

NUMERICAL COMPUTATION OF INTEGRALS IN HIGHER
DIMENSIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ÇANKAYA UNIVERSITY

BY

HAKAN BAYDAR

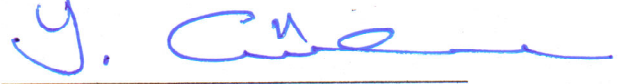
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MATHEMATICS AND COMPUTER SCIENCE

AUGUST 2006

Title of the Thesis : **Numerical Computation of Integrals In Higher Dimensions**

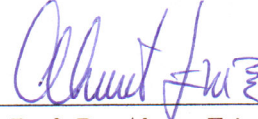
Submitted by **Hakan Baydar**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University



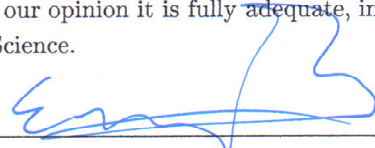
Prof. Dr. Yurdahan Güler
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.



Prof. Dr. Ahmet Eriş
Head of the Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.



Assist. Prof. Dr. Emre Sermutlu
Supervisor

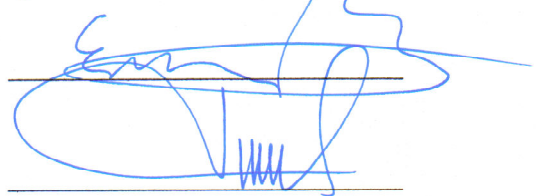
Examination Date : 25.07.2006

Examining Committee Members

Assoc. Prof. Dr. Tanıl Ergenç (METU)



Assist. Prof. Dr. Emre Sermutlu (Çankaya Univ.)



Prof. Dr. Kenan Taş (Çankaya Univ.)

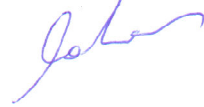


STATEMENT OF NON PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Hakan Baydar

Signature :



Date :

25.07.2006

ABSTRACT

NUMERICAL COMPUTATION OF INTEGRALS IN HIGHER DIMENSIONS

Baydar, Hakan

M.S.c., Department of Mathematics and Computer Science

Supervisor: Assist. Prof. Dr. Emre Sermutlu

August 2006, 35 pages

Quadrature refers to any method for numerically approximating the value of definite integral $\int_a^b f(x)dx$. The goal is to attain a given level of precision with the fewest function evaluations.

The factors that control the difficulty of a numerical integration problem are the dimension of the integral and the smoothness of the integrand f .

Any quadrature method relies on evaluating the integrand f on a finite set of points (called the abscissas or quadrature points), and after processing these evaluations to produce an approximation to the integral. Usually this involves taking a weighted average.

The goal is to determine which points to evaluate and what weight to use so as to maximize performance over a broad class of integrands.

This study reviews Monte Carlo and Newton-Cotes methods of numerical approximation of integrals on both rectangular and nonrectangular regions and contains new routines that can evaluate integrals up to 7 dimensions over arbitrary regions in MATLAB.

The work aims to compare the methods and give some approximation results using our self-written code.

Keywords: Numerical integration, Quadrature, Monte Carlo, Newton-Cotes, MATLAB

ÖZ

YÜKSEK BOYUTLU İNTEGRALLERİN NÜMERİK HESAPLANMASI

Baydar, Hakan

Yüksek Lisans, Matematik-Bilgisayar Bölümü

Tez Yöneticisi: Yrd. Doç. Dr. Emre Sermutlu

Ağustos 2006, 35 sayfa

$\int_a^b f(x)dx$ şeklindeki belirli integralin nümerik olarak yaklaştırılması için herhangi bir method kullanılmasına tümlev alma denir. Amaç en az fonksiyon değerlendirimi ile verilen duyarlılık seviyesinde sonuç elde etmektir.

Nümerik bir integral probleminin zorluğunu kontrol eden faktörler integralin boyutu ve fonksiyonun pürüzsüzlüğüdür.

Her tümlev alma methodu, integrali alınan f fonksiyonunu sınırlı sayıda noktada (absis veya tümlev alınan nokta) hesaplamaya dayanır, daha sonra bu değerler bir yaklaştırım elde etmede kullanılır. Genelde bu ağırlıklı ortalama almayı gerektirir.

Hedef hangi noktalarda fonksiyonun hesaplanacağı ve hangi ağırlıkların kullanılacağıdır, öyle ki integrali alınan fonksiyonlarda en geniş sınıfta maksimum performans elde edilsin.

Bu arařtırmada integrallerin nümerik yaklařtırılmasında kullanılan Monte Carlo ve Newton-Cotes metodları gözden geçirilmiřtir ve MATLAB ile yazılmıř 7. dereceye kadar integralleri herhangi bir bölgede hesaplayabilen yeni programlar içermektedir.

Bu çalıřmada amaç metodları karşılařtırmak ve kendi yazdıđımız kod ile bazı yaklařtırım sonuçlarını vermektedir.

Anahtar Kelimeler: Nümerik integral, Tümler alma, Monte Carlo, Newton-Cotes, MATLAB

ACKNOWLEDGMENTS

Special thanks to my mother, İclal Baydar, who has been self-sacrificing throughout my life and I have always been grateful to my father İsmail Baydar, and my sister, Bahar Baydar for their valuable support.

I would like to thank my advisor Assist. Prof. Dr. Emre Sermutlu for his guidance, advice, encouragements and insight throughout the study of this thesis work.

I will always remember the support of my friends at this thesis; Tansel Avkar, Ahmet Kabarcık and Azmi Kalaycođlu.

At last but not least, I thank to my dear, Gamze Cömert. She has always been a source of motivation for me and she gave me extra power throughout the work.

TABLE OF CONTENTS

STATEMENT OF NON PLAGIARISM	iii
ABSTRACT	iv
ÖZ	vi
ACKNOWLEDGMENTS	viii
TABLE OF CONTENTS	ix
CHAPTERS:	
1 Numerical Integration in 1- Dimension	1
1.1 Newton-Cotes Rules	1
1.2 Composite Rules	3
1.3 Adaptive Quadrature	5
1.4 Gaussian Rules	6
1.5 Monte Carlo Method	11
2 Numerical Integration in Higher Dimensions-Rectangular Regions . . .	13
2.1 Newton-Cotes Method	14
2.2 Gaussian Quadrature	16
2.3 Monte-Carlo Method	16
3 Numerical Integration in Higher Dimensions-Nonrectangular Regions .	18
4 MATLAB Implementation	23
5 Tests and Comparison	28
6 CONCLUSION	35
APPENDIX	A1
REFERENCES	R1

LIST OF TABLES

1.1	Results of NC method	3
1.2	Table of Abscissa and Weight for Gaussian Integration on $[-1, 1]$. . .	10

LIST OF FIGURES

1.1	Composite trapezoidal rule sampled on 1-dimension	5
2.1	Simpson rule sampled on 2-dimensions where $m=4$, $n=2$	15
3.1	A nonrectangular region: $\int_a^b \int_{g(x)}^{h(x)} f(x, y) dy dx$	19
3.2	Newton-Cotes method on a nonrectangular region in 2-dimensions . . .	20
3.3	Monte Carlo method on a nonrectangular region in 2-dimensions . . .	21
5.1	Number of points vs. relative error on log-log scale in 2 and 3 dimensions.	29
5.2	Number of points vs. relative error on log-log scale in 4 and 5 dimensions.	30
5.3	Number of points vs. relative error on log-log scale in 6 and 7 dimensions.	31
5.4	Number of points vs. relative error on log-log scale in 2 and 3 dimensions.	32
5.5	Number of points vs. relative error on log-log scale in 4 and 5 dimensions.	33
5.6	Number of points vs. relative error on log-log scale in 6 and 7 dimensions.	34

CHAPTER 1

Numerical Integration in 1- Dimension

1.1 Newton-Cotes Rules

The basic integration rules use one polynomial to model $f(x)$ in the interval $[a, b]$ for the estimate of $\int_a^b f(x)dx$. The interval $[a, b]$ is divided into equal parts, the integrand $f(x)$ is interpolated at the division points, and the resulting polynomial is integrated to estimate the integral. We set $x_i = a + ih$, $i = 0, 1, 2, \dots, n$ where $h = [b - a]/n$. The rules are of two types : **the closed formulas** use all the points x_i , and **the open formulas** use all but the end points. Both kinds of formulas are called **Newton-Cotes formulas** : They are of the form

$$\int_a^b f(x)dx \approx dh \sum_{i=0}^n w_i f(x_i) \quad \text{closed} \quad (1.1)$$

$$\int_a^b f(x)dx \approx dh \sum_{i=1}^{n-1} w_i f(x_i) \quad \text{open} \quad (1.2)$$

Practically, the constant d may be multiplied into the w . The table below gives the values for the formulas for models of degree 0 through 6.

n = Polynomial degree in the model for $f(x)$
 d = coefficient of h in equation
 Note that the formulas are symmetric about the middle

CLOSED FORMULAS, $h = (b - a)/n$

n	d	w_0	w_1	w_2	w_3	w_4	w_5	w_6
0	1	1						
1	1/2	1	1					
2	1/3	1	4	1				
3	3/8	1	3	3	1			
4	2/45	7	32	12	32	7		
5	5/288	19	75	50	50	75	19	
6	1/140	41	216	27	272	27	216	41

OPEN FORMULAS, $h = (b - a)/(n + 2)$

n	d	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	1	1						
1	3/2	1	1					
2	4/3	2	-1	2				
3	5/24	11	1	1	11			
4	6/20	11	-14	26	-14	11		
5	7/1440	611	-453	562	562	-453	611	
6	8/945	460	-945	2196	-2459	2196	-954	460

Some of the common closed Newton-Cotes formulas with their error term are as follows

3-point Newton-Cotes Rule (Simpson's Rule) :

$$\int_{x_0}^{x_2} f(x)dx \approx \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] \quad (1.3)$$

5-point Newton-Cotes Rule (Boole's Rule) :

$$\int_{x_0}^{x_4} f(x)dx \approx \frac{2h}{45}[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)] \quad (1.4)$$

7-point Newton-Cotes Rule :

$$\int_{x_0}^{x_6} f(x)dx \approx \frac{h}{140}[41f(x_0) + 216f(x_1) + 27f(x_2) + 272f(x_3) + 27f(x_4) + 216f(x_5) + 41f(x_6)] \quad (1.5)$$

For the Simpson's Rule, each of the rule has the following properties: All of the weights are positive, and all but one of the weights are equal. [1] Higher order rules are not useful because they contain negative coefficients which results computation errors.

Example 1.1.1: Let's evaluate $\int_0^1 x^2 e^{-x} dx$.

The table below gives the results using different methods.

Table 1.1: Results of NC method

	Result	Error
Exact	0,16060279414279	0
NC_2	0,16060429956291	$1,50542 \times 10^{-6}$
NC_4	0,16060280536960	$1,2268 \times 10^{-8}$
NC_6	0,16060279421974	$7,69521 \times 10^{-11}$

1.2 Composite Rules

The integral of an interpolating polynomial is rarely good enough to make an accurate estimation. The standard solution to this accuracy problem is to use **composite rules**.

The idea of the composite rules is to use the fact that if $a < c < b$;

$$\int_a^b f(x)dx = \int_a^c f(x)dx + \int_c^b f(x)dx \quad (1.6)$$

by dividing the interval of integration into a number N , say, of equal subintervals and applying a low order quadrature rule to each subinterval.[2]

Thus the composite **composite trapezoidal rule** using N intervals and $h = (b - a)/N$ becomes

$$\int_a^b f(x)dx \approx \frac{h}{2} \left[f(a) + f(b) + 2 \sum_{i=1}^{N-1} f(a + ih) \right] \quad (1.7)$$

The composite trapezoidal rule with N intervals uses $N + 1$ nodes.

The **composite Simpson's rule** with N intervals uses $2N + 1$ points in all since the basic rule uses midpoint of the interval as well as end points. The stepsize is therefore $h = (b - a)/2N$ and the resulting composite formula is

$$\int_a^b f(x)dx \approx \frac{h}{3} \left[f(a) + f(b) + 2 \sum_{i=1}^{N-1} f(a + ih) + 4 \sum_{i=0}^{N-1} f(a + (2i + 1)h) \right] \quad (1.8)$$

One way of finding an accurate result is to begin with a small value of N and then repeatedly to double the number of intervals until two successive estimates agree to within a desired tolerance. This allows the efficient use of the previous results by only computing the new function values.

