Computing Multiple Integrals by MATLAB

Shengxiang Xia[1]
e-mail: summer_5069@yahoo.com.cn
College of Science, Shandong Jianzhu University,
Jinan 250101, China

Abstract.
In this short note, two MATLAB procedures are presented that can be used to automatically evaluate double and triple integrals on irregular regions.

1. Introduction

Since we can not analytically integrate every function, the need for approximate integration formulas is obvious. In addition, there might be situations where the given function can be integrated analytically, and yet an approximation formula may end up being a more efficient alternative to evaluating the exact value of the integral. The MATLAB function quad is available to perform approximate single integrations. MATLAB has a command dblquad which can be used to automatically evaluate a double integral over a rectangular region. If the integration region D is not rectangular, but is vertically or horizontally simple, then a number of options may be carried out:

(1) A change of variables may be used to transform D into a square region R. Then dblquad is used on R.

(2) If the inner integration can be calculated by hand, then the problem can be reduced to a single integral, the MATLAB function quad is available to perform approximate single integrations.

(3) A integrand function $f(x, y)$ can be defined on a rectangular region $R \supseteq D$ (containing the actual integration region D) in such away that $f(x, y) = 0$ for $(x, y) \notin D$; that is, the value of the function becomes zero outside the integration region D, which may result in more computations. Then dblquad is used over the region R.

MATLAB has a built-in triple integrator triplequad which can be used to automatically evaluate a triple integral over a cubic region, it can be used on an irregular region in a similar way as in (3), the other real option is to reduce the triple integral to a single integration and a double integral, then the methods of above are used. Using whatever methods to evaluate multiple integrals is very difficult by hand, unless the region and the integrand function are very simple.

In this paper, we provide two MATLAB procedures that can be used to automatically evaluate double and triple integrals over irregular integration regions, when the multiple integral is written as the iterated integrals. As each iterated integral is based on a composite Gaussian quadrature, we first review Gaussian Quadratures.

[1] * Supported by NSF of China (No. 51078225)

2. Gaussian Quadrature

Consider an ordinary quadrature rule with n points $x_1, x_2, ..., x_n$ and weights $w_1, w_2, ..., w_n$:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

What is the maximal possible degree of precision? (k degree of precision means that the rule is exact on all polynomials of degree at most k). For any number of points, an upper bound is provided in [1]: the degree of precision of an n-point rule is less than $2n$. We shall show that the maximum possible degree of precision $2n$-1 is achieved by a unique n-point rule. Without loss of generality, we may restrict to the interval $[a, b] = [-1, 1]$. Our chief tool will be the Legendre polynomials $P_n(x)$. Recall that deg $P_n = n$ and P_n is $L^2([-1,1])$ - orthogonal to $\wp_{n-1}$ (where $\wp_n$ denotes the space of polynomials of degree $\le n$). Let $x_1 < x_2 < ... < x_n$ denote the roots of P_n, which we know to be distinct and to belong to [-1, 1]. These are called the n Gauss points on [-1, 1]. We define a quadrature rule by $\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} I_n f(x)dx$, where $I_n f(x) \in \wp_{n-1}$ is the Lagrange interpolant to $f(x)$ at the n Gauss points. This is a standard interpolatory quadrature rule:

$$\int_{-1}^{1} I_n f(x) = \sum_{i=1}^{n} w_i f(x_i),\ w_i = \int_{-1}^{1} \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} dx\,.$$

This rule is called the n-point Gauss rule (Gauss-Legendre integration formula).

Theorem 2. 1 [1]. The n-point Gauss rule has degree of precision equal to $2n$-1.

The constants w_i and the roots x_i of the Legendre polynomials have been extensively tabulated. Table 2.1[1] lists these values for n=2, 3, 4, and 5.

An integral $\int_a^b f(x)dx$ over an arbitrary $[a, b]$ can be transformed into an integral over [-1, 1] by using the change of variable:

$$x = \frac{b-a}{2} t + \frac{b+a}{2}\ . \qquad (1)$$

Hence

$$\int_a^b f(x)dx = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{b-a}{2} t + \frac{b+a}{2}\right) dt\,. \qquad (2)$$

This allows the n-point Gauss rule to be applied on any interval $[a, b]$. The composite n-point Gauss rule requires that for a positive integer m, subdivide the interval $[a, b]$ into m subintervals, and then apply the n-point Gauss rule to each subinterval.

