INTRODUCTION TO COMPUTATIONAL PHYSICS

FIRST COURSEWORK

U24200 –Academic Session 2020–2021

INSTRUCTIONS

a) This is worth **50%** of your total mark for this unit.

b) **You must undertake this assignment individually.**

c) Submission method: **by Moodle through the available dropbox**.

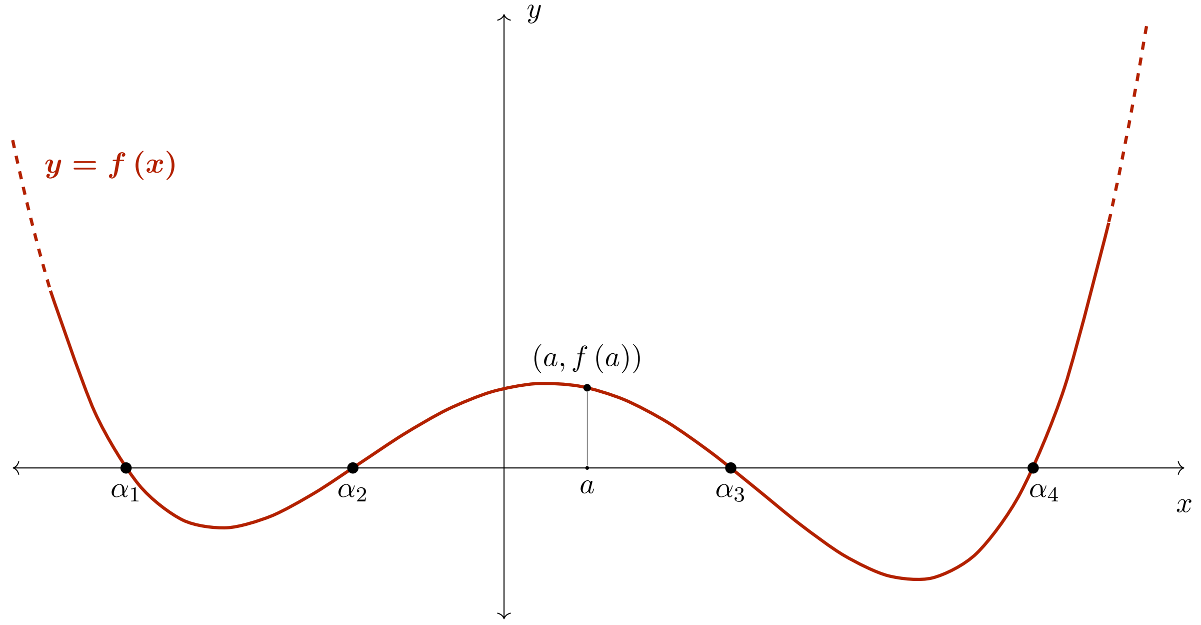
d) Submission deadline: **January 22, 2020, 4PM**

1 Introduction

Let $f(x)$ be a continuous function of a single variable. It can be:

- a simple function, e.g. a polynomial $f(x) = a_n x^n + \cdots + a_1 x + a_0$ (for instance $f(x) = x - 1$, $f(x) = 3x^2 + 4x - 5$, etc)
- or a more complicated function such as $f(x) = \cos x$, $f(x) = e^x$, ...

Having one real input x and one real output $y = f(x)$ for every input, this function can be represented by a two-dimensional graph plotting *abscissae* x against *ordinates* y:



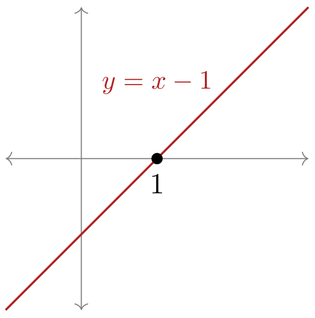
We have highlighted four values $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ in the above example. They are values of x such that $f(x) = 0$, i.e. the graph of f intersects the x-axis. We usually call such values **zeros** (or **roots**) of the function f, or **roots** (or simply *solutions*) of the equation $f(x) = 0$.

