

**An interactive program
to approximate double integrals:
an easy to use interface for Cubpack++**

Bart Maerten and Ronald Cools

Report TW269, June 1997



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

**An interactive program
to approximate double integrals:
an easy to use interface for Cubpack++**

Bart Maerten and Ronald Cools

Report TW 269, June 1997

Department of Computer Science, K.U.Leuven

Abstract

Using **Cubpack++** to approximate a double integral requires from users that they write a small C++ program. Although example programs are provided, this still introduces a treshold. This short note describes an interface to **Cubpack++** that relieves users from any need to write C++.

CR Subject Classification : G.1.4 [Numerical Analysis]: Quadrature and Numerical Differentiation – *adaptive quadrature; multiple quadrature*; G.4 [Mathematical Software]: *efficiency; reliability and robustness*

An interactive program to approximate double integrals: an easy to use interface for **Cubpack++**

Bart Maerten and Ronald Cools

Dept. of Computer Science

Katholieke Universiteit Leuven

Celestijnenlaan 200A

B-3001 Heverlee, Belgium

E-mail: {Bart.Maerten | Ronald.Cools}@cs.kuleuven.ac.be

Abstract

Using **Cubpack++** to approximate a double integral requires from users that they write a small C++ program. Although example programs are provided, this still introduces a threshold. This short note describes an interface to **Cubpack++** that relieves users from any need to write C++.

1 Introduction

A situation that is typical for mathematical software (libraries) is that they are often sandwiched between a user-written main program and a user-written function routine. This situation occurs if one wants to compute the zeros, the extremal points or the integral of a specific function. The user has to provide a program from which he calls a zero-finder, an integration routine, etc. from a library, with the appropriate parameters. He also has to provide a routine that implements the function under investigation that will be called by a library routine. To avoid re-compilation and re-linking, the user will normally write his main program such that it asks interactively for as many parameters as possible. Changing the function will in most cases force him to recompile and re-link. **Cubpack++** is a C++ class library dealing with automatic integration of functions over a large variety of two-dimensional regions [1, 2]. One of the referees of the paper was unhappy because he had to wait a terrible long time if he made small changes to his integrand function. The explanation for the delay was that with a language such as C++ linking might involve much more than linking with a language such as Fortran. To be more specific, some C++ compilers delay the instantiation of template classes until the link stage.

We are sensitive to user comments and we want to make **Cubpack++** more easy to use. In this text we describe an interface to **Cubpack++** which

- avoids any compilation by you¹ and
- avoids that you have to write real C++ code.

This is not meant for very complicated integrands. The more complicated they are, the more they suffer a performance drop. For simple functions performance is no problem.

The necessary background and knowledge on how to write a function parser was found in [3].

In what follows we will explain how you have to enter integrands and what you can do with this interface.

¹We refer to the user of an automatic package as ‘you’ for the sake of conciseness, and apologize to any reader who finds this convention over-familiar.

2 A simple interface for Cubpack++

2.1 Appearance of the interface.

The interface has been kept simple. It is not graphical to keep it as portable as possible. We only provide a subset of the available regions of **Cubpack++** at this moment in the interface.

First you will be prompted to enter the *integrand*. At this stage you can also enter a question mark to obtain information on how to enter this integrand and which elementary functions are available. Then you will be prompted for the *order of the 2 integration variables*. If your integrand uses only one, you will get a warning and the possibility to correct your error, if any.

Secondly you have to enter the *integration domain*. This is done by choosing simple regions from a menu. Every time you select a region, you'll need to specify this region and it will be added to the domain. In this way you build the whole integration domain. Regions are most of the time specified by a set of points. Points are entered as two reals, separated with a space or a comma.

The last step before integration is to enter the *accuracy* you request and to specify a *maximum number of function evaluations* the integrator is allowed to use. In section 2.3 a complete session is given.

After the first integration you will get a small menu. This will offer you 3 possibilities:

1. Integration of another integrand. You will have to give a new domain.
2. Same integrand, but integration over a new domain.
3. Same integrand and domain, but for another accuracy request.

2.2 Syntax of the integrand function

Simple expressions, i.e., 1 line expressions, are the most obvious way to define the function you want to integrate. They use the classical operators like addition, multiplication, powers, exp, cos, Also constants, such as pi, and parentheses can be used. With a \wedge one denotes the power operator. Table 1 contains a list of functions that are implemented. You can use any name you like for the independent variables as long as they differ from built-in functions. *Every statement has to end with a ";"*. *The definition of the function ends with an extra ";"*. For example:

```
cos(x+y+pi);;
```

Numbers can be entered with the usual floating point notation in an expression. A side effect from the function parser is that you can write for example $0.5e-5$ also as $0.5*10^{(-5)}$. Be aware that the second notation will be considered as an expression and will be evaluated, hence it is less efficient.