In the figure below composite trapezoidal rule is applied:

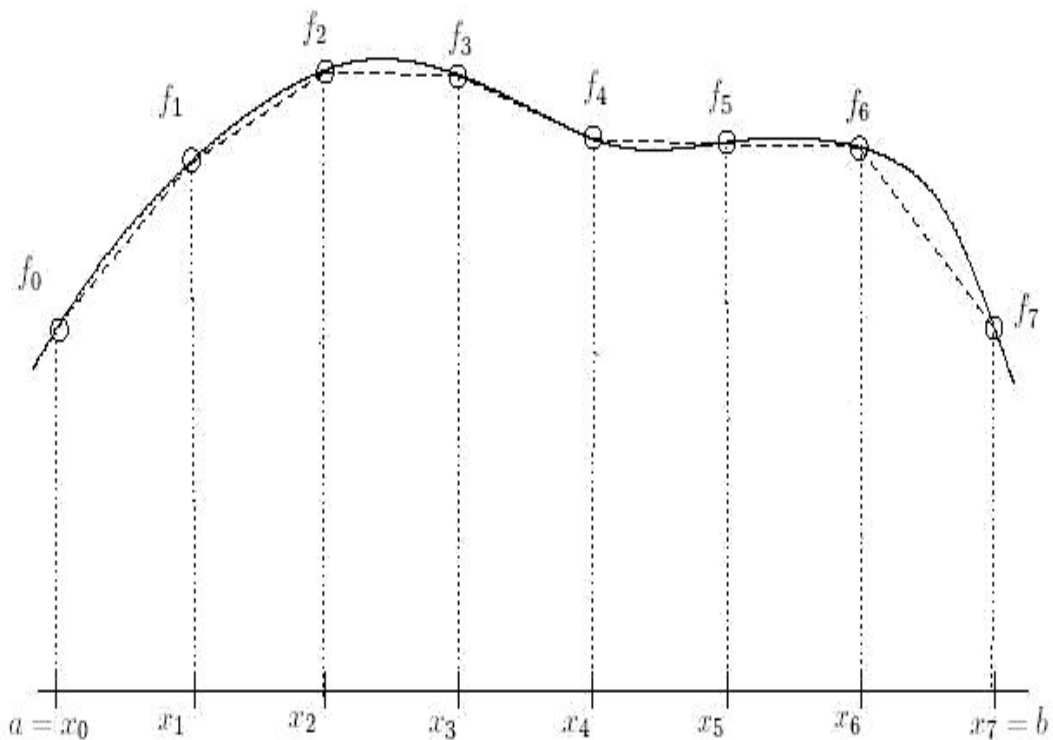


Figure 1.1: Composite trapezoidal rule sampled on 1-dimension

1.3 Adaptive Quadrature

The composite quadrature rules necessitate the use of equally spaced points. Typically, a small step size h was used uniformly across the entire interval of integration to ensure the overall accuracy. This does not take into account that some portions of the curve may have large functional variations that require more attention than other portions of the curve. It is useful to introduce a method that adjusts the step size to be smaller over portions of the curve where a larger functional variation occurs.

Adaptive quadrature involves careful selection of the points where $f(x)$ is sampled. We want to evaluate the function at as few points as possible while approximating the integral to within some specified accuracy. A fundamental additive property of a definite integral is the basis for adaptive quadrature. If c is any point between a and

b , then

$$\int_a^b f(x)dx = \int_a^c f(x)dx + \int_c^b f(x)dx \quad (1.9)$$

The idea is that if we can approximate each of the two integrals on the right to within a specified tolerance, then the sum gives us the desired result. If not, we can recursively apply the additive property to each of the intervals $[a; c]$ and $[c; b]$. The resulting algorithm will adapt to the integrand automatically, partitioning the interval into subintervals with fine spacing where the integrand is varying rapidly and coarse spacing where the integrand is varying slowly.

1.4 Gaussian Rules

All the formulas of Newton-Cotes use values of the functions at equally spaced points. This is convenient when the formulas are combined to form the composite rules, but this restriction of using equally spaced points can significantly decrease the accuracy of the approximation.

Gaussian quadrature uses evaluation points, or nodes that are not equally spaced in the interval. The nodes x_1, x_2, \dots, x_n in the interval $[a, b]$ and coefficients c_1, c_2, \dots, c_n are chosen to minimize the expected error obtained in the approximation

$$\int_a^b f(x)dx \approx \sum_{i=1}^n c_i \cdot f(x_i) \quad (1.10)$$

To minimize the expected error, we assume that the best choice of these values is that which produces the exact results for the largest class of polynomials. [3]

The coefficients c_1, c_2, \dots, c_n in the approximation formula are arbitrary and the nodes x_1, x_2, \dots, x_n are restricted only by the fact that they lie in $[a, b]$, in the interval of integration.

This gives $2n$ parameters to choose. If the coefficients of a polynomial are considered parameters, the class of polynomials of degree at most $2n - 1$ also contains $2n$ parameters. This is the largest class of polynomials for which it is reasonable to expect the formula to be exact. For the proper choice of the values and constants, exactness on this set can be obtained.[3]

To illustrate the procedure for choosing the appropriate constants, we will show how to select the coefficients and nodes when $n = 2$ and the interval of integration is $[-1, 1]$.

Example 1.4.1:

Suppose we want to determine c_1, c_2, x_1 and x_2 so that the integration formula

$$\int_{-1}^1 f(x)dx \approx c_1 f(x_1) + c_2 f(x_2) \tag{1.11}$$

gives the exact result whenever $f(x)$ is a polynomial of degree $2(2) - 1 = 3$ or less, that is when

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 \tag{1.12}$$

for some collection of constant a_0, a_1, a_2 , and a_3 . Because

$$\int (a_0 + a_1x + a_2x^2 + a_3x^3)dx = a_0 \int 1dx + a_1 \int xdx + a_2 \int x^2dx + a_3 \int x^3dx \tag{1.13}$$

this is equivalent to showing that the formula gives exact results when $f(x)$ is $1, x, x^2$ and x^3 . This is the condition we will satisfy. So, we need to find c_1, c_2, x_1 and x_2

with

$$c_1 \cdot 1 + c_2 \cdot 1 = \int_{-1}^1 1 dx = 2, \quad c_1 \cdot x_1 + c_2 \cdot x_2 = \int_{-1}^1 x dx = 0, \quad (1.14)$$

$$c_1 \cdot x_1^2 + c_2 \cdot x_2^2 = \int_{-1}^1 x^2 dx = \frac{2}{3}, \quad c_1 \cdot x_1^3 + c_2 \cdot x_2^3 = \int_{-1}^1 x^3 dx = 0. \quad (1.15)$$

Solving this system of equations gives the unique solution

$$c_1 = 1, \quad c_2 = 1, \quad x_1 = \frac{-\sqrt{3}}{3}, \quad \text{and} \quad x_2 = \frac{\sqrt{3}}{3} \quad (1.16)$$

This result produces the following integral approximation formula:

$$\int_{-1}^1 f(x) dx \approx f\left(\frac{-\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right) \quad (1.17)$$

which gives the exact result for every polynomial of degree 3 or less[3].

The technique in Example 1.4.1 can be used to determine the nodes and coefficient for formulas that gives exact results for higher-degree polynomials, but an alternative method obtains them more easily[3]. The set that is relevant to our problem is the set of *Legendre polynomials* a collection $P_0(x), P_1(x), \dots, P_n(x), \dots$ that has the following properties:

- For each n , $P_n(x)$ is a polynomial of degree n .
- $\int_{-1}^1 P_i(x)P_j(x)dx = 0$ whenever $i \neq j$

The first few Legendre polynomials are

$$P_0(x) = 1, \quad P_1(x) = x, \quad P_2(x) = x^2 - \frac{1}{3}, \quad (1.18)$$

$$P_3(x) = x^3 - \frac{3}{5}x, \quad \text{and} \quad P_4(x) = x^4 - \frac{6}{7}x^2 + \frac{3}{35}. \quad (1.19)$$

The roots of these polynomials are distinct, lie in the interval $(-1, 1)$, have a symmetry with respect to the origin, and, most importantly, are the nodes to use to solve our problem.

The nodes x_1, x_2, \dots, x_n needed to produce an integral approximation formula that will give exact results for any polynomial of degree $2n - 1$ or less are the roots of the n th-degree Legendre polynomial. In addition, once the roots are known, the appropriate coefficient for the function evaluations at these nodes can be found from the fact that for each $i = 1, 2, \dots, n$ we have

$$c_i = \int_{-1}^1 \frac{(x - x_1)(x - x_2) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_n)}{(x_i - x_1)(x_i - x_2) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)} dx \quad (1.20)$$

However, both the roots of the Legendre polynomials and the coefficients are extensively tabulated, so it is not necessary to perform these evaluations. A small sample is given in Table 1.2 [3].

This completes the solution to the approximation problem for definite integrals of functions on the interval $[-1, 1]$. But this solution is sufficient for any closed interval since the simple linear relation

$$t = \frac{2x - a - b}{b - a} \quad (1.21)$$

transforms the variable x in the interval $[a, b]$ into the variable t in the interval $[-1, 1]$.

Then the Legendre polynomials can be used to approximate

$$\int_a^b f(x) dx = \int_{-1}^1 f\left(\frac{(b-a)t + b + a}{2}\right) \frac{b-a}{2} dt \quad (1.22)$$

Using the roots $r_{n,1}, r_{n,2}, \dots, r_{n,n}$ and coefficient $c_{n,1}, c_{n,2}, \dots, c_{n,n}$ given in table 1.2, produces the following approximation formula, which gives the exact result for a polynomial of degree $2n + 1$ or less. The approximation formula is

$$\int_a^b f(x) dx \approx \frac{b-a}{2} \sum_{j=1}^n c_{n,j} f\left(\frac{(b-a)r_{n,j} + b + a}{2}\right) \quad (1.23)$$

Table 1.2: Table of Abscissa and Weight for Gaussian Integration on $[-1, 1]$

n	Roots $r_{n,i}$	Coefficients $c_{n,i}$
2	0.5773502692	1.0000000000
	-0.5773502692	1.0000000000
3	0.7745966692	0.5555555556
	0.0000000000	0.8888888889
	-0.7745966692	0.5555555556
4	0.8611363116	0.3478548451
	0.3399810436	0.6521451549
	-0.3399810436	0.6521451549
	-0.8611363116	0.3478548451
5	0.9061798459	0.2369268850
	0.5384693101	0.4786286705
	0.0000000000	0.5688888889
	-0.5384693101	0.4786286705
	-0.9061798459	0.2369268850

Example 1.4.2:

Consider the problem of finding approximations to $\int_1^{1.5} e^{-x^2} dx$, whose value to seven decimal places is 0.1093643. Gaussian quadrature applied to this problem requires that the integral be transformed into one whose interval of integration is $[-1, 1]$:

$$\int_1^{1.5} e^{-x^2} dx = \int_{-1}^1 e^{-[(1.5-1)t+1.5+1]/2]^2} \left(\frac{1.5-1}{2}\right) dt = \frac{1}{4} \int_{-1}^1 e^{-(t+5)^2/16} dt \quad (1.24)$$

The values in Table 1.2 give the following Gaussian quadrature approximations

$n = 2$:

$$\int_1^{1.5} e^{-x^2} dx \approx \frac{1}{4} [e^{-(5+0.5773502692)^2/16} + e^{-(5-0.5773502692)^2/16}] = 0.1094003, \quad (1.25)$$

$n = 3$:

$$\begin{aligned} \int_1^{1.5} e^{-x^2} dx \approx & \frac{1}{4} [(0.55555555556)e^{-(5+0.7745966692)^2/16} + (0.8888888889)e^{-(5)^2/16} \\ & + (0.55555555556)e^{-(5-0.7745966692)^2/16}] = 0.1093642 \end{aligned} \quad (1.26)$$

Using Gaussian quadrature with $n = 3$ requires three function evaluations and produces an approximations that is accurate to within 10^{-7} . The same number of function evaluations is needed if Simpson's rule is applied to original integral using $h = (1.5 - 1)/2 = 0.25$. This application of Simpson's rule gives the approximations

$$\int_1^{1.5} e^{-x^2} dx \approx \frac{0.25}{3} \left(e^{-1} + 4e^{-(1.25)^2} + e^{(1.5)^2} \right) = 0.1093104 \quad (1.27)$$

a result that is accurate only to within 5×10^{-5} .

For simple problems, the Composite Simpson's rule may be acceptable to avoid the computational complexity of Gaussian quadrature, but for problems requiring expensive function evaluations, the Gaussian procedure should certainly be considered. Gaussian quadrature is particularly important for approximating multiple integrals since the number of function evaluations increases with dimension of integrals.