Two situations can arise, depending on the function $f(x)$ you are working with:

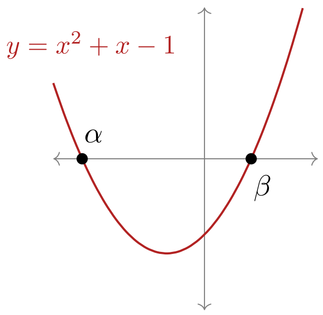
- sometimes finding or describing zeros of a function *symbolically* will be an easy task, e.g.
  - $f(x) = x - 1$ has only one root $\alpha = 1$,
  - $f(x) = x^2 + x - 1$ has two roots $\alpha = \frac{-\sqrt{5}-1}{2}$, $\beta = \frac{\sqrt{5}-1}{2}$ that are easy to find exactly using the *quadratic formula*,
  - $f(x) = \cos(x)$ has infinitely many roots but we know them: $\frac{(2k+1)\pi}{2}$ (k any integer),
  - and $f(x) = e^x$ has no zeros at all, because it is strictly positive on all values of x;

  by *symbolically* we mean roots we can represent exactly because we do not require decimal approximations to write them down (even if we may need approximations to *operate* with them in real-life scenarios, using $\sqrt{5} \simeq 2.23607\ldots$ and $\pi \simeq 3.14159\ldots$);

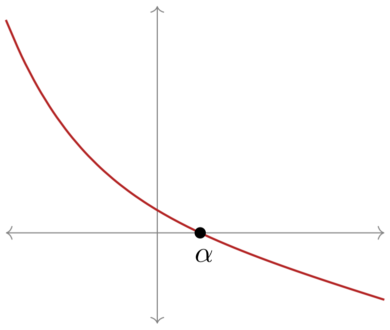
- most oftentimes, however, we may determine the *amount* of roots – but finding one, let alone all, symbolically will be either difficult or impossible. For instance, a thorough study of the roots of $e^{-x} = x$ (i.e. the zeros of function $f(x) = e^{-x} - x$) reveals that there is only one such zero α and it lies in the interval $(0.5, 0.6)$. But we cannot describe α in closed form, unlike the above examples; we cannot write it as an integer, a multiple of a known constant (e.g. π), a combination of square roots or simple functions of known values, etc. The only thing we can do is approximate α with a fixed amount of correct significant digits, e.g. $\alpha \simeq 0.5671432904097838$ with 16 correct decimal digits and $\alpha \simeq 0.56714329040978387299996866221035$ with 32 correct decimal digits.



$f(x) = 0$ equates to $x = 1$



Formula $\frac{-b \pm \sqrt{b^2-4ac}}{2a}$ applied to $ax^2 + bx + c$, $a = 1$, $b = 1$, $c = -1$



Function $f(x) = e^{-x} - x$ has only one zero α but we cannot find it exactly

Numerical solution of equations consists in approximating one or more roots of a given equation $f(x) = 0$ using certain *root-finding algorithms*. Some remarks are in order here:

- these algorithms will work regardless of whether or not we can find the roots symbolically, e.g. for $f(x) = x - 1$ we know the exact solution, $x = 1$, but we can also apply a root-finding algorithm if we want to; if implemented correctly, it will approximate

  $$x \simeq 1.\boxed{\text{large amount of 0's}}\boxed{\text{other digits}} \quad \text{or} \quad x \simeq 0.\boxed{\text{large amount of 9's}}\boxed{\text{other digits}}$$

  and the better the approximation, the more 0's or 9's after the decimal point.

- As said above, most functions will not afford us the luxury of a symbolic solution, and numerical root-finding algorithms to approximate it will be the only tools available.

- Before applying the root-finding algorithm, we need to know the *precision*, i.e. the number of correct digits that we desire, e.g. 16 in the first approximation of α for $e^{-x} = x$ above.

Most root-finding algorithms are **iterative** methods: they entail the creation of a sequence of numbers $\{x_k\}_{k\geq 0}$ such, that every new iteration (i.e. element of the sequence) x_k can be obtained as a fixed function of a fixed number of previous iterations:

$$\boxed{x_k = G\left(x_{k-1}, x_{k-2}, \ldots, x_{k-j}\right)} \qquad G \text{ and } j \text{ being fixed and depending on the method.} \tag{1}$$

This requires, as you will realise yourselves, a set of j initial conditions $x_0, \ldots, x_{j-1}$.

