1991

Algorithm QUAD2D: General Two-Dimensional Quadrature

James V. Lambers

Report Number:
91-069

ALGORITHM QUAD2D: GENERAL TWO-DIMENSIONAL QUADRATURE

James V. Lambers

CSD-TR-91-069
December 1991
(Revised December 1992)

ALGORITHM QUAD2D: GENERAL TWO-DIMENSIONAL QUADRATURE

James V. Lambers*
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
Technical Report CSD-TR-91-069
CAPO Report CER-91-34
December 1991
Revised October 1992
Revised December 1992

Abstract

This algorithm computes integrals of functions over general two-dimensional domains. The domain is represented as the interior of a boundary defined by a set of smooth parameterized pieces. If the domain is multiply connected, then boundary curves are given for each hole as well as for the outside boundary.

Categories and Subject Descriptors: G.1.4. [**Numerical Analysis**]: Quadrature and Numerical Differentiation - *adaptive quadrature*;
G.4 [**Mathematics of Computing**]: Mathematical Software;
G.m [**Mathematics of Computing**]: Miscellaneous - *FORTRAN*

General Terms: Algorithms

Additional Keywords and Phrases: Integration, two-dimensional domains, general shapes, multiply-connected

*Research supported in part by the National Science Foundation Research Experience for Undergraduates program, CCR 86-19 817. Current address, Department of Computer Science, Stanford University, Palo Alto, CA

1 INTRODUCTION

We first describe the algorithm QUAD2D with its full complement of 20 arguments. Later, we identify two other versions: QUADSM (simple QUAD2D) with 18 arguments for domains with no holes, and QUADEZ (easy QUAD2D) with 8 arguments which uses default parameters and routines supplied with this algorithm. This algorithm computes the integral of a function $f(x, y)$ over a general two-dimensional domain D. The notation and terminology of the companion paper [1] is used. The following call to the FORTRAN subroutine QUAD2D returns the value RESULT of the computation:

```
CALL QUAD2D(F, X, Y, XPRIME, YPRIME, NCURVE, ICURVE,
  NPIECE, BRANGE, LCLKW, LDERIV, ERRABS, ERRREL, QBOUTR,
  QBINNR, QTOUTR, QTINNR, DIFFNM, IWKIN, RESULT)
```

The integrand $f(x, y)$ is F. The domain D has boundary pieces

$$x_i(p) = \text{X(P,I)}, \quad y_i(p) = \text{Y(P,I)}, \quad \text{I} = 1, \text{NPIECE}$$

These pieces define NCURVE boundary curves with the first curve the outside boundary. That is, the outside boundary is:

$$(\text{X(P,I)},\text{Y(P,I)}), \text{I} = 1, \text{ICURVE(1)}$$

and the Jth hole's boundary is:

$$(\text{X(P,I)},\text{Y(P,I)}), \text{I} = \text{ICURVE(J)} + 1, \text{ICURVE(J} + 1)$$

as the array ICURVE contains pointers to the last piece of each boundary curve. The two-dimensional array BRANGE contains the initial and final values for the parameter P of each piece of the boundary.

The logical variable LCLKW is .TRUE. if the outside boundary is oriented clockwise. All holes must use this same orientation of their boundaries. The logical variable LDERIV is .TRUE. if the user supplies the derivatives of the boundary parameterization functions, XPRIME and YPRIME, corresponding to X and Y. Otherwise, QUAD2D uses numerical differentiation.

The variables ERRABS and ERRREL govern the accuracy of each subarea integration. The error estimate EST of each such integration satisfies

$$\text{EST} \leq \text{MAX(ERRABS, ERRREL} * | \text{ SUBRES } |)$$

where SUBRES is the computed integral over the subarea. The total quadrature area is the sum of the errors on the subareas. If there are K such subareas, then the actual error could be multiplied by as much as K. One expects a statistical accumulation of the errors which would produce a total error of about $\sqrt{K}$ times the average subarea error.