Note: We apply traditional techniques using either vertical or horizontal partitions (dx or dy respectively) to calculate a definite integral.

Table 2.1 *node points and weights*

n	x_k	w_k
2	$-\frac{\sqrt{3}}{3}, \frac{\sqrt{3}}{3}$	1, 1
3	$-\frac{1}{5}\sqrt{15}, 0, \frac{1}{5}\sqrt{15}$	$\frac{5}{9}, \frac{8}{9}, \frac{5}{9}$
4	$-\frac{1}{35}\sqrt{525+70\sqrt{30}}$, $-\frac{1}{35}\sqrt{525-70\sqrt{30}}$ $\frac{1}{35}\sqrt{525-70\sqrt{30}}$ $\frac{1}{35}\sqrt{525+70\sqrt{30}}$	$-\frac{1}{36}\sqrt{30}+\frac{1}{2}$ $\frac{1}{36}\sqrt{30}+\frac{1}{2}$ $\frac{1}{36}\sqrt{30}+\frac{1}{2}$ $-\frac{1}{36}\sqrt{30}+\frac{1}{2}$
5	-0.90617984593866 - 0.53846931010568 0 0.53846931010568 0.90617984593866	0.23692688505618 0.47862867049937 0.56888888888889 0.47862867049937 0.23692688505618

3. Double integral

In this section, we consider the numerical integration of a function $f(x, y)$ with respect to two variables *x* and *y* over the integration region $D = \{(x, y) \mid a \le x \le b,\ c(x) \le y \le d(x)\}$. The double integral quadratures in this paper are used when Fubini's theorem is valid so a double integral is expressible in terms of iterated integrals.

$$S = \iint_D f(x, y)dxdy = \int_a^b dx \int_{c(x)}^{d(x)} f(x, y)dy. \tag{3}$$

There are different ways to approximate (3) (see [2, 3, 4]). In this note, the numerical formula for this double integration over a two-dimensional region takes the form

$$S(a, b, c(x), d(x)) = \sum_{i=1}^{m} w_i \sum_{j=1}^{n} v_j f(x_i, y_{i,j}). \tag{4}$$

Here the weights w_i, v_j depend on the method of one-dimensional integration we choose.

We introduce a double integration routine "gaussdbl (fun, a, b, c, d, tol)" which uses the composite 5-point Gauss rule for both integrations and calls another routine "gauss2int". The details are given in the routine gaussdbl. The left/right boundary *a*/*b* of integration region given as the second/third input argument must be a number, while the lower/upper boundary *c*/*d* of integration region given as the fourth/fifth input argument may be either a number or a function of

x. The sixth input argument tol is an absolute error tolerance, the default value is 10^{-5}. The program terminates when abs(gauss2int(fun, a, b, c, d, 2M, 2N)- gauss2int(fun, a, b, c, d, M, N))<tol, where *M*/*N* is a positive integer, it is the number of subintervals of [*a*, *b*]/[*c*, *d*].

```
function ss=gaussdbl(fun, a, b, c, d, tol)
if nargin==5
    tol=1.0e-5;
end
M=2; N=2;
Smn=gauss2int(fun, a, b, c, d, 1, 1);
S2mn=gauss2int(fun, a, b, c, d, M, N);
k=1;
while abs(Smn-S2mn)>=tol & k<9
    Smn=S2mn;
    M=2*M;
    N=2*N;
    k=k+1;
    S2mn=gauss2int(fun, a, b, c, d, M, N);
end
ss=S2mn;
function ss=gauss2int(fun, a, b, c, d, M, N)
t=[-0.90617984593866, -0.53846931010568, 0, 0.53846931010568, 0.90617984593866];
A=[0.23692688505618, 0.47862867049937, 0.56888888888889, 0.47862867049937,
0.23692688505618];
hx=(b-a)/M;
for k=1:M+1;
     x(k)=a+(k-1)*hx;
end
ss=0;
for k=1:M
     stt(1:5)=0;
   for j=1:5
        tt(j)=((x(k+1)-x(k))*t(j)+x(k+1)+x(k))/2;
       if isnumeric(c)
            ct(j)=c;
       else
            ct(j)=feval(c, tt(j));  % in case c is given as a function of x
       end
       if isnumeric(d)
            dt(j)=d;
       else
            dt(j)=feval(d, tt(j));
       end
       stt(j)=gaussfy(fun, tt(j), ct(j), dt(j), N);
     end
     ss=ss+sum(A.*stt)*(x(k+1)-x(k))/2;
```

```
 end
function s=gaussfy(fun, x, c, d, N)
% use the composite 5-point Gauss rule for fun over [c, d]
s=0;
for k=1:N+1
      h=(d-c)/N;
      c1(k)=c+h*(k-1);
end
for k=1:N
      s=s+glegend(fun, x, c1(k), c1(k+1));
 end
function I=glegend(fun, x, c, d)
% use the 5-point Gauss rule for fun over [c, d]
if nargin==2
      c=-1; d=1;
end
t=[-0.90617984593866, -0.53846931010568, 0, 0.53846931010568, 0.90617984593866];
A=[0.23692688505618, 0.47862867049937, 0.56888888888889, 0.47862867049937,
0.23692688505618];
for k=1:5
     y(k)=((d-c)*t(k)+c+d)/2;
     fy(k)=feval(fun, x, y(k));
     s1=fy(k);
     if s1==-inf
          s1=-realmax;
    end
    if s1==inf
         s1=realmax;
    end
end
I=A*fy'*(d-c)/2;
```

