Adaptive Cubatures on Triangle
How to implement them

Radu T. Trîmbiţaş

“Babeş-Bolyai” University

ROGER 2007, Königswinter
Outline

1. Introduction
2. A Meta Algorithm for Adaptive Integration
3. Our Approach
4. MATLAB Implementation
5. Examples and Tests
Problem

Our problem: calculate the definite integral

\[ If := \int_B f(x) \, dx \]

\[ f : B \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}, \text{ given integrand, } B \text{ given region.} \]
Our problem: calculate the definite integral

\[ \text{If} := \int_B f(x) \, dx \]

\( f : B \subseteq \mathbb{R}^n \longrightarrow \mathbb{R} \), given integrand, \( B \) given region.

The aim of constructing integration algorithm is to approximate \( \text{If} \) with a given error tolerance \( \varepsilon \) and as few function evaluations as possible.
What is an adaptive integration algorithm

- Adaptive algorithms decide dynamically how many function evaluations are needed. The information for such decisions is derived from numerical experiments based on integrand. In general, no a priori information about the decision process is available. The efficiency and reliability of such algorithms depends upon the subdivision strategy.

- The decision as to whether or not a subregion has to be further subdivided is based on either local and global knowledge. This leads to local and global subdivision strategy respectively.
- Local knowledge is based only on the considered subregion.
- Global knowledge is based on knowledge about all subregions of the integration region.
- In any case, the depth of the subdivision process is determined dynamically.
For the 1D case the basic idea is as follows: Let \([a, b]\) be a bounded interval. In order to compute

\[
I = \int_{a}^{b} f(x) \, dx
\]

we integrate \(f\) using two methods which provide us the approximations \(I_1\) and \(I_2\). If the difference of this two approximations is less than a given tolerance, we accept the better of them, say \(I_2\), as approximate value of integral. Otherwise, we divide \([a, b]\) into two (or three) congruent parts; then proceed recursively on each part.
For the 1D case the basic idea is as follows: Let \([a, b]\) be a bounded interval. In order to compute

\[ I = \int_a^b f(x) \, dx \]

we integrate \(f\) using two methods which provide us the approximations \(I_1\) and \(I_2\). If the difference of this two approximations is less than a given tolerance, we accept the better of them, say \(I_2\), as approximate value of integral. Otherwise, we divide \([a, b]\) into two (or three) congruent parts; then proceed recursively on each part.

The idea is credited to Huygens, but in this form appear in [Davis, Rabinowitz 1984].
Negative results

- [deBoor 1971] it is impossible to construct a correct program that integrates each given function.
- Moreover, for a given program, it is possible to find a function $f$, which is not correctly integrated ( [Kahan 1980]).
- Hence, the task of each implementer is to code programs which function correctly for a class of function as large as possible.
The Rice’s meta algorithm

- The Rice’s meta algorithm 2 [Rice 1975] is an abstract description of the mechanisms involved in adaptive integration.
- It can be used as a starting point for the development of adaptive integration algorithms based on a given formula $Q_N$ with an error estimator $E$.
- We reproduced it here in the form given in [Überhuber 1995]
The Rice’s meta algorithm

Meta algorithm for adaptive integration

**Input:** \( f, B, \varepsilon, Q_N, E \).

**Output:** The approximate integral value \( q \) and the error estimation \( e \).

\[
q := Q(f; B); \quad e := E(f, B);
\]

insert \((B, q, e)\) into the data structure;

while \( e > \varepsilon \) do

choose an element of the data structure (with index \( s \));

Subdivide \( B_s \) into subregions \( B_\ell, \ell = 1, 2, \ldots, L \);

Calculate approximations for integrals over \( B_1, \ldots, B_L \)

\[
q_\ell := Q_N(f; B_\ell), \quad \ell = 1, 2, \ldots, L;
\]