The remaining input arguments are external routines for performing one-dimensional quadrature, numerical differentiation, and workspace initialization. QBOUTR and QBINNR perform outer and inner integration, respectively, on any area that is bounded by an edge of the approximating polygon and an arc of the boundary. On such an area, called a *band*, we evaluate an integral of the form

$$\int_a^b \int_{l(p)}^{y(p)} f(x(p), y) \frac{dx(p)}{dp} \, dy \, dp \tag{1}$$

or

$$\int_a^b \int_{l(p)}^{x(p)} f(x, y(p)) \frac{dy(p)}{dp} \, dx \, dp \tag{2}$$

where $x(p)$ and $y(p)$ are X and Y, respectively, and $l(p)$ denotes the appropriate point on the straight line bounding the area. These routines must be of the form

```
SUBROUTINE QBOUTR(FOUTER, FINNER, LINE, BNDRY1, BNDRY2,
    DERIV, A, B, ERRABS, ERRREL, LREV, QINNER, DDIFF, RESULT,
    ERREST, LSUCC)

SUBROUTINE QBINNR(FINNER, A, B, S, ERRABS, ERRREL, LREV,
    RESULT, ERREST, LSUCC)
```

In QBOUTR, FOUTER is the integrand function for the outer integral. It must have the form

```
DOUBLE PRECISION FUNCTION FOUTER(P, FINNER, LINE, BNDRY1,
    BNDRY2, DERIV, LREV, QBINNR, DIFFNM)
```

where P is the parameter value at which the inner integral should be evaluated. Similarly, FINNER is the integrand $f(x, y)$ for the inner integral. LINE is the function that defines the edge of the polygon that bounds the area as a function of either x or y, whichever is not the inner variable of integration. BNDRY1 is X if the inner integral is evaluated with respect to y, and Y otherwise. BNDRY2 is the other function defining the boundary. DERIV is the derivative of BNDRY1.

By default, FINNER is evaluated at the point (BNDRY1(P),S) where S is the inner variable of integration. If the logical argument LREV is .TRUE., then those arguments are reversed; this implies that LREV is .TRUE. when x is the inner variable of integration. QINNER is the quadrature routine used for evaluating the inner integral; it must have the same form as QBINNR. DDIFF is the numerical differentiation routine used for the inner integral; it must have the form of DIFFNM (see below). In QBINNR, S is the argument

to FINNER which is fixed when evaluating the inner integration (e.g. the value of x if the inner integral is made with respect to y).

In both routines, A and B are the limits of integration, ERRABS and ERRREL are the desired absolute and relative accuracy, RESULT is the estimate of the integral, ERREST is the error estimate, and LSUCC is a logical variable that is .TRUE. if no error condition occurred.

The routines QTOUTR and QTINNR are used for computing outer and inner integrals, respectively, over triangles of the approximating polygon. These routines have the form

```
SUBROUTINE QTOUTR(FOUTER, FINNER, EDGE1, EDGE2, A, B, ERRABS,
    ERRREL, QINNER, RESULT, ERREST)

SUBROUTINE QTINNR(FINNER, A, B, T, ERRABS, ERRREL, RESULT,
    ERREST)
```

FOUTER is the integrand for the outer integral, and it must have the form

```
DOUBLE PRECISION FUNCTION FOUTER(T, FINNER, EDGE1, EDGE2,
    QTINNR)
```

where T is the outer variable (X or Y) of integration. FINNER is the integrand $f(x, y)$ for the inner integral. EDGE1 and EDGE2 are the functions that define the non-vertical edges of the triangle as functions of x. QINNER is the quadrature routine for inner integration, which must have the form of QTINNR. In both routines, A and B are the limits of integration, ERRABS and ERRREL are the desired absolute and relative accuracy, RESULT is the estimate of the integral, and ERREST is the error estimate.