Multiple lines, use of extra variables make it easier to define (long) functions with some repetition patterns. You can assign the result of simple expressions to some extra variables and use them in following statements. The last statement is either an assignment or a simple expression. *The result of the last statement is the value of the integrand.*

```
r=x*x+y*y;  
1/(1.2+sqrt(r))*exp(-r);;
```

If-then-else statements give you the opportunity to define more complicated functions. Within an if-then-else structure you can use also multiple statements. These can be assignments or if-then-else statements. As a rule we expect a sequence of *if-then(-else) structures* to be *followed either by an assignment or a simple expression* (which has to be the last statement).

```

r = x^ 2+y^ 2;
if ((r<0.5) || (r>1.1))
    { z = r*4-pi; }
else
    { z = r/4*pi; };
sqrt(abs(z));

```

Remarks. The number of statements is limited to 20. However, within an if-then-else structure one can also use that number of statements. If-then-else structures don't have a recursion-limit. The maximum length of a variable is 20 characters.

Built-in functions and possibilities. Table 1 shows which functions are implemented. There is also an online help which should give an up to date overview of available functions.

Table 1: Built-in functions, boolean operators, and constants.

functions	inverse functions	boolean operators	constants
^ power		&& and	pi 3.14159265...
abs absolute value		or	
sqrt square root		>	
exp exponential	log natural logarithm	>=	
cos cosine	acos arc cosine	<	
sin sine	asin arc sine	<=	
tan tangent	atan arc tangent	== equality	
cotan cotangent	acotan arc cotangent	!= not equal	

Formally, a statement has a form as described in Table 2. The names between <> express the sort of structure that is expected there. An <expression> can be defined using the classical functions. For a <boolean_expression> one needs some extra functionality by boolean operators. A <multiple_statement> within an if-then(-else) is a sequence of assignments and/or if-then(-else) structures.

Table 2: Structure of statements.

<expression> ;	normal expression
<variable name> = <expression> ;	assignment
if (<boolean_expression>) { <multiple_statement> } ;	if-then
if (<expression>) { <multiple_statement> } else { <multiple_statement> } ;	if-then-else

2.3 An example session.

The example session is for the integration of

$$\int_{\Omega} \frac{1}{\frac{1}{200^2} + (x - 0.7)^2} \cdot \frac{1}{\frac{1}{100^2} + (y - 0.5)^2} dx dy, \quad (1)$$

where Ω is the parallelogram defined by the vertices (0,0), (1,0), (0.3,1), and (1.3,1). This function has an internal peak at (0.7, 0.5).

Enter an integrand or press ? to get some info

```
In>> z1 = 1/(200*200)+(x-0.7)^2;  
      z2 = 1/(100*100)+(y-0.5)^2;  
      1/(z1*z2);;
```

Give the integration variables in the proper order,
separated by space(s) or return(s)

```
In>> x y
```

```
var1: x  
var2: y
```

Define your integration domain by adding regions.
The integration region now consists of 0 subregions.
You can add more if you want.

1. TRIANGLE.
2. PARALLELOGRAM.
3. RECTANGLE.
4. CIRCLE.
5. OUT_CIRCLE.
6. PLANE.
7. INFINITE_STRIP.
8. SEMI_INFINITE_STRIP

0. to end

Enter choice: 2

Give 3 points:

Basepoint A and points B & C at adjacent sides
so that a parallelogram is given by the four
corner points A,B,C,D=B+C-A

In>> 0 0 1 0 0.3 1

Define your integration domain by adding regions.
The integration region now consists of 1 subregions.
You can add more if you want.

1. TRIANGLE.
 2. PARALLELOGRAM.
 3. RECTANGLE.
 4. CIRCLE.
 5. OUT_CIRCLE.
 6. PLANE.
 7. INFINITE_STRIP.
 8. SEMI_INFINITE_STRIP
0. to end

Enter choice: 0

Give absolute error, relative error, maximum evaluations

In>> 0 0.5e-3 100000

Out>> The result is 193625.919529
Out>> with estimated absolute error 96.4
Out>> The number of evaluations used is 30081

1. New integrand (and new region).
 2. Same integrand, new domain.
 3. Integrate (other accuracy).
0. Quit.

Enter choice: 0

End of execution.

