NAG Library Function Document

nag_quad_2d_fin (d01dac)

1 Purpose

nag_quad_2d_fin (d01dac) attempts to evaluate a double integral to a specified absolute accuracy by repeated applications of the method described by Patterson (1968) and Patterson (1969).

2 Specification

```
#include <nag.h>
#include <nagd01.h>
void nag_quad_2d_fin (double ya, double yb,
     double (*phi1)(double y, Nag_Comm *comm),
     double (*phi2)(double y, Nag_Comm *comm),
     double (*f)(double x, double y, Nag_Comm *comm),
     double absacc, double *ans, Integer *npts, Nag_Comm *comm,
     NagError *fail)
```

3 Description

nag_quad_2d_fin (d01dac) attempts to evaluate a definite integral of the form

$$I = \int_a^b \int_{\phi_1(y)}^{\phi_2(y)} f(x,y)\,dx\,dy$$

where a and b are constants and $\phi_1(y)$ and $\phi_2(y)$ are functions of the variable y.

The integral is evaluated by expressing it as

$$I = \int_a^b F(y)\,dy, \qquad \text{where} \qquad F(y) = \int_{\phi_1(y)}^{\phi_2(y)} f(x,y)\,dx.$$

Both the outer integral I and the inner integrals $F(y)$ are evaluated by the method, described by Patterson (1968) and Patterson (1969), of the optimum addition of points to Gauss quadrature formulae.

This method uses a family of interlacing common point formulae. Beginning with the 3-point Gauss rule, formulae using 7, 15, 31, 63, 127 and finally 255 points are derived. Each new formula contains all the points of the earlier formulae so that no function evaluations are wasted. Each integral is evaluated by applying these formulae successively until two results are obtained which differ by less than the specified absolute accuracy.

4 References

Patterson T N L (1968) On some Gauss and Lobatto based integration formulae *Math. Comput.* **22** 877–881

Patterson T N L (1969) The optimum addition of points to quadrature formulae, errata *Math. Comput.* **23** 892

5 Arguments

1: **ya** – double *Input*

On entry: a, the lower limit of the integral.

2: **yb** – double *Input*

On entry: b, the upper limit of the integral. It is not necessary that $a < b$.

3: **phi1** – function, supplied by the user *External Function*

phi1 must return the lower limit of the inner integral for a given value of y.

The specification of **phi1** is:

```
double phi1 (double y, Nag_Comm *comm)
```

1: **y** – double *Input*

On entry: the value of y for which the lower limit must be evaluated.

2: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **phi1**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_quad_2d_fin (d01dac) you may allocate memory and initialize these pointers with various quantities for use by **phi1** when called from nag_quad_2d_fin (d01dac) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

4: **phi2** – function, supplied by the user *External Function*

phi2 must return the upper limit of the inner integral for a given value of y.

The specification of **phi2** is:

```
double phi2 (double y, Nag_Comm *comm)
```

1: **y** – double *Input*

On entry: the value of y for which the upper limit must be evaluated.

2: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **phi2**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_quad_2d_fin (d01dac) you may allocate memory and initialize these pointers with various quantities for use by **phi2** when called from nag_quad_2d_fin (d01dac) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

5: **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, double y, Nag_Comm *comm)
```

1: **x** – double *Input*

2: **y** – double *Input*

On entry: the coordinates of the point (x, y) at which the integrand f must be evaluated.

3: **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_2d_fin (d01dac) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag_quad_2d_fin (d01dac) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

6: **absacc** – double *Input*

On entry: the absolute accuracy requested.

7: **ans** – double * *Output*

On exit: the estimated value of the integral.

8: **npts** – Integer * *Output*

On exit: the total number of function evaluations.

9: **comm** – Nag_Comm *

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

10: **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 ⟨*value*⟩ had an illegal value.

NE_CONVERGENCE

The outer integral has converged, but n of the inner integrals have not converged with 255 points: $n = \langle value \rangle$. **ans** may still contain an approximate estimate of the integral, but its reliability will decrease as n increases.

The outer integral has not converged, and n of the inner integrals have not converged with 255 points: $n = \langle value \rangle$. **ans** may still contain an approximate estimate of the integral, but its reliability will decrease as n increases.

The outer integral has not converged with 255 points. However, all the inner integrals have converged, and **ans** may still contain an approximate estimate of the integral.

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

The absolute accuracy is specified by the variable **absacc**. If, on exit, **fail.code** = NE_NOERROR then the result is most likely correct to this accuracy. Even if **fail.code** = NE_CONVERGENCE on exit, it is still possible that the calculated result could differ from the true value by less than the given accuracy.

8 Parallelism and Performance

nag_quad_2d_fin (d01dac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time taken by nag_quad_2d_fin (d01dac) depends upon the complexity of the integrand and the accuracy requested.

With Patterson's method accidental convergence may occasionally occur, when two estimates of an integral agree to within the requested accuracy, but both estimates differ considerably from the true result. This could occur in either the outer integral or in one or more of the inner integrals.

If it occurs in the outer integral then apparent convergence is likely to be obtained with considerably fewer integrand evaluations than may be expected. If it occurs in an inner integral, the incorrect value could make the function $F(y)$ appear to be badly behaved, in which case a very large number of pivots may be needed for the overall evaluation of the integral. Thus both unexpectedly small and unexpectedly large numbers of integrand evaluations should be considered as indicating possible trouble. If accidental convergence is suspected, the integral may be recomputed, requesting better accuracy; if the new request is more stringent than the degree of accidental agreement (which is of course unknown), improved results should be obtained. This is only possible when the accidental agreement is not better than machine accuracy. It should be noted that the function requests the same accuracy for the inner integrals as for the outer integral. In practice it has been found that in the vast majority of cases this has proved to be adequate for the overall result of the double integral to be accurate to within the specified value.

The function is not well-suited to non-smooth integrands, i.e., integrands having some kind of analytic discontinuity (such as a discontinuous or infinite partial derivative of some low-order) in, on the boundary of, or near, the region of integration. **Warning**: such singularities may be induced by incautiously presenting an apparently smooth interval over the positive quadrant of the unit circle, R

$$I = \int_R (x+y)\,dx\,dy.$$

This may be presented to nag_quad_2d_fin (d01dac) as

$$I=\int_0^1 dy\int_0^{\sqrt{1-y^2}}(x+y)\,dx=\int_0^1\left(\tfrac{1}{2}\left(1-y^2\right)+y\sqrt{1-y^2}\right)dy$$

but here the outer integral has an induced square-root singularity stemming from the way the region has been presented to nag_quad_2d_fin (d01dac). This situation should be avoided by re-casting the problem. For the example given, the use of polar coordinates would avoid the difficulty:

$$I=\int_0^1 dr\int_0^{\frac{\pi}{2}}r^2(\cos v+\sin v)\,dv.$$

10 Example

This example evaluates the integral discussed in Section 9, presenting it to nag_quad_2d_fin (d01dac) first as

$$\int_0^1\int_0^{\sqrt{1-y^2}}(x+y)\,dx\,dy$$

and then as

$$\int_0^1\int_0^{\frac{\pi}{2}}r^2(\cos v+\sin v)\,dv\,dr.$$

Note the difference in the number of function evaluations.

10.1 Program Text

