to derive the weights and abscissae by calling `nag_quad_1d_gauss_wrec` (d01tdc).

10.1 Program Text

```

/* nag_quad_ld_gauss_recm (d01tec) 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       ri, muzero;
    /* Arrays */
    double       *a = 0, *b = 0, *c = 0, *abscissae = 0, *weights = 0, *mu = 0;
    /* Nag Types */
    NagError     fail;

    INIT_FAIL(fail);

    printf("nag_quad_ld_gauss_recm (d01tec) 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

    /* Allocate coefficient, weight and abscissae arrays */
    if (!(a = NAG_ALLOC(n, double)) ||
        !(b = NAG_ALLOC(n, double)) ||
        !(c = NAG_ALLOC(n, double)) ||
        !(mu = NAG_ALLOC(2*n+1, double)) ||
        !(abscissae = NAG_ALLOC(n, double)) ||
        !(weights = NAG_ALLOC(n, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Set up moments w.r.t  $w(x)x^j$  in array mu */
    mu[0] = 2.0;
    for (i = 1; i < 2*n; i = i+2) {
        ri = (double) (i + 2);
        mu[i] = 0.0;
        mu[i+1] = 2.0/ri;
    }

    /* nag_quad_ld_gauss_recm (d01tec).
     * Compute three term recurrence coefficients from moments in  $w(x)x^j$ .
     */
}

```

```

nag_quad_ld_gauss_recm(n, mu, a, b, c, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_quad_ld_gauss_recm (d01tec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

printf("\n      a      b      c\n");
for (i = 0; i < n; i++) {
    printf("%10.5f%10.5f%10.5f\n", a[i], b[i], c[i]);
}

/* nag_quad_ld_gauss_wrec (d01tdc).
 * Compute weights and abscissae for a Gaussian quadrature rule
 * governed by a three-term recurrence relation.
 */
muzero = mu[0];
nag_quad_ld_gauss_wrec(n, a, b, c, muzero, weights, abscissae, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_quad_ld_gauss_wrec (d01tdc).\n%s\n", fail.message);
    exit_status = 2;
    goto END;
}

printf("\n  weights      abscissae\n");
for (i = 0; i < n; i++) {
    printf("%10.5f      %10.5f\n", weights[i], abscissae[i]);
}

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(c);
NAG_FREE(mu);
NAG_FREE(abscissae);
NAG_FREE(weights);

return exit_status;
}

```

10.2 Program Data

None.

10.3 Program Results

nag_quad_ld_gauss_recm (d01tec) Example Program Results

a	b	c
-1.73205	0.00000	1.73205
-1.93649	0.00000	1.11803
-1.97203	0.00000	1.01835
-1.98431	0.00000	1.00623
-1.98997	0.00000	1.00285
-1.00000	0.00000	0.50252

weights	abscissae
0.17132	-0.93247
0.36076	-0.66121
0.46791	-0.23862
0.46791	0.23862
0.36076	0.66121
0.17132	0.93247