Assume you want to fix a precision; more specifically, assume you wish to approximate a root α with p correct decimal digits. You can do so by fixing a certain tolerance $\varepsilon = \frac{1}{2}10^{-p}$, which is what you will use as a criterion to stop your iteration process. Let us represent a few steps of this method:

$$\begin{aligned}
\boldsymbol{x_j} &= G\left(x_{j-1}, x_{j-2}, \ldots, x_0\right) \quad \text{(1st actual step of the method, using only initial conditions)}\\
\boldsymbol{x_{j+1}} &= G\left(\boldsymbol{x_j}, x_{j-1}, \ldots, x_1\right) \quad \text{(2nd step, using previous one } x_j \text{ and some initial conditions)}\\
\boldsymbol{x_{j+2}} &= G\left(\boldsymbol{x_{j+1}}, \boldsymbol{x_j}, x_{j-1}, \ldots, x_2\right) \quad \text{(3rd step)}\\
\vdots \quad & \vdots \quad \vdots\\
\boldsymbol{x_m} &= G\left(\boldsymbol{x_{m-1}}, \boldsymbol{x_m}, \ldots, \boldsymbol{x_{m-j}}\right) \quad (m\text{th step})\\
\vdots \quad & \vdots \quad \vdots
\end{aligned}$$

The process will stop at a step n where two consecutive iterations differ by less than ε:

$$|\boldsymbol{x_n} - \boldsymbol{x_{n-1}}| < \frac{1}{2}10^{-p}, \qquad \text{i.e. } \boldsymbol{x_n} \text{ and } \boldsymbol{x_{n-1}} \text{ share their first } p \text{ decimal digits.} \tag{2}$$

Obviously, the challenge here is finding a function G, depending on the original function f, such that the sequence generated above converges to the root we are looking for: $\lim_{n\to\infty} \boldsymbol{x_n} = \alpha$. Once G is established (and we will not explain the mathematical reasoning behind this now), **condition (2) ensures, in most cases, that the last iteration $\boldsymbol{x_n}$ in our method will share p correct decimal digits with α as well**.

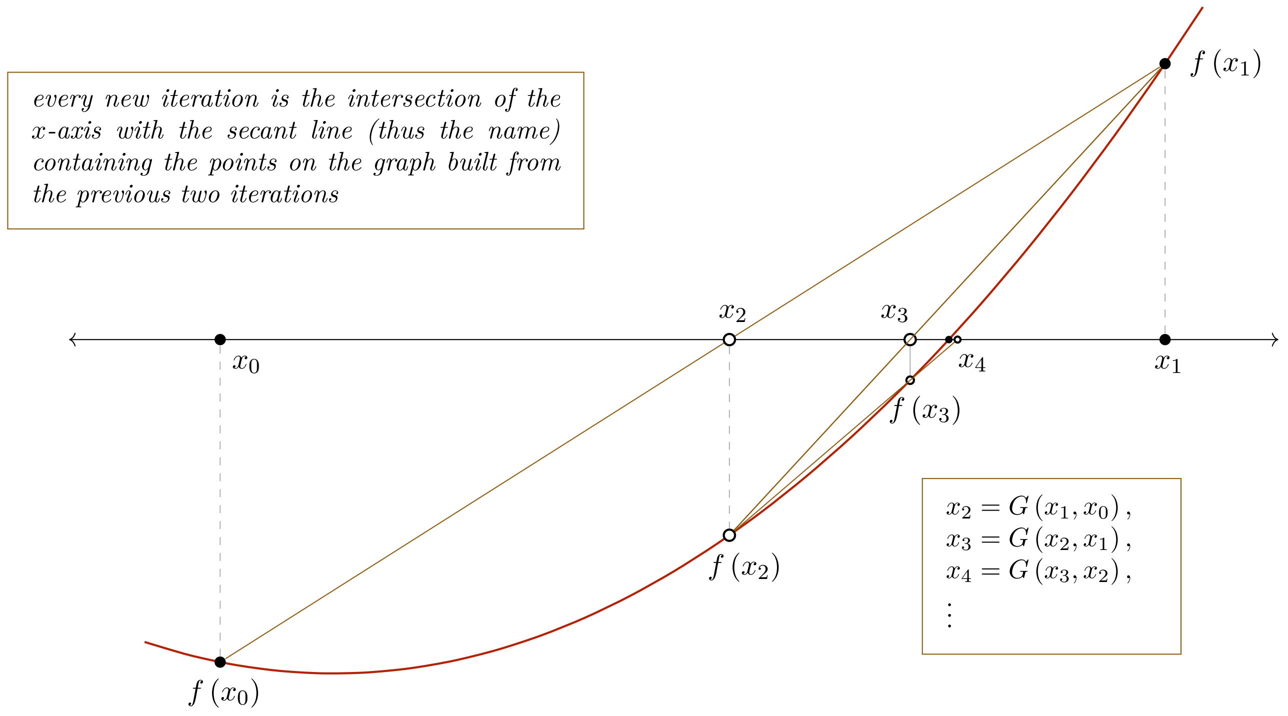
There are many examples of iterative methods, i.e. choices of G:

- the **secant method**, which may or may not converge depending on initial conditions;
- **Newton's method**, which may or may not converge but if it does, is usually faster;
- the **bisection method**, which converges more often but does so extremely slowly;
- other methods, such as Steffensen's method, fixed-point iteration, Halley's method, etc.

We will see the first three methods.

2 The secant method

This is a two-point iterative method, i.e. $j = 2$ meaning every iteration is a function of the previous two. Assume we have chosen two initial conditions x_0, x_1, chosen close to the root α that we are after:


every new iteration is the intersection of the x-axis with the secant line (thus the name) containing the points on the graph built from the previous two iterations
$f(x_1)$
x_0
x_2
x_3
x_4
x_1
$f(x_3)$
$f(x_2)$
$f(x_0)$
$x_2 = G(x_1, x_0)$,
$x_3 = G(x_2, x_1)$,
$x_4 = G(x_3, x_2)$,
$\vdots$

x_4 is already fairly close to (but still distinguishable from) the root. It is reasonable to assume that $x_5, x_6, \ldots$ will be even closer, and proceeding in this manner we will obtain an x_k sharing the desired number of decimal digits with α. Simple geometry shows that the intersection of line $\overline{(x_{n-1}, f(x_{n-1}))\ (x_{n-2}, f(x_{n-2}))}$ with the x-axis is $G(x_{n-1}, x_{n-2}) = x_{n-1} - f(x_{n-1}) \frac{x_{n-1}-x_{n-2}}{f(x_{n-1})-f(x_{n-2})}$, meaning the secant method has the following form:

$$x_n = x_{n-1} - f(x_{n-1}) \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}, \quad n \geq 2, \quad \text{having chosen initial conditions } x_0, x_1 \tag{3}$$

Example. Assume we want to find the largest root of $f(x) = x^2 - 1$. We know how to find it symbolically (it is $= 1$), hence we do not need a numerical method – but that does not mean we cannot apply one. First let us express G in terms of f:

$$G(x_{n-1}, x_{n-2}) = x_{n-1} - f(x_{n-1}) \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})} = x_{n-1} - \frac{\left(x_{n-1}^2 - 1\right)(x_{n-1} - x_{n-2})}{x_{n-1}^2 - x_{n-2}^2} = \frac{x_{n-2}x_{n-1} + 1}{x_{n-2} + x_{n-1}}$$

Thus our iteration method will be the following. We are using pen and calculator instead of Python here, so we can afford simplifying G:

$$x_n = \frac{x_{n-2}x_{n-1} + 1}{x_{n-2} + x_{n-1}}, \qquad n \geq 2.$$

It being a two-point method, we need two initial conditions. This is up to the method user. **Bear in mind that convergence (as well as convergence speed) of the method may depend on an adequate choice of these conditions**. Choosing unwisely can result in convergence to the wrong root (-1), slower convergence to 1, or no convergence at all.

