STUDIA UNIV. "BABEŞ–BOLYAI", MATHEMATICA, Volume **XLVII**, Number 4, December 2002

An adaptive cubature on triangle

GH. COMAN, IOANA POP AND RADU T. TRÎMBIŢAŞ

Dedicated to Professor D.D. Stancu on his 75^{th} birthday

Abstract. Starting from an elementary cubature formula on triangle which is exact on $\mathbb{P}_2^2$ (bivariate polynomials having global degree 2) an adaptive cubature method is devised. Also a MATLAB implementation is given.

1. Introduction

Let us consider the triangle Δ with the vertices V_i, $i = \overline{1,3}$, and the cubature formula (see figure 1):

$$\begin{aligned} P_i &= \frac{1}{2}\left(V_j + V_k\right), \qquad \{i,j,k\} = \{1,2,3\}, \\ I &= \int_T f(x,y)\,\mathrm{d}x\,\mathrm{d}y \approx \frac{\mathrm{area}(\Delta)}{6}\sum_{i=1}^{3} f(P_i). \end{aligned} \tag{1}$$

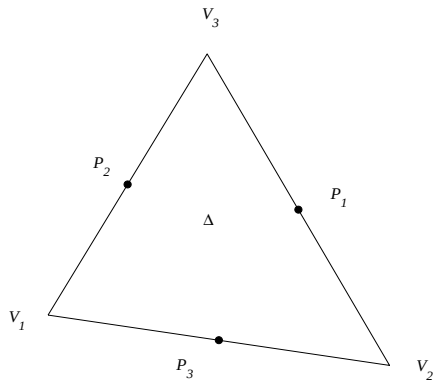

FIGURE 1. The triangle and edges' midpoints.

It can be easily seen that P_i, $i = \overline{1,3}$, are the midpoints of the triangle edges and the formula is exact for each $f \in \mathbb{P}_2^2$ (bivariate polynomials having total degree

Received by the editors: 01.09.2002.

equal to 2). We shall try to turn this formula into and adaptive cubature algorithm. The idea is to combine two elementary cubature formula, one which subevaluates I and one which overestimates it. If the absolute value of the difference of the results provided by these formulae is less than a given tolerance ε, we stop, and the result is the value given by the second formula. Otherwise, we proceed with a subdivision of the triangle, and apply the same method to each triangle of the subdivision.

2. The algorithm

For a detailed description of an adaptive numerical integration algorithm see [2, pp. 166–170]. We can decompose our triangle, denoted by Δ, into four triangles, Δ_1, Δ_2, Δ_3 and Δ_4 determined by vertices and the middle points (a Delaunay triangulation, [1], see Figure 2). The first step will be the formula given by (1) applied to Δ and the second step will be the same, but applied to each of the four triangle of the triangulation, Δ_i, $i = \overline{1,4}$. Let I_1 be the value provided by the elementary formula

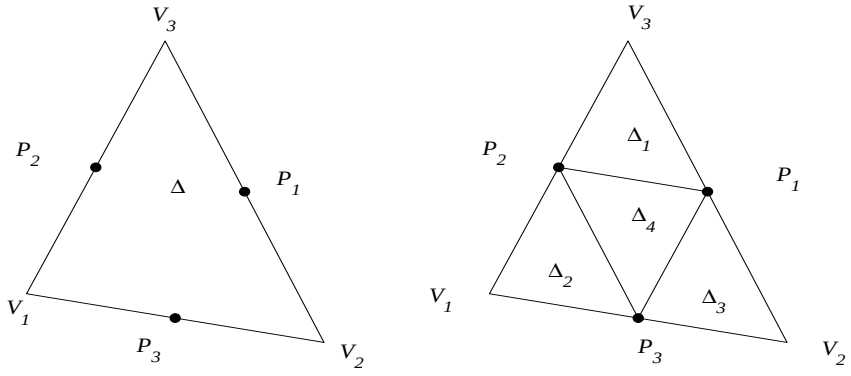

FIGURE 2. Initial triangle and the subdivision

(1), and I_2 the value obtained summing the four values obtained applying (1) to each triangle of the subdivision. A stopping criterion could be

$$|I_1 - I_2| < \varepsilon,$$

where ε is the desired tolerance. If the criterion is not fulfilled, then we apply the same procedure recursively to each triangle of the subdivision.

The detailed description is given in Algorithm 1.

Algorithm 1 Adaptive cubature algorithm on triangle; call $result := \text{adapt}(f, \Delta, \varepsilon)$, where f is the function, Δ is the triangle and ε is the desired tolerance; elem_formula implements the elementary cubature, given by (1)

Let $\Delta_1, \Delta_2, \Delta_3, \Delta_4$ be the triangles determined by vertices and midpoints
$I_1 := \text{elem\_formula}(f, \Delta)$;
$I_2 := \text{elem\_formula}(f, \Delta_1) + \text{elem\_formula}(f, \Delta_2) +$
$\text{elem\_formula}(f, \Delta_3) + \text{elem\_formula}(f, \Delta_4)$;
if $|I_1 - I_2| < \varepsilon$ **then**
$result := I_2$;
else
$result := \text{adapt}(f, \Delta_1, \varepsilon) + \text{adapt}(f, \Delta_2, \varepsilon) +$
$\text{adapt}(f, \Delta_3, \varepsilon) + \text{adapt}(f, \Delta_4, \varepsilon)$;
end if

3. The implementation

For adaptive cubature on triangle implementation see [3, 4].

