CS319: Scientific Computing (with MATLAB)

CS319 Lab 2: Functions and Optimization

Weeks 4+5, 2022

Niall Madden (Niall.Madden@UniversityOfGalway.ie)

Goal: You will develop an algorithm for solving an optimisation problem. This will give a context in which to study some of aspects of coding functions in MATLAB, including

- anonymous functions;
- "file" functions;
- multiple return values.

Submit your code on Blackboard (Labs... Lab 2) by 17:00, Friday 10 Feb. Your program file(s) should include your name, ID number, email address, and a short description of how the code works.
Collaboration is encouraged. Include the name and email address of any person you collaborated with on the assignment, and a statement of what each of you contributed.

1. Optimization

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 $x \in [a, b]$.

2. Anonymous function

As a specific example, we'll try to maximise

$$f(x) = e^{-2x} - 2x^2 + 4x, \tag{1}$$

in the interval $[-1, 2]$, as shown below. The solution is

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

The function is plotted on $[-1, 2]$ in Figure 1, which was generated using the following code. Note the definition of the *anonymous* function, and use of the "dot" operator to compute x^2 as `x.^2`.

Define and plot an anonymous function

```
f = @(x)(exp(-2*x)-2*x.^2 + 4*x);
fplot(f, [-1, 2], 'LineWidth', 2);
grid on;
xlabel('x');
ylabel('e^{-2x}-2x^2 +4x')
```

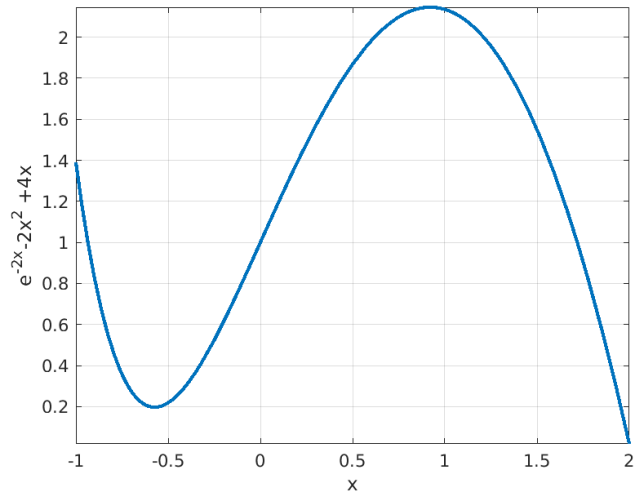

Figure 1: Plot of $f(x)$, using `fplot`

3. Bisection

We'll first solve this problem 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

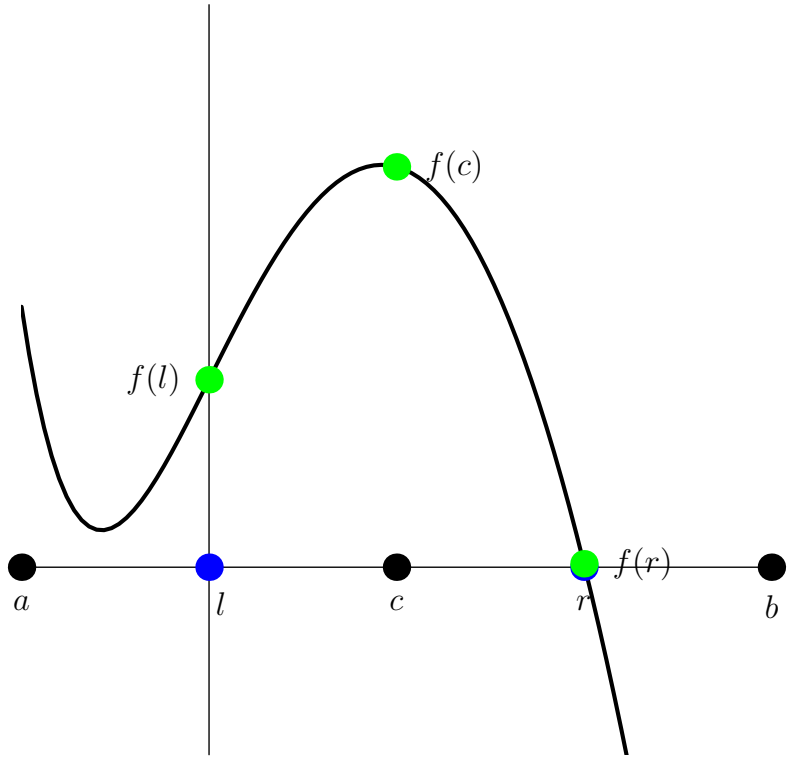

Illustration of the first step of the algorithm

3. Bisection

The following function implements the algorithm in a rather basic way. You'll find it in the file *Bisection.m* on Blackboard and on the bitbucket repository. Test it for the function f defined in (1).

