Assignment #1: Finding Minima of Functions (**total 10 points**), due by 3:00 pm Monday, 30 January 2023

The objective of this assignment is to script numerical methods that can determine roots of simple functions such as a quadratic function. You will implement and compare two distinct finding-root methods: the bisection and the Newton-Raphson. In the last exercise, you will apply one of these methods to solve a physics problem which is to locate the minima of a potential function ruling the interaction of two ions, Na^+ and Cl^-.

1 Potential energy and stable equilibrium

In mechanics, you learned about the potential energy, $U(x)$, of a particle, which varies with the particle's position, x (one-dimensional case). A simple example is the potential energy $U(x) = (1/2)k\,x^2$ for a harmonic oscillator. Consider a particle moving along the x-axis - the force F on the particle is given by

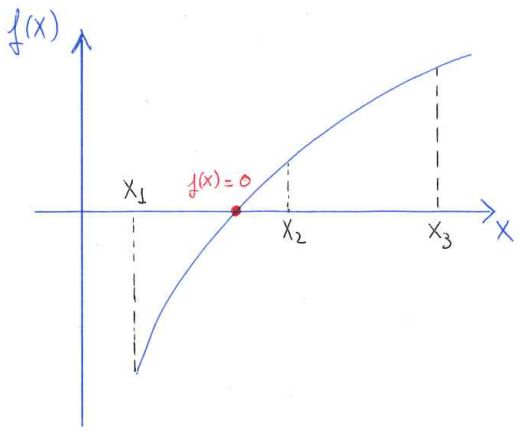
$$F(x) = -\frac{dU(x)}{dx} \tag{1}$$

For the harmonic oscillator potential, the negative gradient of the potential energy is $F(x) = -kx$, which is Hooke's law. The force acting on a particle is zero at a stationary point of the function, i.e. $dU(x)/dx = 0$. Close to a potential energy minimum, the force is always towards the bottom of the potential well, while near a maximum the force is away from the top of the potential energy maximum. So the bottom of a potential well is a point of stable equilibrium, while a maximum of the potential is a point of unstable equilibrium, from which the particle tends to move away under the slightest perturbation. A point of inflection is also a point of unstable equilibrium. It is important to identify the states of stable equilibrium of particles or systems of particles, and this involves finding the minima of potential energy functions. For one-dimensional problems, this means we must find the points where the slope of $U(x)$ is zero, and its second derivative is positive.

1.1 The bisection method for finding roots of a function

The bisection method for finding roots for a function, $f(x) = 0$, is very simple: choose a value for x_1 for which $f(x_1)$ is negative and x_3 for which $f(x_3)$ is positive (see figure below). Then evaluate

$f(x_2)$ for the average of these two values, $x_2 = (x_1 + x_3)/2$. If $f(x_2)$ is positive, replace x_3 by x_2 and if $f(x_2)$ is negative, replace x_1 by x_2. Repeat (or iterate) this procedure until $\mid f(x_2) \mid$ is less than some small value (tolerance) you choose.



1.2 Numerical accuracy in scientific computing

In mathematics there are analytical methods for finding solutions to many forms of equations. However, in many cases, analytic methods are not available and we must seek numerical solutions. Numerical solutions are not, generally speaking, exact solutions to the equation concerned. Instead, they are approximate solutions. For example, in this problem set, we are looking for solutions to the problem $f(x) = 0$, where f is a quadratic function of x, $f(x) = ax^2 + bx + c$, for the sake of simplicity. In this particular case, the analytic solution is given by

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

The numerical methods we will use does not give this result exactly (unless we choose the initial guesses for the root so that we do end up coincidentally with the exact root). Instead, we solve a similar problem in which the root is set to be found once $\mid f(x) \mid < \xi$, where ξ is a sufficiently small number which we choose.

2 Python scripting

(a) Open a new file in your Notebook environment and name it `script.ipynb`.

