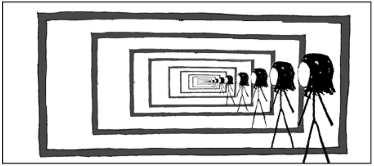
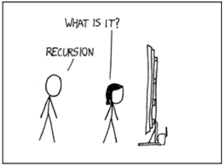
CS319: Scientific Computing

Functions and Quadrature

Dr Niall Madden

Week 4: 5th and 7th, February, 2025





Slides and examples: https://www.niallmadden.ie/2425-CS319

Outline

Slides and examples:
https://www.niallmadden.ie/2425-CS319

Overview of this week's classes

This week, we will study the use of **functions** in C++, which we started (very briefly) at the end of Week 3.

In Scientific Computing, we use the term "**function**" in two different, but related ways:

- A mathematical function, such as $f(x) = e^{-x}$ or $u(x, y) = \sin(\pi x) \cos y$.
- A function we code to preform a task, such a determining if an integer is positive. Or, often, working with mathematical functions: to calculate derivatives at a point (Lab 2) or integrals.

And often we'll combine both ideas!

However, we'll motivate some of this study with a key topic in Scientific Computing: **Quadrature**, which is also known as **Numerical Integration**.

Later, we'll use this as an opportunity to study the idea of **experimental analysis** of algorithms.

- A **Quadrature** method, in one dimension, is a method for estimating definite integrals. The applications are far too numerous to list, but feature in just about every area of Applied Mathematics, Probability Theory, and Engineering, and even some areas of pure mathematics.
- They are methods for estimating integrals of functions. So this gives us two reasons to code functions:
  (i) As the functions we want to integrate;
  (ii) As the algorithms for doing the integration.

But before we get on to actual methods, we'll learn the basics of writing functions in C++.

Functions

A good understanding of **functions**, and their uses, is of prime importance.

Some functions return/compute a single value. However, many important functions return more than one value, or modify one of its own arguments.

For that reason, we need to understand the difference between **call-by-value** and **call-by-reference** ($\longleftarrow$ later).

Functions

Every C++ program has at least one function: `main()`

Example

```
#include <iostream>
int main(void )
{
   /* Stuff goes here */
   return(0);
}
```

Each function consists of two main parts: Header/Prototype and Body/Definition.

1. Header

The Function "header" or **prototype** gives the function's

- return value data type, or `void` if there is none, and
- parameter list data types or `void` if there are none.
- The header line ends with a semicolon.

The prototype is often given near the start of the file, before the **main()** section.

Syntax for function header:

```
ReturnType FnName (type1, type2, ...);
```

Examples:

int compute_lcm(int a, int b); // compute lowest common multiple of a and b

float convert_temp(float f); // convert between temp scales.

bool ComplicatedFunction(float f, double c, int x, int y, char ccc);

Other examples include passing a function as an argument.

----
but not
float, int ComputeTwoThings(int a, int b);
-----

2. Function definition

- The **function definition** can be anywhere in the code (after the header). *(but not inside another function, such as main).*
- First line is the same as the prototype, except variables names need to be included, and that line does not end with a semi-colon.
- That is followed by the body of the function contained within curly brackets.

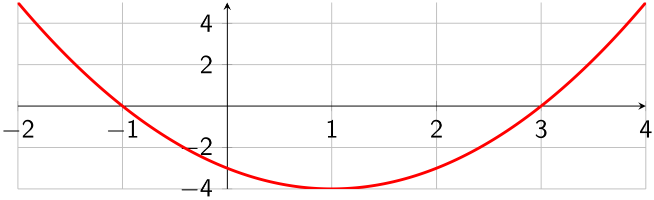
Syntax:

```
ReturnType FnName (type1 param1, type2 param2, ...)
{
    statements
}
```

- `ReturnType` is the data type of the data returned by the function.
- `FnName` the identifier by which the function is called.
- `type1 param1, ...` consists of
  - the data type of the parameter *(e.g., int, double, etc)*
  - the name of the parameter will have in the function. It acts within the function as a local variable. *(name=identifier)*
- the statements that form the function's body, contained with braces `{...}`.

Since this is a course on scientific computing, we'll often need to define mathematical functions from $\mathbb{R} \to \mathbb{R}$ (more or less), such as $f(x) = e^{-x}$. Typically, such functions map one or more `double`s onto another `double`.

The example we'll look at is $f(x) = x^2 - 2x - 3$.



00MathFunction.cpp

```
#include <iostream>
#include <iomanip>

double f(double x) // x^2 - 2x - 3
{
  return (x*x - 2*x -3);
}

int main(void){
  double x;
  std::cout << std::fixed << std::showpoint;
  std::cout << std::setprecision(2);
  for (int i=0; i<=10; i++)
  {
    x = -1.0 + i*.5;
    std::cout << "f("<< x << ")="<< f(x) << std::endl;
  }
  return(0);
}
```

Functions — E.g, Prime?

In this example, we write a function that takes an non-negative integer input and checks it its a composite (`true`) or prime (`false`).

