Numerical Methods

Some example applications in C++

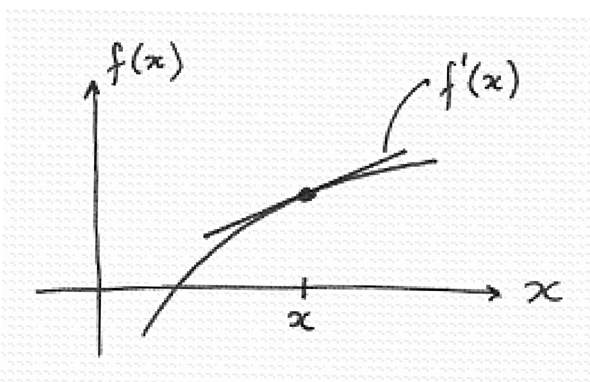
Introduction

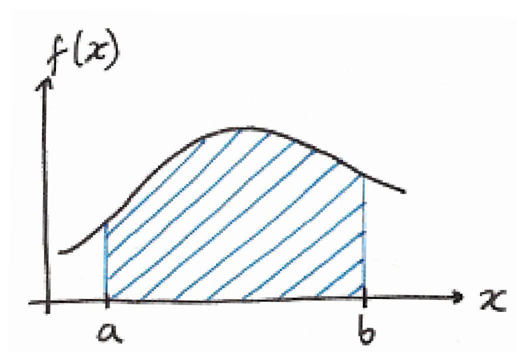
Numerical methods apply algorithms that use *numerical* approximations to solve mathematical problems.

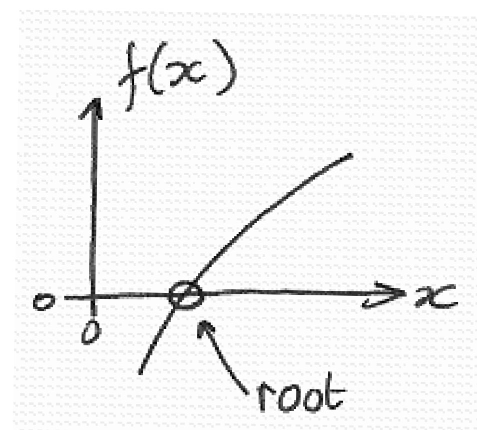
This is in contrast to applying *symbolic analytical* solutions, for example *Calculus*.

We will look at very basic, but useful *numerical* algorithms for:

1. Differentiation
2. Integration
3. Root finding







Taylor's Expansion

Key to the formulation of numerical techniques for differentiation, integration and root finding is Taylor's expansion:

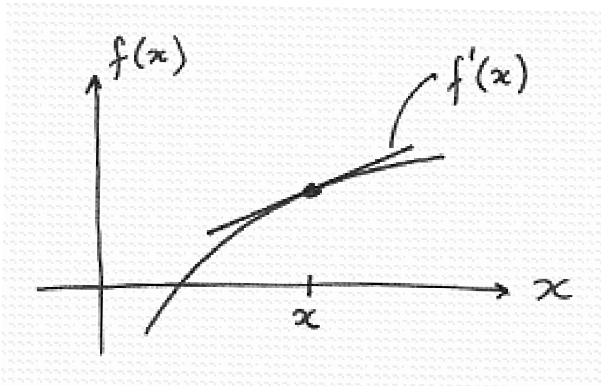
$$f(x+h) = f(x) + \frac{h^1}{1!} f'(x) + \frac{h^2}{2!} f''(x) + \frac{h^3}{3!} f'''(x) + \ldots$$

The value of a function at $x + h$ is given in terms of the values of derivatives of the function at x

The general idea is to use a small number of terms in this series to approximate a solution.

In some cases we can improve on the solution by iterating the procedure $\Rightarrow$ ideal task for a computer.

1. Numerical differentiation

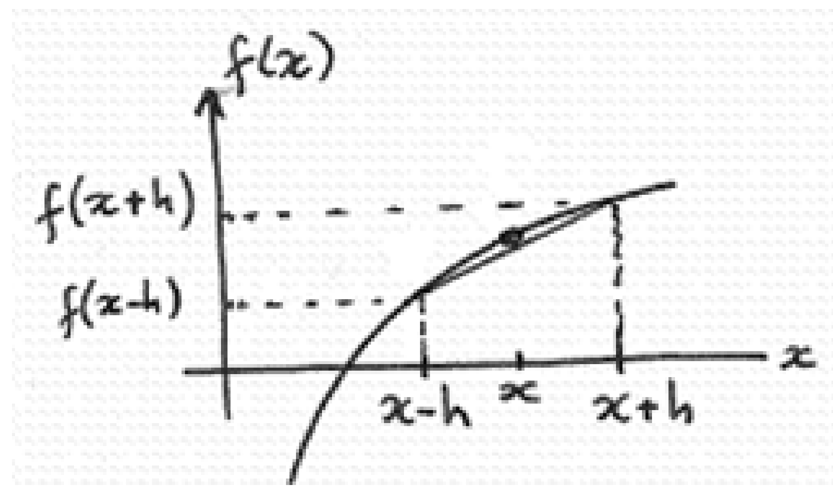


Aim

Given a function *f(x)*, we wish to calculate the derivative *f'(x)*; that is, the gradient of the function at *x*.