(b) As we learned in the previous week, a Python script is a sequence of instructions following a certain syntax that we enter in a file which usually has the descriptor `.ipynb` for typical Notebook environments. Enter the lines below in your `script.ipynb` file.

```
import matplotlib.pylab as plt
import numpy as np
a = 1
b = 1
c = -6
x = np.arange(-5.0, 5.0, 0.2)
plt.plot(x, a * x * x + b * x + c)
plt.plot(x, 0.0 * x)
plt.show()
```

Save `script.ipynb` in your Notebook. In order to generate an array of values for x, from -5.0 to 5.0 in steps of 0.2, `script.ipynb` uses the `arange` function in `numpy`. The two `plot()` functions plot the parabola determined by the values of a, b and c and a horizontal axis at $f(x) = 0$ for visualization. The plot does not appear on screen until the `show()` command is executed. Run your script and check how your plot looks like.

(c) Your script will not run until you execute its cell in your Notebook. Run your script once more and see the plot it produces when you change your parabola to $f(x) = x^2 + 3x - 4$.

3 Python script to find the roots of a parabolic function

(a) Choose a parabola with a minimum and two REAL roots (other than the ones given above). Determine its roots analytically.

(b) Write a Python script to plot your chosen parabola and the coordinate set with an appropriate ordinate and abscissa range (and so both roots can be seen). Your parabola should be implemented as a function in your script just as we did in the practice exercises.

(c) Define variables x_1, x_2 and x_3 to your script with x_1 and x_3 initialised to values for which $f(x_1) < 0$ and $f(x_3) > 0$ and $x_2 = (x_1 + x_3)/2$. Add `if` statements to the script which check that the variables are correctly initialised and warn the user if they are not.

(d) Use `if` and `else` statements which update x_1 or x_3 to x_2 according to whether $f(x_2)$ is greater or less than zero and print the updated values of x_1 and x_3 for verification.

(e) Plot the new point on your parabola $(x_2, f(x_2))$ in a colour different from the colour used for the parabola curve. Visualize your plot to check if the new point is displayed correctly.

(f) Add a `while` loop to iteratively update x_2 using the rule $x_2 = (x_1 + x_3)/2$ as long as the absolute value of $f(x_2)$ is greater than a tolerance value `tol`, initialised to `tol = 0.0001` before the `while` loop. If your script does not terminate in a few seconds it may be that it has an infinite `while` loop. To terminate it manually, interrupt it via the Notebook Kernel.

(g) Add a statement which prints the values of x_2 and $f(x_2)$ after the absolute value of $f(x_2)$ becomes less than `tol`. The Python absolute value function is `math.fabs(argument)` called within `math` package. Remember to indent both the `while` loop and the `if-else` statements.

(h) Run the script and check that it gives you a correct root for the parabola. Modify the script so that it will give the other root and then run it again.

(i) Use the script to find how the number of steps required to find the root of the parabola depends on the value of `tol` by adding a variable `nsteps` into the `while` loop which is incremented by one each time the loop is executed. Note that the variable `nsteps` must be initialised to zero.

(j) Plot a graph of `nsteps` versus the logarithm of `tol` to the base 10. The Python log function is `math.log10(argument)`.

(k) **IMPORTANT:** Graphs from (e) and (j) should be included in your report **(1 point)**.

(l) **IMPORTANT:** Attach the `.ipynb` files you generated in item (j) in addition to your report **(1 point)**.

4 The Newton-Raphson method for finding roots of a function

The Newton-Raphson (NR) method is based on the Taylor series expansion of a function about a specific point $x = x_0 + \epsilon$,

$$f(x_0 + \epsilon) = f(x_0) + f'(x_0)\epsilon + \frac{1}{2} f''(x_0)\epsilon^2 + \dots \tag{3}$$

where $f(x_0)$, $f'(x_0)$, and $f''(x_0)$ are the values of f at x_0 and its first and second derivatives, respectively. When we apply the NR method to root finding, x_0 is the current estimate for a root and we want to know which step size and direction will take us to an improved estimate for the root. The method can be applied to find the minima of a function of a single variable or to a function of many variables.