Example 1 Use routine gaussdbl to approximate $S = \iint_D y^2 sin^2(x+y)\cos x dxdy$, for the region D in the plane described by $-\pi/2 \le x \le \pi/2,\ -\pi \le y \le \pi$.

Solution: The exact value of S is $\frac{2\pi^3}{3} - \frac{\pi}{3}$. We calculate S by gaussdbl, we get S = 19.62365356938493 and error = 3.81650e-10.

Remarks. (a) We compare gauss2int (or gaussdbl) with MATLAB program dblquad and TwoD[4]. gauss2int used 1400 evaluations to compute S to obtain a result in error by 3.85×10^{-10}. With 10(×144) evaluations of the integrand, TwoD computed an approximation with error 1.67×10^{-9} and dblquad used 1574 evaluations to obtain a result in error by 1.1×10^{-7}.

(b) For $I=\int_0^2\int_0^{d(x)}(xy)^{-0.1}dydx$, with $d(x)=3\left(1-(\frac{x}{2})^{3/2}\right)^{2/3}$ (see [4]). gauss2int used 89375 evaluations to calculate I in error by 8.72×10^{-4}, with 190(×144) evaluations of the integrand, TwoD compute an approximation with error 2.29×10^{-4}. We see that approximations of I computed by gauss2int and TwoD are very close.

(c) For the families (2), (3) in [4], with same tolerance, we compare gaussdbl with MATLAB program dblquad and TwoD [4], and we find that approximations computed by gaussdbl and TwoD are very close, gaussdbl and TwoD appear to be more efficient than dblquad.

(d) For the families (2) in [4], with same tolerance, we use gaussdbl, TwoD and dblquad to compute 500 integrals, gaussdbl took 11.20s, dblquad took 14.17s, and TwoD took 2.56s. For the families (3) in [4], gauss2int used 93600 evaluations to compute 20 integrals, which took 19.49s, dblquad used 451718 evaluations, which took 43.48s, and TwoD used 10902 evaluations , which took 7.61s, and all approximations computed by gauss2int, dblquad and TwoD are almost same.

By above (a), (b), (c) and (d), we see that gauss2int (gaussdbl) appears to be more efficient than dblquad and TwoD appears to be more efficient than gauss2int.

Example 2 Use routine gaussdbl to approximate $S=\iint_D\frac{x+y}{x^2+y^2}dxdy$, for the region D in the plane described by $x^2+y^2\le1$ and $x+y\ge1$.

Solution: The integral region D is shown shaded in Figure 1. The integration region D is written as the iterated integral $S=\iint_D\frac{x+y}{x^2+y^2}dxdy=\int_0^1dx\int_{1-x}^{\sqrt{1-x^2}}\frac{x+y}{x^2+y^2}dy$, it is also expressible as $S=\int_0^{\frac{\pi}{2}}d\theta\int_{\frac{1}{\sin\theta+\cos\theta}}^1\left(\frac{r(\sin\theta+\cos\theta)}{r^2}\right)rdr=\int_0^{\frac{\pi}{2}}d\theta\int_{\frac{1}{\sin\theta+\cos\theta}}^1(\sin\theta+\cos\theta)dr$. It is easily to show that $S=2-\frac{\pi}{2}$.

