

CS319: Scientific Computing (with C++)

## CS319 Lab 6: Numerical Integration 2

Week 8 (6+7 March, 2025)

### Goal:

- ▶ Derive and Implement Simpson's Rule in 2D;
- ▶ Use a Jupyter Notebook to verify its convergence.
- ▶ Compare with a Monte-Carlo Method.

### Deadline:

Submit your work for the assignment on Slide 10 by 17:00, Wednesday 12 March.

# 1. Review

1. Review your notes from Weeks 6 and 7 on Numerical Integration in one and two dimensions.
2. Complete Lab 5: from that you should have working code for the Trapezium, **Simpson's**, and Boole's Rule for estimating a definite integral of a function of one variable. (If you've yet to complete Lab 5, focus on implementing Simpson's Rule).

## 2. 1. Trapezium Rule in 2D

In class we did a rough derivation of the Trapezium Rule in 2D, and implemented it as a function `Trap2D()`.

Presently, we want to the two-dimensional Simpson's Rule. To have a single framework which works for both, we'll consider a slightly different presentation of the Trapezium Rule.

First, let's write the general formulation for a one-dimensional quadrature method:

- ▶ Define the (equally spaced) **quadrature points**:  $\{x_0, x_1, \dots, x_N\}$ .  
If we set  $h = (b - a)/N$ , then  $x_i = a + ih$ .
- ▶ Define the associated **quadrature points**:  
 $y_0 = f(x_0), y_1 = f(x_1), \dots, y_N = f(x_N)$ .
- ▶ Suppose we have a set of **quadrature weights**  $w_0, w_1, \dots, w_N$ .

## 2. 1. Trapezium Rule in 2D

- Then the general form of a quadrature rule is

$$\int_a^b f(x)dx \approx \sum_{i=0}^N w_i y_i.$$

- For the Trapezium Rule in one-dimension, we have

$$w_0 = \frac{h}{2}, \quad w_1 = h, \quad w_2 = h, \dots, \quad w_{N-1} = h, \quad w_N = \frac{h}{2}.$$

The advantage of defining the method this way is that

- It is easy to apply this approach to other methods, such as Simpson's Rule or Boole's Rule.
- It is easy to generalise to higher dimensions.

## 2. 1. Trapezium Rule in 2D

Dealing with the last point first, we know that computing

$$\int_a^b \int_a^b f(x_1, x_2) dx_1 dx_2$$

can be thought of as integrating first in  $x_1$ , and integrating then in  $x_2$ .

In the same way, we can apply our quadrature rule in each direction...

## 2. 1. Trapezium Rule in 2D

That is

$$\int_a^b \int_a^b f(x_1, x_2) dx_1 dx_2 \approx \sum_{i=0}^N w_i \left( \sum_{j=0}^N w_j y_{i,j} \right)$$

where  $y_{i,j} := f((x_1)_i, (x_2)_j)$ . Expanding that, we get

$$\int_a^b \int_a^b f(x_1, x_2) dx_1 dx_2 \approx \sum_{i=0}^N \sum_{j=0}^N w_i w_j y_{i,j} \quad (1)$$

This is implemented in the program `Quad2D.cpp`, which can be downloaded from

<https://www.niallmadden.ie/2425-CS319/lab6/Quad2D.cpp> Note that the quadrature weights are specified in `main()`, and the `Quad2D()` function uses them.

## 2. 1. Trapezium Rule in 2D

The code in `Quad2D.cpp` applies the 2D Trapezium Rule to estimating

$$\int_0^1 \int_0^1 e^{x_1+x_2} dx_1 dx_2, \quad (2)$$

with  $N = 16$  intervals in each coordinate direction.

1. Adapt that code in `01CheckConvergence.cpp` from Week 6 (or Lab 5) so that the 2D Trapezium Rule is used to estimate the solution to (2) for various values of  $N$ , and the results are output as a NumPy array that can be copied into a Jupyter notebook.
2. Tip: make sure you allocate and de-allocate memory for `x1`, `x2`, and `y` within the `for`-loop that applies the method for various values of  $N$ .
3. Adapt the Jupyter notebook at <https://www.niallmadden.ie/2425-CS319/lab6/CS319-Lab6-Q1.ipynb> to verify the convergence of the method.

## 2. 1. Trapezium Rule in 2D

4. Further modify the code so that it reports how much time a call to the 2D Trapezium function takes for a given  $N$ . (See code from [Lab1-Q3.cpp](#) from Lab 1 for an example of how to do this).
5. Have the C++ code output the times for each  $N$  as a numpy array. Again, copy that into the Jupyter Notebook. Determine  $K$  and  $r$  where the time taken, for a given  $N$  is

$$t(N) \approx KN^r.$$



## 3. 2: Simpson's Rule in 2D

Based on our study of the Trapezium Rule, it is relatively easy to derive the 2D Simpson's Rule.

Recall first the 1D version:

$$\int_a^b f(x)dx \approx \sum_{i=0}^N w_i y_i.$$

where

$$w_0 = w_N = \frac{1}{3}h,$$

$$w_1 = w_3 = \cdots = w_{N-1} = \frac{4}{3}h,$$

$$w_2 = w_4 = \cdots = w_{N-2} = \frac{2}{3}h,$$

Now the method is exactly the same as in (1), except using these new weights.

### ASSIGNMENT

- (a) Write a C++ program that implements Simpson's Rule for estimating the solution to (2).
- (b) Your code should report the error, and time taken, for various values of  $N$ .
- (c) Use a Jupyter notebook to determine how the error, and time taken, depends on  $N$ .
- (d) Upload both your C++ code, and the Jupyter notebook (as PDF or HTML). Make sure all files contain, at the very least, your name and ID number. However, you should document your C++ code based on the verbal feedback from Lab 4.

## 5. Extra: Monte Carlo Methods

The approach given here is useful:

- ▶ We have a general framework for implementing any 1D method.
- ▶ We have a general framework for extending any method from 1D to 2D.
- ▶ In fact, this approach works in the “obvious” way if we extend to three or more dimensions...
- ▶ ... However, it does get very slow. This is known as the **“curse of dimensionality”**. For example, if we taken  $N = 1,000$  in one dimensional, we have to compute 1,000,000 values of  $f$  in two dimensions. In general, the time scales like  $N^d$  in  $d$  dimensions.

**Monte Carlo** methods are a useful way of estimating high dimensional integrals. They are fast, if not especially accurate...

## 5. Extra: Monte Carlo Methods

### Monte Carlo Algorithm

- ▶ Choose some value of  $N$  and compute an  $(N + 1) \times (N + 1)$  array,  $z$ , of values of  $f$ , as we did in previous methods.
- ▶ Choose some value  $M$  such that  $M \ll N^2$ . Usually, taking  $M = N$  is OK.
- ▶ Compute

$$Q_M := \frac{V}{M} \sum_{k=0}^M z_{i_k} j_k,$$

where  $i_k$  and  $j_k$  are randomly chosen integers in the range  $[0, N]$ . (That is, for each  $k$ , choose a random  $i_k$  and  $j_k$ ).

Try implementing this method, and comparing with the results for our other methods, in terms of both accuracy and time. You don't have to submit your work for this part.