1.5 Monte Carlo Method

The Monte Carlo method is an alternative way of computing integrals numerically which is not based on polynomial interpolation. In this method we choose random

points between the integral limits. The weight of each point in the calculation is assumed to be 1, then we evaluate the function value at each random point x , so the desired solution is the average of these values, that is

$$\int_a^b f(x)dx \approx \frac{1}{n} \sum_{i=1}^n f(x_i), \quad \text{where } a < x_i < b \quad (1.28)$$

As guessed, the effectiveness of the method is depending on the number of points used, that is the more number of points we use, the better result we obtain. The error in the approximation decrease as $\frac{1}{\sqrt{n}}$, where n is the number of points, as proved in [4].

CHAPTER 2

Numerical Integration in Higher Dimensions-Rectangular Regions

In his paper Ronald Cools classifies the integrals into 3 ranges [5]:

Range I: integrals of dimensions 3 to about 6 or 7

Range II: dimensions 7 or 8 to about 15

Range III: dimensions greater than 15

In Range I, they considered adaptive methods based on rules exact for polynomials are the most important tools. Range II was considered to be borderline range. Here much depends on the smoothness of the integrand. In this range adaptive Monte Carlo methods definitely become competitive, but only low accuracy is achievable. Range III was considered 'really high' dimensional. Monte Carlo is applicable. One should not hope for more than 2 digits accuracy.

For the approximation of low dimensional integrals a reasonable collection of techniques and software is available. The approximation of multiple integrals is however a different problem. In this research area, one suffers from so-called curse of dimen-

sionality; 'the computational complexity grows exponentially with the dimension [5].'

2.1 Newton-Cotes Method

The method used in one dimensional integrals can be modified to use in the approximation of multiple integrals[6]. For the sake of simplicity lets consider the double integral

$$\int_a^b \int_c^d f(x, y) dy dx = \int_R \int f(x, y) dA = \int_a^b \left(\int_c^d f(x, y) dy \right) dx \quad (2.1)$$

for some constant numbers a, b, c, d and in the same manner A is a rectangular regions. Simpson's rule is a way to illustrate the approximation, also other rules can be used.

To apply the Composite Simpson's rule, we divide the region R by partitioning the intervals $[a, b]$ and $[c, d]$ into even number of subintervals n, m , respectively. So the step sizes are $h = (b - a)/n$ and $k = (d - c)/m$. The preferred rule is applied firstly to approximate

$$\int_c^d f(x, y) dy \quad (2.2)$$

We choose Composite Simpson's rule, let $y_j = c + jk, j = 1 \dots, m$

$$\int_c^d f(x, y) dy \approx \frac{k}{3} \left[f(x, y_0) + 2 \sum_{j=1}^{(m/2)-1} f(x, y_{2j}) + 4 \sum_{j=1}^{m/2} f(x, y_{2j-1}) + f(x, y_m) \right] \quad (2.3)$$

Evaluating over the second integrand

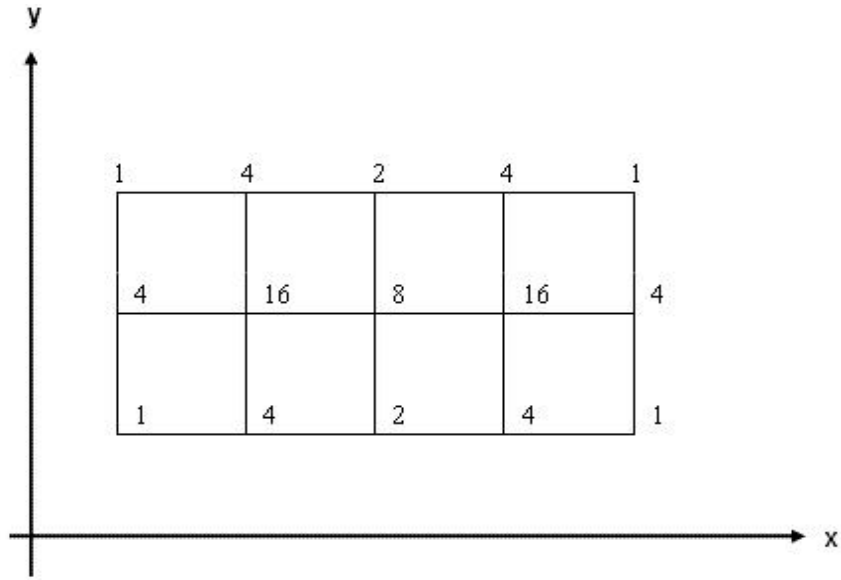


Figure 2.1: Simpson rule sampled on 2-dimensions where $m=4$, $n=2$

$$\int_a^b \int_c^d f(x, y) dx dy \approx \frac{k}{3} \left[\int_a^b f(x, y_0) dx + 2 \sum_{j=1}^{(m/2)-1} \int_a^b f(x, y_{2j}) dx \right. \\ \left. + 4 \sum_{j=1}^{m/2} \int_a^b f(x, y_{2j-1}) dx + \int_a^b f(x, y_m) dx \right] \quad (2.4)$$

Again using the Composite Simpson's rule (1.8) by dividing the interval of integration $[a, b]$, we approximate each integral above.

The same technique can be used for the approximations of higher order integrals. If we use n function evaluations in one dimension, then we will use n^d function evaluations in d dimensions.

2.2 Gaussian Quadrature

An alternative method to approximate multiple integrals is Gaussian quadrature.

Using a 2-d integral as an example we have

$$\int_a^b \int_c^d f(x, y) dy dx \approx \sum_{i=1}^N w_i \left(\int_c^d f(x_i, y) dy \right) \quad (2.5)$$

So

$$\int_a^b \int_c^d f(x, y) dy dx \approx \sum_{i=1}^N \sum_{j=1}^N w_i v_j f(x_i, y_j) \quad (2.6)$$

where (w_i, x_i) and (v_i, y_i) are the weights and abscissae of the rules used in the respective dimensions.

Higher dimensional integrals can be approximated by the same strategy.

2.3 Monte-Carlo Method

In Monte Carlo integration the method to evaluate $\int \int_R f(x, y) dy dx$ is that; chosen within the rectangular area A , the integral of the function f is estimated as the area of A multiplied by the average value of the function evaluations at the points below the curve. That is;

$$I = A \frac{\sum_{i=1}^n f(x_i, y_i)}{n} \quad (2.7)$$

The dimension change does not affect the procedure.

The fundamental disadvantage of Monte Carlo integration is that its accuracy increases only as the square root of N , the number of sampled points. If your accuracy

requirements are modest, or if your computer is good enough in performance, then the technique is highly recommended.

CHAPTER 3

Numerical Integration in Higher Dimensions-Nonrectangular Regions

In general a multiple integral over a nonrectangular region is of the form

$$I = \int_a^b \int_{L_1}^{U_1} \cdots \int_{L_{n-1}}^{U_{n-1}} f(x_1, x_2, \dots, x_n) dx_n dx_{n-1} \cdots dx_1 \quad (3.1)$$

where $L_1 = L_1(x_1)$, $U_1 = U_1(x_1)$, \dots , $L_{n-1} = L_{n-1}(x_1, x_2, \dots, x_{n-1})$ and $U_{n-1} = U_{n-1}(x_1, x_2, \dots, x_{n-1})$ and a and b are constant.

Multiple integrals over variable regions are numerically more difficult to evaluate compared to integrals over rectangular regions.

We have both theoretical and practical problems. The theoretical problem is that we have formulas that give exact results only for the case where all intermediate functions in the integral are polynomials up to a certain order. This imposes certain constraints on integral limits and it is usually impossible to check unless we evaluate the integral analytically. Finding the area itself an integration problem, so we can not use the formulas of the type

$$I = A \sum f_i \cdot w_i \quad (3.2)$$

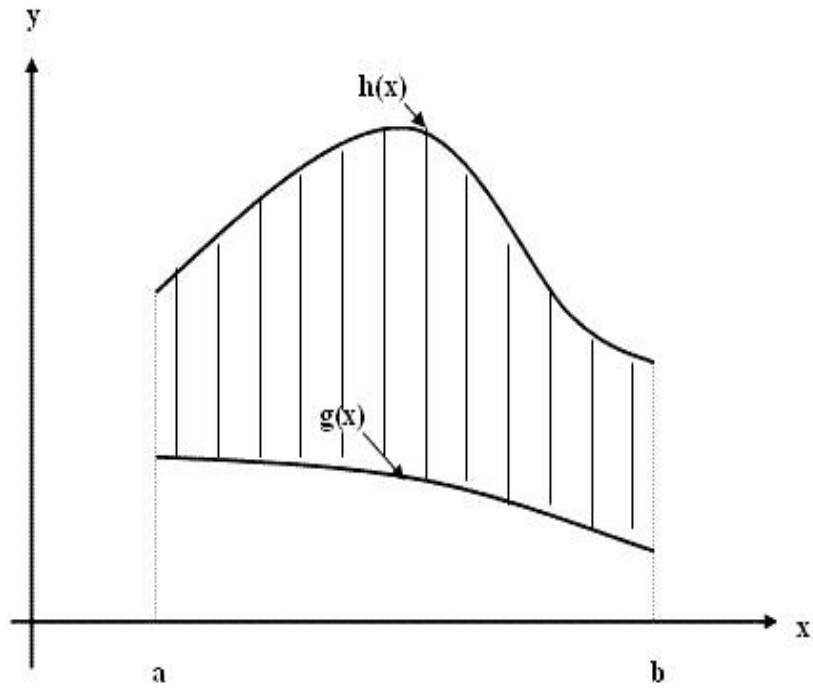


Figure 3.1: A nonrectangular region: $\int_a^b \int_{g(x)}^{h(x)} f(x, y) dy dx$

If we use Newton-Cotes methods in 2 or higher dimensions for nonrectangular regions, the points will not be regularly spaced as seen in Figure 3.2. This causes many difficulties in implementation. Therefore we have used the simplest rule, that is the trapezoidal rule in this case.

Let

$$I = \int_a^b \int_{g(x)}^{h(x)} f(x, y) dy dx. \quad (3.3)$$

Analytically this integral will be evaluated as

$$I = \int_a^b F(x, y) \Big|_{g(x)}^{h(x)} dx = \int_a^b \Phi(x) dx. \quad (3.4)$$

But numerically, we will start to determine the nodes using limits of the outer

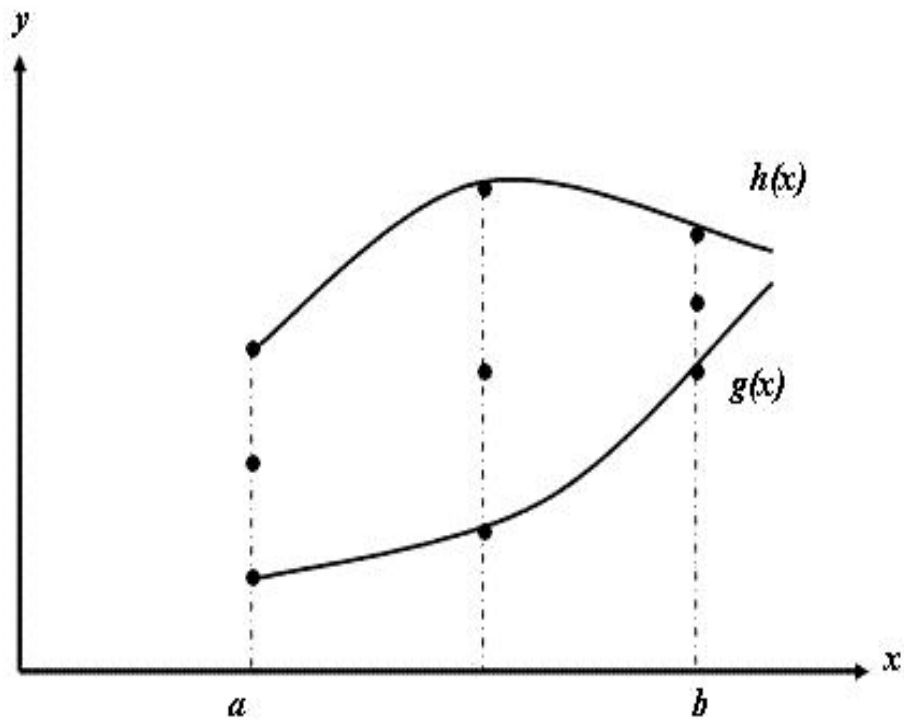


Figure 3.2: Newton-Cotes method on a nonrectangular region in 2-dimensions

integral first. The nodes are:

$$x_i = a + \Delta x \cdot i, \quad y_{ij} = g(x_i) + \Delta y_i \cdot j \quad (3.5)$$

where $i = 0, 1, \dots, n$, $\Delta x = (b - a)/n$, $j = 0, 1, \dots, k$ and $\Delta y_i = (h(x_i) - g(x_i))/k$.