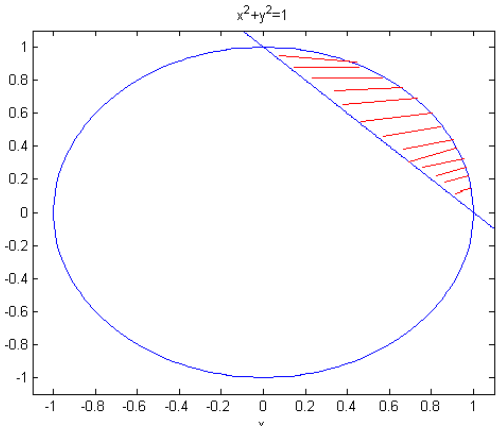

Figure 1 *the integral region D*

We constructed a MATLAB program "exa2" in order to use the routines gaussdbl, TwoD and dblquad for computing *S*.

```
function exa2
c_x=@(theta)1./(sin(theta)+cos(theta));
fun2=@(theta, r)sin(theta)+cos(theta);
tic
SS1=gaussdbl(fun2, 0, pi/2, c_x, 1)
toc
err1=SS1-(2-pi/2)
tic
SS2=TwoD(fun2, 0, pi/2, c_x, 1)
toc
err2=SS2-(2-pi/2)
tic
SS3=dblquad(@fexa2,0, pi/2, 0, 1)
toc
err3=SS3-(2-pi/2)
 function v=fexa2(theta, r)
% Set fexa2(theta, r)=0 outside region.
v=(sin(theta)+cos(theta)).*(r>=1./(sin(theta)+cos(theta)));
end
end
```

We run the program "exa2.m" to get the following results.

```
>> exa2
SS1 = 0.429203673205172
Elapsed time is 0.006496 seconds.
err1 = 6.89e-014
SS2 = 0.429203673205103
Elapsed time is 0.006716 seconds.
err2 =  -1.11e-016
SS3 = 0.429200625654125
Elapsed time is 0.201904 seconds.
err3 =  -3.05e-006
```

Example 3 Use routine gaussdbl to approximate the volume of the solid Ω bounded by the plane $z=2-x$ and the surface $z=4-x^2-y^2$ as displayed in Figure 2.

Solution: The integral region D is determined by the circle $(x-\frac{1}{2})^2+y^2=\frac{9}{4}$, that is $D=\{(x,y)\,|\left(x-\frac{1}{2}\right)^2+y^2\le\frac{9}{4}\}=\{(x,y)\,|-\sqrt{2-x^2+x}\le y\le\sqrt{2-x^2+x},\ -1\le x\le 2\}$. Thus the volume of the solid Ω

$$V=\iint_D\left[\left(4-x^2-y^2\right)-(2-x)\right]dS=\int_{-1}^{2}dx\int_{-\sqrt{2-x^2+x}}^{\sqrt{2-x^2+x}}\left[\left(4-x^2-y^2\right)-(2-x)\right]dy\,.$$

We know that the exact value of V is $\frac{81}{32}\pi$. We constructed a MATLAB program "exa3" in order to use the routines gaussdbl, TwoD and dblquad for computing V.

