Instructor: Dr. S. Hughes

Due date: Sunday, Feb 18th (midnight)

Great news☺: we are giving you 2 weeks to complete this next assignment. Bad news: it contains a bit more questions than usual. So the next two lab weeks will cover this one assignment.

Lab 4 - Roots and Nonlinear Equations

Keywords: Root Solvers, Systems of Nonlinear Equations, BiSection Method, Secant Method, Newton' Method, Some Graphics, Generalized Newton's Method for Higher Dimensions

Task: Go through all the questions and coding exercises below and hand in a single modularized Python code where each part runs each part of the question(s). Remember and include your surname at the beginning of the code name.

Code Submissions on onQ: Attach a single (preferable, or no more than several) .py (called LabX_YourLastName.py) and single .py.txt (same code, same name, just a different extension).

Marks: Coding Skills, Efficiency, Correct Results and Clear Graphs and Good Presentation of Results (6); Good use of Comments (2); Good Coding Structure and Readability (2). Total: (10) [Note: We may increase the marks for this one as it is a bit longer than the previous ones, or count as double weight.]

Reminder: All codes must run under Python 3.x. and the Spyder IDE (check this before submitting). Section off the code by using #%% (and label section in code, e.g., Q1(b) $\cdots$), and add useful comments, so we can step through it easily.

Acknowledgements: Please comment on any help that you received from others for completing the Lab assignment.

Some Background (in addition to lecture notes):

Root Finding Algorithms - Wikipedia
Trial and error searching (part 1) - Rubin Landau Lecture
Trial and error searching (part 2) - Rubin Landau Lecture

Question 1

(a) Use the Bisection approach to solve for

$$f(x) = \frac{1}{x-3}, \tag{1}$$

in the interval (0,5). Use a *tol* (tolerance) of 10^{-9} (pass it). Is your answer correct - justify yes or no? Also plot the function between say 2.5 and 3.5.

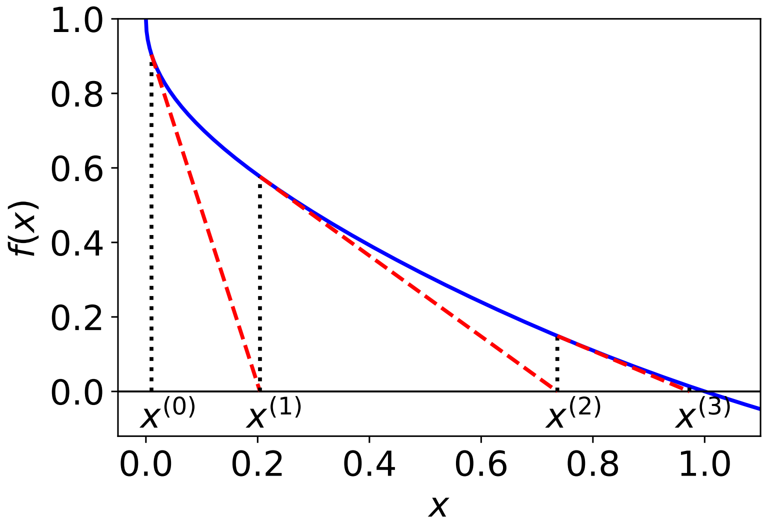
(b) In the class notes, I discussed the example roots of the functions,

$$f(x) = \exp[x - \sqrt{x}] - x, \tag{2}$$

and we considered the root near 2.5.

Locate the other root near 1 using your Bisection code, and a sensible window ('bracket').

Next write a Newton solver, which now requires the derivative function, and verify if the same answer is obtained within the stated accuracy. In both cases, use a *tol* (tolerance) of 10^{-8}. You can start with an initial guess of say $x_0 = 0.01$. Use the solution to reproduce a graph similar to that below[A], to see what is happening in the algorithm as a function of iteration.



(c) When using Newton's method, it can be annoying to start from different initial guesses and end up with the same root, even if there are multiple roots. A trick is to suppress an already found root and then use the function

$$u(x) = \frac{f(x)}{x-a}, \tag{3}$$

where a is the root already found. Implement this approach to Eq. (2), using the root $a = 1$. Check that the other root, near 2.5, is found for all the four initial conditions: $x_0 = 2.0, 0.5, 4.0, 0.1$.

[A] It does not have to look identical of course, just show the same qualitative features, including the text labels.

Question 2

(a) Find the depth h to which a sphere with radius $R = 1$ m sinks when placed in water as a function of the density of the sphere ρ. The mass of the sphere is $m = (4/3)\pi R^3 \rho_{\text{sph}}$. The volume of the sphere submerged to a depth of h is given by the expression: $V = (1/3)\pi(3Rh^2 - h^3)$. Assume the density of water is $\rho_{\text{water}} = 1$ (in relative units) and the density of the sphere is in the range $0 < \rho_{\text{sph}} < 1$ (i.e., the sphere will float, and $h < 2R$).

Solve the resulting non-linear equation in h using the Bisection method. Hint: you can exploit Archimedes' principle: `https://en.wikipedia.org/wiki/Archimedes'_principle`, and ignore any effects caused by surface tension. You can window between $h = 0$ and $h = 2$.

(b) Next, plot the surface of the sphere and the surface of the water. You could look up some Python documentation on `meshgrid` and surface plots, such as `plot_surface` and `counterf`.