Another difficulty is representing the limits and choosing appropriate points within the region.

MATLAB routines, `dblquad` and `triplequad` do not evaluate integrals in nonrectangular regions. In higher dimensions, there is no routine in MATLAB.

Since there are not many programs and there is a need we decided to write routines in MATLAB.

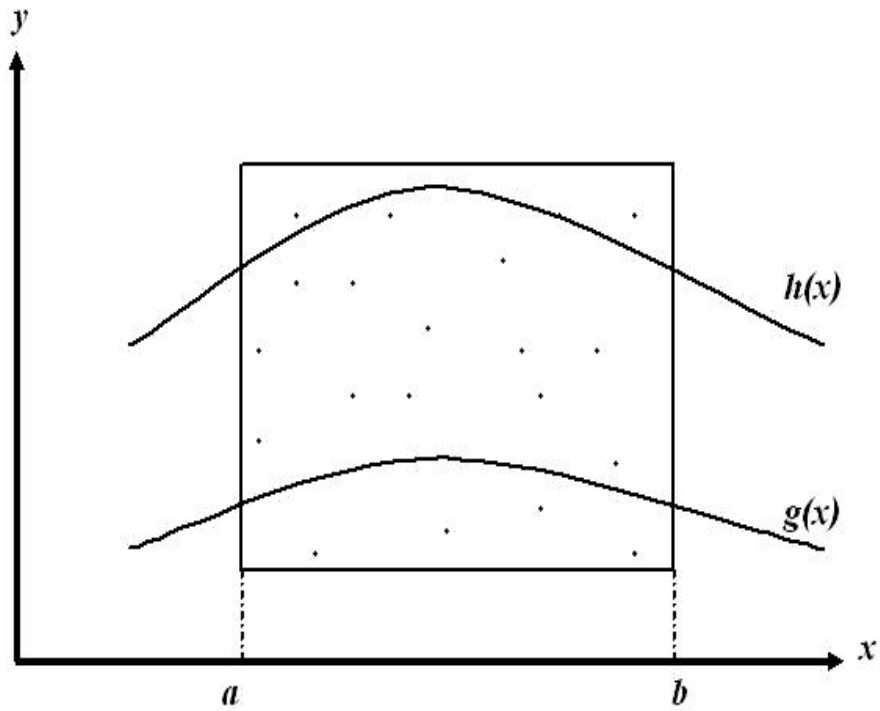


Figure 3.3: Monte Carlo method on a nonrectangular region in 2-dimensions

In nonrectangular regions the Monte Carlo method proceeds as follows; First a rectangular region is chosen such that it includes the region of integration(Figure 3.3). Then random points are taken within this rectangle. Then the area is estimated as the area of this rectangular region times the ratio of the points falling inside the region. Then to find the result of the integral we multiply this area by the average value of the function values at the random points that fall in the region of integration. As the number of dimensions grow Monte Carlo method is said to be more convenient on nonrectangular regions because the error does not depend on the dimension.

Our aim in this work is;

- To write routines in MATLAB that can evaluate integrals up to 7 dimensions on arbitrary regions.
- To compare the effectiveness of Newton-Cotes and Monte Carlo methods as number of dimensions grow.

CHAPTER 4

MATLAB Implementation

In MATLAB, quadrature routines `dblquad` and `triplequad` are used for high dimensions. That means integrals in 4 and higher dimensions can not be evaluated. Another problem is that it is unable to evaluate integrals in nonrectangular regions. Because of these reasons we decided to write our own routines.

We wrote 4 programs:

`recquad.m`: evaluates integrals in rectangular regions using Newton-Cotes closed 7 points rule.

`recquadmc.m`: evaluates integrals in rectangular regions using Monte Carlo method.

`varquad.m`: evaluates integrals in any regions using trapezoidal rule.

`varquadmc.m`: evaluates integrals in any region using Monte Carlo method.

Here are the explanation and methodology for each routine;

recquad.m

In this routine the user gives the function in terms of variables x_1, x_2, \dots, x_d the limits (all numbers) and the approximate number of points n to be used. Newton-Cotes 7 points, closed rule is used. Higher methods are not stable because they contain negative terms which results in computation errors. The program evaluates integrals in rectangular regions up to 7 dimensions.

If the given number of points is more than 1.000.000 the `subdiv` routine divides the region into 2^d parts, where d is the dimension. Otherwise the vectors grow too large for RAM, the computer uses hard disk, and the program slows down. We use 7 point Newton-Cotes rule $k+1$ times. Therefore the number of points in one dimension will be $(7 + 6k)$. In d dimensions, if we do not use `subdiv`, the total number points will be $(7 + 6k)^d$, if we use once $2^d(7 + 6k)^d$. For example, given 1000 points by the user in 2 dimensions we take $(1000^{1/2} - 1)$ points (ceiled) in each dimension and $(1000^{1/2} - 1)/6$ (ceiled) points in each small rectangular region. Because of the ceiling the exact total number of points is more than the given one. We practically need to know the exact number of points used to compare the methods Newton-Cotes and Monte Carlo.

The main part is the `Coreint` function, which gives the weight of the points. It divides the rectangular region into equally spaced points and then these points are vectorized in 1 dimensional vectors by the help of repeating the matrix (`repmat`) and transforms it into a column vector ($x = x(:)$). This really makes the routine fast to evaluate the result. Then these vectors are multiplied by Newton-Cotes weights. The

weight vector is obtained.

Then the weights are multiplied by the function values and this result is divided by the number of points, which gives the weighted average. Then this is multiplied by the multi-dimensional volume.

recquadmc.m

This program evaluates integrals in rectangular regions using Monte Carlo method. It is able to evaluate integrals between 1-7 dimensions. The user gives the function in terms of the variables x_1, x_2, \dots, x_n , the limits and the approximate number of points n to be used. The number of points in one dimension is ceiling of $(n)^{1/d}$ where d is the dimension of the integral. So the exact number of points is $[\text{ceil}(n)^{1/d}]^d$. If needed `subdiv` is used like in `recquad.m`.

In this routine; the main idea is that; we obtain random numbers between 0 and 1 by the random number generator of MATLAB. Then each of these random numbers are transferred to region of integration, that is, if the random number is r , L is the lower limit, U is the upper limit, r transferred to $L + (U - L) * r$. Then the integrand is evaluated at these new random points. The average of results are obtained by dividing the sum of these to the total number of points. The average value is then multiplied by the length of each dimensions(i.e. in 3-dimensions volume of the region)

varquad.m

This routine is written for evaluating integrals up to 7 dimensions on any region. The user gives the function, limits and the number of points. First two limits are

numbers, second two are functions of x_1 , the second two are functions of x_1 and x_2 etc. Here because of the complications of the integration region we are not subdividing the area. This results in a practical limit in the number of points the user may enter. This limit depends on the RAM of the computer, for our system it was around one million.

We decided to use trapezoidal rule for both its simplicity and adaptability to any number of points.

In rectangular case our approximation for the infinitesimal area dx_1, dx_2, \dots, dx_n has a fixed size but in this case it varies. So we have to take this into account.

The routine first calculates the points in the region, then the volume elements. Both are transformed into one dimensional arrays for simplicity in calculations. Then the function is evaluated at each point, multiplied by the volume elements and summed to obtain the result.

varquadm.c.m

This routine is written for integrals up to 7 dimensions with variable limits. It uses Monte Carlo method. The user gives the function, limits and the number of points. The exact number of points is the one ceiled to the nearest hundred.

To find a large enough rectangular region that contains the given region of integration random points are generated by MATLAB and transferred to region of integration like in `recquadm.c.m`. The random points are evaluated at each upper and lower limits of the integral. Then the minimum of the lower limits and the maximum

of upper limits is obtained. A small value is added to maximum and subtracted from the minimum value to guarantee a rectangular region that contains the curve inside.

After finding a large enough rectangular region, we generate random points in this region. Now, we have to find the points that fall within the region of integration. We constructed a matrix of ones. Then we tested whether the points are inside or outside the region using `round` and `ceil` functions. If it is inside, the corresponding matrix entry is one and zero otherwise. Afterwards the function values matrix is multiplied by this matrix of zeros and ones. So the values outside the region are omitted. Then the average value of function is obtained by dividing the sum of function values to the number of points. Finally this value is multiplied by the multi-dimensional volume.

Although we are evaluating the function at some unnecessary points using this way, the results are much more faster than using a loop to determine whether the points fall inside or outside the region.

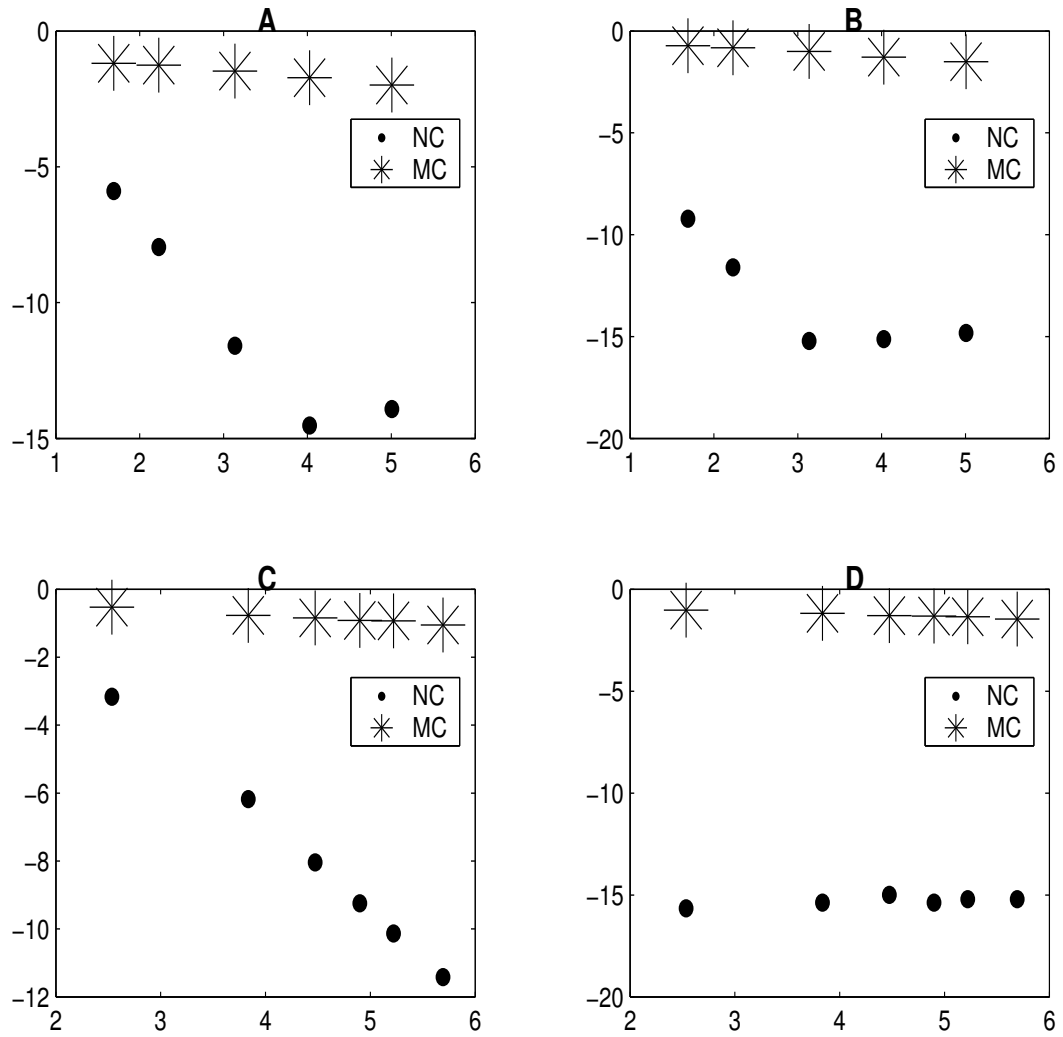
CHAPTER 5

Tests and Comparison

In this chapter some tests results and related graphics are given. In every comparison of an example the number of points taken in each method are the same. We have to say that as a natural result of the algorithm of the Monte Carlo method, in every test different results are obtained, because it uses random points. So we took the average value of absolute value of errors in 100 runs in every case. One should say that as the number of points in the tests increases the time for the result also increases. For the comparison of time for Monte Carlo method and Newton-Cotes method we can say that if the number of points are same, the elapsed times are approximately the same. In the graphs the x-axis is the logarithm of the number of points in base 10; the y-axis is the logarithm of absolute error in base 10. The relative error is defined as:

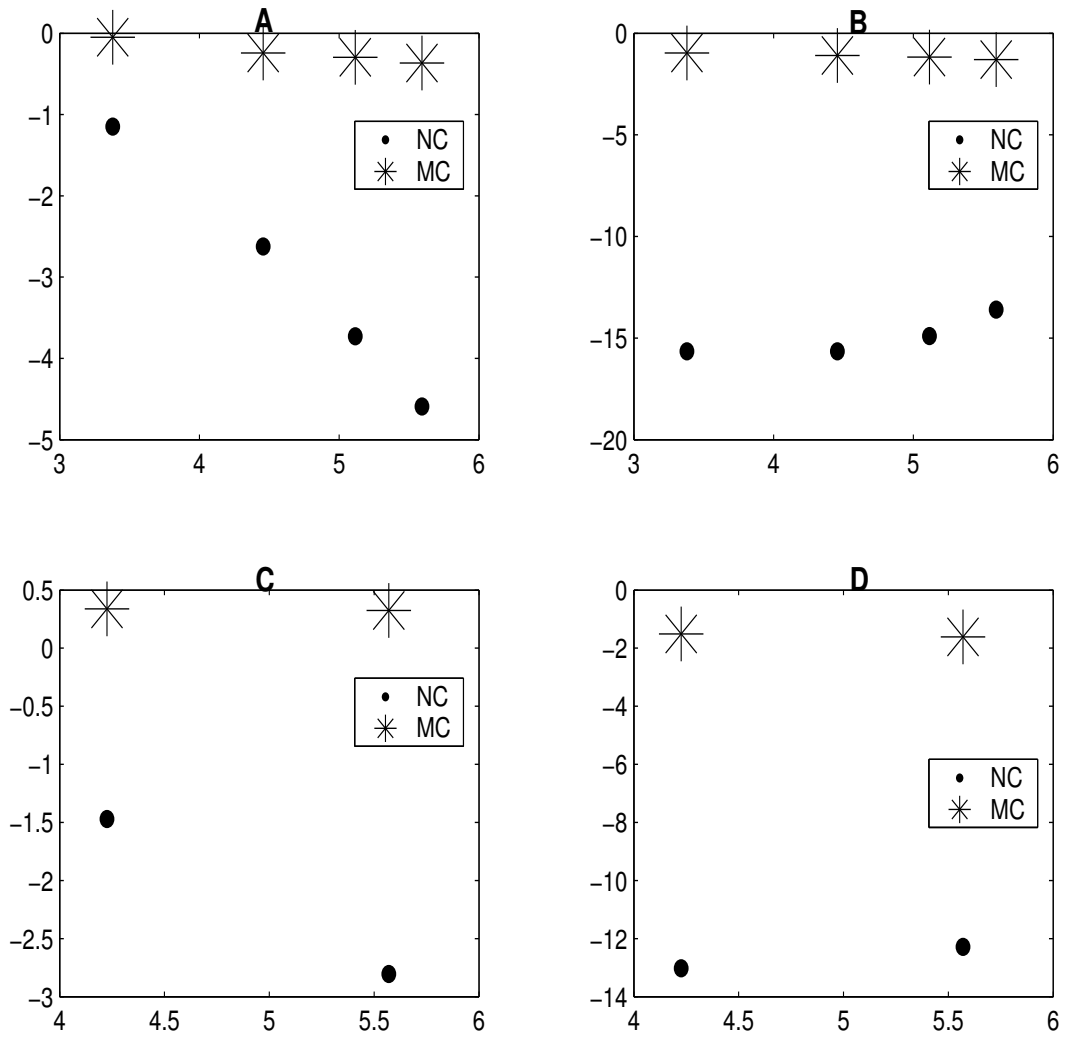
$$e = \left| \frac{I_{approximate} - I_{exact}}{I_{exact}} \right| \quad (5.1)$$

The graphs are drawn in MATLAB. In each page first the results are given and below the related integrals are given. As we expected the error gets bigger as the dimension grows, also the results are better in rectangular regions than nonrectangular ones.



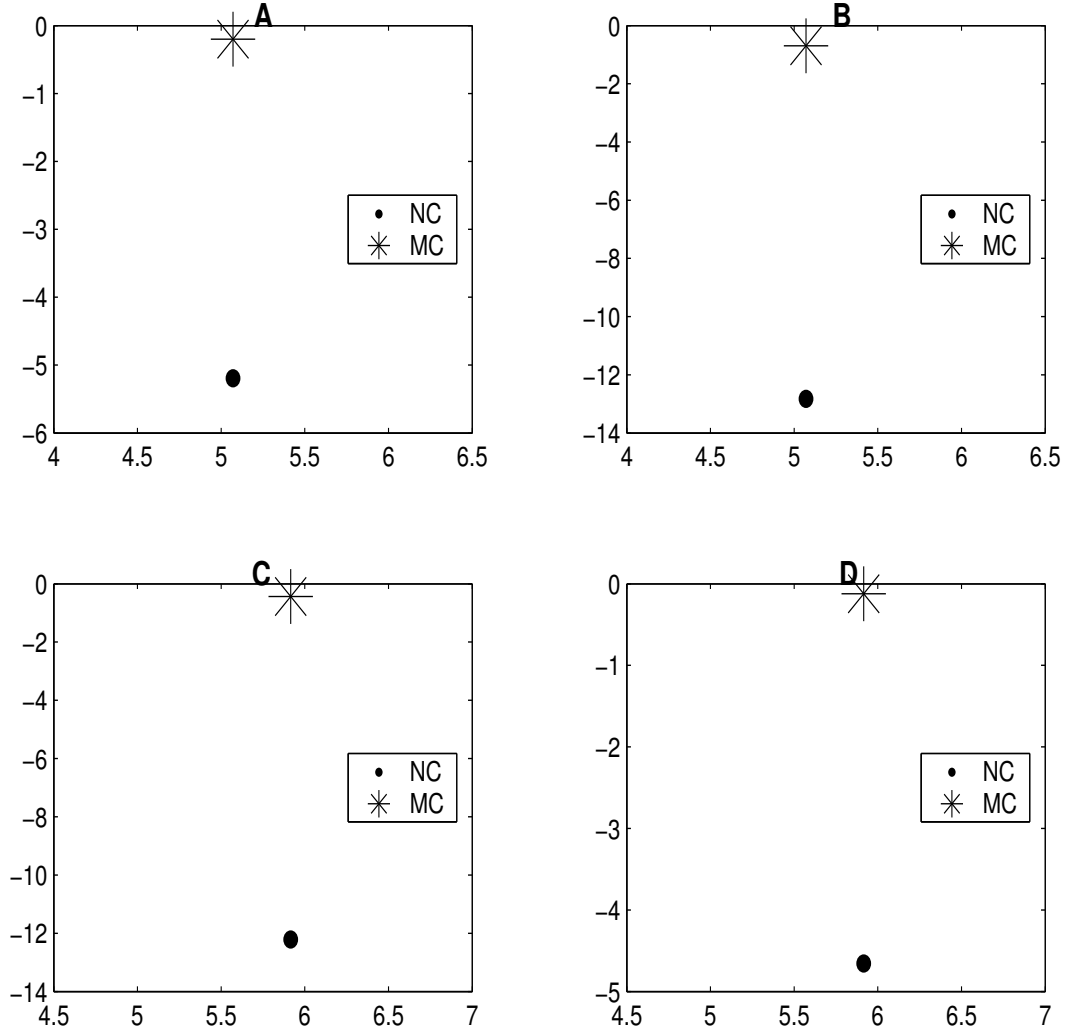
$$\begin{aligned}
 \text{A)} I &= \int_{3/2\pi}^4 \int_1^2 (x_1/x_2) \, dx_2 \, dx_1 \\
 \text{B)} I &= \int_{\pi/200}^{2\pi} \int_1^2 (\sin(100x_1)3x_2^2) \, dx_2 \, dx_1 \\
 \text{C)} I &= \int_0^{\sqrt{\pi}} \int_2^e \int_3^1 (x_1 \sin(x_1^2) \frac{1}{x_2} e^{x_3}) \, dx_3 \, dx_2 \, dx_1 \\
 \text{D)} I &= \int_0^1 \int_0^1 \int_0^3 (x_1 + x_2 + x_3) \, dx_3 \, dx_2 \, dx_1
 \end{aligned}$$

Figure 5.1: Number of points vs. relative error on log-log scale in 2 and 3 dimensions.



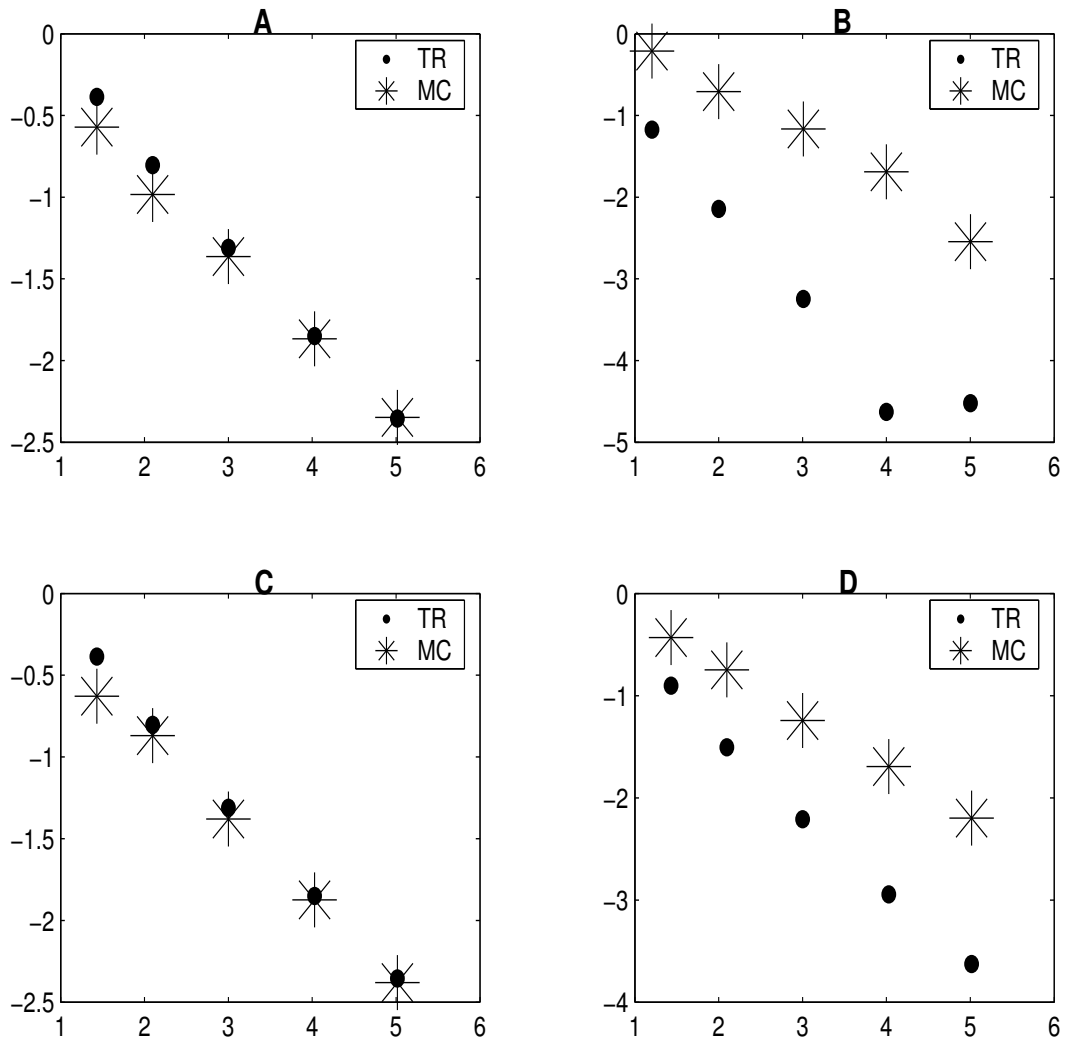
$\mathbf{A}) I = \int_0^1 \int_0^2 \int_0^3 \int_0^4 (x_1 e^{x_2} + 3x_3 + e^{4x_4}) dx_4 dx_3 dx_2 dx_1$
$\mathbf{B}) I = \int_0^1 \int_0^2 \int_0^3 \int_0^4 (15x_1^2 + 20x_2 + 5x_3 + 25x_1x_4 + 25) dx_4 dx_3 dx_2 dx_1$
$\mathbf{C}) I = \int_1^e \int_0^\pi \int_0^1 \int_{\pi/2}^0 \int_0^{\pi/2} (\log(x_1) \frac{1}{1+x_2^2} e^{x_3} \sin(10x_4) \cos(x_5)) dx_5 dx_4 dx_3 dx_2 dx_1$
$\mathbf{D}) I = \int_0^1 \int_0^2 \int_0^3 \int_0^4 \int_0^5 (18x_1^2 + 24x_2 + 6x_3 + 6x_4x_5 + 216) dx_5 dx_4 dx_3 dx_2 dx_1$

Figure 5.2: Number of points vs. relative error on log-log scale in 4 and 5 dimensions.



