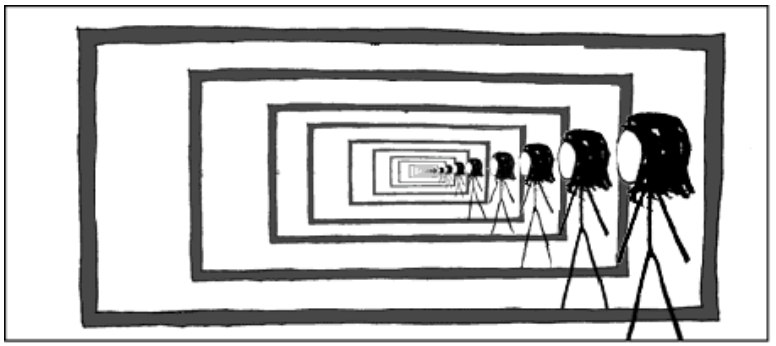
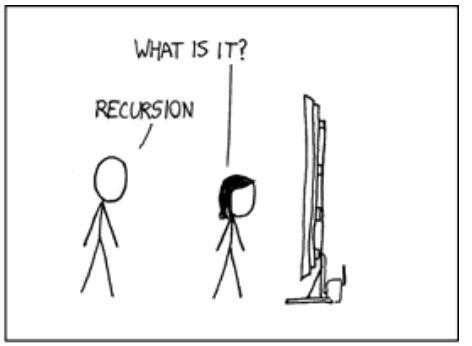
CS319: Scientific Computing

Functions and Quadrature

Dr Niall Madden

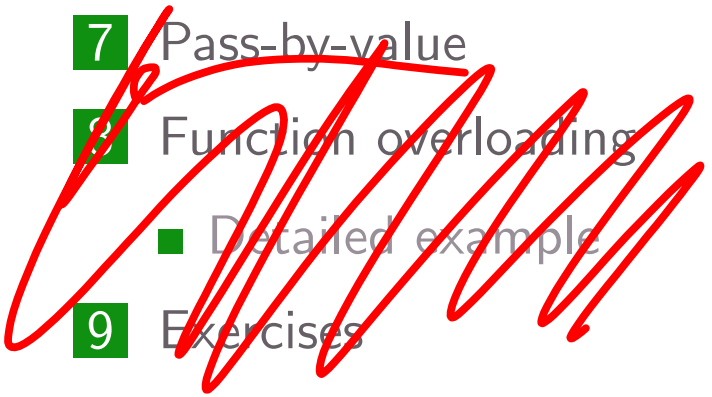
Week 4: 4th and 6th, February, 2026





Slides and examples: https://www.niallmadden.ie/2526-CS319

0. Outline



Slides and examples:
https://www.niallmadden.ie/2526-CS319

1. Overview of this week's classes

Last week, we began studying the use of **functions** in C++.

We'll motivate some of this study with a key topic in Scientific Computing: **Quadrature**, which is also known as **Numerical Integration**. The concept was briefly introduced at the end of last Friday's class.

We'll also use this as an opportunity to study the idea of **experimental analysis** of algorithms.

. . . . . .

However, we are still studying how to write functions in C++. For each new concept, we'll write a new version of *QuadratureV0x.cpp*.

2. Numerical Integration (again)

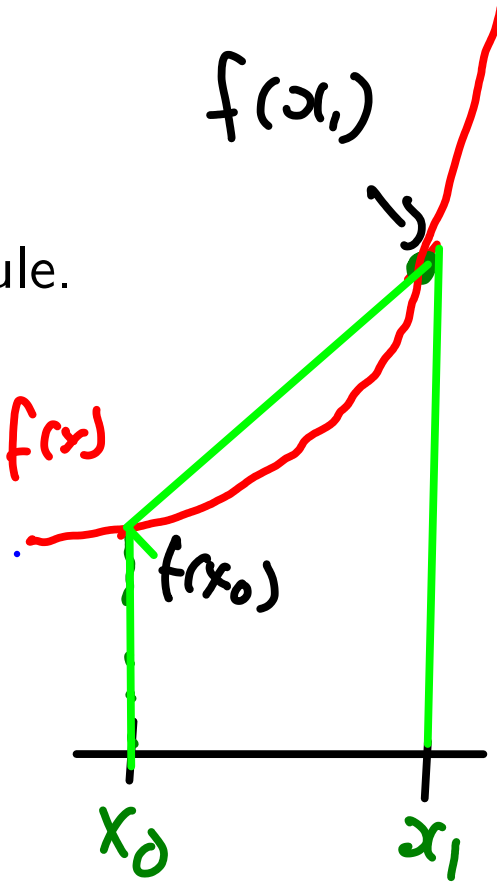
We finished Week 3 by starting to develop an algorithm for estimating definite integrates of one-dimensional functions:

$$\int_a^b f(x)dx,$$

using one of the simplest methods: the Trapezium Rule.