For example, choose $x_0 = 0, x_1 = 2$. Let us fix precision $\varepsilon = \frac{1}{2}10^{-16}$ and start our process:

$$\begin{aligned}
x_0 &= 0 \quad \text{(initial condition chosen by us)} \\
x_1 &= 2 \quad \text{(initial condition chosen by us)} \\
x_2 &= G(x_1, x_0) = \frac{x_0 x_1 + 1}{x_0 + x_1} = \frac{0 \cdot 2 + 1}{0 + 2} = \frac{1}{2} \\
x_3 &= G(x_2, x_1) = \frac{x_1 x_2 + 1}{x_1 + x_2} = \frac{2 \cdot \frac{1}{2} + 1}{2 + \frac{1}{2}} = \frac{4}{5} = 0.8 \\
x_4 &= G(x_3, x_2) = \frac{x_2 x_3 + 1}{x_2 + x_3} = \frac{\frac{1}{2} \cdot \frac{4}{5} + 1}{\frac{1}{2} + \frac{4}{5}} = \frac{14}{13} \simeq 1.076923076923076923 1 \\
x_5 &= G(x_4, x_3) = \frac{x_3 x_4 + 1}{x_3 + x_4} = \frac{\frac{4}{5} \cdot \frac{14}{13} + 1}{\frac{4}{5} + \frac{14}{13}} = \frac{121}{122} \simeq 0.99180327868852459016 \\
x_6 &= G(x_5, x_4) = \frac{3280}{3281} \simeq 0.99969521487351417251 \\
x_7 &= G(x_6, x_5) = \frac{797162}{797161} \simeq 1.00000125445173559 67 \\
x_8 &= G(x_7, x_6) = \frac{5230176601}{5230176602} \simeq 0.99999999980880186730 \\
x_9 &= G(x_8, x_7) = \frac{8338590849833284}{8338590849833285} \simeq 0.99999999999999998801 \\
x_{10} &= G(x_9, x_8) = \frac{87224605504560089535585254}{87224605504560089535585253} \simeq 1.0000000000000000000000000001146. \\
x_{11} &= G(x_{10}, x_9) = \frac{1454660594681285404315232913246121223340241}{1454660594681285404315232913246121223340242} \simeq 0.\boxed{42 \text{ digits} = 9}3125544
\end{aligned}$$

We have $|x_{11} - x_{10}| < \frac{1}{2}10^{-16}$, thus x_{11} is our last iteration and the process can stop. Our approximation for the root is $x_{11} = 0.999\cdots$, and if we round it up to 16 digits we get 1.

Needless to say, choosing other initial conditions will change the numbers (and potentially, the number of iterations needed) in the above process. EXERCISE: try this yourselves, e.g. $x_0 = 1/2$ and $x_1 = 3/2$.

Example. Let us try $f(x) = e^{-x} - x$ shown in page 2. Unlike the previous example, we now do not have a simple representation of root α. We know (after looking at its graph) that it lies between 0.5 and 0.6, thus these two can be the initial conditions. Our iteration function will be

$$\begin{aligned}
G(x_{n-1}, x_{n-2}) &= x_{n-1} - f(x_{n-1}) \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})} = x_{n-1} - \frac{\left(e^{-x_{n-1}} - x_{n-1}\right)(x_{n-1} - x_{n-2})}{x_{n-2} - e^{-x_{n-2}} + e^{-x_{n-1}} - x_{n-1}} \\
&= \frac{e^{x_{n-2}} x_{n-2} - e^{x_{n-1}} x_{n-1}}{e^{x_{n-1}} \left(e^{x_{n-2}}(x_{n-2} - x_{n-1}) - 1\right) + e^{x_{n-2}}}
\end{aligned}$$

