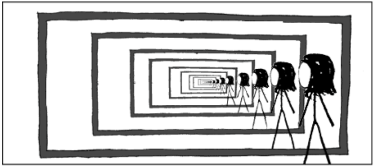
Annotated slides from 4pm class

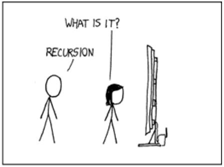
CS319: Scientific Computing

Quadrature, and Functions in C++

Dr Niall Madden

Week 4: **9am and 4pm** 31 January, 2024





Slides and examples: https://www.niallmadden.ie/2324-CS319

Outline

Class times

9am: Section 1, 2, and 3.1

1. Overview of this week's classes
   - Why quadrature?
2. Functions (again)
   - Header
   - Function definition
   - E.g, Prime?
   - `void` functions
3. Numerical Integration
   - The basic idea
   - The code

     Start here 4pm
   - Trapezium Rule as a function
4. Functions as arguments to functions
5. Functions with default arguments
6. Pass-by-value
7. Function overloading
8. Detailed example

Slides and examples:
https://www.niallmadden.ie/2324-CS319

02QuadratureV01.cpp (headers)

```
// 02QuadrateureV01.cpp:
// Trapezium Rule (TR) quadrature for a 1D function
// Author: Niall Madden
// Date: 31 Jan 2024
// Week 04: CS319 - Scientific Computing
#include <iostream>
#include <cmath>   // For exp()

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

f is the function we will integrate.

Numerical Integration

The code

02QuadratureV01.cpp (main)

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

TR = Trapezium Rule

$$\int_0^1 e^x dx = e^x\Big|_0^1 = e^1 - e^0 \text{ "="} \text{ exp(1.0) - 1.0;}$$

Numerical Integration

02QuadratureV01.cpp (main continued)

```
22   double h=(b-a)/double(N);
     double Int_f_TR = (h/2.0)*f(0.0);
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);
   }
```

f(a)

~ $\frac{h}{2} f(x_0)$

22: N is an int, so we write (b-a)/double(N) so that it is temporarily converted to a floating point number. "Casting".

02QuadratureV01.cpp (main continued)

```
22    double h=(b-a)/double(N);
      double Int_f_TR = (h/2.0)*f(0.0);
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);
    }
```

Line 25: note that $x_i = a + ih$

So this is $\sum_{i=1}^{n-1} h f(x_i)$.

Typical output:

```
N=10, Trap Rule=1.71971, error=0.00143166
N=20, Trap Rule=1.71864, error=0.00035796
N=40, Trap Rule=1.71837, error=8.94929e-05
```

It appears that, as N increases, the error decreases.

So, it works!!

Numerical Integration — 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`~~. N.
- The function implements the Trapezium Rule for these values, and the globally defined `f`.

03QuadratureV02.cpp(header)

```
   // 03QuadrateureV03.cpp: Trapezium Rule as a function
 2 // Trapezium Rule (TR) quadrature for a 1D function
   // Author: Niall Madden
 4 // Date: 31 Jan Feb 2024
   // Week 04: CS319 - Scientific Computing
 6 #include <iostream>
   #include <cmath>   // For exp()
 8 #include <iomanip> // for, e.g., std::setprecision

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

Line 10: we've combined header & definition.

03QuadratureV02.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);
```

Trapezium Rule as a function

03QuadratureV02.cpp(function)

```cpp
double TrapRule(double a, double b, int N)
{
  double h=(b-a)/double(N);
  double QFn = (h/2.0)*f(0.0);
  for (int i=1; i<N; i++)
    QFn += h*f(a+i*h);
  QFn += (h/2.0)*f(b);
  return(QFn);
}
```

a

compare with lines 22-26 in V01.

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.

Functions as arguments to functions

04QuadratureV04.cpp(header)

```
// 04QuadrateureV03.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);
```

define f and g

Here "Fn" is a place-holder.

However, the 1st argument to TrapRule():
must be a function, must take a double as input,
and must return a double.

Functions as arguments to functions

04QuadratureV04.cpp (part of main())

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

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

\t = "tab"

not

OR

N=10;

Functions as arguments to functions

04QuadratureV04.cpp (TrapRule())

```
double TrapRule(double Fn(double), double a,
                double b, int N)
{
  double h=(b-a)/double(N); a
  double QFn = (h/2.0)*Fn(~~0.0~~);
  for (int i=1; i<N; i++)
    QFn += h*Fn(a+i*h);
  QFn += (h/2.0)*Fn(b);

  return(QFn);
}
```

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

N
↓

```
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 definition of 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 first ~~link~~ line in the function definition.

header/prototype.

Functions with default arguments

To do, this we specify the value of N in the **function prototype**. You can see this in `05QuadratureV04.cpp`. In particular, note Line 10:

`05QuadratureV04.cpp` (line 10)

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

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

More precisely, if I don't specify the 4th argument, it is taken to be 10. (The name of that value, N, is just a place-holder, but is required).

Functions with default arguments

Important:

Finished here **31/1/2024** at 5pm

- 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);
```

✗

Since a & b have default values, so too must N.