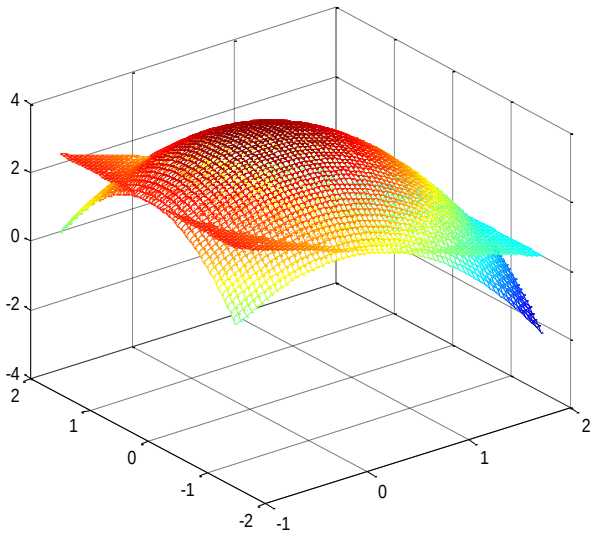

Figure 2 *the solid* Ω

```
function exa3
x=[-1:0.05:2];
y=[-3/2:0.05:3/2];
[X, Y]=meshgrid(x, y);
exa3_f1=inline('4-x.^2-y.^2', 'x', 'y');
exa3_f2=inline('2-x', 'x', 'y');
exa3_fun=@(x, y)2-x.^2-y.^2+x;
Z1=exa3_f1(X, Y);
Z2=exa3_f2(X, Y);
mesh(X, Y, Z1);
hold on;
mesh(X, Y, Z2);
hold off
c_x=@(x)-sqrt(2-x.^2+x);
d_x=@(x)sqrt(2-x.^2+x);
tic
V1=gaussdbl(exa3_fun, -1, 2, c_x, d_x)
toc
err1=81/32*pi-V1
tic
V2=TwoD(exa3_fun, -1, 2, c_x, d_x)
toc
err2=81/32*pi-V2
tic
```

```
V3=dblquad(@fexa3, -1, 2, -2,2)
toc
err3=81/32*pi-V3
function v=fexa3(x, y)
% Set fexa3(x, y)=0 outside region.
v=(2-x.^2-y.^2+x).*(-sqrt(2-x.^2+x)<=y).*(y<=sqrt(2-x.^2+x));
end
end
```

We run the program "exa3.m" to get the following results.

```
>> exa3
V1 =  7.952155747734767
Elapsed time is 0.598901 seconds.
err1 =  6.5666e-007
V2 = 7.952156282679086
Elapsed time is 0.032663 seconds.
err2 = 1.2172e-007
V3 = 7.952162828115892
Elapsed time is 0.274504 seconds.
err3 = -6.4237e-006
```

4. Triple integral

Suppose that a triple integral can be written as the iterated integral $V = \iiint_{\Omega} f(x,y,z)dxdydz$ $= \int_a^b dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} f(x,y,z)dz$, and let $g(x,y) = \int_{z_1(x,y)}^{z_2(x,y)} f(x,y,z)dz$, $h(x) = \int_{y_1(x)}^{y_2(x)} g(x,y)dy$, $V = \int_a^b h(x)dx$. We introduce a triple integration routine gausscub(fun, a1, a2, b1, b2, c1, c2, tol) that calls another routine "gauss3int(fun, a1, a2, b1, b2, c1, c2, N1, N2, N3)" which uses the composite 5-point Gauss rule for each integral. The computational steps are as follows.

(1) Subdivide the interval $[a, b]$ into N_1 subintervals, apply the 5-point Gauss rule to each subinterval $[x_k, x_{k+1}]$, and apply the composite 5-point Gauss rule for $h(x)$ over $[a,b]$, which calls another routine "gaussfx2()" for computing $h(x_k)$ (see (2)).

(2) For fixed x_k, subdivide the interval $[y_1(x_k), y_2(x_k)]$ into N_2 subintervals, apply the 5-point Gauss rule to each subinterval, and apply the composite 5-point Gauss rule for $g(x_k, y)$ over $[y_1(x_k), y_2(x_k)]$ to get $h(x_k)$, which calls another routine "gaussfx3()" for computing $g(x_k, y_j)$ (see (3)).

(3) For fixed x_k, y_j subdivide the interval $[z_1(x_k, y_j), z_2(x_k, y_j)]$ into N_3 subintervals, apply the 5-point Gauss rule to each subinterval, and apply the composite 5-point Gauss rule for $f(x_k, y_j, z)$ over $[z_1(x_k, y_j), z_2(x_k, y_j)]$ to get $g(x_k, y_j)$.

The details are given in the routine gausscub(fun, a1, a2, b1, b2, c1, c2, tol). The front/back boundary a_1/a_2 of variable x of integration region given as the second/third input argument must be a number, the left/right boundary b_1/b_2 of variable y of integration region given as the