For integrating over the bands, QUAD2D needs a numerical differentiation routine for differentiating one of the boundary functions, X or Y. The differentiation routine DIFFNM has the form

```
DOUBLE PRECISION FUNCTION DIFFNM(F, P, IPC, BGSTEP, ERRTOL)
```

F is either X or Y. P and IPC are the arguments to F at which the derivative is computed. BGSTEP is the beginning step size for a finite difference scheme, and ERRTOL is the desired relative accuracy.

In addition, QUAD2D uses an external routine to initialize workspace for the quadrature routines, IWKIN. It accepts a single integer argument.

2 HEURISTIC CONSTANTS

Algorithm QUAD2D is parameterized by several heuristic constants. There may need to be changed for domains with badly behaved boundary curves or tuned to improve efficiency for particular classes of domains. These constants, along with their default values, are described below:

DEFMAG	The number of subdivisions made in each piece of the boundary during a boundary scan (default = 10^3)
RMAXMG	The maximum number of subdivisions allowed (default = 10^9)
RMULT	The factor by which the magnification is multiplied when the polygon needs to be refined (default = 10)
BGSTEP	Beginning step size for the numerical differentiation routine (default = 10^{-2})
ERRTOL	Error tolerance for the numerical differentiation routine (default = 10^{-9})
NWORK	The argument to IWKIN (default = 9024)
MAXVTX	The maximum number of vertices allowed in an approximating polygon (default = 500)
TOL	Tolerance parameter for changing orientation during a boundary scan (default = 1.25, must lie between 1 and 2)

3 USING QUAD2D

The QUAD2D software package consists of source code and documentation for the following: a library of FORTRAN routines including QUAD2D and its subprograms, a library of customized IMSL quadrature and differentiation routines, a C program called *xdplot* for plotting domains in the X-Windows environment, and a library of FORTRAN routines called Q2PLOT that helps produce plotting data for *xdplot*. In addition, FORTRAN programs for 52 test cases are included. The test cases are derived from the set of 26 parameterized domains in [1]. We now illustrate how to compile and run QUAD2D on these test cases with a transcript from a sample session. It is assumed that the UNIX is the computing environment and % is the system prompt. UNIX commands are given in italics.

```
% ls -F
Makefile   bin/   domains/   lib/      quad2d/
README     doc/   imsl/      q2plot/   xdplot/
```

The six directories *domains, imsl, lib, q2plot, quad2d*, and *xdplot* contain the software. The directory *doc* contains manual pages (in Tex format) for using *quad2d* and *xdplot*. The directory *bin* is for executable code after compilation. Every directory has a README file which describes its contents. The *Makefile* files are UNIX scripts to compile programs, create libraries, etc. If a different operating system is used, then these must be replaced.

To compile all of the libraries and *xdplot*, type

```
% make install
```

When this job is complete, the libraries and the *xdplot* executable are installed as shown:

```
% ls lib
libimsl.a   libq2plot.a   libquad2d.a
% ls bin
xdplot
```

Now change to the *domains* directory to run the test cases as indicated below.

```
% cd domains
% ls -F
Makefile        dom17.f  farea.f    q9nrsr.f   testall*
README          dom19.f  input/     r9nrmn.f   testdom.f
dom10.f         dom26.f  output/    r9nrxc.f
dom12.f         dom9.f   q9nrbc.f   r9nryc.f
```

The source files for the program *testdom* contain the definitions of the 26 domains from the set. Each source file with the prefix "dom" is a complete program for testing the domain with the indicated number; e.g. *dom12.f* is the source code for a program to test domain 12. *Testdom* uses numerical differentiation, and all of the other programs may be used with symbolic or numerical differentiation. Now compile all of the programs in this directory as follows:

```
% make all
% ls -F
Makefile    dom17.f   dom9.f     q9nrsr.o   testdom.f
README      dom17.o   dom9.o     r9nrmn.f   testdom.o
dom10*      dom19*    farea.f    r9nrmn.o
dom10.f     dom19.f   farea.o    r9nrxc.f
dom10.o     dom19.o   input/     r9nrxc.o
dom12*      dom26*    output/    r9nryc.f
dom12.f     dom26.f   q9nrbc.f   r9nryc.o
dom12.o     dom26.o   q9nrbc.o   testall*
dom17*      dom9*     q9nrsr.f   testdom*
```