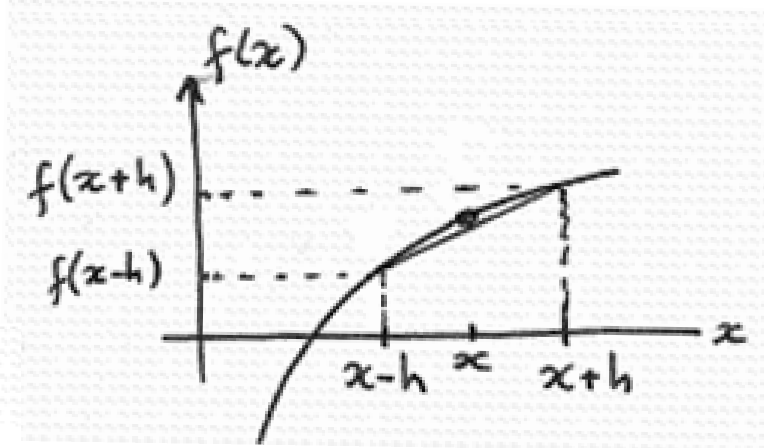
The Central Difference Approximation, CDA, provides an approximation to this gradient:

$$\mathrm{CDA} = \frac{f(x+h) - f(x-h)}{2h} \simeq f'(x)$$



Proof

$$CDA = \frac{f(x+h) - f(x-h)}{2h} \approx f'(x)$$



Proof:

Taylor's expansion,

$$f(x+h) = f(x) + \frac{hf'(x)}{1!} + \frac{h^2 f''(x)}{2!} + \frac{h^3 f'''(x)}{3!} + \dots$$

$$f(x-h) = f(x) - \frac{hf'(x)}{1!} + \frac{h^2 f''(x)}{2!} - \frac{h^3 f'''(x)}{3!} + \dots$$

$$\Rightarrow CDA = f'(x) + \frac{h^2}{6} f'''(x) + O(h^4)$$

i.e $CDA \approx f'(x)$

the error $\approx \frac{h^2}{6} f'''(x)$.

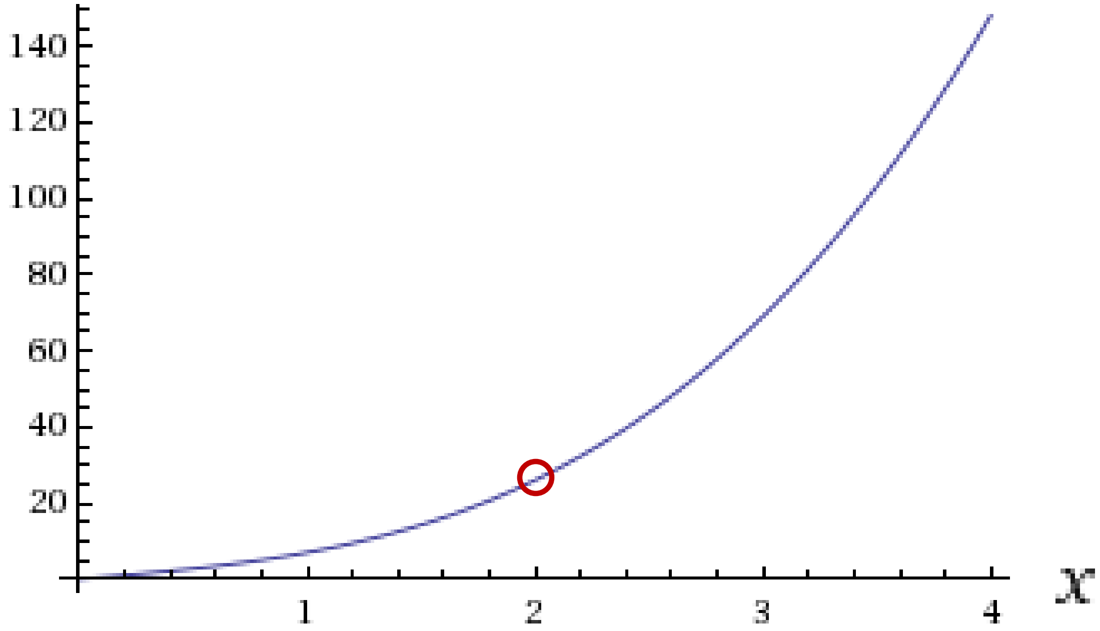
The approximation improves as the size of *h* reduces.

Limited precision in the computer prevents us from making *h* very small!

Problem

For the following function, calculate the derivative at $x = 2$

$$f(x) = 2x^3 + 5x$$



Algorithm

1. Define the function:

$$f(x) = 2\,x^3 + 5\,x$$

2. Set the parameters:

x = 2, h = 0.01

3 Calculate the CDA:

$$CDA = \frac{f(x+h) - f(x-h)}{2h}$$

4 Output the result.

C++ code

```
// Central-Difference Approximation (CDA)
// for the derivative of a function f(x).
// Here, f(x)=2*x^3+5*x, h=0.01, x=2.0.

#include <iostream>
using namespace std;

double f(double x) { return 2*x*x*x + 5*x; }

int main() {

  double x=2.0, h=0.01;
  double cda = (f(x+h)-f(x-h))/(2*h);
  cout << "f'(" << x << ") = " << cda << endl;

}
```

$f(x) = 2\,x^3 + 5\,x$

Output

```
f'(2) = 29.0002
```

Verification

The program gives us `f'(2) = 29.0002`

We can verify that this is what we expect:

From Calculus
$f(x) = 2x^3 + 5x$
$f'(x) = 6x^2 + 5$
$f''(x) = 12x$
$f'''(x) = 12$

The function here is $f(x) = 2x^3 + 5x$
From calculus we can obtain $f'(x) = 6x^2 + 5$
and so the exact solution for $f'(2)$ is $6*2^2 + 5 = \underline{29.0000}$