Specifically, we learned that one can approximate

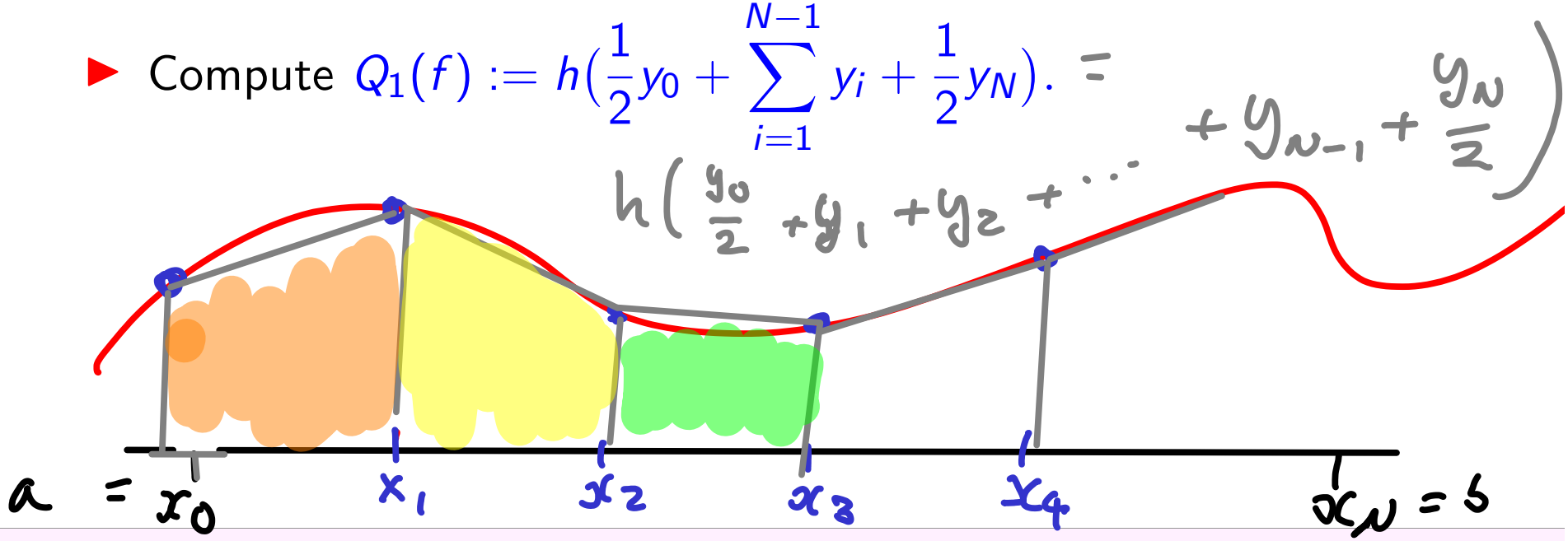
$$\int_{x_0}^{x_1} f(x)df \approx \frac{1}{2}(x_1 - x_0)\big(f(x_0) + f(x_1)\big).$$



The can be extended to give the following algorithm.

The Trapezium Rule

- Choose the number of intervals N, and set $h = (b - a)/N$.
- Define the quadrature points $x_0 = a$, $x_1 = a + h$, $\ldots x_N = b$. In general, $x_i = a + ih$.
- Set $y_i = f(x_i)$ for $i = 0, 1, \ldots, N$.
- Compute $Q_1(f) := h(\frac{1}{2}y_0 + \sum_{i=1}^{N-1} y_i + \frac{1}{2}y_N)$. $= h(\frac{y_0}{2} + y_1 + y_2 + \cdots + y_{N-1} + \frac{y_N}{2})$