You might find the following resources useful: `https://matplotlib.org/stable/gallery/mplot3d/surface3d.html` and `https://problemsolvingwithpython.com/06-Plotting-with-Matplotlib/06.16-3D-Surface-Plots/`, but basically explore the `matplotlib` docs for the different graph options.

When I set $\rho = 0.8$, I found $h \approx 1.4257$; or if I set $\rho = 0.55$, then $h \approx 1.0668$ and my Python code generates the following simple plot (your plot need not look like this, and indeed could be much clearer):

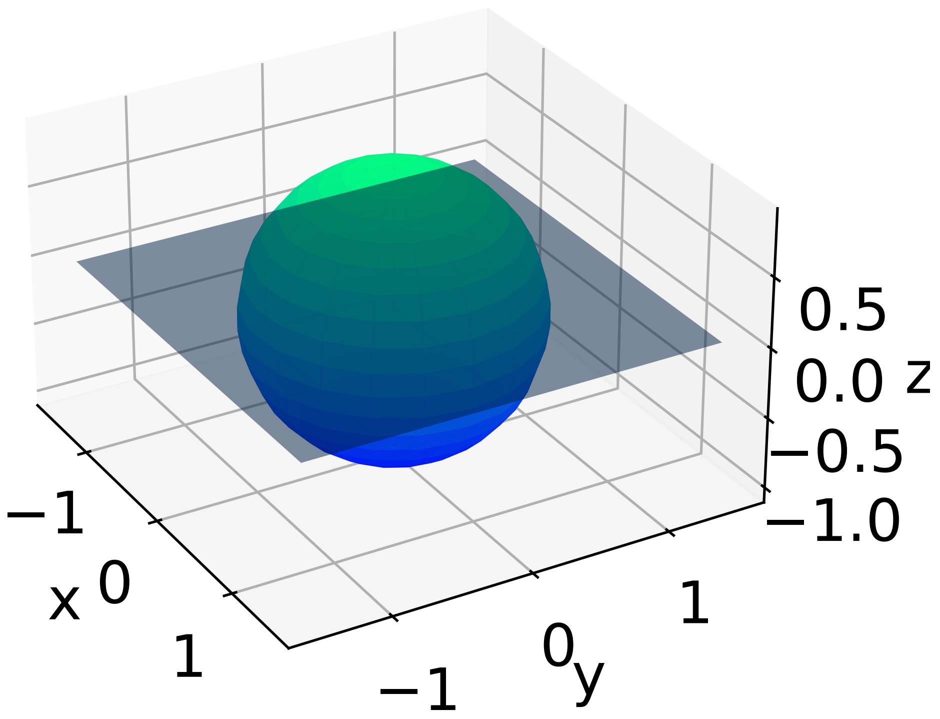


Figure 1: Simple graphical visualization to the solution to problem 2(b).

Question 3

In this question, we want to explore and implement the generalized Newton's Method, for solving coupled nonlinear equations. *It's time to get serious!*

(a) First, set up two subroutines that return the following function sets;

$$\texttt{Set 1:}\ \begin{cases} f_0 &= x_0^2 - 2x_0 + x_1^4 - 2x_1^2 + x_1 \\ f_1 &= x_0^2 + x_0 + 2x_1^3 - 2x_1^2 - 1.5x_1 - 0.05, \end{cases} \tag{4}$$

and

$$\texttt{Set 2:}\ \begin{cases} f_0 &= 2x_0 - x_1\cos(x_2) - 3 \\ f_1 &= x_0^2 - 25(x_1-2)^2 + \sin(x_2) - \pi/10 \\ f_2 &= 7x_0\exp(x_1) - 17x_2 + 8\pi, \end{cases} \tag{5}$$

The subroutines should return the entire set as NumPy arrays, so do not write separate subroutines for each f_0, f_1, etc, collect them as one functions containing all parts. So your function could look something like this, e.g., for Set 2:

```python
def fs2(xs):
    x0, x1, x2 = xs
    ...
    return np.array([f0,f1,f2])
```

(b) Next, set up a subroutine that returns the numerical Jacobian, where you will pass $\mathbf{f}, \mathbf{x}$, with a default numerical step size, $h = 10^{-4}$. This routine should return the Jacobian matrix for any arbitrary function, so should work for either **fs2** or **fs1**. For convenience in using the generalized Newton's Method, also return $\mathbf{f}(\mathbf{x})$, using a default $tol = 10^{-8}$.

(c) Implement the generalized Newton's Method, to solve for the two function sets above, using an initial $\mathbf{x}$ value of $\mathbf{x} = [1.0, 1.0]$ or $\mathbf{x} = [1.0, 1.0, 1.0]$ (depending on the function to be called). Obtain the solution for $\mathbf{x}_{\text{roots}}$ and check the value of $\mathbf{f}(x_{\text{roots}})$ is close to zero (print all values).

Note, you can use your Gaussian Elimination code from the previous problem set, with or without pivoting (pivoting is naturally more robust, but is not needed to solve this particular question). Alternatively you can use one from Numpy (e.g., `import numpy.linalg as la` and then `la.solve(A,b)`).

(d) For the case of Set 2, write an analytical Jacobian function, where you can evaluate the partial derivates analytically or using `SymPy` (but it is easy to do by hand!). Check if the answer agrees with the one from the numerical Jacobian. If it agrees within the numerical precision, check what value of h starts to yield a noticeable difference.