```
/* nag_quad_2d_fin (d01dac) 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 <nagx01.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static double NAG_CALL phi1(double y, Nag_Comm *comm);
  static double NAG_CALL phi2a(double y, Nag_Comm *comm);
  static double NAG_CALL fa(double x, double y, Nag_Comm *comm);
  static double NAG_CALL phi2b(double y, Nag_Comm *comm);
  static double NAG_CALL fb(double x, double y, Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  static double ruser[5] = { -1.0, -1.0, -1.0, -1.0, -1.0 };
  Integer exit_status = 0;
  Integer npts, i;
  double absacc, ans, ya, yb;
  Nag_Comm comm;
  NagError fail;
  INIT_FAIL(fail);

  printf("nag_quad_2d_fin (d01dac) Example Program Results\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /* Input arguments */
#ifdef _WIN32
  scanf_s("%lf %lf", &ya, &yb);
#else
  scanf("%lf %lf", &ya, &yb);
#endif
#ifdef _WIN32
  scanf_s("%lf", &absacc);
#else
  scanf("%lf", &absacc);
#endif

  for (i = 1; i <= 2; i++) {
    /* nag_quad_2d_fin (d01dac).
     * Two-dimensional quadrature over a finite region.
     */
    switch (i) {
    case 1:
      printf("\nFirst formulation\n");
      nag_quad_2d_fin(ya, yb, phi1, phi2a, fa, absacc, &ans, &npts,
                      &comm, &fail);
      break;
    case 2:
      printf("\nSecond formulation\n");
      nag_quad_2d_fin(ya, yb, phi1, phi2b, fb, absacc, &ans, &npts,
                      &comm, &fail);
      break;
    }
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_quad_2d_fin (d01dac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    printf("Integral = %9.4f\n"
           "Number of function evaluations = %5" NAG_IFMT "\n", ans, npts);
  }

END:
  return exit_status;
}

static double NAG_CALL phi1(double y, Nag_Comm *comm)
{
  if (comm->user[0] == -1.0) {
    printf("(User-supplied callback phi1, first invocation.)\n");
    comm->user[0] = 0.0;
  }
  return 0.0;
}

static double NAG_CALL phi2a(double y, Nag_Comm *comm)
{
  if (comm->user[1] == -1.0) {
    printf("(User-supplied callback phi2a, first invocation.)\n");
    comm->user[1] = 0.0;
  }
  return (sqrt(1.0 - pow(y, 2)));
}

static double NAG_CALL fa(double x, double y, Nag_Comm *comm)
{
  if (comm->user[2] == -1.0) {
    printf("(User-supplied callback fa, first invocation.)\n");
    comm->user[2] = 0.0;
  }
  return (x + y);
}

static double NAG_CALL phi2b(double y, Nag_Comm *comm)
{
  if (comm->user[3] == -1.0) {
    printf("(User-supplied callback phi2b, first invocation.)\n");
    comm->user[3] = 0.0;
  }
  return (0.5 * nag_pi);
}

static double NAG_CALL fb(double x, double y, Nag_Comm *comm)
{
  if (comm->user[4] == -1.0) {
    printf("(User-supplied callback fb, first invocation.)\n");
    comm->user[4] = 0.0;
  }
  return (pow(y, 2) * (cos(x) + sin(x)));
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_quad_2d_fin (d01dac) Example Program Results

First formulation
(User-supplied callback phi1, first invocation.)
(User-supplied callback phi2a, first invocation.)
(User-supplied callback fa, first invocation.)
Integral =    0.6667
Number of function evaluations =   189

Second formulation
(User-supplied callback phi2b, first invocation.)
(User-supplied callback fb, first invocation.)
Integral =    0.6667
Number of function evaluations =    89
```