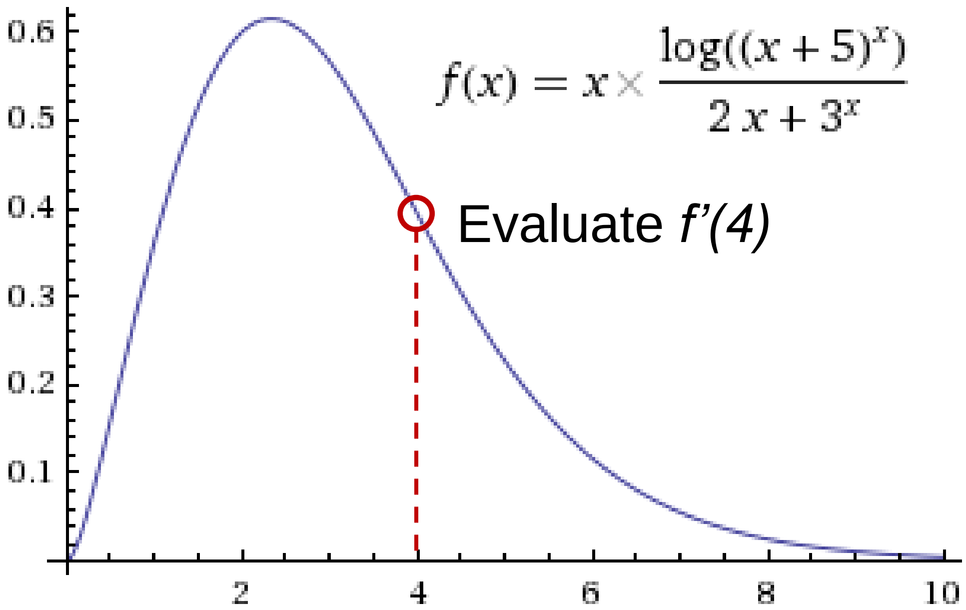
We see that the error in the CDA is $29.0002 - 29.0000 = \underline{0.0002}$

From analysis of Taylor's expansion we predict the error in the CDA as $\approx h^2 f'''(x)/6$
$= 0.01^2.12/6 = \underline{0.0002}$

Our algorithm is working as predicted.

A more difficult problem

So far the CDA does not look so useful, we have only solved a trivial problem. Let's try a more difficult function:



Analytical solution

$$f'(x) = \frac{(2x + 3^x)x^2 + (x+5)(2x+3^x)x\log(x+5) - 3^x(x+5)(x\log(3)-1)\log((x+5)^x)}{(x+5)(2x+3^x)^2}$$

$$\approx -0.1863498$$

Adapt the C++ code for the new calculation

```
// Central-Difference Approximation (CDA)
// for the derivative of a function f(x).
#include <iostream>
#include <cmath>
using namespace std;

double f(double x) {
  return x*log(pow(x+5,x))/(2*x+pow(3,x));
}

int main() {

  double x=4.0, h=0.01;
  double cda = (f(x+h)-f(x-h))/(2*h);
  cout << "f'(" << x << ") = " << cda << endl;

}
```

Output

```
f'(4) = -0.186348
```

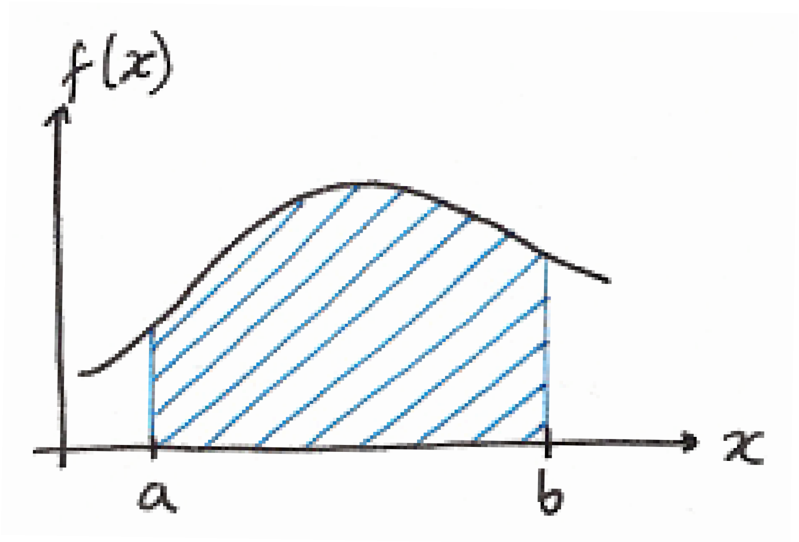
The error is +0.000002

2. Numerical integration

Aim

We wish to perform numerically the following integral:

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



This is simply the area under the curve *f(x)* between *a* and *b*.

