CS319: Scientific Computing (with C++)

CS319 Lab 3: Optimization (and functions in C++)

Week 5 (13-14 Feb, 2025)

Goal: You will develop an algorithm for solving an optimisation problem. This will give a context in which to study some of the intricacies of functions in C++, including

- global and local variables;
- pass-by-reference;
- functions as arguments to other functions.

Unfortunately, it is quite difficult to plot functions in C++. But that gives us an opportunity to do a little Python coding too.

You don't have to submit your work this week: we'll develop the idea some more next week to include a Newton-type method for this problem.

1. Optimization

"**Optimisation**" is the process of finding a maximum or minimum value of some function. It is one of the most important, and challenging, problems in computational science. If anything, it is more important than ever, since it is the engine that drives the latest methods in deep learning and generative AI.

For the purposes of this lab, "**optimisation**" means finding the point at which a given function achieves its maximum value. Typical optimization problems you might be familiar with include

- at what speed does my car have the best fuel efficiency?
- what is the maximum height a ball will reach if thrown with particular initial velocity?

The problems can be posed in a mathematical framework:

- We'll take a given function, f, which we call the **objective function**.
- We find the value of m that maximises f in a given interval, $[a, b]$. That is, find m such that $a \leq m \leq b$, and $f(m) \geq f(x)$ for all $x \in [a, b]$.

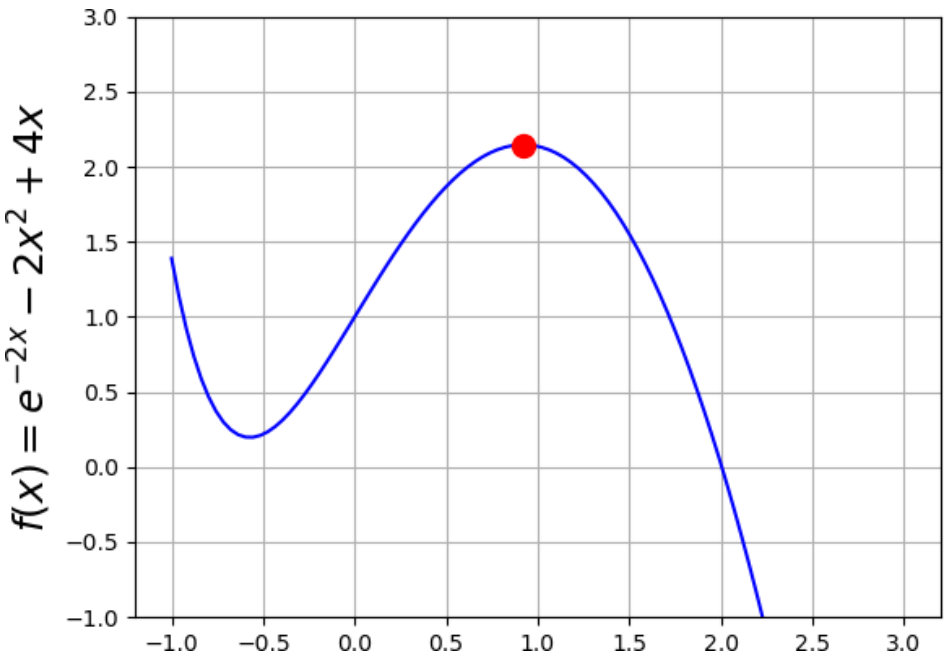
1. Optimization

As a specific example, we'll try to maximise

$$f(x) = e^{-2x} - 2x^2 + 4x$$

in the interval $[-1, 3]$, as shown below. The solution is

$$m = (e^{2x} - 1/2)e^{-2x} \approx 0.9207028302185.$$



2. A little Python

As we progress through CS319, we'll get more adept at switching between C++ and Python.

In particular, we'll use Python for visualisation. The figure on the previous slide was generated using a few lines of Python. Here is how to do it:

- Connect the https://cloudjupyter.universityofgalway.ie and log in using the details sent to you by maths-sto@universityofgalway.ie earlier this semester.
- Start a new notebook using the 2324-CS319 kernel.
- Enter the following code.

PlotF.py

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-1, 3, 100) # values of x to use
f = lambda x: np.exp(-2*x) - 2*x**2 + 4*x
m = 0.9207; # value of x which gives maximum (to digits)