Bisection.m

```
function c = Bisection(ObjFn, a, b)

while ( (b-a) > 1e-6)
    c = (a+b)/2.0; % center
    l = (a+c)/2.0; % left
    r = (c+b)/2.0; % right

    if ( (ObjFn(c) > ObjFn(l)) && (ObjFn(c) > ObjFn(r)) )
        a=l;
        b=r;
    elseif ( ObjFn(l) > ObjFn(r) )
        b=c;
    else
        a=c;
    end
end
```

3. Bisection

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

Q1. In MATLAB, each function has its own **workspace**. That means that a (file) function know only about the variables passed to it. However, a variable can be is defined as `global`, in which case it will belong to all workspaces. (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 `Bisection` function stops iterating when the interval containing the maximum has width 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.

Do this by writing a short script which defines and plots the function, f, using the code snippet on Slide 3.
The script should call the `Bisection` function, and also output (using *`fprintf`*) the value of x at which f is maximised.

Now, somewhere before the `Bisection` function is called, add the lines

3. Bisection

```
1    global TOL;
     TOL=1.0e-5;
```

Next, within the `Bisection.m` file, also add the line "`global TOL`". Finally, adjust the condition in the `while` loop so that it iterates until $(b - a) < TOL$.

3. Bisection

Q2. 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 the `Bisection` function as follows:

- The first line of code will be:

```
function [c, IterCount] = BisectionV02(ObjFn, a, b)
```

  This means that it will return two values: the maximizing value of x, and also the number of iterations taken.
- Set the variable *IterCount* to zero just before the start of the `while` loop. At each step of the `while` loop it should increment *IterCount* by 1.
- Modify the script that calls the `Bisection` function so that it takes two return values, e.g.,

```
1 [Max_x, IterationCount] = BisectionV02(f, -1, 2 );
```

  It should also output the value of `IterationCount`.

3. Bisection

Q3. It is possible that the `while` loop in the `Bisection` function does not terminate. To avoid an infinite loop, add a new parameter to the parameter list of the `Bisection()` function that controls the maximum number of iterations the algorithm will take.

- The user should be prompted for their preferred value;
- A suitable warning message should be generated if the convergence is not achieved.

Algorithm 2: Newton

The Bisection method just described is quite robust: providing that the function is continuous, it will find an approximation of its maximum in the desired interval. However, there are much faster methods, the most important being **Newton's Method for Optimisation:** choose an initial guess x_0, and set

$$x_{k+1} = x_k - f'(x_k)/f''(x_k) \quad \text{for} \quad k = 0, 1, 2, \ldots.$$

Implement this method. Note that it is different from Bisection in that one only provides a single initial guess, and also that we must provide both f' and f''.

Assignment

(i) Write an implementation of the **Newton Algorithm** as a "file" function. Like your `Bisection()` function, it should take the maximum number of iterations as an argument. It should iterate until the absolute value of the difference bewteen two sucessive iterations is less than the tolerance (which is a `global` variable), or the number of iterations exceeeds the maximum allowed.

(ii) Write a script file that tests both your `Bisection` and `Newton` optimisers for finding the local maximum of the function

$$g(x) = x + \sin(2x) \tag{2}$$

in the interval $[-1, 2]$, in the case of Bisection, and with an initial guess of $x_0 = 0.5$, in the case of Newton.

(iii) The script should prompt the user for the maximum number of iterations.

(iv) Having called both optimizers, the script should should output the estimates they compute, and the number of iterations the used.

Submit your **three** files to the "Lab 2" section of 2122-CS319 on Blackboard:

- One function file with the *`Bisection()`* function;
- One function file with the *`Newton()`* function;
- One script file that tests them both for the function g in (2).

Deadline: 17.00, Friday 10 Feb.