For example, $\int_2^4 \left(5\,x + 2\,x^3\right) d\,x = 150$

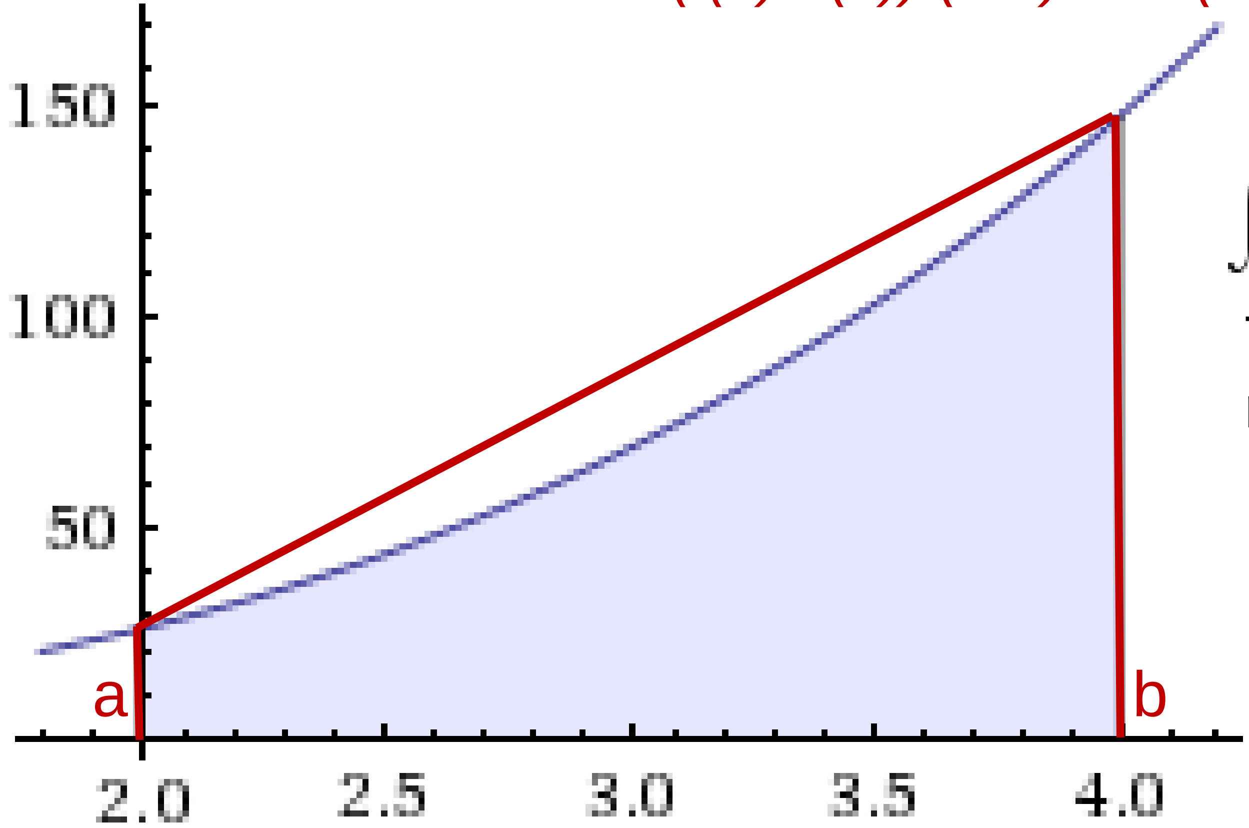
How can we perform this numerically?

Formulating an algorithm

A first approximation can be obtained by forming a trapezoid.

$$f(x) = 2x^3 + 5x$$

Trapezoid area
= ½ (f(2)+f(4)) (4-2) = ½ (26+148) (2) = ***174***.



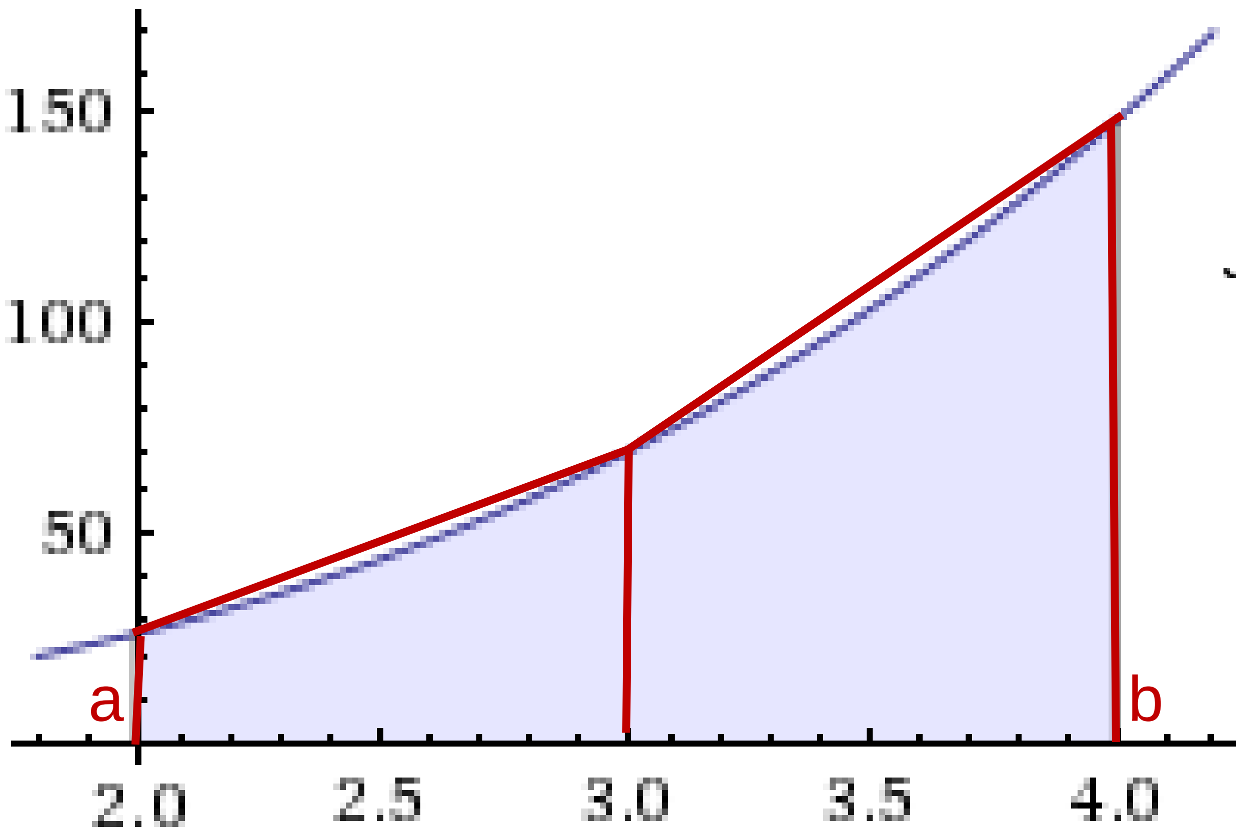
$$\int_2^4 (5x + 2x^3)\, dx = 150$$

The error in the result is16%

An improved approximation can be obtained by forming
<u>two</u> trapezoids.

