Numerical Integration

Leonardo Trentini, Odile Montarnal
Programming Concepts in Scientific Computing
MATH-458

December 16, 2022

1 Introduction

In this project, carried out as part of the Programming Concepts in Scientific Computing course, we developed an implementation of three different algorithms for numerical integration in one or two dimension.

In particular, this project aims at computing:

$$I = \int_\Omega f(\mathbf{x}) d\mathbf{x} \qquad \text{with} \quad \Omega = (a, b) \times (c, d)$$

Using either Midpoint, Trapezoidal or Cavalieri-Simpson integrators [1].

Our program might be used as a "black-box" by a generical user. For instance, this user doesn't have to be aware of the internal functioning of the code. In theory, he actually doesn't have to know how to code either: he might just clone our repository, build, run the main and follow the instructions on screen.
However, our code is designed in way such that a more expert user can easily understand its functioning and extend it for his personal purposes.

2 Usage

First of all, you need to clone the repository

```
git clone https://github.com/leotrentini22/integration.git
```

Then, build and run 'main.cpp'. This can be done with and IDE (we used Clion), or directly in the terminal, by going to the folder where you cloned our repository and typing these commands:

```
mkdir build
cd build
cmake ..
make
./main
```

Finally, just follow the instructions on screen.

3 Typical Program Execution (the Flow)

Once compiled and run, the program will show to user the instructions to integrate a generical function in 1 or 2 dimensions.
In particular, the program will ask the user:

- the dimension (either 1 or 2)
- the domain extremes (2 extremes in 1 dimension, 4 extremes in 2 dimensions)
- the number(s) of partitions (namely, in how many partitions divide our domain)
- the method to choose (Midpoint, Trapezoidal or Cavalieri-Simpson)

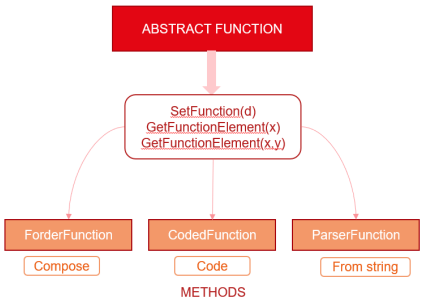
- the function to integrate. In particular, the user will be asked to insert a string containing the desired function. The string can contain variables (x in 1 dimension, (x, y) in 2 dimensions) and common "standard" functions, such as $cos(x)$ or $exp(x + y)$ (see 4)

For each of these steps, several checks were added, so that a user might not crash the program. For example, the dimension can only be set to 1 or 2, and if a user tries to set it to a different number, for example 3 or -1, the program will show an error and will ask the user to insert the dimension again.

After these passages, the integral will be calculated and the result will be printed on the screen.

4 List of Features

The Main is written as concise as possible: all the set up of the problem is done in a class, called **Initialization**, that contains all the methods to print instructions to user, get inputs, run the calculation and print the result (either to the screen or into a file).



To get the function from user, we have written an abstract class, called **Abstract Function**. This class contains only pure abstract methods, as it is a mother classes to three daughter classes, that are three different manners to get the function from the user:

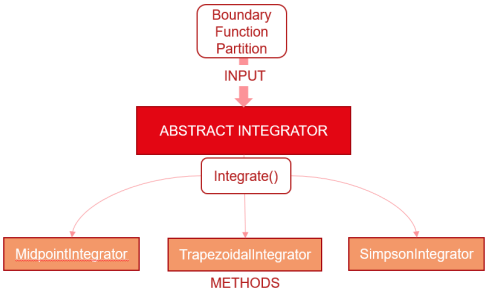
- Parser Function
- Forder Function
- Coded Function

In particular, our program uses **Parser Function** to complete this task. This class contains methods to get a function from a string. It uses an external library to parse the string and evaluate it, and this library is installed when cloning our repository. However, the code is abstracted enough and it shouldn't be a problem to change this library and use another one in case of need.

We also developed two other classes to get the function to integrate from the user. **Forder Function** is designed in a way such that the user can iteratively compose the function, by inserting different numbers to the screen. The starting point would be $f(x) = x$ (or $f(x, y) = x$ in 2 dimensions), then the user can apply exponential, sinusoidal, add a constant, multiply for a constant (and the same for y in two dimensions). We decided not to use this class in our program, because we found it harder to understand for a generic user and less complete and exhaustive than Parser Function.

The third daughter class is **Coded Function**: in this class, the user can directly write a function inside the code. If we want to use this class, we have to recompile every time we want to change the function to integrate, but on the other hand there are no inputs needed from the user during the flow. For these reasons, this class is used to run the Google Tests (see 5).

We chose not to allow a generic user to decide which manner to use to insert the function, because we thought that Parser Function had definitely the best behaviour, for the reasons explained above. In addition, it would be difficult to explain these different behaviours in a synthetic way on the screen to a generic user, and in this form we would also prevent unwanted outputs.



However, a user can choose the class to use by changing just a line in the Initialization class.

The other family of classes that we implemented is directly connected to the numerical integration that the program has to carry out. The mother class is **Abstract Integrator**, that has as private members the **domain extremes**, the **number(s) of partitions** and the **function to integrate**. Moreover, this class contains useful public methods, that will be inherited by the daughter classes, and the pure virtual method **Integrate**, that will be only carried out in the daughter classes.

The three daughter classes contain the three different numerical algorithms:

- **Midpoint Integrator**
- **Trapezoidal Integrator**
- **Simpson Integrator**

These algorithms are implemented for 1 and 2 dimensions, but are easily extendible to more dimensions.

For a deeper view of our features, in **Doxyfile** we provide instructions in order to build a complete documentation for our project with Doxygen. To use it, first you need to install Doxygen. Then open our Doxyfile with **Doxywizard** (installed within Doxygen) and run.

5 Tests

We implemented two different tests on our program.
At first, in **test integrators** we use Coded Function and **Google Tests** to evaluate the validity of our numerical methods. The results of numerical integration are compared to the known results with a certain tolerance of precision.
On the other hand, to test the correctness of Initialization class, we implemented in **test main** a series of assertions that evaluate if, given wrong inputs by the user, our methods behave as expected.

6 Possible developments

During our work, we found some developments that might be done, although we preferred to concentrate our efforts on a precise task and leave these developments for a future attempt.

Our program can be easily extended to **more dimensions**, following the pattern that we used for the second dimension.
It would be possible also to add **more numerical methods**, such as the Booles Rule [2], following the pattern provided by Abstract Integrator.
Moreover, the code can be extended to **stranger domains**. This implies that a new abstract class should be created so that the way to get the domain can be changed easily. The domain might be passed as a function, using Function Parser class. Then this domain can be divided in simpler domains, assembled by union.
In addition, our program might be extended to work in **complex domains**. This would need to create a class of **complex numbers** and then overload all the methods previously presented. However, this extension would require to investigate better how does integration work in the theory of complex spaces [3].

References

[1] A. Quarteroni and F. Saleri, "Scientific computing with matlab and octave," *Springer Berlin*, 2006.

[2] M. Abramowitz and I. A. Stegun, "Handbook of mathematical functions with formulas, graphs, and mathematical tables," *Dover*, 1972.

[3] J. Stalker, "Complex analysis: Fundamentals of the classical theory of functions," *Springer*, 1998.