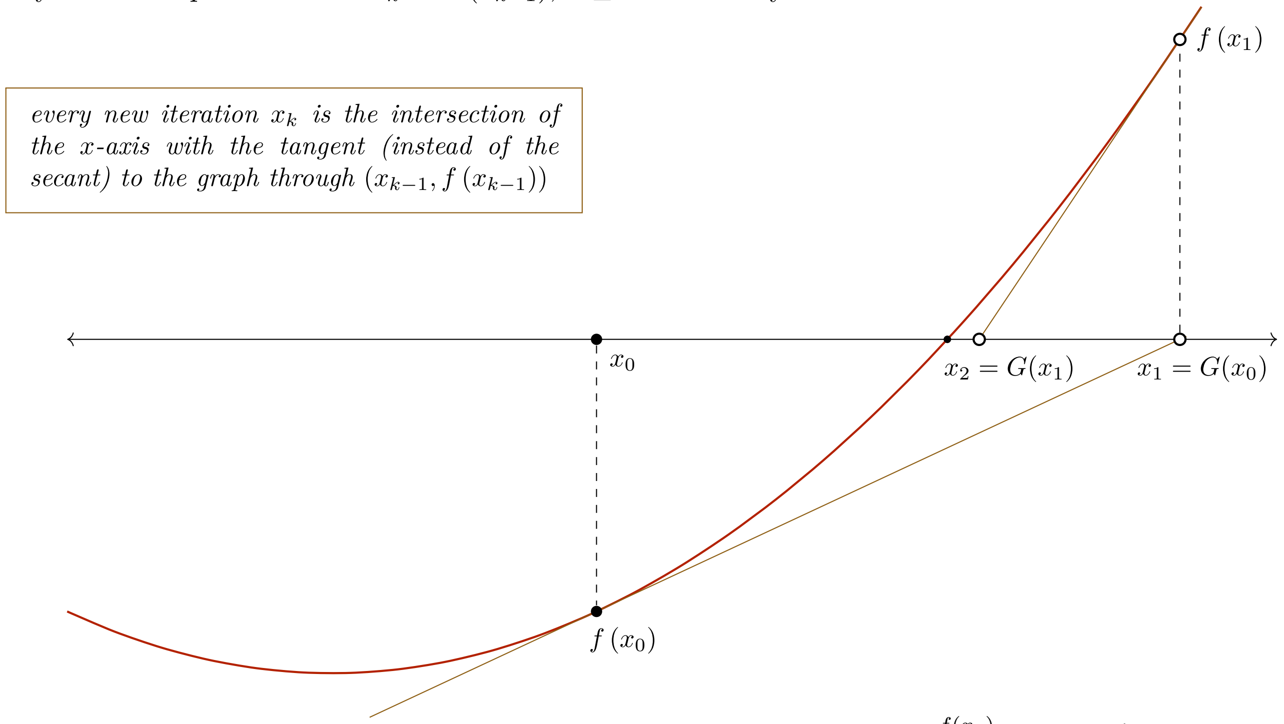
thus our process will be:

$$\begin{aligned}
x_2 &= G(x_1, x_0) = G(0.6, 0.5) = \frac{e^{0.5} 0.5 - e^{0.6} 0.6}{e^{0.6} \left(e^{0.5}(0.5 - 0.6) - 1\right) + e^{0.5}} \simeq 0.56754458483730139 47 \\
x_3 &= G(x_2, x_1) = \frac{e^{x_1} x_1 - e^{x_2} x_2}{e^{x_2} \left(e^{x_1}(x_1 - x_2) - 1\right) + e^{x_1}} \simeq 0.5671409166735748153, \\
x_4 &= G(x_3, x_2) = \frac{e^{x_2} x_2 - e^{x_3} x_3}{e^{x_3} \left(e^{x_2}(x_2 - x_3) - 1\right) + e^{x_2}} \simeq 0.5671432905821386311 \\
&\vdots \quad \vdots \quad \vdots
\end{aligned}$$

until $x_7 \simeq 0.567143290409783873$ which satisfies $|x_7 - x_6| < \frac{1}{2}10^{-16}$, thus we have our approximation 0.567143290409783873 of the root with 16 correct decimal digits.

3 Newton's Method

Also called **Newton-Raphson method**, it is a one-point algorithm, i.e. $j = 1$: every iteration only needs the previous one: $x_k = G(x_{k-1})$, $k \geq 1$. Thus only one initial condition is needed:


every new iteration x_k is the intersection of the x-axis with the tangent (instead of the secant) to the graph through $(x_{k-1}, f(x_{k-1}))$
$f(x_1)$
x_0
$x_2 = G(x_1)$
$x_1 = G(x_0)$
$f(x_0)$

Basic Geometry and Calculus yield an explicit expression: $G(x_k) = x_k - \frac{f(x_k)}{f'(x_k)}$, where f' is the **derivative** of f. Thus our method will be

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}, \quad n \geq 1, \quad \text{having chosen initial condition } x_0 \tag{4}$$

This method usually requires less iterations than secant to approximate the root with the same precision. Same as with secant, convergence to the desired root is not guaranteed here. Try to think of situations leading to lack of convergence.