Testdom accepts data from the standard input. The data consists of a domain number in the range 1 to 26, and two parameters. All of the data must be supplied on one line. The command, the input and the output (the quadrature results) are shown:

```
% testdom
16   1.25   0.75
     3.77953623210047
```

The *input* directory contains data files for *testdom* as shown below. The parameters in these files are the parameters listed in the table in [1] of computed areas for instances of the first 26 domains.

% *ls input*

01a	03b	06a	08b	11a	13b	16a	18b	21a	23b	26a
01b	04a	06b	09a	11b	14a	16b	19a	21b	24a	26b
02a	04b	07a	09b	12a	14b	17a	19b	22a	24b	
02b	05a	07b	10a	12b	15a	17b	20a	22b	25a	
03a	05b	08a	10b	13a	15b	18a	20b	23a	25b	

To run *testdom* on all of these data files, type

% *testall*

When *testall* is finished, the *output* directory contains output files which contain the output corresponding to the input files; e.g. *output/25b* contains the output corresponding to the data in *input/25b*:

% *cat output/25b*
3.57278155318597

To run any of the other programs, simply type its name. For example, for *dom19*, we have

% *dom19*
12.44057452601734

For further testing, it is helpful to be familiar with the programs. The main program in the file *dom19.f* is typical of all these programs:

```
C=============================================================
C
C     QUAD2D
C     NUMERICAL QUADRATURE IN TWO DIMENSIONS
C     BY JAMES LAMBERS
C
C=============================================================
C     PROGRAM DOM19
C
C     THIS PROGRAM IS USED TO INTEGRATE A FUNCTION OVER DOMAIN 19
C     OF THE 26 TEST DOMAINS ASSOCIATED WITH THE ALGORITHM QUAD2D.
C     THE SYMBOLIC DERIVATIVES OF THE BOUNDARY PARAMETER FUNCTIONS
C     ARE GIVEN SO THAT NUMERICAL DIFFERENTIATION CAN BE AVOIDED.
C     THESE ARE THE FUNCTIONS XPRIME AND YPRIME.
C
C        DOMAIN 19 = COMPLEX REGION WITH SHARP INTERIOR REENTRANT POINT
C        ******** THIS DOMAIN REQUIRES A VERY FINE GRID **************
C        ORIENTATION IS COUNTERCLOCKWISE
C        PARAMETERS MAP DOMAIN BY
C              X = U*(1.+A*U**2*(V-2.)**2/10.)
```

```
C               Y = V*(1.+B*U**2*(V-1.)**2/10.)
C        DEFAULT = (0, 0)
C                   NBOUND = 7
C      BRANGE 1 = 0      0     0   0   0   0   0.64   0
C      BRANGE 2 = 0.64   0.6   1   1   1   1   2      1
C
      IMPLICIT DOUBLE PRECISION(A-H, O-Z)
      PARAMETER(NCURVE = 1, NPIECE = 7)
      COMMON / C1PARM / A, B, PI
      LOGICAL LCLKW, LDERIV
      DIMENSION BRANGE(2,NPIECE), ICURVE(NCURVE)
      EXTERNAL X, Y, XPRIME, YPRIME, FAREA,
     A     DQDAG, DQDAG2, DQAND, DQAND2, DDERIV, IWKIN
C
C             ********* SET THE DOMAIN PARAMETERS HERE
      A = 1.0D0
      B = 0.0D0
      PI = DATAN(1.0D0) * 4.0D0
      ICURVE(1) = 7
      BRANGE(1,1) = 0.0D0
      BRANGE(2,1) = 0.6D0
      BRANGE(1,2) = 0.0D0
      BRANGE(2,2) = 1.0D0
      BRANGE(1,3) = 0.0D0
      BRANGE(2,3) = 1.0D0
      BRANGE(1,4) = 0.0D0
      BRANGE(2,4) = 1.0D0
      BRANGE(1,5) = 0.0D0
      BRANGE(2,5) = 1.0D0
      BRANGE(1,6) = 0.64D0
      BRANGE(2,6) = 2.0D0
      BRANGE(1,7) = 0.0D0
      BRANGE(2,7) = 1.0D0
      LCLKW = .FALSE.
      LDERIV = .TRUE.
      ERRABS = 0.0000000000001D0
      ERRREL = 0.0000000000001D0
             ******** SELECT THE QUADRATURE ROUTINE TO USE
C     CALL QUAD2D(FAREA, X, Y, XPRIME, YPRIME, NCURVE,
C    A     ICURVE, NPIECE, BRANGE, LCLKW, LDERIV, ERRABS,
C    B     ERRREL, DQDAG, DQDAG2, DQAND, DQAND2, DDERIV, IWKIN,
C    C     TOTAL)
```

```
C     CALL QUADSM(FAREA, X, Y, XPRIME, YPRIME,
C    A     NPIECE, BRANGE, LCLKW, LDERIV, ERRABS,
C    B     ERRREL, DQDAG, DQDAG2, DQAND, DQAND2, DDERIV,
C    C     IWKIN, TOTAL)
C     TOTAL = QUADEZ(FAREA, X, Y, NPIECE, BRANGE,
C    A     LCLKW, ERRABS, ERRREL)
      CALL Q2PLOT(X, Y, NCURVE, ICURVE, NPIECE,
     A     BRANGE, LCLKW)
C     WRITE(*,900) TOTAL
 900  FORMAT(F20.14)
      STOP
      END
```