QuadratureV01.cpp (headers)

```
// QuadrateureV01.cpp:
// Trapezium Rule (TR) quadrature for a 1D function
// Author: Niall Madden
// Date: Feb 2026
// Week 04: CS319 - Scientific Computing
#include <iostream>
#include <cmath>   // For exp()

double f(double); // prototype
double f(double x) {  return(exp(x)); } // definition
```

$$f(x) = e^x.$$

QuadratureV01.cpp (main)

```
12 int main(void )
   {
14   std::cout << "Using the TR to integrate f(x)=exp(x)\n";
     std::cout << "Integrate f(x) between x=0 and x=1.\n";
16   double a=0.0, b=1.0;
     double Int_f_true = exp(1)-1;
18   std::cout << "Enter value of N for the Trap Rule: ";
     int N;
20   std::cin >> N; // Lazy! Should do input checking.
```

$f(x) = e^x$

$$\int_0^1 f(x)\,dx = e^x\Big|_0^1 = e^1 - e^0 = e - 1.$$

QuadratureV01.cpp (main continued)

```
22   double h=(b-a)/double(N);
     double Int_f_TR = (h/2.0)*f(a);
24   for (int i=1; i<N; i++)
       Int_f_TR += h*f(a+i*h);
26   Int_f_TR += (h/2.0)*f(b);

28   double error = fabs(Int_f_true - Int_f_TR);

30   std::cout << "N=" << N << ", Trap Rule=" << Int_f_TR
                << ", error=" << error << std::endl;
32   return(0);
}
```

note we cast N as a double

23: $Int\_f\_TR = \frac{h}{2} f(a)$

24-25 : $Int\_f\_TR = Int\_f\_TR + h f(x_i)$

Typical output:

```
N=2,   Trap Rule=1.75393, error=0.0356493   = 3.56e-2
N=4,   Trap Rule=1.72722, error=0.00894008  = 8.94e-3
N=8,   Trap Rule=1.72052, error=0.00223676  = 2.23e-3
N=16, Trap Rule=1.71884, error=0.0005593    = 5.59e-4
N=32, Trap Rule=1.71842, error=0.00013983   = 1.39e-4
```

As N increases, the error decreases.
And it seems that the error is proportional to N^{-2}.

4. V02: Trapezium Rule as a function

Next it makes sense to write a **function** that implements the Trapezium Rule, so that it can be used in different settings.

The idea is pretty simple:

- As before, `f` will be a globally defined function.
- We write a function that takes as arguments `a`, `b` and `N`.
- The function implements the Trapezium Rule for these values, and the globally defined `f`.

4. V02: Trapezium Rule as a function

QuadratureV02.cpp (header)

```
// QuadrateureV02.cpp: Trapezium Rule as a function
// Trapezium Rule (TR) quadrature for a 1D function
// Author: Niall Madden
// Date: Feb 2026
// Week 04: CS319 - Scientific Computing
#include <iostream>
#include <cmath>   // For exp()
#include <iomanip>

double f(double x) {  return(exp(x)); } // definition
double TrapRule(double a, double b, int N);
```

New!

4. V02: Trapezium Rule as a function

QuadratureV02.cpp (main)

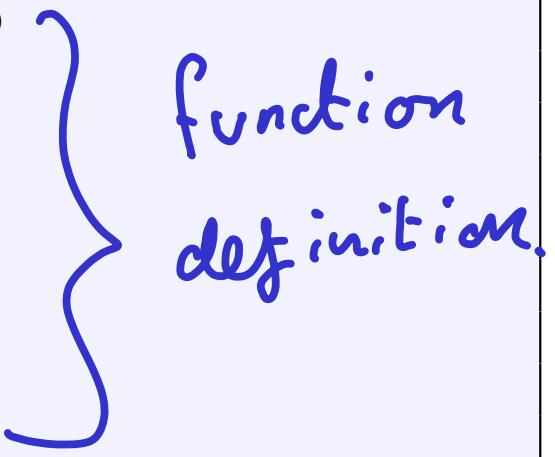
```
int main(void )
{
  std::cout << "Using the TR to integrate in 1D\n";
  std::cout << "Integrate between x=0 and x=1.\n";
  double a=0.0, b=1.0;
  double Int_true_f = exp(1)-1; // for f(x)=exp(x)

  std::cout << "Enter value of N for the Trap Rule: ";
  int N;
  std::cin >> N; // Lazy! Should do input checking.

  double Int_TR_f = TrapRule(a,b,N);
  double error_f = fabs(Int_true_f - Int_TR_f);

  std::cout << "N=" << std::setw(6) << N <<
    ", Trap Rule=" << std::setprecision(6) <<
    Int_TR_f << ", error=" <<  std::scientific <<
    error_f << std::endl;
  return(0);
```

call the function

4. V02: Trapezium Rule as a function

QuadratureV02.cpp (function)

```
34 double TrapRule(double a, double b, int N)
   {
36   double h=(b-a)/double(N);
     double QFn = (h/2.0)*f(a);
38   for (int i=1; i<N; i++)
       QFn += h*f(a+i*h);
40   QFn += (h/2.0)*f(b);
     return(QFn);
42 }
```



5. V03: Functions as arguments to functions

We now have a function that implements the Trapezium Rule. However, it is rather limited, in several respects. This includes that the function, `f`, is hard-coded in the `TrapRule` function. If we want to change it, we'd edit the code, and recompile it.