Trapezoid area
= ½ (f(2)+f(3)) (3-2) + ½ (f(3)+f(4)) (4-3) = ***156***

$$f(x) = 2x^3 + 5x$$



$$\int_2^4 (5x + 2x^3)\, dx = 150$$

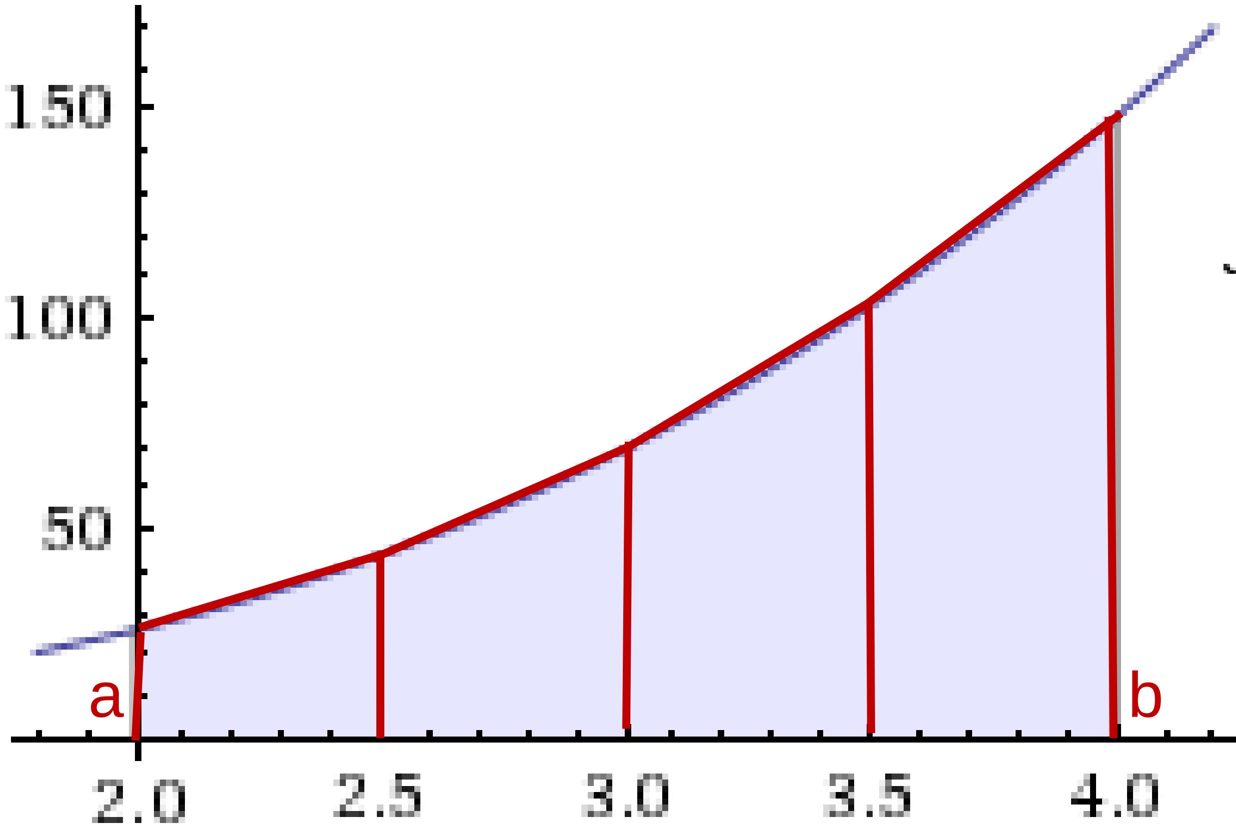
The error in the result is 4%

Four trapezoids.

Trapezoid area

= ½ (f(2.0)+f(2.5)) (2.5-2.0) + ½ (f(2.5)+f(3.0)) (3.0-2.5)
+ ½ (f(3.0)+f(3.5)) (3.5-3.0) + ½ (f(3.5)+f(4.0)) (4.0-3.5)
*= **151.5***

$$f(x) = 2x^3 + 5x$$



$$\int_2^4 (5x + 2x^3)\,dx = 150$$

The error in the result is 1%

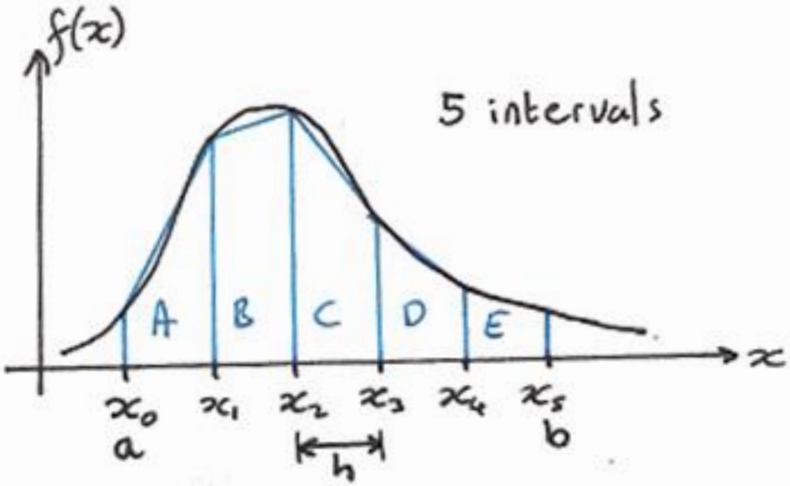
The error $\propto 1/n^2$ where n is the number of trapezoids.