Calculate corresponding error estimates;

\[
e_\ell := E(f; B_\ell), \quad \ell = 1, 2, \ldots, L;
\]

remove old data \((B_s, q_s, E_s)\) from the data structure;

Insert \((B_1, q_1, e_1), \ldots, (B_L, q_L, e_L)\) into the data structure;

\[
q := \sum_i q_i; \quad e := \sum_i e_i;
\]

end while
The case of triangle


This allow:

- A larger degree of generality
- To restart the algorithm for refinement, performing the continuation of previous work.
Basic Elements

- A collection of triangles organized in a heap; $M$ is the current number of triangles
- A quadrature rule $Q$ to produce a local estimate to the integral over each triangle of the collection
- A procedure for error estimation $E$
- A strategy for picking the next triangle to be processed — in our case the triangle on the top of the heap
- A subdivision strategy
The algorithm

Initialize the triangle collection; $M := m$;
Compute $\hat{Q}_i$ and $\hat{E}_i$, $i = 1, 2, \ldots, m$
$\hat{Q} = \sum_{i=1}^m \hat{Q}_i$; $\hat{E} = \sum_{i=1}^n \hat{E}_i$;
while $\hat{E} > \varepsilon$ do
    {Control}
    Pick the triangle $T_k$ on top of heap;
    {Subdivision}
    Divide $T_k$ in $p$ parts;
    {Process triangles}
    Compute $\hat{Q}_k^{(i)}$, $\hat{E}_k^{(i)}$, $i = 1, \ldots, p$;
    {Update}
    $\hat{Q} := \hat{Q} + \sum_{i=1}^p \hat{Q}_k^{(i)} - \hat{Q}_k$; $\hat{E} := \hat{E} + \sum_{i=1}^p \hat{E}_k^{(i)} - \hat{E}_k$;
    Replace triangle $T_k$ by $p$ new triangles;
    $M := M + p - 1$;
end while
Data structures

- A collection of triangles $\text{Tri}$, organized as an array of triangles. Information for each triangle:
  - $V_1, V_2, V_3$ - pointers to vertices (see Vertex below)
  - $VI$ - approximate of the integral
  - $EE$ - error estimation

- A collection of vertices, $\text{Vertex}$, organized as a matrix with two columns (coordinates)

- A heap of pointers to triangles, $\text{Heap}$. Ordered by $EE$. Triangle with maximum $EE$ on top.
Cubature rules

- The user may choose the cubature rule. A procedure that initializes the nodes and the coefficients is specified at invocation.
- The rule must be given in fully symmetric form, as in [Stroud 71] or in Ronald Cools’ Encyclopedia of cubature formula [Encyclopedia].
- A procedure evaluates the cubature formula given in fully symmetric form.
- Supported: a 37 point PI rule of degree 13 [Berntsen, Espelid 1990] and a seven point PI rule of degree 5, due to Radon.
Our Approach

Error estimation

- Two methods:
  - Embedded cubature formulas.
  - Null rules.
Our Approach

Error estimation

Two methods:

- Embedded cubature formulas.
Error estimation

- Two methods:
- Embedded cubature formulas.
- Null rules.
Embedded cubature formulas

- We shall use two cubature formulas

\[ Q_j[f] = \sum_{i=1}^{N_j} f(x_{ij}, y_{ij}), \quad j \in \{1, 2\}, \]

with degree of exactness \( d_j \), \( d_1 < d_2 \), where \( N_j \) is the number of nodes for \( Q_j \).

- In order to reduce the number of function evaluation (and so the amount of work) one tries to choose \( Q_1 \) and \( Q_2 \) such that

\[ \{(x_{i1}, y_{i1} : i = 1, \ldots, N_1) \subset \{x_{i2}, y_{i2} : i = 1, \ldots, N_2\}. \]

A pair \((Q_1, Q_2)\) having this property is called an embedded pair.

