Homework 1

Due: Wednesday 11:59pm, Sep 25, 2024, on Gradescope

Notes: Please read the syllabus for information on programming expectations, the late homework policy, and the collaboration policy. Please submit both the written and coding portions of your homework on Gradescope. The written portion should be submitted as a PDF (prepared however you like, but LaTeX is excellent), with plots and tables in the appropriate places. Please mark the location of each problem in your Gradescope submission—this makes it much easier for us to grade.

Plotting and formatting: Put thought into presenting your data and results in the clearest way possible. Think about what information you want to convey, and how to convey it most effectively. Sometimes, using a table can be useful, but at other times data is best presented using a plot. All tables and plots should be accompanied by some discussion of what we are intended to learn from them.

Choose proper ranges and scales (e.g., semilogx, semilogy, loglog), always label axes, and give meaningful titles. Make sure your axis numbering and labels are large enough to be readable—getting this right will require some exploration in MATLAB, or whatever language you are using. If you do print numbers, in MATLAB for example, use `fprintf` to format the output nicely. Use `format compact` and other format commands to control how MATLAB prints things. When you create figures using MATLAB (or Python or Julia), please try to export them in a vector graphics format (.eps, .pdf, .dxf) rather than raster graphics or bitmaps (.jpg, .png, .gif, .tif). Vector graphics-based plots avoid pixelation and thus look much cleaner.

Code: Try to write clean, concise, and easy-to-read code, with meaningful variable names. Your code must be well-commented. Every line of code should be explained by comments inside the code, unless it is absolutely self-explanatory. Commenting your code is like "showing your work", and grading will take this into account in a similar manner.

You should submit four code files for this assignment: `bisection_tol.m`, `newton_tol.m`, `secant_tol.m`, and `prob4.m`.

---

1. Choose some continuous function $f(x)$ which has exactly one root on the interval $[-1, 1]$ (choose something somewhat interesting; for example, don't choose $f(x) = x$).

   (a) **[3 pts]** Plot the function. Then use the `MATLAB` function `fzero` to find the root; consider this to be the exact solution x^* of $f(x) = 0$.

   (b) **[12 pts]** Using the code `bisection.m` provided to you, run the bisection method for `itermax` $= 30$ iterations. Plot the error $e_k = |x_k - x^*|$ versus the iteration number k for $k = 1, 2, \ldots, 50$ (use `semilogy` to plot the error on a logarithmic scale). Also plot the error bound $E_k = (b - a)/2^k$ discussed in class on the same plot (here, $a = -1$ and $b = 1$). What do you observe? Is the error bound satisfied?

(c) **[5 pts]** Explain why e_k appears as a straight line on your log-lin plot.

2. (a) **[4 pts]** Modify the `bisection.m`, `newton.m`, and `secant.m` codes presented in class to take an additional input: an error tolerance `tol`. You can call these codes `bisection_tol.m`, `newton_tol.m`, and `secant_tol.m`. These functions should terminate when the absolute value of the difference between two successive iterates is less than or equal to `tol`. The codes should also return an additional output `niter`, which is the total number of iterations taken to reach convergence. Each function should still take in the input `itermax`, the maximum number of iterations, and should still take this many iterations if the error tolerance is not reached (with `niter` returned as `itermax` in this case).

   (b) **[4 pts]** Consider the function $f(x) = \cos(\exp(x))$. The only solution of $f(x) = 0$ in $[-1, 1]$ is $x^* = \log(\pi/2)$. Setting `tol` $= 10^{-10}$, verify that the actual error achieved by each of the three methods for this problem is close to, or less than, the specified `tol`. Report your results and write down the lines of code you used to generate them.

   (c) **[4 pts]** Is the error estimation strategy used above (measuring the absolute difference between two successive iterates) completely robust? In other words, is it possible for these programs to return a solution with error larger than the specified tolerance? Explain why or why not, and if such a scenario is possible, propose a modification to the error estimation strategy which is more robust. Is it possible to devise an error estimation strategy which is completely robust, i.e., which cannot possibly return a solution with error larger than the specified tolerance, for the bisection method? How about for Newton's method and the secant method?

   (d) **[8 pts]** Let's assume we trust our original error estimation strategy. Make a plot of the number of iterations taken to solve $f(x) = 0$ versus the specified error tolerance, for tolerances `tol` $= 10^{-1}, 10^{-2}, \ldots, 10^{-10}$. Use the function f given above, with $[a, b] = [-1, 1]$ for the bisection method, $x_0 = 0$ for Newton's method, and $x_0 = 0$, $x_1 = 1$ for the secant method. Make a curve on this plot for each of the three methods (with an appropriate legend to distinguish them; you might want to make the x axis logarithmic and the y axis linear using `semilogx`). What do you observe?