We have implemented this algorithm in MATLAB[1]. The implementation follows the description given by algorithm 1. We introduced an auxiliary input parameter, `trace`, which (when is nonzero) allows us to obtain information about the execution and to represent graphically the process of computing. When trace is set, the additional output parameter, `stat` gives us the number of function evaluation and the number of triangles. Some optimizations which save several function evaluations are possible. Since the value I_1 is the value of the integral on the triangle Δ_4, we compute it once and give it as an input parameter further. We can do the same thing with the values of function in midpoints. Here is the MATLAB source code:

```
function [vi,stat]=mpcubatd2vb(f,x,y,err,trace)
%cubature with middle points, exact for P_2^2%
%call vi=mpcubatd2(f,x,y,err,trace)
%f - the function
%x,y - coordinates of vertices
```

[1] MATLAB©is a trademark of MathWorks, Inc., Natick, MA 01760-2098

```
%err - the error
%trace - tracing  indicator

global FEN TRIN

if nargin<5
    trace=0;
else
    if trace
        clf
        FEN=0; TRIN=0;
    end
end
if nargin < 4
    err=1e-3;
end
[xp,yp]=midpoints(x,y); %subdivision
fp=feval(f,xp,yp);
area=1/2*abs(det([x(:),y(:),ones(3,1)]));
I1=area/3*sum(fp);
vi=quadrg2(f,x,y,xp,yp,fp,err,area,I1,trace);
if trace & (nargout==2)
    FEN=FEN+3;
    stat=struct('nev',FEN,'ntri',TRIN);
end

function vi=quadrg2(f,x,y,xp,yp,fp,err,area,I1,trace)
%cubature with midpoints, internal use
%call vi=quadrg2(f,x,y,xp,yp,fp,err,trace)
%f - the function
%x,y - coordinates of vertices
%xp,yp - midpoints  coordinates
```

```
%fp - value of f in barycenter
%err - the error
%area - area of the triangle
%I1 - the first estimation (elementary formula)
%trace - tracing  indicator

global FEN TRIN
if trace
    fill(x,y,'r','FaceColor','none','EdgeColor','k'); hold on;
    axis equal
    plot(xp,yp,'ok');
    %pause
end
v1x=[xp(2),xp(1),x(3)]; v1y=[yp(2),yp(1),y(3)];
[P1Mx,P1My]=midpoints(v1x,v1y);
v2x=[x(1),xp(3),xp(2)]; v2y=[y(1),yp(3),yp(2)];
[P2Mx,P2My]=midpoints(v2x,v2y);
v3x=[xp(3),x(2),xp(1)]; v3y=[yp(3),y(2),yp(1)];
[P3Mx,P3My]=midpoints(v3x,v3y);
[P4Mx,P4My]=midpoints(xp,yp);
fP1M=feval(f, P1Mx, P1My);
fP2M=feval(f, P2Mx, P2My);
fP3M=feval(f, P3Mx, P3My);
if trace
    FEN=FEN+9;
    TRIN=TRIN+4;
end
zz=area/12;
arean=area/4;
I11=zz*sum(fP1M);
I12=zz*sum(fP2M);
I13=zz*sum(fP3M);
```

```
I14=zz*(fP1M(3)+fP2M(1)+fP3M(2));
I2=I11+I12+I13+I14;
if abs(I2-I1)<err
  vi=I2;
else
 vi=quadrg2(f,v1x,v1y,P1Mx,P1My,fP1M, err,arean,I11,trace)+...
     quadrg2(f,v2x,v2y,P2Mx,P2My,fP2M,err,arean,I12,trace)+...
     quadrg2(f,v3x,v3y,P3Mx,P3My,fP3M,err,arean,I13,trace)+...
     quadrg2(f,xp,yp,P4Mx, P4My,[fP1M(3),fP2M(1),fP3M(2)],+...
       err,arean,I14,trace);
end %if

function [bx,by]=midpoints(x,y)
bx=[x(2)+x(3),x(1)+x(3),x(1)+x(2)]/2;
by=[y(2)+y(3),y(1)+y(3),y(1)+y(2)]/2;
```

4. **A numerical example**

We wish to approximate

$$\int_{\Delta} f(x,y)\,\mathrm{d}x\,\mathrm{d}y,$$

where Δ is the triangle with vertices $V_1(0,0)$, $V_2(1,0)$, $V_3(0,1)$ and f is

$$f(x,y) = y\sin x.$$

The graph of f is given in Figure 3.