fourth/fifth input argument may be either a number or a function of x, while the lower/upper boundary c_1/c_2 of variable z of integration region given as the sixth/seventh input argument may be either a number or a function of x, y. The eighth input argument tol is an absolute error tolerance, the default value is 10^{-5}. The program terminates when abs(gauss3int(fun, a1, a2, b1, b2, c1, c2, 2*N1, 2*N2, 2*N3)- gauss3int(fun, a1, a2, b1, b2, c1, c2, N1, N2, N3))<tol, where N_1/N_2 /N_3 is a positive integer, it is the number of subintervals of [a_1, a_2]/ [b_1, b_2]/ [c_1, c_2].

```
function vv=gausscub(fun, a1, a2, b1, b2, c1, c2, tol)
if nargin==7
    tol=1.0e-5;
end
M=2;
Vmn=gauss3int(fun, a1, a2, b1, b2, c1, c2, 1, 1, 1)
V2mn=gauss3int(fun, a1, a2, b1, b2, c1, c2, M, M, M)
k=1;
while abs(Vmn-V2mn)>=tol & k<10
    Vmn=V2mn;
    M=2*M;
     k=k+1;
    V2mn=gauss3int(fun, a1, a2, b1, b2, c1, c2, M, M, M)
end
vv=V2mn;

function ss=gauss3int(fun, a1, a2, b1, b2, c1, c2, N1, N2, N3)
t=[-0.90617984593866 -0.53846931010568 0 0.53846931010568 0.90617984593866];
A=[0.23692688505618  0.47862867049937 0.56888888888889 0.47862867049937
0.23692688505618];
hx1=(a2-a1)/N1;
for k=1:N1+1;
    x1(k)=a1+(k-1)*hx1;
end
ss=0;
for k=1:N1
   for j=1:5
        ta(j)=((x1(k+1)-x1(k))*t(j)+x1(k+1)+x1(k))/2;
        sta(j)=gaussfx2(fun, ta(j), b1, b2, c1, c2, N2, N3);
   end
   ss=ss+sum(A.*sta)*hx1/2;
 end

function ss=gaussfx2(fun, x1, b1, b2, c1, c2, N2, N3)
% use the composite 5-point Gauss rule for fun over [b1, b2]
t=[-0.90617984593866 -0.53846931010568 0 0.53846931010568 0.90617984593866];
A=[0.23692688505618  0.47862867049937 0.56888888888889 0.47862867049937
0.23692688505618];
```

```
    if isnumeric(b1)
        by1=b1;
    else
        by1=feval(b1, x1);  % in case b1 is given as a function of x
    end
    if isnumeric(b2)
        by2=b2;
    else
        by2=feval(b2, x1); % in case b2 is given as a function of x
  end
  hx2=(by2-by1)/N2;
  for k=1:N2+1;
      x2(k)=by1+(k-1)*hx2;
  end
  ss=0;
  for k=1:N2
     for j=1:5
        tb(j)=((x2(k+1)-x2(k))*t(j)+x2(k+1)+x2(k))/2;
        stb(j)=gaussfx3(fun, x1, tb(j), c1, c2, N3);
     end
      ss=ss+sum(A.*stb)*hx2/2;
 end

function s=gaussfx3(fun, x1, x2, c1, c2, N)
% use the composite 5-point Gauss rule for fun over [c1, c2]
t=[-0.90617984593866 -0.53846931010568 0 0.53846931010568 0.90617984593866];
A=[0.23692688505618  0.47862867049937 0.56888888888889 0.47862867049937
0.23692688505618];
  if isnumeric(c1)
      cz1=c1;
  else
      cz1=feval(c1, x1, x2);  % in case b1 is given as a function of x, y
  end
  if isnumeric(c2)
      cz2=c2;
  else
      cz2=feval(c2, x1, x2); % in case b2 is given as a function of x, y
  end
  h=(cz2-cz1)/N;
  s=0;
  for k=1:N+1
      cc1(k)=cz1+h*(k-1);
  end
  for k=1:N
    for j=1:5
       x3(j)=((cc1(k+1)-cc1(k))*t(j)+cc1(k+1)+cc1(k))/2;
```

```
        fx3(j)=feval(fun, x1, x2, x3(j));
        s1=fx3(j);
    if s1==-inf
        s1=-realmax;
    end
    if s1==inf
        s1=realmax;
    end
        fx3(j)=s1;
    end
        s=s+A*(fx3)'*h/2;
end
```

We discuss the performance of gausscub and triplequad on a family of

$$\int_0^1\int_0^1\int_0^1 \cos(2\pi\xi_1+\tau_1x_1+\tau_2x_2+\tau_3x_3)dx_3dx_2dx_1. \qquad (5)$$

Each problem is constructed as follows: First ξ_1 is picked randomly from $[0,\ 1]$. The parameters $\tau_1,\ \tau_2$ and τ_3 are picked randomly from $[0,\ 1]$ and then scaled so that $\tau_1+\tau_2+\tau_3=15$. For the family (5), gausscub used 1137500 evaluations of the integrand to compute the 100 integrals which took 19.47s, and triplequad used 801422 evaluations, which took 93.22s. We compare the approximations computed by gausscub and triplequad with the approximations computed by MATLAB function "int" for the 100 integrals, and find that gausscub appears to be notably more efficient than triplquad.