- The difference \(|Q_1[f] - Q_2[f]|\) is used as an error estimation for \( Q_1 \).
Our Approach

Null rules

Definition

[Lyness 1965] A rule

\[ N[f] = \sum_{i=0}^{n} u_i f(x_i) \quad (1) \]

is a null rule iff it has at least one nonzero weight, and in addition \( \sum_{i=0}^{n} u_i = 0 \). A null rule has the degree \( d \) if it integrates to zero all basic monomials of degree \( \leq d \) and fail to do so for a monomial of degree \( d + 1 \).

- A null rule of the form (1) has the degree at most \( n - 1 \).
- Null rules may be used as estimations of error.
- An estimation based on a single null rule is sometimes unreliable; in practice one uses combination of null rules of various degrees.[Berntsen, Espelid 1991]
Our Approach

Error estimation using null rules

\[
\text{Compute}
\]

\[
e_j := N_j[f], j = 1, \ldots, 2k;
\]

\[
E_j := (e_{2j-1}^2 + e_{2j}^2)^{1/2}, j = 1, \ldots, k;
\]

\[
r_j := E_j / E_{j+1}, j = 1, \ldots, k - 1;
\]

\[
\text{if } r > 1 \text{ then}
\]

\[
\hat{E} = 10 \max_j E_j \{\text{Nonasymptotic}\}
\]

\[
\text{else if } 1/2 \leq r \text{ then}
\]

\[
\hat{E} := 10r^1 E_1 \{\text{Weakly-asymptotic}\}
\]

\[
\text{else}
\]

\[
E = 10 \cdot 4r^3 E_1 \{\text{Strongly-asymptotic}\}
\]

\[
\text{end if}
\]

- Since cubature rules and null rules are based on the same set of nodes, they are evaluated simultaneously.

- Embedded cubatures are considered combination of a cubature rule \(Q_1\) and a null rule \(Q_1 - Q_2\).
Subdivision

- The simplest subdivision is in four congruent triangles, determined by vertices and midpoints of edges
- A more flexible method
- Use subdivision directions parallel to the sides of the triangle
- 4th differences parallel to the sides are computed
- Let $e$ be a unit vector along one side of triangle $T_k$, $h$ the length of the side, $C$ the barycenter. Define the measure of variation of $f$ in direction $e$:

$$D(e) = h^q \left| f \left( C - \frac{4}{15} he \right) - 4f \left( C - \frac{2}{15} he \right) + 6f(C) - 4f \left( C + \frac{2}{15} he \right) + f \left( C - \frac{4}{15} he \right) \right|$$

(2)

- Three heuristic constants $q, \rho_1, \rho_2$ are involved
Our Approach

Subdivision

- Define $D_a = D(a / \|a\|)$, $D_b = \ldots$ using (2)
- The triangle is divided into four or three triangles according to magnitude of $D_a$, $D_b$, $D_c$
- Requires 13 new function evaluations
Subdivision types

$S_1$

$S_2$

$D_a/\rho_1 \quad D_c \quad D_b \quad D_a$

$D_a/\rho_2 \quad D_c \quad D_b \quad D_a$

$S_3$

$S_4$

$D_c \quad D_a/\rho_2 \quad D_b \quad D_a$

$D_c \quad D_b \quad D_a/\rho_2 \quad D_a$
Algorithm — Choice of subdivision

1. Compute estimates $D_a$, $D_b$, $D_c$ for sides $a$, $b$, $c$
2. Relabel the sides so that $D_a \geq D_b \geq D_c$
3. If $D_c \geq D_a/\rho_1$ then choose $S_1$;
   else if $D_b \geq D_a/\rho_2$ and $D_c \geq D_a/\rho_2$ then choose $S_2$;
   else if $D_b \geq D_a/\rho_2$ and $D_c < D_a/\rho_2$ then choose $S_3$;
   else choose $S_4$
   end if
MATLAB implementation - main function