The exact value is

$$\cos(1) - 1/2 \approx 0.04030230586814,$$

obtained with the following Maple session:

```
> Digits:=20;
```

$$Digits := 20$$

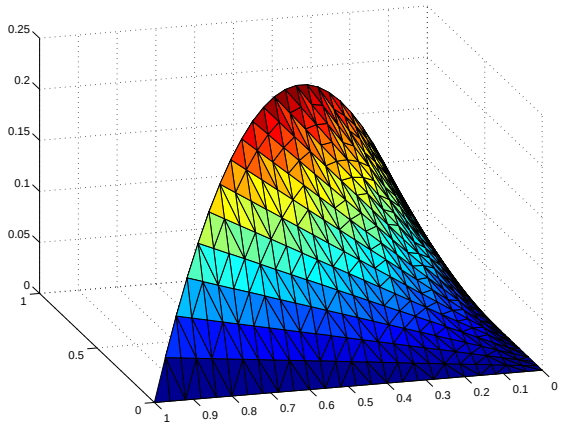

FIGURE 3. the graph of f

```
> w:=int(int(sin(x)*y,y=0..1-x),x=0..1);
```

$$w := \cos(1) - \frac{1}{2}$$

```
> evalf(w);
```

$$.04030230586813971740$$

For $\varepsilon = 10^{-3}$ we need no subdivision, as we can see from the following MATLAB session fragment:

```
>> [vi2,stat]=mpcubatd2vb(@fintegr,x,y,1e-3,1)

vi2 =

   0.04028255698461


stat =

     nev: 12
    ntri: 4
```

For $\varepsilon = 10^{-4}$ a subdivision was done (see Figure 4). The results are as follows:

```
>> [vi2,stat]=mpcubatd2vb(@fintegr,x,y,1e-4,1)

vi2 =

   0.04030110314738


stat =

     nev: 48
    ntri: 20
```

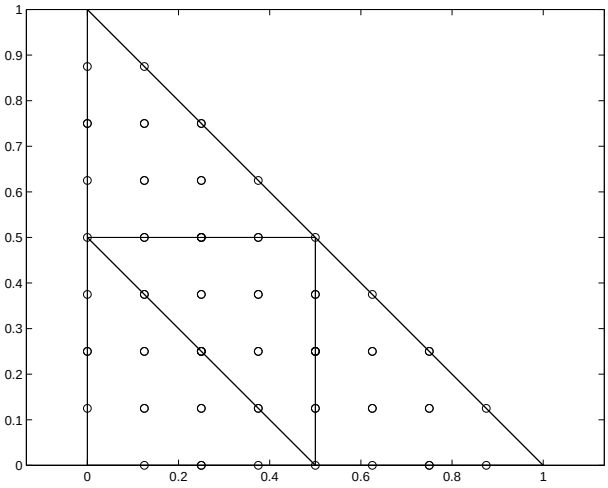

FIGURE 4. Computing the integral for $\varepsilon = 10^{-4}$. One subdivision and 48 function evaluation needed

Finally, we try for $\varepsilon = 10^{-6}$ (see Figure 5)

```
>> [vi2,stat]=mpcubatd2vb(@fintegr,x,y,1e-6,1)
```

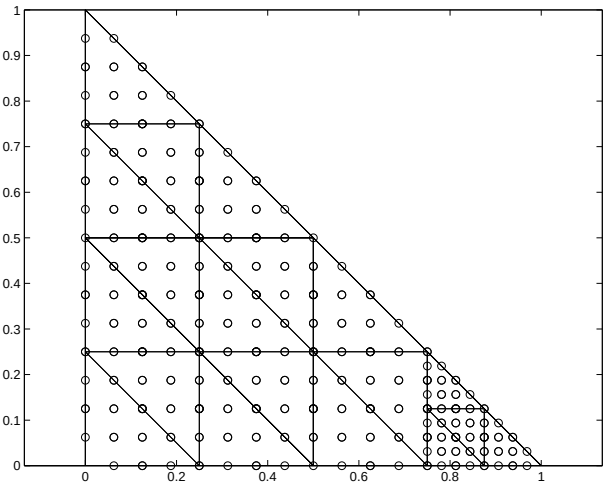

FIGURE 5. Computing the integral for $\varepsilon = 10^{-6}$. 228 function evaluation needed

```
vi2 =

   0.04030231573315

stat =

     nev: 228
    ntri: 100
```

References

[1] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf – *Computational Geometry. Algorithms and Applications*, Springer Verlag, Berlin, Heidelberg, New York, 1997.

[2] C. Überhuber – *Computer Numerik*, Band 2, Springer Verlag, Berlin, Heidelberg, New York, 1995.

[3] D. P. Laurie – Algorithm 584: CUBTRI: Automatic Cubature over a Triangle, ACM TOMS, vol. 8, No. 2, 1982, pp. 210–218.

[4] R. Cools, D. Laurie and L. Pluym – Cubpack++: A C++ Package for Automatic Two-Dimensional Cubature, ACM TOMS Vol. 23, No. 1, 1997, pp. 1–15.

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE,
BABEŞ-BOLYAI UNIVERSITY, 3400 CLUJ-NAPOCA, ROMANIA