Formulating an algorithm

Generalising the procedure:

We want the integral $\int_a^b f(x)\,dx$.

First consider approximating with five trapezoids:



$$A = \frac{h}{2}(f(x_0)+f(x_1))$$
$$B = \frac{h}{2}(f(x_1)+f(x_2))$$
$$C = \frac{h}{2}(f(x_2)+f(x_3))$$
$$D = \frac{h}{2}(f(x_3)+f(x_4))$$
$$E = \frac{h}{2}(f(x_4)+f(x_5))$$

$$h = \frac{x_5 - x_0}{5} = \frac{b-a}{5}$$

let $f_i \equiv f(x_i)$

$A+B+C+D+E$ = Extended Trapezoidal Formula (ETF).

$$h(f_0/2 + f_1 + f_2 + f_3 + f_4 + f_5/2)$$

For n intervals

$$ETF = h\left(\frac{f_0}{2} + f_1 + f_2 + f_3 + \dots + f_{n-1} + \frac{f_n}{2}\right)$$

with $h = \frac{b-a}{n}$ $\quad x_i = a + ih, \; i = 0,1,2,\dots,n$

Algorithm

1. *Define the function:* $f(x) = 2\,x^3 + 5\,x$

2. *Set the limits of the integral, and the number of trapezoids:*

   a = 2, b = 4, n = 100

3. *Set* $h = \frac{b-a}{n}$

4. *Calculate the ETF as*

$$ETF = h\left(\frac{f_0}{2} + f_1 + f_2 + f_3 + \cdots + f_{n-1} + \frac{f_n}{2}\right)$$

with $f_i = f(x_i)$ $x_i = a + ih$, $i = 0, 1, 2, \cdots, n$

5. *Output the result.*

C++ code

$$ETF = h\left(\frac{f_0}{2} + f_1 + f_2 + f_3 + \cdots + f_{n-1} + \frac{f_n}{2}\right)$$

with $f_i = f(x_i)$ $x_i = a + ih$, $i = 0, 1, 2, \cdots, n$

$$h = \frac{b-a}{n}$$

```cpp
// Numerical integration via the Extended
// Trapezoidal Formula (ETF)
#include <iostream>
using namespace std;

double f(double x) { return 2*x*x*x + 5*x; }

int main() {

  double a=2.0, b=4.0;
  int n=100;
  double h = (b-a)/n;

  double etf = (f(a)+f(b))/2;
  for (int i=1; i<n; i++) etf = etf + f(a+i*h);

  etf = etf * h;

  cout << "The integral = " << etf << endl;

}
```