- We code a flexible set of functions which allow to select the cubature formula and null rules. There exists also a restart facility that allows the refinement of previous result. The syntax of main function, CubatureTriang is

\[
[\text{result}, \text{ee}, \text{stat}, \text{Tri}, \text{Vertex}, \text{VI}, \text{EE}] = \text{CubatureTriang}(F, \text{Tri}, ..., \text{Vertex}, \text{VI}, \text{EE}, \text{opt}, \text{varargin})
\]

- Parameters:
  
  \(F\) - function to be integrated
MATLAB implementation - main function

- We code a flexible set of functions which allow to select the cubature formula and null rules. There exists also a restart facility that allows the refinement of previous result. The syntax of main function, CubatureTriang is

\[
[\text{result}, \text{ee}, \text{stat}, \text{Tri}, \text{Vertex}, \text{VI}, \text{EE}] = \text{CubatureTriang}(F, \text{Tri}, \ldots, \text{Vertex}, \text{VI}, \text{EE}, \text{opt}, \text{varargin})
\]

- Parameters:
  Tri - collection of triangles
MATLAB implementation - main function

- We code a flexible set of functions which allow to select the cubature formula and null rules. There exists also a restart facility that allows the refinement of previous result. The syntax of main function, CubatureTriang is

\[
\text{[result, ee, stat, Tri, Vertex, VI, EE]} = \text{CubatureTriang(F, Tri, ... Vertex, VI, EE, opt, varargin)}
\]

- Parameters:
  Vertex - collection of vertices
We code a flexible set of functions which allow to select the cubature formula and null rules. There exists also a restart facility that allows the refinement of previous result. The syntax of main function, CubatureTriang is:

\[
\text{[result, ee, stat, Tri, Vertex, VI, EE] = CubatureTriang(F, Tri, ... Vertex, VI, EE, opt, varargin)}
\]

Parameters:

- **VI** - value of integral for a triangle
We code a flexible set of functions which allow to select the cubature formula and null rules. There exists also a restart facility that allows the refinement of previous result. The syntax of main function, CubatureTriang is

\[
\text{CubatureTriang}(F, \text{Tri}, \ldots, \text{Vertex}, \text{VI}, \text{EE}, \text{opt}, \text{varargin})
\]

Parameters:
EE - error estimation for a triangle
MATLAB implementation - main function

- We code a flexible set of functions which allow to select the cubature formula and null rules. There exists also a restart facility that allows the refinement of previous result. The syntax of main function, CubatureTriang is

\[
\text{[result, ee, stat, Tri, Vertex, VI, EE] = CubatureTriang(F, Tri, ... Vertex, VI, EE, opt, varargin)}
\]

- Parameters:
  opt - options: errabs, errel, restart, initf, trace, nfev -
MATLAB implementation - main function

- We code a flexible set of functions which allow to select the cubature formula and null rules. There exists also a restart facility that allows the refinement of previous result. The syntax of main function, CubatureTriang is

  \[
  [\text{result}, \text{ee}, \text{stat}, \text{Tri}, \text{Vertex}, \text{VI}, \text{EE}] = \text{CubatureTriang}(F, \text{Tri}, ..., \\
  \text{Vertex, VI, EE, opt, varargin})
  \]

- Parameters:

  \text{initf} - initialization function; return cubature parameters: weights, nodes, null rules type;

  call: \([W,G,m,p]=\text{initf}\)
MATLAB implementation - main function

- We code a flexible set of functions which allow to select the cubature formula and null rules. There exists also a restart facility that allows the refinement of previous result. The syntax of main function, `CubatureTriang` is

\[
\text{[result, ee, stat, Tri, Vertex, VI, EE]} = \text{CubatureTriang}(F, \text{Tri}, ... \text{Vertex, VI, EE, opt, varargin})
\]

- Parameters:
  - `result` - approximate of integral
MATLAB implementation - main function