3. This problem will explore a plotting strategy to empirically measure the convergence orders of different root-finders. We know that the bisection method has convergence order $p = 1$, Newton's method has convergence order $p = 2$, and the secant method has convergence order $p = (1 + \sqrt{5})/2$. For this problem, we will use the original `bisection.m`, `newton.m`, and `secant.m` codes which I've provided to you. We will use the same function $f(x)$ as in Problem 2, with the same values for initial guesses as in Problem 2d.

   (a) **[5 pts]** Plot the error $e_k = |x_k - x^*|$ versus the iteration number k for each of the three methods, for $k = 1, 2, \ldots, 50$, on the same plot (make an appropriate legend to distinguish the curves). Use a linear scale on the x axis and a logarithmic scale on the y axis. What do you observe? Is it easy to deduce the convergence orders

of the three methods from this plot?

(b) **[5 pts]** Argue that for k sufficiently large, a method of order p should satisfy

$$\log e_k \approx c + p \log e_{k-1},$$

where e_k is the error at iteration k, and c is some constant.

(c) **[10 pts]** Make a scatter plot (using the `scatter` function) of $\log e_k$ versus $\log e_{k-1}$ (distinguishing the different methods, for example, by using different colored markers, with appropriate labeling in the legend). Also plot the lines $y = px$, for $p = 1, 2, (1+\sqrt{5})/2$ on the same plot (using `hold on` and the `plot` function). Describe what you observe, and explain it. Can you roughly identify the convergence orders of the three methods from this approach? Is there any challenge to doing so? (Note: You may see some points which appear to be outliers compared with your expectation. Do not worry about these; you can think about the explanation, or ask me, but are not required to explain this here.)

4. In this problem, you will write a code that finds *all* of the roots of the function $f(x) = x^5 - 3x^2 + 1$ in the interval $[-2, 2]$. Submit a single program, `prob4.m`, which carries out the task in parts (a) and (b), along with the values of the roots you obtain. You can use any of the functions you have written so far, or which have been provided to you, in this problem. This exercise outlines a typical use case of the Newton's method; we use a robust but slow-converging method (like bisection) to obtain a good initial guess, and then use Newton's method to refine the guess to a highly accurate result.

   (a) **[15 pts]** Using the fact that the roots of f on this interval are separated by at least 0.25, write a program that (i) uses the bisection algorithm to obtain estimates of the roots that are accurate to within 0.1, and (ii) uses these estimates as initial guesses in Newton's method, refining them to obtain the roots to within 10^{-12} error. Report the values of the roots you obtain. (Hint: Some of the intervals you use might not contain a root; in this case, the `bisection` function provided to you will return an empty result. The MATLAB function `isempty` allows you to check if a variable is empty.)

   (b) **[5 pts]** Consider the root in $[-0.75, -0.5]$. Using this scheme, how many Newton iterations were required to obtain the root to within 10^{-12} accuracy? How many iterations of the bisection method, starting from $a = -1$, $b = -0.5$ would have been required to obtain the same accuracy?

5. In this problem, you will derive the convergence rate of the secant method.

   (a) **[5pts]** Show that the secant method

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k)$$

can be rewritten in the form:

$$x_{k+1} = \frac{x_k f(x_{k-1}) - x_{k-1} f(x_k)}{f(x_{k-1}) - f(x_k)}. \tag{1}$$

(b) **[5pts]** Now, denote the root of f by x^*, so that $f(x^*) = 0$. Also assume that f is twice continuously differentiable and that $f' > 0$ and $f'' > 0$ in a neighborhood of x^*. Define the quantity φ to be:

$$\varphi(x_k, x_{k-1}) = \frac{x_{k+1} - x^*}{(x_k - x^*)(x_{k-1} - x^*)},$$

where x_{k+1} is as in (1). Show, for a fixed value of x_{k-1}, that

$$\lim_{x_k \to x^*} \varphi(x_k, x_{k-1}) = \frac{f'(x^*) - \frac{f(x_{k-1}) - f(x^*)}{x_{k-1} - x^*}}{f(x^*) - f(x_{k-1})}$$

(c) **[5pts]** Use this to show that

$$\lim_{k \to x^*} \varphi(x_k, x_{k-1}) = \lim_{x_{k-1} \to x^*} \lim_{x_k \to x^*} \varphi(x_k, x_{k-1}) = \frac{f''(x^*)}{2f'(x^*)}.$$

Hint: Before taking the limit, divide the numerator and denominator of the previous expression by $x^* - x_{k-1}$. Then use l'Hôpital's rule.

(d) **[5pts]** Next, assume that the secant method has convergence order p, that is to say that

$$\lim_{k \to \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} = A < \infty.$$

Using the above results, sketch an argument that $p - 1 - 1/p = 0$, and therefore that $p = (1 + \sqrt{5})/2$. Show also that $A = \left(\frac{f''(x^*)}{2f'(x^*)}\right)^{p/(1+p)}$. This does not need to be rigorous, but the idea should be clear.