PHYS 331 – Introduction to Numerical Techniques in Physics

Homework 3: Root-Finding II, and Order of Convergence

Due Friday, Sept. 14, 2018, at 11:59pm.

Problem 1 – Comparison of the Bisection and Newton-Raphson Methods (15 points)

Note: This problem is paper only. No Python files should be submitted.

You are stranded on a desert island with a solar calculator that only has the four basic arithmetic operators (+,–,×,÷). You also have a ruler on hand, and want to construct a cubical box that has a volume of exactly 600 cm^3 (it's a long story), within a tolerance *in the volume* of 0.1 cm^3 (*i.e.*, acceptable values of $V \pm \sigma_V = 600.0 \pm 0.1$ cm^3). To accomplish this, you need to compute the desired length of the cube, L, in centimeters, fairly accurately. Remembering the good old days when you took PHYS 331, you recall that the Newton-Raphson method provides a way that you can use your simple calculator to iteratively solve for L.

(a) First, define a function, $f(x)$, such that at least one of the roots is the desired cube length, L.

(b) Using Newton-Raphson, write an iteration function (*i.e.*, x_{n+1} = **only** a function of x_n) using only the operators that are available on your solar calculator. **Simplify the function** as much as possible so that you can perform each iteration with the minimum number of operations on your calculator. What is the minimum number of operations needed to compute your iteration function?

(c) Starting with x_0 = the integer value closest to the true answer, perform the Newton-Raphson iteration, making a list of x_0, x_1, x_2, *etc* as well as their associated volumes, $V_0 = x_0^3$, $V_1 = x_1^3$, *etc*, until you reach the desired tolerance in the volume. How many iterations were required to get the stated accuracy?

(d) In the above problem we defined our error in a non-traditional way because our application required it. We effectively defined the error by how close our function $f(x)$ came to 0 with each iteration (which was done by seeing how close the volume was to the target value – note that these are equivalent statements). It is more conventional to define error in terms of the parameter x and not the function space $f(x)$. Let's determine the relationship between the error in volume, σ_V, and the error in cube length, σ_L. This can be done by propagation of error, which you may have learned somewhat in PHYS 118, as follows:

$$V = L^3$$

$$\sigma_V = \sqrt{\sigma_L{}^2 \left(\frac{\partial V}{\partial L}\right)^2} = \left|\sigma_L \cdot 3L^2\right|$$

$$\sigma_V = 3\sigma_L L^2$$

(1) Using the definition for the error in cube length as the difference between successive iterations ($\sigma_L = |x_{n+1} - x_n|$), what would be your estimate for the error in the volume at your last iteration performed in part (c) above? How does this compare to the true error in the volume at that last iteration? (*Report answers here and below to only 3 significant digits; more digits are unnecessary in this context*).

(2) If we wanted to solve this problem using a conventional root-finding function, we would need to define our tolerance in terms of L and not V. In this problem, what would be an appropriate tolerance value to set? (Assume that you are not limited by what the solar calculator can do.)

(3) If we were to perform the bisection method of root-finding with a starting interval of 1, and define the error as the size of the interval, how many iterations would it take to achieve the desired tolerance you computed above? How does this compare to the Newton-Raphson method, and why?

(e) Consider how you would perform Newton-Raphson or bisection to find the root with your calculator in practice, including all of the steps needed. Write out each of the steps you would use for each method, and the number of operations in each step. Add up the total number of arithmetic operations needed for each method to achieve the desired tolerance, using the number of iterations from your answers in parts (c) and (d). There may be different answers depending on your approach and assumptions; make sure to explain. Which method requires fewer operations?

Extra Credit (+2 points): Write a Python script to run the simulation for the above scenario using the bisection method. Use a starting interval of 1 with endpoints on the integers that bracket the root, and set it up to halt when the difference between the volumes corresponding to estimated root and the true root is < 0.1 cm^3 (similar to how we did for the Newton-Raphson method above). How many iterations are needed? Is this the same or different than your answer for bisection in part (d) above? If it is different, why is that? Note that your computational work should be submitted as a *HW1EC.py* file, with your written answers in your pdf file.

Problem 2 – Newton-Raphson Root-Finding (10 points)
Consider the real roots of the following 5th order polynomial, which, being of order >4, may not have an analytical solution:

$$f(x) = x^5 - 3x^3 + 15x^2 + 27x + 9$$

(a) Write a Python function of this function, `func(x)`, and explore plotting it over different ranges to find all **real** roots. Provide a final, nice plot of the function over a limited range of x where all of the real roots can be viewed easily. How many real roots are there and what are their approximate locations?

(b) Write the Newton-Raphson iteration equation for solving the roots for this function. Write out the entire expression as a function of x_n so that it can be readily implemented in Python in the next step.

(c) Now, implement the root-finding of the above as a Python function `Newt(xstart,tol)`, where `xstart` is the starting value, and `tol` is the desired tolerance. The function should return the root found. In this case, we define the error as the difference between the computed root in successive iterations (*i.e.*, $| x - x_{old} |$). Measure each root to a tolerance of 10^{-15}.

You do not need to put everything in `Newt`; you might wish to have `Newt` call other functions that you define above `Newt`. Think about how to break up the problem into appropriately sized chunks.

Problem 3 – A Computational Physics Problem (12 points). Do problem 15 of problem set 4.1 in the textbook (natural frequencies of a uniform cantilever beam). Use the supplied *HW3p3template.py*, which is the "safe" Newton-Raphson function from the textbook. Do not modify this function. Note that the moment of inertia for a rectangular cross-section is $I=bh^3/12$, where b is width and h is height. Add to the supplied .py file any plots, functions, and comments needed to show your thinking processes, and make sure all results (including plots and answers) display automatically when executing your file. (Hint: pay attention to units.) Choose a tolerance that provides at least 4 significant digits in your final solution.

Problem 4 – Order of Convergence (13 points). In class we discussed different orders of convergence for bisection compared to Newton-Raphson. While the bisection order is easily proved, the order for N-R is non-trivial to prove analytically. Instead, here you will experimentally verify the order of convergence for N-R based on how error reduces with successive iterations.

(a) Similar to Homework 2 Problem 3, modify the Newton-Raphson function provided in the new template file, *HW3p4template.py*, with a diagnostic tool to track the estimated error with each

iteration. The function `newtonRaphsonMOD` has already been partially modified to remove unwanted code that will help you with this task. Estimate the error **by the absolute value of the difference between successive iterations**, and have the function **only** output a numpy array of floats containing the errors at each iteration.

(b) Use this modified function to find the real root of

$$f(x) = (x+10)(x-25)(x^2+45)$$

that exists in the range of x=(0,40) (*i.e.*, use (0,40) as your starting bracket). Plot the error versus number of iterations for 10 iterations. After how many iterations is the error effectively zero (to within machine precision?)

(c) Determine whether the order of convergence from the above data is more consistent with m=1, m=1.5, or m=2. Use the fact that:

$$\varepsilon_{n+1} = c\varepsilon_n^{\,m}$$

and thus when the correct value of m is chosen,

$$\frac{\varepsilon_{n+1}}{\varepsilon_n^{\,m}} \approx const \quad in \quad n \tag{1}$$

The best way to do this is by trial and error to see which value of m is most consistent with the error convergence model. (There really isn't an analytical way to do this – believe me I tried!) Is the value of m what you expected?