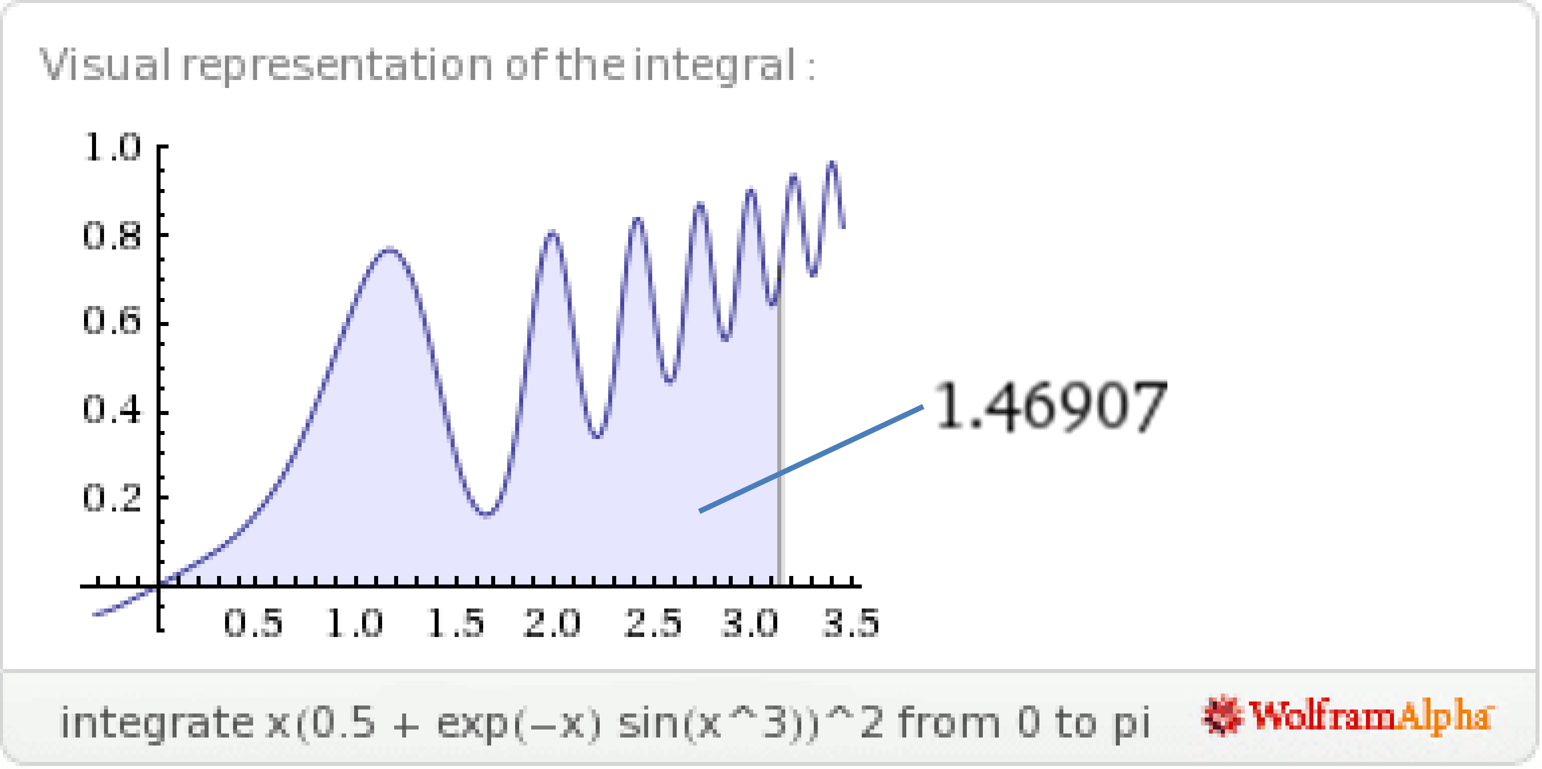
Output

```
The integral = 150.002
```

Error = 0.002

A more difficult problem

$$\int_0^{\pi} x\left(\frac{1}{2}+e^{-x}\sin(x^3)\right)^2 dx$$



Adapt the previous C++ code

```
#include <iostream>
#include <cmath>
using namespace std;

double f(double x) {
  return x * pow(0.5+exp(-x)*sin(x*x*x),2); }

int main() {

  double a=0.0, b=M_PI;
  int n=100;
  double h = (b-a)/n;

  double etf = (f(a)+f(b))/2;
  for (int i=1; i<n; i++) etf = etf + f(a+i*h);

  etf = etf * h;

  cout << "The integral = " << etf << endl;

}
```

Output

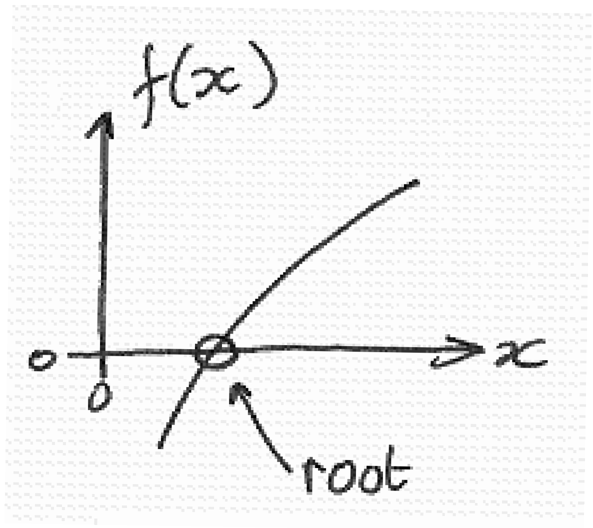
```
The integral = 1.46937
```

The error is +0.00030

3. Root finding

Aim

We wish to find the root x_0 of the function $f(x)$; i.e. $f(x_0) = 0$.



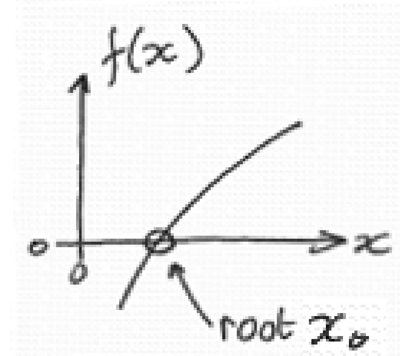
How can we perform this numerically?

There are many ways to do this.

We will implement the Newton-Raphson method....

Formulating an algorithm

let x_0 be the root of a function $f(x)$
ie $f(x_0)=0$. let x be an estimate
of x_0, and ε be the error in this estimate;
i.e $\varepsilon = x - x_0$ or $x_0 = x - \varepsilon$.



If we can obtain a good estimate of ε
then we can improve our root estimate

$$x_{i+1} = x_i - \varepsilon_i$$ iteratively.

Obtaining an error estimate:

Taylor's expansion:

$$0 = f(x_0) = f(x - \varepsilon)$$

$$= f(x) - \frac{\varepsilon f'(x)}{1!} + \frac{\varepsilon^2 f''(x)}{2!} - \frac{\varepsilon^3 f'''(x)}{3!} + \cdots$$

dropping $O(\varepsilon^2)$ terms gives

$$0 \approx f(x) - \varepsilon f'(x)$$

$$\Rightarrow \quad \varepsilon \approx f(x)/f'(x)$$

Then

$$\boxed{x_{i+1} = x_i - f(x_i)/f'(x_i)}$$

d(x)

Newton-Raphson iterative formula for the root of $f(x)$.

The algorithm so far:

1. *define f(x) and d(x)*
2. *Initialise x*
3. *Iterate:*

   e = f(x)/d(x)

   x = x – e
4. *Output x*

But how many iterations?