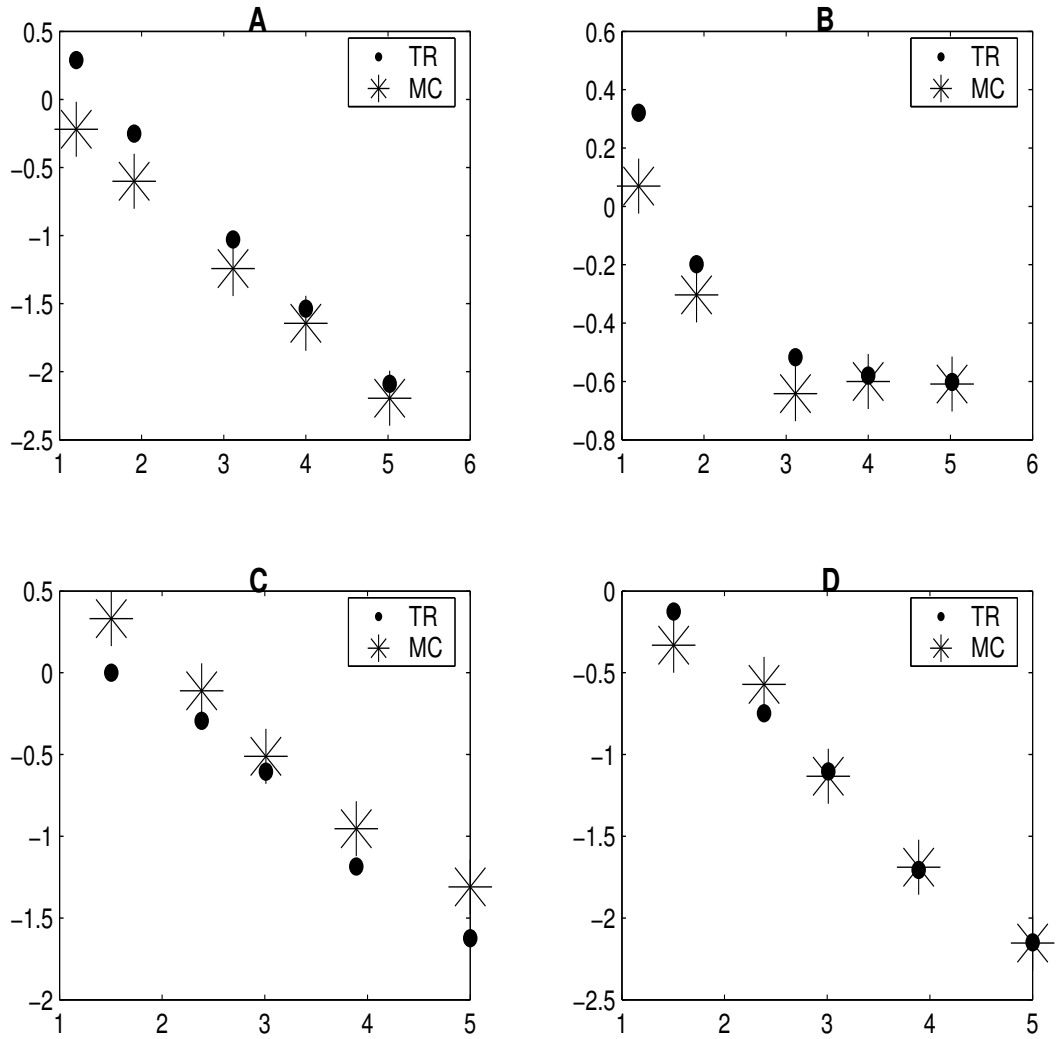
$$\begin{aligned}
 \text{A)} I &= \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 (x_1 \cos(x_1^2) x_2 x_3 x_4 x_5 x_6) \, dx_6 \, dx_5 \, dx_4 \, dx_3 \, dx_2 \, dx_1 \\
 \text{B)} I &= \int_0^1 \int_0^2 \int_0^3 \int_0^4 \int_0^5 \int_0^6 (3x_1^2 + 4x_2 + x_3 + x_4 x_5 + x_6^2) \, dx_6 \, dx_5 \, dx_4 \, dx_3 \, dx_2 \, dx_1 \\
 \text{C)} I &= \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 (x_1^2 + 4x_1 x_3 + 4x_4^3 + x_5^2 x_6 - 2x_7) \, dx_7 \, dx_6 \, dx_5 \, dx_4 \, dx_3 \, dx_2 \, dx_1 \\
 \text{D)} I &= \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 (x_1 x_2 x_3^2 + \frac{1}{x_4} x_5 x_6^3 e^{x_7}) \, dx_7 \, dx_6 \, dx_5 \, dx_4 \, dx_3 \, dx_2 \, dx_1
 \end{aligned}$$

Figure 5.3: Number of points vs. relative error on log-log scale in 6 and 7 dimensions.



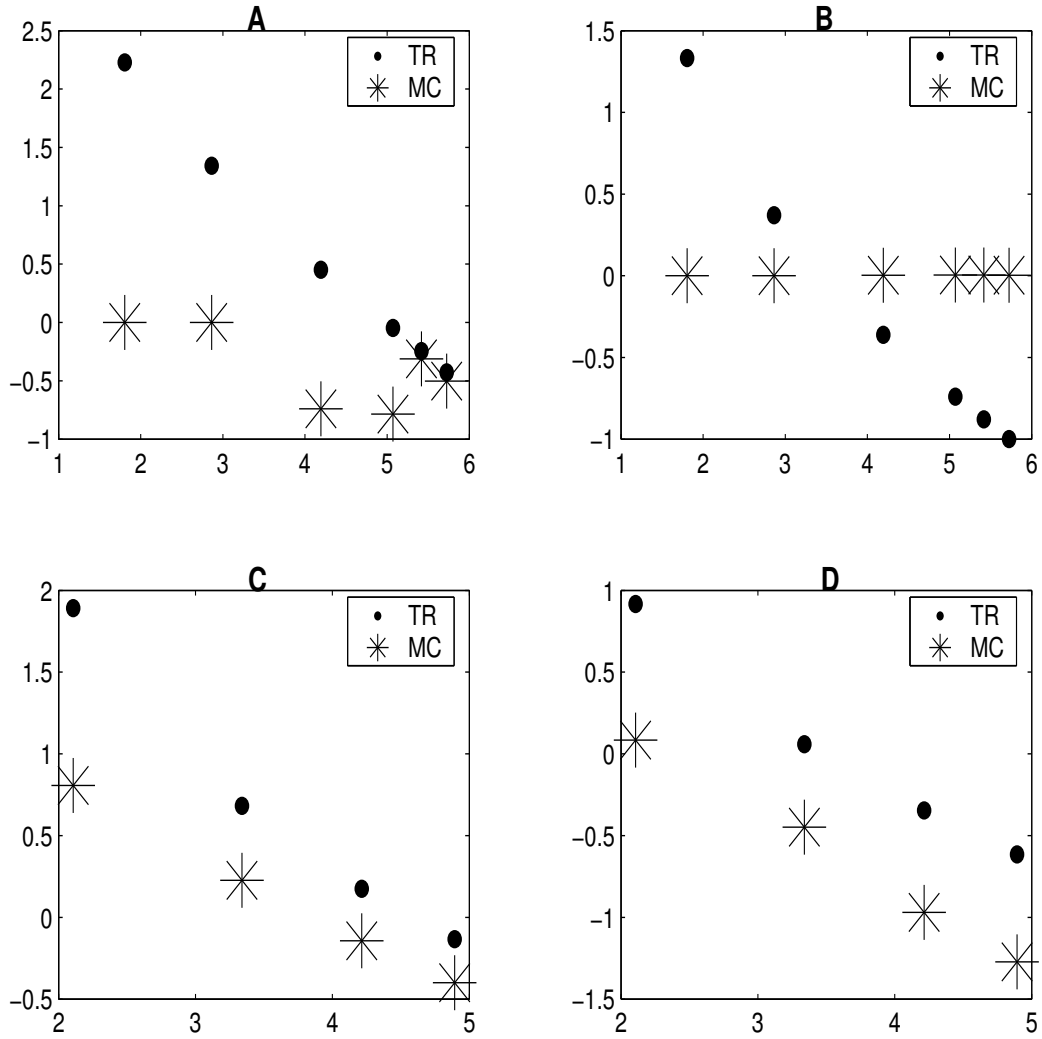
$$\begin{aligned}
 \text{A)} I &= \int_0^2 \int_0^{2-x_1} (x_1^2 + e^{x_2}) dx_2 dx_1 \\
 \text{B)} I &= \int_0^2 \int_0^{\sin x_1} (\cos(x_1) * x_2) dx_2 dx_1 \\
 \text{C)} I &= \int_0^2 \int_0^{\sqrt{4-x_1^2}} \int_0^{4-2x_2} (x_1) dx_3 dx_2 dx_1 \\
 \text{D)} I &= \int_1^3 \int_0^{6-2x_1} \int_0^{\frac{1}{3}(6-2x_1-x_2)} (1) dx_3 dx_2 dx_1
 \end{aligned}$$

Figure 5.4: Number of points vs. relative error on log-log scale in 2 and 3 dimensions.



$$\begin{aligned}
 \text{A)} I &= \int_0^1 \int_0^{x_1} \int_0^{x_1+x_2} \int_0^{x_1} (x_1^2 + 2x_3 + x_2^3 x_4) \, dx_4 dx_3 dx_2 dx_1 \\
 \text{B)} I &= \int_1^2 \int_0^{x_1} \int_0^{x_2} \int_0^{x_3} (\log(1 + x_1 x_2)) \, dx_4 dx_3 dx_2 dx_1 \\
 \text{C)} I &= \int_0^\pi \int_0^{\pi/2} \int_0^{x_2} \int_0^{x_3} \int_0^{x_2+x_3} (\sin(x_1 + x_2)) \, dx_5 dx_4 dx_3 dx_2 dx_1 \\
 \text{D)} I &= \int_1^2 \int_0^{\log(x_1)} \int_0^{x_1} \int_0^{x_1} \int_0^{x_1} (e^{-x_1-x_2} x_3 x_4 x_5) \, dx_5 dx_4 dx_3 dx_2 dx_1
 \end{aligned}$$

Figure 5.5: Number of points vs. relative error on log-log scale in 4 and 5 dimensions.



$$\begin{aligned}
 \text{A)} I &= \int_0^1 \int_0^{x_1} \int_0^{x_2} \int_0^{x_2^2} \int_0^{x_3^4} \int_0^{x_2 x_5} (3x_1 + x_2 + x_3 x_4 - x_5 x_6^2) dx_6 dx_5 dx_4 dx_3 dx_2 dx_1 \\
 \text{B)} I &= \int_0^1 \int_0^{1-x_1} \int_0^{x_2} \int_0^{x_3} \int_0^{x_4} \int_0^{1-x_1-x_2-x_3-x_4-x_5} (1) dx_6 dx_5 dx_4 dx_3 dx_2 dx_1 \\
 \text{C)} I &= \int_0^1 \int_0^{x_1} \int_0^{x_2} \int_0^{x_3} \int_0^{x_4} \int_0^{x_5} \int_0^{x_6} (x_1 e^{x_1-x_2}) dx_7 dx_6 dx_5 dx_4 dx_3 dx_2 dx_1 \\
 \text{D)} I &= \int_0^1 \int_0^1 \int_0^{x_1} \int_0^{x_2} \int_0^{x_3} \int_0^{x_4} \int_0^{x_5} (x_1^2 + x_2 x_3 + x_4 x_7) dx_7 dx_6 dx_5 dx_4 dx_3 dx_2 dx_1
 \end{aligned}$$

Figure 5.6: Number of points vs. relative error on log-log scale in 6 and 7 dimensions.