- We code a flexible set of functions which allow to select the cubature formula and null rules. There exists also a restart facility that allows the refinement of previous result. The syntax of main function, `CubatureTriang` is

\[
\text{[result, ee, stat, Tri, Vertex, VI, EE] = CubatureTriang(F, Tri, ... Vertex, VI, EE, opt, varargin)}
\]

- Parameters:
  - `ee` - error estimation
MATLAB implementation - main function

We code a flexible set of functions which allow to select the cubature formula and null rules. There exists also a restart facility that allows the refinement of previous result. The syntax of main function, CubatureTriang is

\[
\text{[result, ee, stat, Tri, Vertex, VI, EE]} = \text{CubatureTriang(F, Tri, ... Vertex, VI, EE, opt, varargin)}
\]

- Parameters:
  - stat - statistics: number of function evaluations, number of triangles, success/failure.
MATLAB implementation - cubature and null rules

- Cubature rule and null rules are given in fully symmetric form.
- The function `fselcub` approximate the integral and the error on the current triangle.
- The selection of cubature and null rules is performed via the `initf` parameter of `CubatTri`. Implemented:
  - Berntsen & Espelid 13 degree formula with eight null rules, function `BerntsenEspelid`
  - Embedded 5-7 degree cubature formula, function `ecf57` [Cools, Haegemans 1988].
  - Embedded 5-7 degree cubature formula, function `ecf58` [Laurie 1982].
- The user can code his own function if he/she obeys the call syntax.
MATLAB implementation - Data structures management

- Function `NewVertex` inserts a new vertex into the `Vertex` matrix.
- Function `NewTriangle` inserts a triple of pointers (indices) to the vertices of triangle into the array `Tri`.
- Function `InsertIntoHeap` takes a pointer to the current triangle and its error estimation and insert the pointer into heap at an appropriate place, updating the heap.
- Function `ExtractMaxFromHeap` extract the top triangle from heap and update the data structures.
Examples and tests - Test families

Test family
1. \( f_1(x, y) = (|x - \beta_1| + y)^{d_1} \)
2. \( f_2(x, y) = \begin{cases} 1 & \sqrt{(x - \beta_1)^2 + (y - \beta_2)^2} < d_2 \\ 0 & \text{otherwise} \end{cases} \)
3. \( f_3(x, y) = \exp(-\alpha_1|x - \beta_1| - \alpha_2|y - \beta_2|) \)
4. \( f_4(x, y) = \exp(-\alpha_1^2(x - \beta_1)^2 - \alpha_2^2(y - \beta_2)^2) \)
5. \( f_5(x, y) = (\alpha_1^{-2} + (x - \beta_1)^2)^{-1}(\alpha_2^{-2} + y^2)^{-1} \)
6. \( f_6(x, y) = (\alpha_1^{-2} + (x - \beta_1)^2)^{-1}(\alpha_2^{-2} + (y - \beta_2)^2)^{-1} \)
7. \( f_7(x, y) = \cos(2\pi\beta_1 + \alpha_1 x + \alpha_2 y) \)

Attributes
- X-axis singularity
- Discontinuous
- \( C_0 \) function
- Gaussian
- X-axis peak
- Internal peak
- Oscillatory

- \( d_j \) - difficulty parameters, \( j = 1, \ldots, 7 \)
- \( \alpha_1, \alpha_2, \beta_1, \beta_2 \) - random parameters uniformly distributed on \([0, 1]\).
- \( \alpha_1, \alpha_2 \) scaled such that \( \alpha_1 + \alpha_2 = d_j \)
Examples and Tests

Family 1

Test - family 1

(a) Graph of $f_1$

(b) Evaluation points

Figure: Test for family 1
Test - family 2

(a) Graph of $f_2$  
(b) Evaluation points

**Figure**: Test for family 2
Test - family 3

(a) Graph of $f_3$

(b) Evaluation points

**Figure:** Test for family 3
Test - family 4

(a) Graph of $f_4$

(b) Evaluation points

Figure: Test for family 4
Test - family 5

(a) Graph of $f_5$

(b) Evaluation points

Figure: Test for family 5
Test - family 6

(a) Graph of $f_6$

(b) Evaluation points

Figure: Test for family 6
Examples and Tests

Family 7

Test - family 7

(a) Graph of $f_7$

(b) Evaluation points

Figure: Test for family 7
Test with restart

Family 7, first call for \texttt{errabs=1e-6}, then restart with \texttt{errabs=1e-8}.