To run *xdplot* on domain 19, one must edit this main program to comment out the calls to the QUAD2D interfaces and uncomment the call to Q2PLOT. Then one edits the *Makefile* in this directory, the first few lines of which are shown below:

```
FORTRAN=        f77
FFLAGS=         -O
LIBDIR=         ../lib
QLIB=           quad2d
LIB=        ${LIBDIR}/lib${QLIB}.a ${LIBDIR}/libimsl.a
```

Change the QLIB variable to *q2plot* so that the object files for domain 19 will be linked with the Q2PLOT library instead of the QUAD2D library. Then, assuming the UNIX environment variable DISPLAY is set, one proceeds as follows:

% *make dom19*
% *dom19 — xdplot -f*

4 PERFORMANCE

The performance of quadrature programs is normally measured in terms of the number of integrand, $f(x, y)$, evaluations used. For multidimensional quadrature the other algorithmic components are clearly larger than for traditional one dimensional quadrature. However, the number of integrand evaluations is inherently $O(n^d)$ where d is the dimension and n measures the work for a similar one dimensional problem. Thus measuring performance by counting $f(x, y)$ evaluations should continue to be reasonable for multidimensional quadrature.

The QUAD2D algorithm reduces the general integration problem to a set of integrations over triangles (using the program DQTRGO and DQTRG1) and over bands next to the boundaries (using the programs DQBNDO and DQBNDI). These integrations are, in turn, done with one dimensional quadrature. We have tested the performance of QUAD2D using IMSL library routines as illustrated by the sample program at the end of the previous

section. The triangle integration use DQAND, a standard Gauss quadrature algorithm (DQAND2 is a "copy" of DQAND so it can be used recursively). The band integrations use DQDAG, an efficient adaptive quadrature algorithm (again, DQDAG2 is a "copy" of DQDAG).