CHAPTER 6

CONCLUSION

In this research we conclude that it is much harder to evaluate integrals in non-rectangular regions than rectangular ones. Also as number of dimensions increases the number of points necessary to get a certain error level increases very fast. In high dimensions it is also more difficult or impossible to use higher order formulas that give better results. We can say that when it is possible to apply Newton-Cotes method in dimensions between 1-7, the results are definitely better than Monte Carlo method. But the necessary number of function evaluations to get a specified tolerance in Newton Cotes method exceeds the capacity of present computers by the increase in dimension. This suggests that in dimensions higher than 7-10 Monte Carlo method may be the only possibility to get a feasible result. In nonrectangular regions more work needs to be done to implement Monte Carlo method.

APPENDIX

recquad.m

```
%This program evaluates numerical integrals over rectangular
%regions up to 7 dimensions.
%Use the format: number of points, function, limits.
%Express the function in terms of x1,x2 etc.
%Example: To evaluate int_0^1 int_0^4 x1*x2 dx2 dx1
%using (approximately) 1000 points, write:
%recquad(1000,'x1*x2',0,1,0,4).
%Written by: Emre Sermutlu and Hakan Baydar in 2006.

function [tot,int]=recquad(n,f1,varargin) dim=(size(varargin,2)/2);
if dim~=1 && dim~=2 && dim~=3 && dim~=4 &&dim~=5 && dim~=6 && dim~=7
    error('Dimension must be between 1 and 7')
end

L(1)=varargin{1};U(1)=varargin{2};
if dim >=2;L(2)=varargin{3}; U(2)=varargin{4}; end
```

```

if dim >=3;L(3)=varargin{5}; U(3)=varargin{6};end
if dim >=4;L(4)=varargin{7}; U(4)=varargin{8}; end
if dim >=5;L(5)=varargin{9}; U(5)=varargin{10};end
if dim >=6;L(6)=varargin{11};U(6)=varargin{12};end
if dim >=7;L(7)=varargin{13};U(7)=varargin{14};end

while n>1000000

    [L,U]=subdiv(L,U);

    n=ceil(n/2^dim);

end

m0=ceil((n^(1/dim)-1)/6); m=6*m0+1;

tot=m^dim*size(L,1)    %Total number of points

S=0; for i=1:size(L,1)

    S=S+Coreint(m,f1,L(i,:),U(i,:));

end int=S;

function ara=Coreint(m,f1,L,U) dim=size(L,2); V=1; for i=1:dim

    y(i,:)=linspace(L(i),U(i),m);

    A= repmat(y(i,:),m^(dim-i),m^(i-1));A=A(:);

    X(:,i)=A;

    V=V*(U(i)-L(i));

end

```

```
if dim==1;f=vectorize(inline(char(f1),'x1'));F=feval(f,X(:,1));end
```

```
if dim==2;f=vectorize(inline(char(f1),'x1','x2'));
```

```
F=feval(f,X(:,1),X(:,2));end
```

```
if dim==3;f=vectorize(inline(char(f1),'x1','x2','x3'));
```

```
F=feval(f,X(:,1),X(:,2),X(:,3));end
```

```
if dim==4;f=vectorize(inline(char(f1),'x1','x2','x3','x4'));
```

```
F=feval(f,X(:,1),X(:,2),X(:,3),X(:,4));end
```

```
if dim==5;f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5'));
```

```
F=feval(f,X(:,1),X(:,2),X(:,3),X(:,4),X(:,5));end
```

```
if
```

```
dim==6;f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5','x6'));
```

```
F=feval(f,X(:,1),X(:,2),X(:,3),X(:,4),X(:,5),X(:,6));
```

```
end
```

```
if
```

```
dim==7;f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5','x6','x7'));
```

```
F=feval(f,X(:,1),X(:,2),X(:,3),X(:,4),X(:,5),X(:,6),X(:,7));
```

```
end
```

```

C0=[82 216 27 272 27 216];

C0= repmat(C0,1,(m-1)/6);C0(1)=41;C0(m)=41; C=ones(m^dim,1);

for

i=1:dim

    D=repmat(C0,(m)^(dim-i),(m)^(i-1));D=D(:);C=C.*D;

end

ara=V*sum(F.*C)/(sum(C));

function [alt,ust]=subdiv(L,U) LLL=[];UUU=[];

n=size(L,2);m=size(L,1); M=(L+U)/2;

for j=1:m

    for i=1:n

        ara=repmat([L(j,i) M(j,i)],2^(n-i),2^(i-1));LL(:,i)=ara(:);

        ara2=repmat([M(j,i) U(j,i)],2^(n-i),2^(i-1));UU(:,i)=ara2(:);

    end

    LLL=[LLL;LL];clear LL;

    UUU=[UUU;UU];clear UU;

end

alt=LLL; ust=UUU;

```


recquadmc.m

```
%this program evaluates numerical integrals over rectangular
%regions up to 7 dimensions, using Monte Carlo method.
%Use the format: number of points, function, limits.
%Express the function in terms of x1,x2 etc.
%Example: To evaluate  $\int_0^1 \int_0^4 x_1 x_2 dx_2 dx_1$ 
%using 1000 points, write:
%recquadmc(1000,'x1*x2',0,1,0,4).
%Written by: Emre Sermutlu and Hakan Baydar in 2006.
```

```
function [tot,int]=recquadmc(n,f1,varargin)

dim=(size(varargin,2)/2);

if dim~=1 && dim~=2 && dim~=3 && dim~=4 &&
    dim~=5 && dim~=6 && dim~=7
    error('Dimension must be between 1 and 7')
end

L(1)=varargin{1};U(1)=varargin{2};

if dim>=2;L(2)=varargin{3};U(2)=varargin{4}; end
if dim>=3;L(3)=varargin{5}; U(3)=varargin{6}; end
if dim>=4;L(4)=varargin{7}; U(4)=varargin{8}; end
```

```

if dim>=5;L(5)=varargin{9}; U(5)=varargin{10};end
if dim>=6;L(6)=varargin{11};U(6)=varargin{12};end
if dim>=7;L(7)=varargin{13};U(7)=varargin{14};end

j=0; while n>1000000
    [L,U]=subdiv(L,U);
    n=ceil(n/2^dim);
    j=j+1
end

m=ceil(n^(1/dim));
tot=m^dim*size(L,1)    %Total number of points
S=0;
for i=1:size(L,1)
    S=S+Coreint(m,f1,L(i,:),U(i,:));
end
int=S;

function ara=Coreint(m,f1,L,U) dim=size(L,2); V=1;

x1=rand(m,1);x1=L(1)+(U(1)-L(1))*x1;;x1=x1(:);V=(U(1)-L(1));

if dim>=2;
x2=rand(m,1);x2=L(2)+(U(2)-L(2))*x2;x2=x2(:);V=V*(U(2)-L(2));
end
end

```

```

if dim>=3;

x3=rand(m,1);x3=L(3)+(U(3)-L(3))*x3;x3=x3(:);V=V*(U(3)-L(3));

end

if dim>=4;

x4=rand(m,1);x4=L(4)+(U(4)-L(4))*x4;x4=x4(:);V=V*(U(4)-L(4));

end

if dim>=5;

x5=rand(m,1);x5=L(5)+(U(5)-L(5))*x5;x5=x5(:);V=V*(U(5)-L(5));

end

if dim>=6;

x6=rand(m,1);x6=L(6)+(U(6)-L(6))*x6;x6=x6(:);V=V*(U(6)-L(6));

end

if dim>=7;

x7=rand(m,1);x7=L(7)+(U(7)-L(7))*x7;x7=x7(:);V=V*(U(7)-L(7));

end

if dim==1;

f=vectorize(inline(char(f1),'x1'));F=feval(f,x1);

end

```

```
if dim==2;

f=vectorize(inline(char(f1),'x1','x2'));F=feval(f,x1,x2);

end

if dim==3;

f=vectorize(inline(char(f1),'x1','x2','x3'));F=feval(f,x1,x2,x3);

end

if dim==4;

f=vectorize(inline(char(f1),'x1','x2','x3','x4'));F=feval(f,x1,x2,x3,x4);

end

if dim==5;

f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5'));

F=feval(f,x1,x2,x3,x4,x5);

end

if dim==6;

f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5','x6'));

F=feval(f,x1,x2,x3,x4,x5,x6);

end

if dim==7;
```

```

f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5','x6','x7'));
F=feval(f,x1,x2,x3,x4,x5,x6,x7);

end

ara=V*sum(F)/m;

function [alt,ust]=subdiv(L,U)

LLL=[];UUU=[];

n=size(L,2);m=size(L,1); M=(L+U)/2;

for j=1:m

    for i=1:n

        ara= repmat([L(j,i) M(j,i)],2^(n-i),2^(i-1));LL(:,i)=ara(:);

        ara2=repmat([M(j,i) U(j,i)],2^(n-i),2^(i-1));UU(:,i)=ara2(:);

    end

    LLL=[LLL;LL];clear LL;

    UUU=[UUU;UU];clear UU;

end alt=LLL; ust=UUU;

```

varquad.m

```
%This program evaluates numerical integrals over ANY
%region up to 7 dimensions.
%Use the format: number of points, function, limits.
%Express the function in terms of x1,x2 etc.
%Example: To evaluate int_0^1 int_x1^(exp(x1)) x1*x2 dx2 dx1
%using (approximately) 1000 points, write:
%varquad(1000,'x1*x2',0,1,'x1','exp(x1)').
%Written by: Emre Sermutlu and Hakan Baydar in 2006.

function int=varquad(n,f1,varargin)

dim=(size(varargin,2)/2);

if dim~=1 && dim~=2 && dim~=3 && dim~=4 &&
    dim~=5 && dim~=6 && dim~=7
    error('Dimension must be between 1 and 7')
end

m=ceil((n^(1/dim)));

tot=m^dim %Total number of points

L1=varargin{1}; U1=varargin{2};

X=zeros(m^dim,dim); A=linspace(L1,U1,m); A= repmat(A,m^(dim-1),1);
```

```

X(:,1)=A(:);

kose=ones(1,m);kose(1)=0.5;kose(m)=0.5;

kose1= repmat(kose,m^(dim-1),1);kose1=kose1(:);

Delta= repmat((U1-L1)/(m-1),m^dim,1).*kose1;

if dim>=2

L2=varargin{3};fL2=vectorize(inline(char(L2),'x1'));
U2=varargin{4};fU2=vectorize(inline(char(U2),'x1'));

    for i=1:m

        k=m^(dim-1)*i;

        A=linspace(fL2(X(k,1)),fU2(X(k,1)),m);

        A= repmat(A,m^(dim-2),1);

        X([k-m^(dim-1)+1:k],2)=A(:);

    end

kose2= repmat(kose,m^(dim-2),m);kose2=kose2(:);

d=((fU2(X(:,1))-fL2(X(:,1)))/(m-1)).*kose2; Delta=Delta.*d;

end

if dim>=3

L3=varargin{5};fL3=vectorize(inline(char(L3),'x1','x2'));
U3=varargin{6};fU3=vectorize(inline(char(U3),'x1','x2'));

    for i=1:m^2

        k=m^(dim-2)*i;

```

```

        A=linspace(fL3(X(k,1),X(k,2)),fU3(X(k,1),X(k,2)),m);
        A= repmat(A,m^(dim-3),1);
        X([k-m^(dim-2)+1:k],3)=A(:);
    end

kose3=repmat(kose,m^(dim-3),m^2);kose3=kose3(:);
d=((fU3(X(:,1),X(:,2))-fL3(X(:,1),X(:,2)))/(m-1)).*kose3;
Delta=Delta.*d;
end

if dim>=4
L4=varargin{7};fL4=vectorize(inline(char(L4),'x1','x2','x3'));
U4=varargin{8};fU4=vectorize(inline(char(U4),'x1','x2','x3'));

    for i=1:m^3
        k=m^(dim-3)*i;
        A=linspace(fL4(X(k,1),X(k,2),X(k,3)),fU4(X(k,1),X(k,2),X(k,3)),m);
        A= repmat(A,m^(dim-4),1);
        X([k-m^(dim-3)+1:k],4)=A(:);
    end

kose4=repmat(kose,m^(dim-4),m^3);kose4=kose4(:);
d=((fU4(X(:,1),X(:,2),X(:,3))-fL4(X(:,1),X(:,2),X(:,3)))/(m-1)).*kose4;
Delta=Delta.*d;
end

if dim>=5