Fortunately, it is relatively easy to give the name of one function as an argument to another.

The following example shows how it can be done.

5. V03: Functions as arguments to functions

QuadratureV03.cpp (header)

```
// QuadrateureV03.cpp: Trapezium Rule as a function
// that takes a function as argument
// Week 04: CS319 - Scientific Computing
#include <iostream>
#include <cmath>   // For exp()
#include <iomanip>

double f(double x) {  return(exp(x)); } // definition
double g(double x) {  return(6*x*x); } // definition

double TrapRule(double Fn(double), double a, double b,
                int N);
```

New: function to be integrated

Here "Fn" is a place-holder

5. V03: Functions as arguments to functions

QuadratureV03.cpp (part of main())

```
20    std::cout << "Which shall we integrate: \n"
                << "\t 1. f(x)=exp(x) \n\t 2. g(x)=6*x^2?\n";
22    int choice;
      std::cin >> choice;
24    while (!(choice == 1 || choice  == 2) )
      {
26      std::cout << "You entered " << choice
                  <<". Please enter 1 or 2: ";
28      std::cin >> choice;
      }
30    double Int_TR=-1; // good place-holder
      if (choice == 1)
32      Int_TR = TrapRule(f,a,b,10);
      else
34      Int_TR = TrapRule(g,a,b,10);

36    std::cout << "N=10" << ", Trap Rule="
                << std::setprecision(6) << Int_TR  << std::endl;
38    return(0);
}
```

5. V03: Functions as arguments to functions

QuadratureV03.cpp (TrapRule())

```
42 double TrapRule(double Fn(double), double a,
                  double b, int N)
44 {
     double h=(b-a)/double(N);
46   double QFn = (h/2.0)*Fn(a);
     for (int i=1; i<N; i++)
48     QFn += h*Fn(a+i*h);
     QFn += (h/2.0)*Fn(b);

     return(QFn);
52 }
```

Fn is a function that takes a double as input & returns a double

6. V04: Functions with default arguments

In our previous example, we wrote a function with the header

```
double TrapRule(double Fn(double), double a, double b,
       int  N);
```

And then we called it as

```
Int_TR = TrapRule(f,a,b,10);
```

That is, when we were not particularly interested in the value of `N`, we took it to be 10.

It is easy to adjust the function so that, for example, if we called the function as

```
Int_TR = TrapRule(f,a,b);
```

it would just be assumed that $N = 10$. All we have to do is adjust the function header.

6. V04: Functions with default arguments

To do, this we specify the value of N in the **function prototype**. You can see this in QuadratureV04.cpp. In particular, note Line 10:

QuadratureV04.cpp (line 10)

```
10 double TrapRule(double Fn(double), double a,
                   double b, int N=10); // default N=10
```

This means that, if the user does not specify a value of N, then it is taken that $N = 10$.

6. V04: Functions with default arguments

Important:

- You can specify default values for as many arguments as you like. For example:

```
double TrapRule(double Fn(double), double a=0.0,
        double b=1.0, int N=10);
```

- If you specify a default value for an argument, you must specify it for any following arguments. For example, the following would cause an error.

```
double TrapRule(double Fn(double), double a=0.0,
        double b=1.0, int N);
```

So these are valid (and the same):
Q1 =TrapRule(f); Q1=TrapRule(f,0); Q1=TrapRule(f,0,1)
and Q1=TrapRule(f,0,1,10);

6. V04: Functions with default arguments

Important:

- You can specify default values for as many arguments as you like. For example:

```
double TrapRule(double Fn(double), double a=0.0,
                double b=1.0, int N=10);
```

- If you specify a default value for an argument, you must specify it for any following arguments. For example, the following would cause an error.

```
double TrapRule(double Fn(double), double a=0.0,
                double b=1.0, int N);
```

Finished here Wednesday.