plt.plot(x, f(x), 'b-', m,f(m),'ro', markersize=10)
plt.ylim(-1,3)
plt.grid(visible=True)
plt.ylabel('$f(x)=e^{-2x} - 2x^2 + 4x$', fontsize=18)
```

3. Bisection

Today we'll solve the optimisation problem (in C++) using a *bisection* algorithm:

1. Choose points a and b (where $a < b$) such that the maximum of f in between a and b.
2. Take c to be the midpoint of a and b.
3. Take l to be the midpoint of the left interval, $[a, c]$, and r to be the midpoint of the right interval, $[c, b]$.
4. Compare $f(l)$, $f(c)$ and $f(r)$, and
   - if $f(c)$ is largest set $a = l$, and $b = r$.
   - if $f(l)$ is largest keep a, and set $b = c$.
   - if $f(r)$ is largest keep b, and set $a = c$.
5. Repeat the algorithm until $b - a$ is less than some prescribed tolerance.

The first step is illustrated on the next slide. In this case, since $f(c) > f(l) > f(r)$, for the second step we will set $a = l$ and $b = r$.

3. Bisection

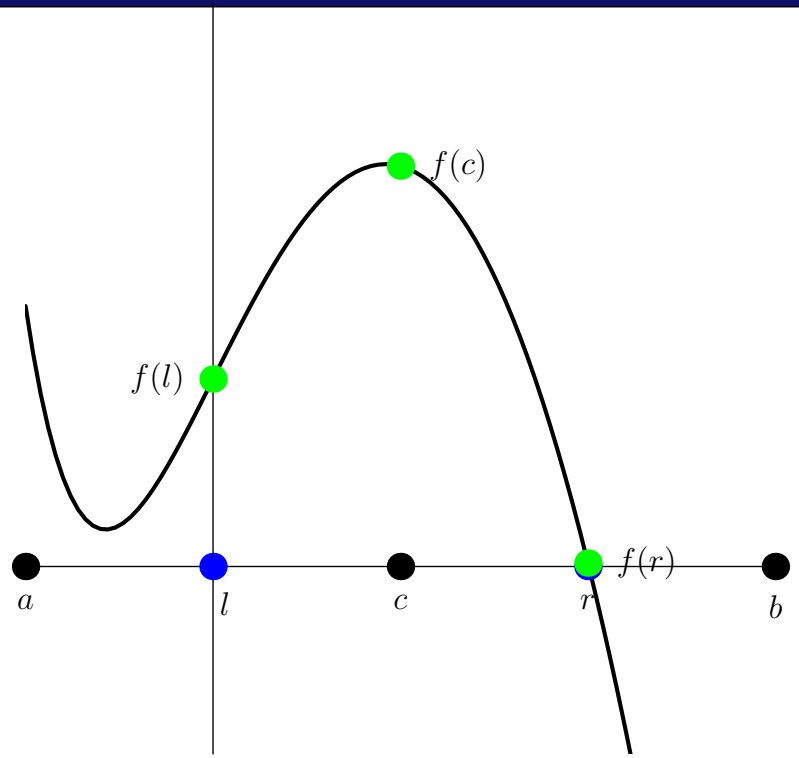


Illustration of the first step of the algorithm

3. Bisection

There is a basic implementation of this at:
https://www.niallmadden.ie/2324-CS319/lab3/Bisection.cpp.

Bisection.cpp

```
38 double Bisection(double a, double b)
   {
40   double c = (a+b)/2.0;
     while ( (b-a) > 1e-6)
42   {
       c = (a+b)/2.0;
44     double l = (a+c)/2.0, r=(c+b)/2.0;

46     if ( (f(c) > f(l)) && (f(c) > f(r)) )
       {
48       a=l;
         b=r;
50     }
       else if ( f(l) > f(r) )
52       b=c;
       else
54       a=c;
     }
56   return(c);
   }
```

3. Bisection

This implementation of the bisection algorithm has several shortcoming. Your task is to make as many of the following improvements as possible.

(a) In C++ a variable is **local** to the function, or code block, where it is defined. However, if the variable is defined outside of any function, then it is **global**, and shared by all functions. (It is very rare that the use of global variables is good practice; it reduces modularity and complicates code re-use. We are using one here just so that we know how they work). The termination criterion is that the interval containing the maximum is less that 10^{-6}. Different applications may require different bounds. Introduce a **global** variable for this bound, the value of which the user can choose.

3. Bisection

(b) To verify the efficiency of an optimisation algorithms, we need to record some basic statistics of its operation, such as the number of iterations (i.e., steps through the loop) taken.
Modify your code so that a variable which records the number of iterations is passed (by reference) the bisection function. This should be reported by the `main()` function. What data type will you use for this new variable? Why?
(Note: typically, the most expensive part of the computation is the function call to `f()`. A better approach would be to count the number of times that is evaluated).

(c) It is possible that the `while` loop in the `Bisection()` function does not terminate. To avoid an infinite loop, add an argument to the `bisection()` function that controls the maximum number of iterations the algorithm will take.

- In your code, it should have a default value (in the argument list) of 10;
- The user should be prompted for their preferred value;
- A suitable warning message should be generated if the convergence is not achieved.

(d) The program provided with this lab finds the maximum of the objective function $f(x) = e^{-2x} - 2x^2 + 4x$ in the interval $[-1, 3]$. To make our `Bisection()` implementation more general, we would like to pass the name of the objective function to be optimised as one of the parameters. As we saw in Week 4, this is reasonably easy in C++. Just add something like the following to the parameter list:

```
double ObjFn(double)
```

Then :

- you give the name of the objective function you would like optimised as an argument to the bisection function;
- the objective function must take a single `double` as its argument and return a double.

Make this change to your code. Verify it works by finding the maximum of two different objective functions.