```



```

L5=varargin{9};fL5=vectorize(inline(char(L5),'x1','x2','x3','x4'));
U5=varargin{10};fU5=vectorize(inline(char(U5),'x1','x2','x3','x4'));

    for i=1:m^4

        k=m^(dim-4)*i;

        A=linspace(fL5(X(k,1),X(k,2),X(k,3),X(k,4)),fU5(X(k,1),...
X(k,2),X(k,3),X(k,4)),m);

        A= repmat(A,m^(dim-5),1);

        X([k-m^(dim-4)+1:k],5)=A(:);

    end

kose5=repmat(kose,m^(dim-5),m^4);kose5=kose5(:);

d=((fU5(X(:,1),X(:,2),X(:,3),X(:,4))-fL5(X(:,1),X(:,2),X(:,3),...
X(:,4)))/(m-1)).*kose5; Delta=Delta.*d; end

if dim>=6

L6=varargin{11};fL6=vectorize(inline(char(L6),'x1','x2','x3','x4','x5'));
U6=varargin{12};fU6=vectorize(inline(char(U6),'x1','x2','x3','x4','x5'));

    for i=1:m^5

        k=m^(dim-5)*i;

        A=linspace(fL6(X(k,1),X(k,2),X(k,3),X(k,4),X(k,5)),...
fU6(X(k,1),X(k,2),X(k,3),X(k,4),X(k,5)),m);

        A= repmat(A,m^(dim-6),1);

        X([k-m^(dim-5)+1:k],6)=A(:);

    end

kose6=repmat(kose,m^(dim-6),m^5);kose6=kose6(:);

```

```

d=((fU6(X(:,1),X(:,2),X(:,3),X(:,4),X(:,5))-fL6(X(:,1),X(:,2),...
X(:,3),X(:,4),X(:,5)))/(m-1)).*kose6; Delta=Delta.*d; end

if dim>=7

L7=varargin{13};

fL7=vectorize(inline(char(L7),'x1','x2','x3','x4','x5','x6'));

U7=varargin{14};

fU7=vectorize(inline(char(U7),'x1','x2','x3','x4','x5','x6'));

    for i=1:m^6

        k=m^(dim-6)*i;

        A=linspace(fL7(X(k,1),X(k,2),X(k,3),X(k,4),X(k,5),X(k,6)),...
fU7(X(k,1),X(k,2),X(k,3),X(k,4),X(k,5),X(k,6)),m);

        A= repmat(A,m^(dim-7),1);

        X([k-m^(dim-6)+1:k],7)=A(:);

    end

kose7=repmat(kose,m^(dim-7),m^6);kose7=kose7(:);

d=((fU7(X(:,1),X(:,2),X(:,3),X(:,4),X(:,5),X(:,6))-fL7(X(:,1),...
X(:,2),X(:,3),X(:,4),X(:,5),X(:,6)))/(m-1)).*kose7; Delta=Delta.*d;

end

if dim==1;

f=vectorize(inline(char(f1),'x1'));

F=feval(f,X(:,1));

end

```

```

if dim==2;

f=vectorize(inline(char(f1),'x1','x2'));

F=feval(f,X(:,1),X(:,2));

end

if dim==3;

f=vectorize(inline(char(f1),'x1','x2','x3'));

F=feval(f,X(:,1),X(:,2),X(:,3));

end

if dim==4;

f=vectorize(inline(char(f1),'x1','x2','x3','x4'));

F=feval(f,X(:,1),X(:,2),X(:,3),X(:,4));

end

if dim==5;

f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5'));

F=feval(f,X(:,1),X(:,2),X(:,3),X(:,4),X(:,5));

end

if dim==6;

f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5','x6'));

F=feval(f,X(:,1),X(:,2),X(:,3),X(:,4),X(:,5),X(:,6));

```

```
end
```

```
if dim==7;
```

```
f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5','x6','x7'));
```

```
F=feval(f,X(:,1),X(:,2),X(:,3),X(:,4),X(:,5),X(:,6),X(:,7));
```

```
end
```

```
int=sum(F.*Delta);
```

varquadmc.m

```
%This program evaluates numerical integrals over ANY
%region up to 7 dimensions, using Monte Carlo Method.
%Use the format: number of points, function, limits.
%Express the function in terms of x1,x2 etc.
%Example: To evaluate  $\int_0^1 \int_{x_1}^1 x_1 x_2 dx_2 dx_1$ 
%using (approximately) 1000 points, write:
%varquadmc(1000,'x1*x2',0,1,'x1','exp(x1)').
%Written by: Emre Sermutlu and Hakan Baydar in 2006.
```

```
function int=varquadmc(n,f1,varargin)
```

```
dim=(size(varargin,2)/2);
```

```
if dim~=1 && dim~=2 && dim~=3 && dim~=4 &&
```

```
    dim~=5 && dim~=6 && dim~=7
```

```
    error('Dimension must be between 1 and 7')
```

```
end
```

```
m=ceil(n/100)+100;
```

```
L1=varargin{1};U1=varargin{2};
```

```
x1=rand(n,1);x1=L1+(U1-L1)*x1;x1=x1(:);V=(U1-L1); Z=ones(n,1);
```

```

if dim>=2

LL2=varargin{3};fL2=vectorize(inline(char(LL2),'x1'));
UU2=varargin{4};fU2=vectorize(inline(char(UU2),'x1'));

t1=rand(m,1);t1=L1+(U1-L1)*t1;t1=t1(:);

L2=min(fL2(t1));U2=max(fU2(t1)); D=U2-L2;L2=L2-D/20;U2=U2+D/20;

x2=rand(n,1);x2=L2+(U2-L2)*x2; x2=x2(:);V=V*(U2-L2);

Y=x2-fL2(x1);M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y);

Y=fU2(x1)-x2;M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y);

end

```

```

if dim>=3

LL3=varargin{5};fL3=vectorize(inline(char(LL3),'x1','x2'));
UU3=varargin{6};fU3=vectorize(inline(char(UU3),'x1','x2'));

t2=rand(m,1);t2=L2+(U2-L2)*t2;t2=t2(:);

L3=min(fL3(t1,t2));U3=max(fU3(t1,t2));

D=U3-L3;L3=L3-D/20;U3=U3+D/20; x3=rand(n,1);x3=L3+(U3-L3)*x3;

x3=x3(:);V=V*(U3-L3);

Y=x3-fL3(x1,x2);M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y);

Y=fU3(x1,x2)-x3;M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y);

end

```

```

if dim>=4

LL4=varargin{7};fL4=vectorize(inline(char(LL4),'x1','x2','x3'));

```

```

UU4=varargin{8};fU4=vectorize(inline(char(UU4),'x1','x2','x3'));
t3=rand(m,1);t3=L3+(U3-L3)*t3;t3=t3(:);
L4=min(fL4(t1,t2,t3));U4=max(fU4(t1,t2,t3));
D=U4-L4;L4=L4-D/20;U4=U4+D/20; x4=rand(n,1);x4=L4+(U4-L4)*x4;
x4=x4(:);V=V*(U4-L4);
Y=x4-fL4(x1,x2,x3);M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y);
Y=fU4(x1,x2,x3)-x4;M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y);
end

if dim>=5
LL5=varargin{9};fL5=vectorize(inline(char(LL5),'x1','x2','x3','x4'));
UU5=varargin{10};fU5=vectorize(inline(char(UU5),'x1','x2','x3','x4'));
t4=rand(m,1);t4=L4+(U4-L4)*t4;t4=t4(:);
L5=min(fL5(t1,t2,t3,t4));U5=max(fU5(t1,t2,t3,t4));
D=U5-L5;L5=L5-D/20;U5=U5+D/20; x5=rand(n,1);x5=L5+(U5-L5)*x5;
x5=x5(:);V=V*(U5-L5);
Y=x5-fL5(x1,x2,x3,x4);M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y);
Y=fU5(x1,x2,x3,x4)-x5;M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y); end

if dim>=6 LL6=varargin{11};
fL6=vectorize(inline(char(LL6),'x1','x2','x3','x4','x5'));
UU6=varargin{12};
fU6=vectorize(inline(char(UU6),'x1','x2','x3','x4','x5'));
t5=rand(m,1);t5=L5+(U5-L5)*t5;t5=t5(:);

```

```

L6=min(fL6(t1,t2,t3,t4,t5));U6=max(fU6(t1,t2,t3,t4,t5));
D=U6-L6;L6=L6-D/20;U6=U6+D/20; x6=rand(n,1);x6=L6+(U6-L6)*x6;
x6=x6(:);V=V*(U6-L6);
Y=x6-fL6(x1,x2,x3,x4,x5);M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y);
Y=fU6(x1,x2,x3,x4,x5)-x6;M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y); end

if dim>=7 LL7=varargin{13};
fL7=vectorize(inline(char(LL7),'x1','x2','x3','x4','x5','x6'));
UU7=varargin{14};
fU7=vectorize(inline(char(UU7),'x1','x2','x3','x4','x5','x6'));
t6=rand(m,1);t6=L7+(U7-L7)*t6;t6=t6(:);
L7=min(fL7(t1,t2,t3,t4,t5,t6));U7=max(fU7(t1,t2,t3,t4,t5,t6));
D=U7-L7;L7=L7-D/20;U7=U7+D/20; x7=rand(n,1);x7=L7+(U7-L7)*x7;
x7=x7(:);V=V*(U7-L7);
Y=x7-fL7(x1,x2,x3,x4,x5,x6);M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y);
Y=fU7(x1,x2,x3,x4,x5,x6)-x7;M=max(abs(Y));Y=Y/M;Z=Z.*ceil(Y); end

if dim==1;f=vectorize(inline(char(f1),'x1'));F=feval(f,x1);end

if dim==2;
f=vectorize(inline(char(f1),'x1','x2'));F=feval(f,x1,x2);
end

if dim==3;

```



```
f=vectorize(inline(char(f1),'x1','x2','x3'));F=feval(f,x1,x2,x3);
```

```
end
```

```
if dim==4;
```

```
f=vectorize(inline(char(f1),'x1','x2','x3','x4'));F=feval(f,x1,x2,x3,x4);
```

```
end
```

```
if dim==5;
```

```
f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5'));F=feval(f,x1,x2,x3,x4,x5);
```

```
F=feval(f,x1,x2,x3,x4,x5);
```

```
end
```

```
if dim==6;
```

```
f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5','x6'));F=feval(f,x1,x2,x3,x4,x5,x6);
```

```
F=feval(f,x1,x2,x3,x4,x5,x6);
```

```
end
```

```
if dim==7;
```

```
f=vectorize(inline(char(f1),'x1','x2','x3','x4','x5','x6','x7'));F=feval(f,x1,x2,x3,x4,x5,x6,x7);
```

```
F=feval(f,x1,x2,x3,x4,x5,x6,x7);
```

```
end
```

```
int=sum(F.*Z)*V/n;
```

REFERENCES

- [1] **Horwitz A.** (2001), *A Version of Simpson's Rule for Multiple Integrals*, *Journal of Computational and Applied Mathematics*, 1-11. Vol. 134
- [2] **Turner P.R.** (1994), *Numerical Analysis*, Macmillan, England
- [3] **Burden R.L., Faires J.D.** (2001), *Numerical Analysis*, Brooks/Cole, U.S.A.
- [4] **Evans M., Swartz T.** (2000), *Approximating Integral via Monte Carlo and Deterministic Methods*, Oxford University Press, United States.
- [5] **Cools R.** (2002), *Advances in Multidimensional Integration*, *Journal of Computational and Applied Mathematics*, 1-12. Vol. 149
- [6] **Kincaid D., Cheney W.** (2002), *Numerical Analysis: Mathematics and Scientific Computing*, Brooks/Cole, U.S.A.
- [7] **Rice J.R.** (1983), *Numerical Methods, Software, and Analysis*, McGraw-Hill, U.S.A.
- [8] **F. Bornemann, D. Laurie, S. Wagon, and J. Waldvoge** (2004), *The SIAM 100-Digit Challenge: A Study in High-Accuracy Numerical Computing*, SIAM, USA
- [9] **W. Gander and W. Gautschi** (2000), *Adaptive Quadrature Revisited*, *BIT Numerical Mathematics*, 84- 101. Vol.40