Example. Return to $f(x) = e^{-x} - x$. $f'(x) = -e^{-x} - 1$, thus our iteration function will be

$$G(x) = x - \frac{f(x)}{f'(x)} = x - \frac{e^{-x} - x}{-e^{-x} - 1} = \frac{x+1}{e^x + 1},$$

Choose initial condition $x_0 = 1$ and the first iterations of (4) become:

$$\begin{aligned}
x_1 &= G(x_0) = x_0 - \frac{f(x_0)}{f'(x_0)} = \frac{2}{1+e} \simeq 0.53788284273999024150 \\
x_2 &= G(x_1) = x_1 - \frac{f(x_1)}{f'(x_1)} \simeq 0.56698699140541323888 \\
x_3 &= G(x_2) = x_2 - \frac{f(x_2)}{f'(x_2)} \simeq 0.56714328598912294440 \\
x_4 &= G(x_3) = x_3 - \frac{f(x_3)}{f'(x_3)} \simeq 0.567143290409783869 \\
x_5 &= G(x_4) = x_4 - \frac{f(x_4)}{f'(x_4)} \simeq 0.567143290409783873;
\end{aligned}$$

iteration x_5 shares at least 16 digits with x_4 ($|x_5 - x_4| < \frac{1}{2}10^{-16}$), thus the mathematical discussion we have omitted here implies that it also shares those digits with the actual root. Thus we can stop here and 0.5671432904097838 is our approximation. EXERCISE: try an initial condition x_0 closer to the root, say 0.55, and observe the algorithm converge even faster.

4 The bisection method

This is the most intuitive method and has the advantage of converging regardless of our choice of initial conditions *if the zero entails a change in sign*, as well as not requiring derivatives or complicated manipulation of our function $f(x)$ – but is also the slowest method available.

The idea behind bisection is simple: if a root α lies between a and b (i.e. inside the interval $[a, b]$) and we divide, or *bisect*, this interval into two subintervals of equal length, then α belongs to either the left half $\left[a, \frac{a+b}{2}\right]$, or the right half $\left[\frac{a+b}{2}, b\right]$. Once we know which of the two subintervals it belongs to, we apply the exact same argument to that subinterval: dividing it into two, and discerning which half contains α. Each subinterval will be $1/2$ the size of the previous one, thus we will be progressively cornering α in smaller intervals until we have an interval whose length is smaller than the precision tolerance $\frac{1}{2}10^{-p}$.

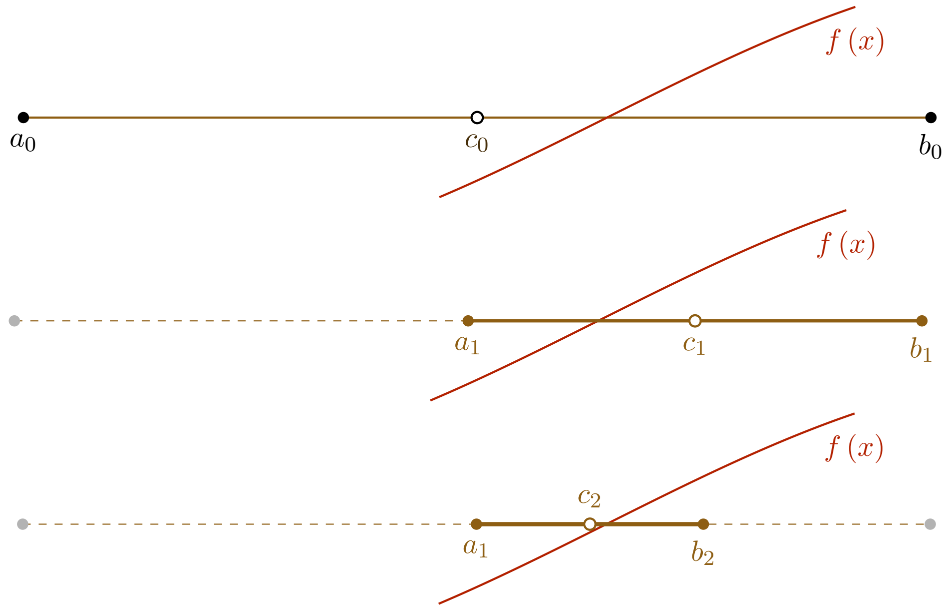
Let us describe the iteration. We need two initial conditions a_0, b_0 (extremes of the initial interval containing α) and then proceed as follows for $n = 0, 1, \ldots,$ until $|a_n - b_n| < \frac{1}{2}10^{-p}$.