\begin{figure}[h]
\centering
\begin{subfigure}{0.45\textwidth}
\includegraphics[width=\textwidth]{figure1a}
\caption{First step}
\end{subfigure}
\begin{subfigure}{0.45\textwidth}
\includegraphics[width=\textwidth]{figure1b}
\caption{Second step, restart}
\end{subfigure}
\caption{Test for family 7 with restart}
\end{figure}
Number of function evaluation - family 4

\( \varepsilon = 10^{-2}, 10^{-4}, \ldots, 10^{-10}, \) 500 samples for each error
Number of failures - family 4

$\varepsilon = 10^{-2}, 10^{-4}, \ldots, 10^{-10}$, 500 samples for each error
Number of correct digits - family 4

\[ \varepsilon = 10^{-2}, 10^{-4}, \ldots, 10^{-10}, \text{500 samples for each error} \]
Number of function evaluation - family 7

\[ \varepsilon = 10^{-2}, 10^{-4}, \ldots, 10^{-10}, \text{500 samples for each error} \]
Number of failures - family 7

$\epsilon = 10^{-2}, 10^{-4}, \ldots, 10^{-10}$, 500 samples for each error

![Graph showing the average number of failures for test family 7]
Number of correct digits - family 7

\[ \varepsilon = 10^{-2}, 10^{-4}, \ldots, 10^{-10}, \text{500 samples for each error} \]
References I

J. Berntsen, T. O. Espelid:
Algorithm 706: DCUTRI: An algorithm for Adaptive Cubature Over a Collection of Triangles,

J. Berntsen, T. O. Espelid:
Error Estimation in Automatic Quadrature Routines,

J. Berntsen, T. O. Espelid:
*Degree 13 Symmetric Quadrature Rules for the Triangle*,
References II

T. O. Espelid, A. Genz:
*On the Subdivision Strategy in Adaptive Cubature Algorithms for Triangular Regions*,

Carl de Boor:
On writing automatic integration algorithm,

R. Cools, A. Haegemans:
An embedded pair of cubature formulae of degree 5 and 7 for the triangle,
References III

R. Cools, D. P. Laurie, L. Pluym: 
Cubpack++: A C++ package for Authomatic Two-dimensional Cubature, 

Philip J. Davis, Philip Rabinowitz: 
*Methods of Numerical Integration*, 2nd edition, 

William M. Kahan: 
Handheld calculator evaluates integrals, 

D. P. Laurie: 
Algorithm 584: CUBTRI: Automatic Cubature over a triangle, 
References IV

J. Rice:
A metaalgorithm for adaptive quadrature,
*JACM*, 22(1975), pp. 61–82.

I. Somogyi, R. Trîmbiţas:
The study of an adaptive algorithm for some cubature formulas on triangle,
*Studia UBB*.

I. Somogyi, R. Trîmbiţas:
On an adaptive algorithm based on embedded cubature formulas on triangle,

A. H. Stroud:
*Approximate Calculation of Multiple Integrals*,
References V

Cr. Überhuber,
*Computer Numerik*, Band II,

Ronald Cools:
Encyclopedia of Cubature Formulae
www.cs.kuleuven.be/cwis/research/nines/research/ecf

J. Nievergelt, H. Hinterberger, K. C. Sevcik:
The grid file: An adaptable, symmetric, multikey file structure,

H. Samet:
*The Design and Analysis of Spatial Data Structure*,
Addison -Wesley, Reading, Ma., 1991.
References VI