3 A link with gnuplot

The interface is non-graphical, but we want the user to be able to see where the integration routine performed function evaluations. For this we provide a link with the function plotting utility “gnuplot”.

Cubpack++ first asks the user if he wants to see the evaluation points. So, before the session given in the previous section **Cubpack++** will show you the following line.

```
Do you want a plot of the evaluation-points ? (n,N/y,Y) y
```

The sequel of the session is exactly as showed in the previous section. The only change you will notice is that after the result is printed, you will have a window with the evaluation points and that you first have to press the enter key to proceed. The window with the evaluation points then disappears. Note that a file named ‘evalplot.gp’ is created in the current directory to store the evaluation points. This file can become large if many points are used!

```
*****
```

```
Out>> The result is 193625.919529
Out>> Estimated abs err 96.4
Out>> Number of evaluations: 30081
```

```
*****
```

```
Hit return to continue
```

The result of the previous example is presented in Figure 1.

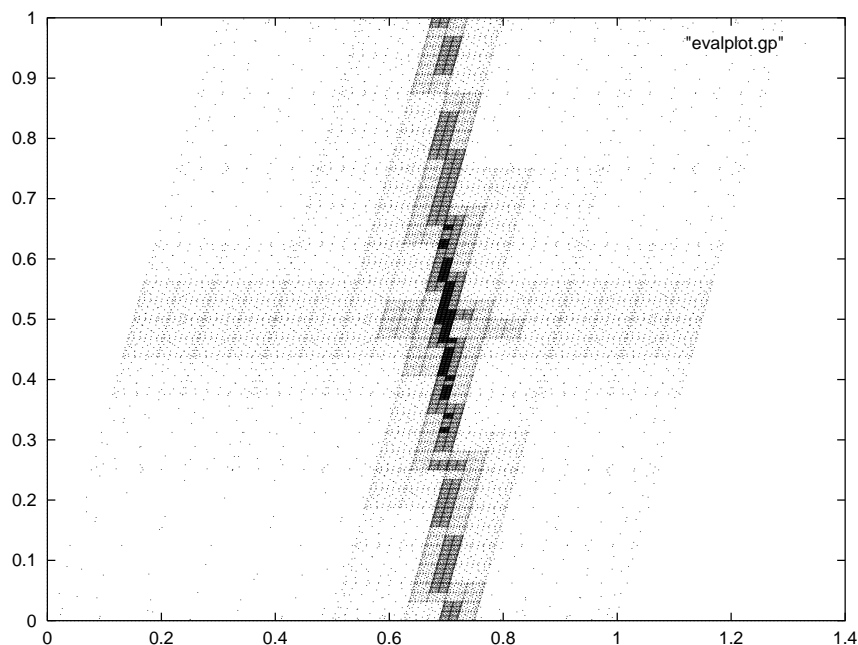


Figure 1: Evaluation points for the integration of integral (1)

4 Availability

We try to make **Cubpack++** as portable as possible. To find out whether our code is portable or not, we try all compilers we can get. This situation will remain as long as the ANSI-standard is not finalised and as long as the available compilers do not implement this standard.

One can download the software from <http://www.cs.kuleuven.ac.be/~ronald/>. A list of compilers that compile the current version of **Cubpack++** successfully is given in Table 3. The steps you have to do to get **Cubpack++** at work for you, are described in the file INSTALL.ITF in the parent directory.

Table 3: Systems for which **Cubpack++** has proven it works.

Compiler	Version	Operating System
g++/gcc	2.7.2	Ultrix 4.4
		SunOS 5.5.1
x1C		IBM AIX Version 3.2
CC	SPARCompiler C++ 4.1	SunOS 5.5.1

References

- [1] R. Cools, D. Laurie, and L. Pluym, *Algorithm 764: Cubpack++: A C++ Package for Automatic Two-Dimensional Cubature*, ACM Transactions on Mathematical Software **23** (1997), no. 1, 1–15.
- [2] ———, *A User Manual for Cubpack++ version 1.1*, Report TW255, Dept. of Computer Science, K.U.Leuven, March 1997.
- [3] J. Lewi, K. de Vlaminc, E. Steegmans, and I. Van Horebeek, *Software Development by LL(1) Syntax Description*, Wiley, Chichester, 1992.