Example 4 Use routine gausscub to approximate the triple integral

$$V=\iiint_{\Omega}(x+2z)\sin y dv=\int_0^{\frac{\pi}{4}}dy\int_0^{y}dz\int_0^{y+z}(x+2z)\sin y dx.$$

Solution: The order of iterated integral is $y\leftarrow z\leftarrow x$, for using routines gausscub and triplequad, the integrand and the bound functions of the integral should be defined in this order (see the following MATLAB program "exa4"). The exact value of V is

$$\frac{17}{8}\sqrt{2}\pi-\frac{17}{2}\sqrt{2}-\frac{17}{768}\sqrt{2}\pi^3+\frac{17}{64}\sqrt{2}\pi^2.$$

```
function exa4
b2=inline('y', 'y');
c2=inline('y+z ', 'y ', 'z ');
exam_4=inline('(x+2*z)*sin(y)', 'y', 'z', 'x ');
VV=(17*sqrt(2)*pi/8-17*sqrt(2)/2-17*sqrt(2)*pi^3/768+17*sqrt(2)*pi^2/64);
tic
V1=gausscub(exam_4, 0, pi/4, 0, b2, 0, c2)
toc
err1=VV-V1
tic
```

```
V2=triplequad(@fexa4, 0, pi/4, 0, pi/4, 0, pi/2)
toc
err2=VV-V2
function v=fexa4(y, z, x)
% Set fexa4(x, y, z)=0 outside region.
v=(x+2*z).*sin(y).*(z<=y).*(x<=(y+z));
end
end
```

We run the program "exa4.m" to get the following results.

```
>> exa4
V1 = 0.157205682755273
Elapsed time is 6.960517 seconds.
err1 = 2.914e-015
V2 = 0.157214226369327
Elapsed time is 1.141406 seconds.
err2 = -8.5436e-006
```

We see that the degree of approximation of the result evaluated by the routine gausscub is very high, and gausscub appears to be more efficient than triplequad.

Example 5 Find the triple iterated integral

$$V = \int_0^1 \int_0^{\pi} \int_0^{\pi} -3ze^{-xy-z^2}(cos(xy) - 10cos(xy)xy + 3xysin(xy) + 4cos(xy)x^2y^2 - sin(xy))dzdydx.$$

Solution: We constructed a MATLAB program "exa5" in order to use the routines gausscub and triplequad for computing *V*.

```
function exa5
s=1.27124615015737694510572433816 69; % the value computed by Matlab function int
fevals=0;
tic
s1=gausscub(@fun1, 0, 1, 0, pi, 0, pi)
toc
err1=s1-s
fprintf(['This cost ', num2str(fevals),' evaluations of f. \n'])
fevals=0;
tic
s2=triplequad(@fun1, 0, 1, 0, pi, 0, pi)
toc
err2=s2-s
fprintf(['This cost ',num2str(fevals),' evaluations of f. \n'])

 function v=fun1(x, y, z)
% Set fun1(x,y,z)=0 outside region.
fevals=fevals+1;
```

```
v=-3.*z.*exp(-x.*y-z.^2).*(cos(x.*y)-10.*cos(x.*y).*x.*y+3*x.^2.*sin(x.*y).*y.^2
+4.*cos(x.*y).*x.^2.*y^2-sin(x.*y));
end
end
```

We run the program "exa5.m" to get the following results.

```
>> exa5
s1= 1.271246152898202
Elapsed time is 0.19 seconds.
err1 = 2.7408e-009
This cost 11375 evaluations of f.
s2= 1.271246466089101
Elapsed time is 1.8 seconds.
err2 = 3.16e-007
This cost 15692 evaluations of f.
```

Example 6 Use gausscub and triplequad to approximate $\iiint_{\Omega} \sqrt{xyz}\,dxdydz$, here Ω is the region in the first octant bounded by the cylinder $x^2+y^2=4$, the sphere $x^2+y^2+z^2=4$, and the plane $x+y+z=8$ as displayed in Figure 3.