In order to have a linear equation to solve for an improved approximation for a root, the Taylor series for f about the point x_0 is truncated at the term which is linear in ϵ,

$$f(x_0 + \epsilon) = f(x_0) + f'(x_0)\epsilon + O(\epsilon^2) \tag{4}$$

The term $O(\epsilon^2)$ in this equation indicates that the series was truncated and terms of order ϵ^2 and higher were omitted. Now set $f(x_0 + \epsilon) = 0$ so that $x_0 + \epsilon$ is an approximation for a root. It is not the exact root since the Taylor series for f was truncated. From the truncated Taylor series, the condition $f(x_0 + \epsilon) = 0$ is equivalent to

$$f(x_0) + f'(x_0)\epsilon = 0 \tag{5}$$

from which we obtain (and just setting $\epsilon \equiv \epsilon_0$)

$$\epsilon_0 = -\frac{f(x_0)}{f'(x_0)} \tag{6}$$

which is the first-order adjustment to the root. By letting $x_1 = x_0 + \epsilon_0$ and calculating a new ϵ_1, and so on, the process can be repeated until it converges to a fixed point (which gets closer to the real root) using

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{7}$$

where n now refers to iteration steps. This expression summarizes the NR method.

(a) Write a new Python script which plots the parabola you chose for the previous exercise and its derivative using Python function definitions. Note that you do not need the constant c to evaluate the derivative of the parabola $f(x) = ax^2 + bx + c$ and so the arguments for the derivative function definition should be (x, a, b) only.

(b) Define a variable x and initialise it to 1.

(c) Add a `while` loop to update x iteratively using the NR rule $x \coloneqq x - f(x)/f'(x)$ (the symbol ":=" indicates an assignment) as long as the absolute value of $f(x)$ is greater than a tolerance value of `tol= 0.0001`. Remember that you need to indent the code within the `while` loop.

(d) Plot the initial and the converged points on your parabola with a different colour from the colour used to draw the parabola curve.

(e) Print the values of x and $f(x)$ before and after the absolute value of $f(x)$ becomes less than `tol`.

(f) Run the script and check that it gives you a correct root for the parabola.

(g) Modify (if necessary) the script so that it will give the other root and then run it again to obtain that root. If you cannot think how to modify the script to do this, think about how the NR algorithm works and therefore what you need to change in order to obtain the other root.

(h) Use the script to find how the number of steps required to find the root of the parabola depends on the value of `tol` by adding a variable `nsteps` into the `while` loop which is incremented each time the loop is executed.

(i) Plot a graph of `nsteps` versus the logarithm of `tol` to the base 10 (as in the previous exercise).

(j) Compare the efficiencies of the bisection and NR methods in your report by comparing the graphs of the number of iterations taken to reach a root for a given value of `tol`.

(k) **IMPORTANT:** Graphs from (d) and (i) should be included in your report **(1 point)**.

5 Python script to find the minima of a potential energy function

The interaction potential energy between two ions such as Na^+ and Cl^- is given by

$$U(x) = A\exp\left(-x/p\right) - \frac{e^2}{4\pi\epsilon_0}\frac{1}{x} \tag{8}$$

The two terms represent short range, Pauli repulsion of electron clouds and long range electrostatic attraction of oppositely charged ions. The numerical values of $e^2/(4\pi\epsilon_0)$, A, and p are 1.44 eV nm, 1090 eV and 0.033 nm, respectively. Use these values in your script whose algorithm will be described below.

Previously, you used the NR method to find roots of functions by solving $f(x) = 0$ numerically. NR can also be used to find minima or maxima of functions by solving $df/dx = 0$ for x, since the derivative of the function vanishes at a maximum or minimum point. To apply the NR method to find the root of $dU/dx = 0$ we need the rule

$$x \coloneqq x - \frac{U'(x)}{U''(x)} \tag{9}$$

where U' and U'' are the first and second derivatives of $U(x)$ with respect to x.

(a) Write a Python script to plot the function $U(x)$. First, define the function $U(x)$ in your script. Plot the potential $U(x)$ in the range of $x = 0.01$ to 1 nm.