`01IsComposite.cpp` (header)

```cpp
// 01IsComposite.cpp
// An example of a simple function.

#include <iostream>

bool IsComposite(int i);
```

Calling the `IsComposite` function

`01IsComposite.cpp` (main)

```
int main(void )
{
  int i;

  std::cout << "Enter a natural number: ";
  std::cin >> i;

  std::cout << i << " is a " <<
    (IsComposite(i) ? "composite":"prime") << " number."
            << std::endl;

  return(0);
}
```

Notice use of ?: operator.

Defining the `IsComposite` function

`01IsComposite.cpp` (function definition)

```
bool IsComposite(int i)
{
  int k;
  for (k=2; k<i; k++)
    if ( (i%k) == 0)
      return(true);

  return(false); // If we get to here, i has no divisors between 2 and i-1
}
```

Function ends first time a "return" is encountered.

Most functions will return some value. In rare situations, they don't, and so have a `void` return value.

`02Kth.cpp` (header)

```
// 02Kth.cpp:
// Another example of a simple function.
// Author: Niall Madden
// Date: 05 Feb 2025
// Week 04: CS319 - Scientific Computing
#include <iostream>
void Kth(int i);
```

Puzzle: What is the next term in the sequence:

s, t, n, d, r, d, t, h, t, h, t, h, ...

02Kth.cpp (main)

```
int main(void )
{
  int i;

  std::cout << "Enter a natural number: ";
  std::cin >> i;

  std::cout << "That is the ";
  Kth(i);
  std::cout << " number." << std::endl;

  return(0);
}
```

Sequence is 1st, 2nd, 3rd, 4th, 5th, 6th, 7th, 8th, 9th, 10th
11th, 12th, ... 19th, 20th,
21st, 22nd, ...

02Kth.cpp (function definition)

```
// FUNCTION KTH
// ARGUMENT: single integer
// RETURN VALUE: void (does not return a value)
// WHAT: if input is 1, displays 1st, if input is 2, displays 2nd,
// etc.
void  Kth(int i)
{
  std::cout << i;
  i = i%100;
  if ( ((i%10) == 1) && (i != 11))
    std::cout << "st";
  else if ( ((i%10) == 2) && (i != 12))
    std::cout << "nd";
  else if ( ((i%10) == 3) && (i != 13))
    std::cout << "rd";
  else
    std::cout << "th";
}
```

remainder on dividing by 100.

$\to i = 1, 21, 31, \ldots$

Numerical Integration

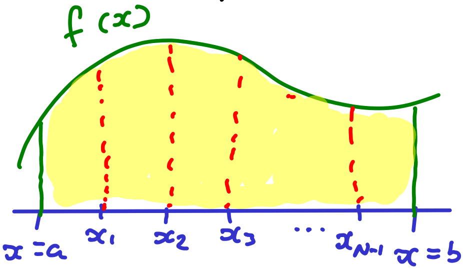
Numerical integration is an important topic in scientific computing. Although the history is ancient, it continues to be a hot topic of research, particularly when computing with high-dimensional data.

In this section, we want to estimate definite integrates of one-dimensional functions:

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

We'll use one of the simplest methods: the Trapezium Rule.

$\int_a^b f(x)\,dx$ is "area bounded above by $f(x)$, if $f>0$, and the x-axis, as well as between $x=a$, $x=b$.



- Choose an int N.
- Set $h = \dfrac{(b-a)}{N}$
- Let $x_i := a + ih$ for $i = 0, 1, \dots, N$.

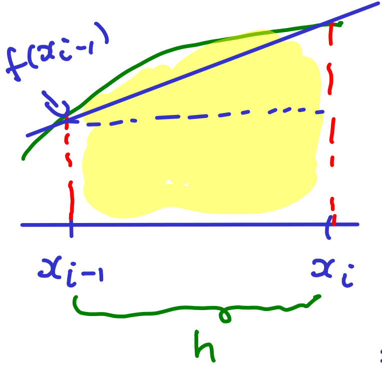
So $x_0 = a$ .. $x_N = b$

$x_1 = a+h$

$x_2 = a+2h$

We know $\int_a^b f(x)\,dx = \int_{x_0}^{x_1} f(x)\,dx + \int_{x_1}^{x_2} f(x)\,dx + \cdots + \int_{x_{N-1}}^{x_N} f(x)\,dx.$

Also



$$\int_{x_{i-1}}^{x_i} f(x)\,dx \simeq h\,f(x_{i-1}) + \frac{h}{2}\left(f(x_i) - f(x_{i-1})\right)$$

$$= \frac{h}{2}\left[f(x_{i-1}) + f(x_i)\right]$$

Consequently

$$\int_a^b f(x)\,dx \simeq \frac{h}{2} f(x_0) + h f(x_1) + h f(x_2) + \cdots + \frac{h}{2} f(x_N).$$

03QuadratureV01.cpp (headers)

```
// 03QuadrateureV01.cpp:
// Trapezium Rule (TR) quadrature for a 1D function
// Author: Niall Madden
// Date: 06 Feb 2025
// Week 04: CS319 - Scientific Computing
#include <iostream>
#include <cmath>   // For exp()

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

03QuadratureV01.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.
```

03QuadratureV01.cpp (main continued)

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

  double error = fabs(Int_f_true - Int_f_TR);

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

$x_i = a + ih.$

Typical output:

Finished here
Wed

N	Error
2	3.565×10^{-2}
4	8.9×10^{-3}
8	2.2×10^{-3}
16	5.5×10^{-4}

It works!