What is done in iteration n:

- At this stage we have $[a_n, b_n]$ containing α. Compute the midpoint $c_n = \frac{a_n+b_n}{2}$.

  Option 1: If $f(a_n) f(c_n) < 0$ that means $\alpha \in [a_n, c_n]$. Start iteration $n+1$, replacing $[a_n, b_n]$ by the smaller interval $[a_n, c_n]$, and repeat the process.

  Option 2: If $f(b_n) f(c_n) < 0$ the right half-interval $[c_n, b_n]$ contains α. Start iteration $n+1$ by defining $[a_{n+1}, b_{n+1}]$ to be this interval $[c_n, b_n]$, and repeat the process.



Needless to say, iteration $n = 3$ in the above example would arise from taking $a_3 = c_2$ and $b_3 = b_2$.

5 The Actual Coursework

Write a Python program to approximate the roots of any function. It must contain, at least, the following functions.

1. A routine `initial_plot` plotting the graph of any $f(x)$, then giving the user the option to change the range interval and create new plots, or stop plotting altogether. You will use this function to zoom in or out until you have spotted a root of f, thus giving you good candidates for initial condition(s) close to it before you implement the methods in 2 below.

2. Functions `Secant`, `Newton`, `Bisection` for approximating roots numerically. Each of these functions should have at least the following parameters passed to them as inputs:

- the function $f(x)$ itself and, in one of the three methods, its derivative $f'(x)$;
- the initial condition(s) `x0`,... required by the given method;
- the tolerance `tol` determining the precision with which you wish to approximate the root.

Your functions must write initial conditions and all iterations in an external file. Think of ways in which to fill out this external file with at least two columns, so that:

- one of the columns corresponds to the index $k = 0, \ldots$ determining the iteration;
- and another one for the iteration x_k itself.

Make sure you write this in a way that makes it readable from another Python routine.

3. Function `read_from_file` reading an external file and returning the column corresponding to the iterations of whatever method (`Secant`, `Newton` or `Bisection`) filled the file originally. If the file is not arrayed in proper columns, `read_from_file` should detect this and notify the user.

4. A routine `final_plot` doing the exact same thing as `final_plot`, and in addition having slightly larger, properly labelled dots for some of the iterations, along with captions containing all the relevant information.

Let `XXXXXX` be your student number. All that is needed for you to upload to Moodle is:

- One Python file `UPXXXXXX_MAIN.py` containing the main body of your program.
- One Python file `UPXXXXXX_METHODS.py` containing your methods and, perhaps, other routines.
- Optional `UPXXXXXX_FUNCTIONS.py` if you wish to define your different $f(x), f'(x)$ in a separate file.
- One Jupyter notebook showcasing applications of, and comments about, your Python project.
- A few sample `.txt` files containing iterations of all three methods applied to functions.
- A few initial plots, showing how you located and zoomed into roots of at least one function (you can skip this if your Jupyter file contains plots)
- A final plot obtained from `final_plot`. Again, you can skip this if your Jupyter file contains plots.

Mark ranges: if your program

- does none of these: run without errors, produce a plot or a numerical output: **0-45 marks**
- fails to do *at least* one of the things described above: **30-60 marks**
- runs without errors and produces *one* plot, but lacks what is said below: **45-70 marks**
- same as item above and the process is correct for arbitrary $f(x)$: **60-90 marks**
- same as item above and a good Jupyter notebook is present: **80-100 marks**

Mark ranges overlap because the global layout of your program and your ability to detect glitches will also count.