We have an estimate of the error $\varepsilon \approx f(x) / f'(x)$

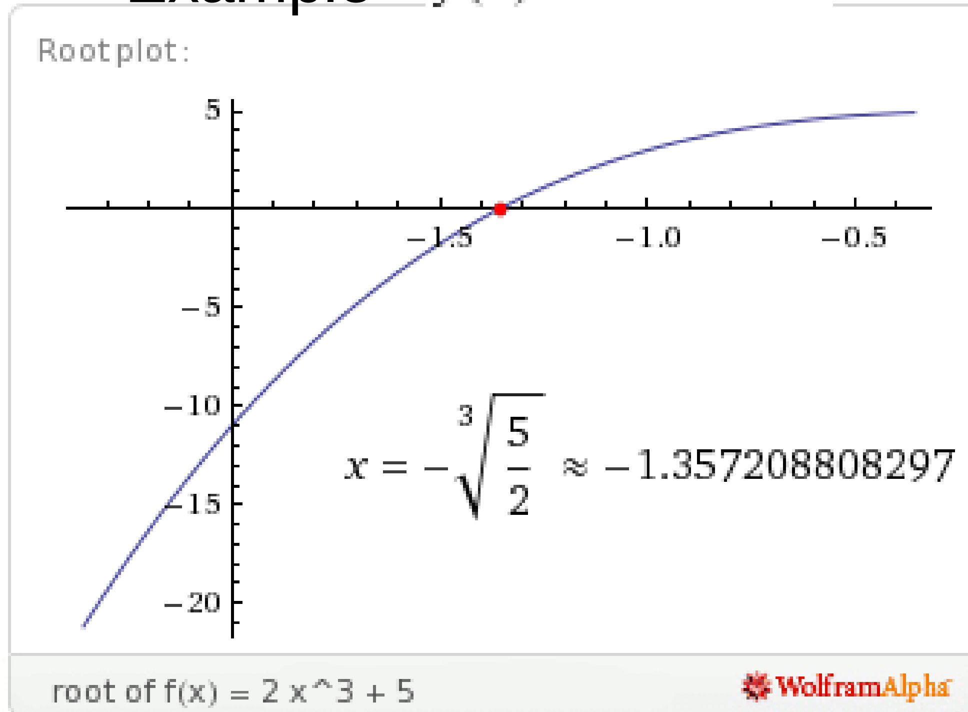
Use this to form a termination condition that requires 6 decimal place accuracy:

“ *iterate until* $\boldsymbol{\varepsilon} < 10^{-9}$ ”

Algorithm

1. *define f(x) and d(x)*
2. *initialise x*
3. *iterate:*
   e = f(x)/d(x)
   if $\boldsymbol{\varepsilon} < 10^{-9}$ *terminate*
   x = x – e
4. *Output x*

Example $f(x) = 2\,x^3 + 5$



C++ code

$$f(x) = 2x^3 + 5$$
$$f'(x) = 6x^2$$

```
// Newton-Raphson method for the root of f(x)
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

double f(double x) { return 2*x*x*x + 5; }
double d(double x) { return 6*x*x; }

int main() {

  cout << setprecision(9) << fixed;
  double e, x = -1.5;

  while (true) {
    e = f(x)/d(x);
    cout << "x = " << x << endl;
    if (fabs(e)<1.0e-6) break;
    x = x - e;
  }

}
```

Output

```
x = -1.500000000
x = -1.370370370
x = -1.357334812
x = -1.357208820
```

The number of correct digits *doubles* on every iteration (rapid convergence)!

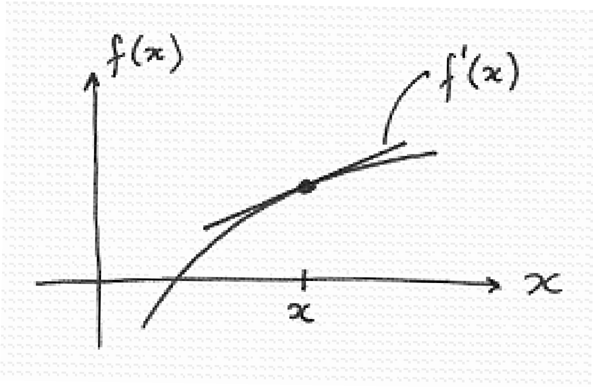
7 decimal place accuracy

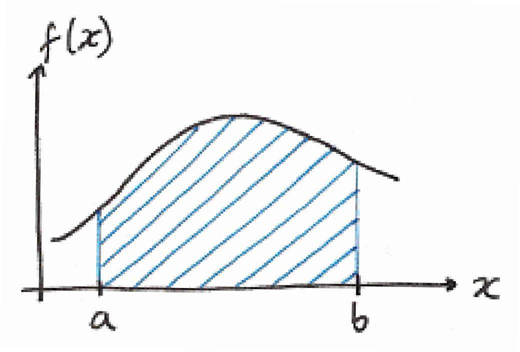
$$f(x) = 2x^3 + 5$$

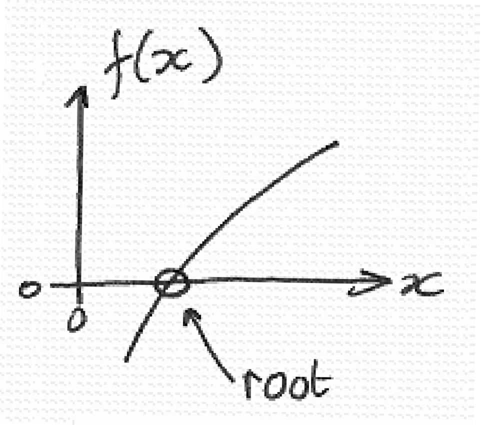
$$x = -\sqrt[3]{\frac{5}{2}} \approx \boxed{-1.357\,208\,8}08\,297$$

Finally

In this lecture we have looked at *Numerical Methods*.







More about numerical methods can be found at:

http://en.wikipedia.org/wiki/Numerical_methods