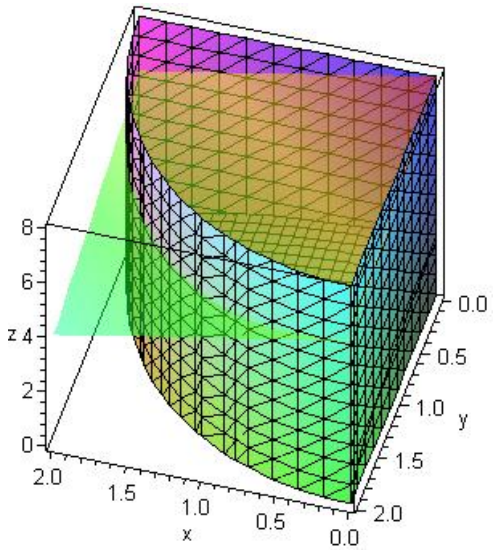

Figure 3 *the region* Ω

Solution: We rewrite the above triple integral as the iterated integral.

$$V=\iiint_{\Omega}\sqrt{xyz}\,dxdydz=\int_0^2 dx\int_0^{\sqrt{4-x^2}} dy\int_{\sqrt{4-x^2-y^2}}^{8-x-y}\sqrt{xyz}\,dz\,.$$

We constructed a MATLAB program "exa6" in order to use the routine gausscub and triplequad for computing V.

```
function exa6
v=20.34426877; % the value computed by the maple function int
cxy=inline('8-x-y', 'x', 'y');
dxy=inline('sqrt(4-x^2-y^2)', 'x', 'y');
bx=inline('sqrt(4-x^2)', 'x');
fevals=0;
tic
v1=gausscub(@fexa6, 0, 2, 0, bx, dxy, cxy, 0.01)
err1=v1-v
fprintf(['This cost ', num2str(fevals),' evaluations of f. \n'])
toc
fevals=0;
tic
v2=triplequad(@fun1, 0, 2, 0, 2, 0, 8, 0.00001)
err2=v2-v
fprintf(['This cost ', num2str(fevals),' evaluations of f. \n'])
toc

function w=fexa6(x, y, z)
fevals=fevals+1;
w=sqrt(x*y*z);
end

function w=fun1(x, y, z)
% Set fun1(x, y, z)=0 outside region.
fevals=fevals+1;
w=sqrt(x*y*z).*(y<=sqrt(4-x.^2)).*(z>=sqrt(4-x.^2-y.^2)).*(z<=(8-x-y));
end
end
```

We run the program “exa6.m” to get the following results.

```
>> exa6
v1=20.352902524600498
err1=0.008633754600499
This cost 11375 evaluations of f.
Elapsed time is 1.049463 seconds.
v2=20.312806858385787
err2= -0.031461911614212
This cost 209684 evaluations of f.
Elapsed time is 25.553564 seconds.
```

5. Conclusion

The two MATLAB procedures gaussdbl and gausscub introduced in this paper can be used to automatically evaluate double and triple integrals respectively on irregular regions, which can be taught as basic algorithms in, for example, a numerical integration course. The performances of

gausscub and triplequad (gaussdbl, TwoD and dblquad) are discussed on several examples, and it is shown that the gaussdbl and gausscub are effective and capable programs for approximating numerically double and triple integrals. Although gausscub appears to be more efficient than triplequad for many examples, yet the degrees of accuracy of approximations computed by gausscub are very poor for some complicated integrals, especially for integrands with peaks. We need to develop more efficient algorithms for triple integrals.

Acknowledgement. The author is very grateful to the referees for valuable comments and corrections.

References

[1] Richard L. Burden， J. Douglas Faires. Numerical Analysis (Seventh Edition). Brooks/ Cole Thomson, Pacific Grove, CA, 2001.
[2] F.N. Fritsch, D.K. Kahaner, and J.N. Lyness, Double integration using one-dimensional adaptive quadrature routines: a software interface problem, ACM Trans. Math. Software, 7 (1981) 46-75.
[3] J. Lyness, When not to use an automatic quadrature routine, SIAM Review, 25 (1983) 63-87.
[4] L.F. Shampine, Matlab Program for Quadrature in 2D, Appl. Math. Comp., 202 (2008) 266-274.