PHYS 331 – Introduction to Numerical Techniques in Physics

Homework 3: More Root-Finding and Convergence…

Due Friday, Sept. 15, 2017, at 11:59pm.

Problem 1 – Comparison of the Bisection and Newton-Raphson Methods (10 points)

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 750 cm^3 (it's a long story), within a tolerance of 0.1 cm^3 (*i.e.*, acceptable values of $V = 750 \pm 1$ 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} = some 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.

(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) If you had used the bisection method with a starting interval of 1 (*i.e.*, the two adjacent integers that bracket the solution), how many iterations would have been required to achieve the needed tolerance?

(e) Compare the number of arithmetic operations needed per iteration for the two methods. (Assume that you are not able to do any arithmetic in your head).

Note: while you are welcome to perform the operations in part (c) using Python, you do not need to submit your code for this problem. All answers should be recorded in your *HW3.pdf* file.

Problem 2 – Choosing a Root-Finding Algorithm that Converges (20 points)

Consider the real roots of the following 5th order polynomial, which, being of order >4, may not have an analytical solution:

$$f(x) = x^5 - 3x^3 + 15x^2 + 29x + 9$$

(a) Write a Python function of this function, `func(x)`, and explore plotting it over different ranges to find all real roots. In `main()`, make a final, nice plot of the function over a limited range of x where all of the real roots can be viewed easily. For purposes of comparison, use this same range of x for all plots in the following sections.

(b) Write the Newton-Raphson iteration equation for solving the roots for this function.

(c) Now, convert the above into a fixed-point equation of the form $g(x)=x$. Write this as a Python function `gNewt(x)`. Calculate the derivative (it's messy – don't worry about multiplying it out, just keep it factorized for simplicity.) Write this as a Python function `dgNewt(x)`. Make a plot of the absolute value of the derivative overlaid with the line $y=1$. (Scale the plot as needed to observe

the desired features). Is the convergence criterion for fixed point iteration satisfied in the region of some or all of the roots? Is this behavior what you would expect?

(d) Instead of using Newton-Raphson, generate a different fixed point solution for the roots of $f(x)=0$ by subtracting both sides by $29x$ then dividing by -29, to create a function $g(x)=x$. Write this as a function `g1(x)`. Calculate the derivative, and write this as a function `dg1(x)`. As above, make a plot of the absolute value of dg and $y=1$, and discuss the convergence properties about each root.

(For fun, you might also try other strategies for creating different $g(x)=x$ and run them through this analysis; in the interests of your time I'm giving you this solution because it has neat properties to explore).

(e) Write a fixed-point iteration function, `fixed_pt(g,xstart,tol)` that accepts arguments of a Python function `g`, floating point variable `xstart` that represents the starting point for the iteration, and tolerance in x `tol`. The function should return the answer as a single, floating point variable after the tolerance is reached. Since, in this method, we don't have a bracket $|a-b|$, you should instead estimate the tolerance as the absolute difference between successive iterations, $|x_{n+1}-x_n|$. While you are welcome to explore adding error checking to your function, in the interests of time we won't be focusing on that in this problem, and you can keep your function simple if you wish.

(f) Apply your fixed point iteration method using both `gNewt` and `g1`. Attempt to measure each root to a tolerance of 10^{-20} using both functions and different starting values. What happens when you seed the fixed-point iteration in a region that is divergent? Do your theoretical predictions of convergence (or not) jive with your computational explorations?

Problem 3 – A Computational Physics Problem (10 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.

Problem 4 – Order of Convergence (10 points).

(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 corresponding to the error 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 $x=(0,15)$, and 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 $m=1$ or $m=2$. Is the value what you expected? Hint: you might consider computing ratios of adjacent iterations. It is non-trivial (and not recommended) to try to fit the error versus number of iterations to an analytical solution for m.