The areas (i.e., $f(x,y) = 1$) of each of the two instances of the 26 domains was computed with accuracy requests ERRABS and ERREL ranging from 0.1 to 10^{-13}. The principal observations are:

1. *The triangle integration is rather efficient.* The accuracy requests were usually met using 36 $f(x,y)$ evaluations, the minimum number possible using DQAND twice. Note that $36 = 6 \times 6$ and six is the minimum in one dimension for DQAND. This is to be expected since DQAND is exact except for round-off effects for this integration. A few times 72 $f(x,y)$ evaluations were used reflecting the effect of round-off on the accuracy test.

2. *The band integration is less efficient.* The minimum number of $f(x,y)$ evaluations for DQDAG is 21, so that using it twice requires at least $441 = 21 \times 21$ $f(x,y)$ evaluations. Not that adaptive integration is needed here to provide high reliability for domains with complex shapes. One level or more of adaption is used for 15 domains (numbers 2, 4, 5, 6, 9, 11, 12, 13, 14, 16, 18, 19, 23, 24, 25, and 26) resulting in a band integration with at least $1323 = 21 \times 21 \times 3$ evaluations. Domain 13 has instances where more adaption is used, requiring $3087 = 21 \times 21 \times 7$ evaluations for some bands. Domains 9, 12, 19, and 26 are the most difficult shapes and always require considerable adaption. These domains have cases were a band required over 10,000 $f(x,y)$ evaluations, the maximum observed was $21609 = 21 \times 21 \times 49$ evaluations.

Since every domain requires at least a few band integrations, the number of $f(x,y)$ evaluations is always over 1000. The total number of evaluations for ERRABS $= 10^{-13}$ and one instance of each of the 26 domains is listed below.

Domain	1	2	3	4	5	6	7	8	9
f Evaluations	1836	2277	2421	6975	10098	16884	3888	2421	18657
Domain	10	11	12	13	14	15	16	17	18
f Evaluations	2385	3267	53190	2718	7668	2934	4329	4914	7407
Domain	19	20	21	22	23	24	25	26	
f Evaluations	26586	2934	1908	1908	5868	5724	2790	21087	

In most cases, the number of evaluations is not reduced at all if the accuracy requested is reduced. This occurs only for the domains with very high counts. For example reducing the accuracy request to 10^{-2} for domains 12, 19, and 26 reduces the evaluation counts to 11736, 8946, and 9621, respectively.

That the use of full adaptive quadrature for the band integration is inefficient is seen by replacing DQDAG for the inner integration QBINNR by the simpler, non adaptive IMSL

routine DQAND. This routine uses a fixed sequence of Gauss quadrature points to achieve a requested accuracy. It has the ability to accept a quadrature estimate using only 6 function evaluations. Thus its use can require as few as $21 \times 6 = 126$ $f(x, y)$ evaluations for a band integration, 28.6% as many as using DQDAG for both QBOUTR and QBINNR. The actual overall improvement, of course, depends on how many triangle integrations there are, the details of DQDAG's adaption in the outer integration, as well as the logic in DQAND. The observed percentages in the number of $f(x, y)$ evaluations required for these 26 integrations average 33.9% and ranges from 29.1% to 37.6%. The values obtained for the quadrature did not change significantly. It is plausible that the special nature of the band integrations can be exploited to devise substantially more efficient quadrature than can be achieved by composing two one-dimensional quadrature methods.

Sample tests were run replacing $f(x, y) = 1$ by other simple, smooth functions and the results are qualitatively unchanged.

5 ACKNOWLEDGMENTS

I wish to thank Xing Kang Fu, John R. Rice and Pok-Yin Yu for testing QUAD2D and suggesting improvements in the code and documentation.

References

[1] J. V. Lambers and J. R. Rice, Numerical Quadrature For General Two-dimensional Domains. CSD-TR-91-067, Computer Sciences, Purdue University, September, 1991