(b) Differentiate $U(x)$ with respect to x (on paper) and plot a graph of $-dU/dx$ versus x. This is the force acting on the particle. Add a statement to your script so that it also plots $-dU/dx$. Observe that $-dU/dx$ is zero at the minimum of $U(x)$. Explain why this is so in your report.

(c) Differentiate dU/dx (on paper) to obtain $U''(x)$ and write the expressions for $U'(x)$ and $U''(x)$ in your report.

(d) Add instructions to your script so that it finds the value of x for which $U(x)$ is a minimum using the NR method. Note that the minimum is close to 0.2 nm so that $x = 0.2$ nm is a good value to initialise x to.

(e) **IMPORTANT:** Graphs from (a) and (b) should be included in your report. In the plot of $U(x)$, mark (with a symbol) the minimum of the potential that your numerical procedure found. What is the value of x that you obtained for the minimum of $U(x)$? And what is the minimum of the interaction potential energy? Discuss your findings. **(1 point)**.

(f) **IMPORTANT:** Attach the `.ipynb` files you generated for item (e) in addition to your report **(1 point)**.

6 Report writing guide (5 points)

After generating your codes and results for the exercises above, you will summarize them in the form of a written report. Please, follow the instructions provided in the report template located in the sub-module of the assignment to write your report. All margins, font style/size, line spacing, and page limits set on the template need to be respected. **We will deduct a significant amount of points from reports that do not align with the template provided or we will not be able to assess the report, meaning we will have to deduct -5/10 points from the total marks.**

There is a `docx` file (generated in MS Word editor) available in the sub-module of the assignment that can, in principle, be used and exported to other text editors. However, some of its pre-defined macros may not be read correctly by other editors. Moreover, inserting text on the editable `docx` file may shift or duplicate some of the original headers. Make sure your written content obeys the format set in the read-only `PHYS 381 Assignment 1 Template.pdf` file. In case you have problems with the `docx` file provided, a suggestion is to start a blank document in the text editor of your choice and set the layout as below:

- Page size: 8.5" × 11" (letter);
- Margins: 2.54 cm (top, bottom, left, right);
- Font type and size: Times New Roman, 12 pt, non-italic, non-bold;
- Line spacing: between 1.0 and 1.15 pt;
- Figures must be numbered and contain captions. Axes must carry names and their quantities must be expressed with the proper units whenever required. Do not insert too large figures just to cover space. Choose figure sizes typically seen in textbooks and scientific manuscripts.
- Follow the exact page configuration and order as described in the report template, e.g., title page on page 1, abstract on page 2, introduction on page 3, etc. The title page needs to contain the exact same information as in the report template.

6.1 Submission

The due date for the submission of your report is depicted on the first page of the assignment. Once you conclude the writing of your report in accordance with the template provided, generate a `pdf` file and name it as `report_assignment_1.pdf` (since this is report #1). You will upload the `pdf` file of your report plus your Notebook `.ipynb` files requested in the assignment to the Gradescope platform. You can upload multiple files to the Gradescope platform and you can resubmit your work until the due date. We will test-run all your submitted codes to test for errors. We will also check if the results/figures presented in your report match the output of your codes.

For Groups: If you worked in a group, remember to include your group members in Gradescope after you submit all the files on behalf of the group. You will see an option "Add Group Member" after you upload all files to Gradescope. Students can work in groups of up to 4 members.

Autograder option: Ignore the autograder results that may appear after you upload files to Gradescope. There is no autograder configured for this assignment since everything will be evaluated manually by the PHYS 381 Team.

Please, remember to add comments in all your submitted Notebook codes! Pure code lines without explanatory comments or written Markdown cells will have reduced marks.

Standard `.py` Python codes are not accepted in this submission. Work only with Notebook environments. If any member of the group is experiencing problems with their coding environment, please, contact the instructor immediately.

ONLY `pdf` (for the written report) AND `.ipynb` (for codes) FILES ARE ACCEPTED.

*****