Whenever calculations are needed to solve a problem, those calculations (and, usually, your code) must be submitted as part of the homework assignment.

Homework must be submitted electronically. Unless express permission has been given in advance by the instructor for a late homework submission, a 30% percent penalty will be deducted for each late day (or part of a late day).

In printing non-integers, be sure to use scientific notation and to show at least 6 decimal digits following the decimal point.

Exercise 1.1. (Sequences and rates of convergence.) Here are four sequences:

(a) $x_k = \gamma^k^2$ for $k = 0, \ldots$, with $\gamma = 0.95$;

(b) $x_{k+1} = \frac{x_k + 1}{x_k}$, with $x_0 = 9$;

(c) $x_{k+1} = \sqrt{x_k}$ with $x_0 = 3.2$;

(d) $x_{k+1} = \frac{x_k^2}{2x_k - 1}$, with $x_0 = 12$.

(a) Show that each sequence converges to a limit and give the limit $x^*$. Do this in two ways: (i) give a mathematical derivation, including an explanation for each step, and (ii) compute 10 terms (or more, if necessary) of the sequence that are visibly converging to the limit.

(b) Given the value of $x^*$, obtain more information about the first 10 terms of each sequence. For the $k$th term, compute

- $x_k - x^*$;
- $|x_k - x^*| / |x_{k-1} - x^*| \%$ Test 1 in Handout 3
- $|x_k - x_{k-1}| / |x_{k-1} - x^*| \%$ Test 2 in Handout 3

Determine whether the sequence is converging linearly or superlinearly using (i) a mathematical analysis of the given sequence and (ii) your computed numbers.

Exercise 1.2. (Bisection.) This is a “classic” homework problem that anyone who wants to understand bisection should experience.

Consider a nonlinear scalar-valued twice-continuously differentiable function $f(x)$ and assume that you are given two points $a_0$ and $b_0$, the endpoints of an interval such that $f(a_0)f(b_0) < 0$. Write a generic code that applies the bisection algorithm to $f$ in this interval. The input to this code should include a routine that calculates $f(x)$ for any $x$, and the two points $a_0$ and $b_0$.

Your program should stop in one of the following three ways: (i) your code finds $\bar{x}$ where $f(\bar{x}) = 0$, (ii) the length of the latest interval of uncertainty is less than or equal to a specified tolerance $xtol$, or (iii) $maxit$ bisection steps have been performed, where the value of $maxit$ is specified.
At each bisection step, print the iteration index \( k \), \( a_k \), \( f(a_k) \), \( b_k \), \( f(b_k) \), and \(|a_k - b_k|\).

Consider the function
\[
\tilde{f}(x) = f(x) = x^{11} - 11x^{10} + 55x^9 - 165x^8 + 330x^7 - 462x^6 + 462x^5 - 330x^4 + 165x^3 - 55x^2 + 11x - 1, \quad (1.1)
\]
which is algebraically equivalent to \((x - 1)^{11}\).

(a) Show mathematically that \( \tilde{f} \) has only one zero, at \( x^* = 1 \).

(b) Let \( a_0 = 0.9 \), \( b_0 = 1.2 \), \( \text{maxit} = 6 \), and \( \text{xtol} = 0.001 \). Run your bisection program with \( \text{maxit} = 8 \), evaluating \( f \) in exactly the form given in (1.1)—do not simplify the algebra!

Does \( x^* \) lie in the final bisection interval \([a_k, b_k]\)? Explain whether or not the results are what you expected. If not, explain why. What do these results tell us about the practical reliability of the bisection algorithm?

(c) Run your bisection program a second time with \( \text{maxit} = 6 \), this time applied to \( \tilde{f}(x) = (x - 1)^{11} \), evaluated in this form, using the same \( a_0, b_0, \text{maxit} \), and \( \text{xtol} \) as in part (b). Was your bisection routine successful? Comment on any differences between the results of (b) and (c).

**Exercise 1.3.** The purpose of this problem is to give experience with several methods for one-dimensional zero-finding.

Three methods (and associated code) will be considered: your own codes for Newton’s method and the pure secant method, and \texttt{fzero}, the MathWorks implementation of Richard Brent’s \texttt{zeroin}, probably the most used zero-finding method in the world.

The function to which our zero-finding methods will be applied is
\[
f(x) = x^3 + 0.43x^2 - 1.935x - 0.495. \quad (1.2)
\]

The graph of this function is shown below because it may provide insights into some of the behavior that you will observe. This function has three simple real zeros: \((-1.5, -0.25, 1.32)\).

(a) Write codes that implement the two zero-finding methods (pure Newton and pure secant) that were discussed in class.

These codes to be written will by definition be unrealistic because the value of \( x^* \), the exact zero, is assumed to be known, so that the error in each iterate can be calculated exactly. Note, however, that the logic of each iteration does not require \( x^* \).
Each of your two codes should stop (i) if an exact zero $x^*$ is found, i.e., a point such that $f(x^*) = 0$, (ii) after `maxit` iterations, and (iii) if $|f|$ is less than `ftol`, for specified values of `maxit` and `ftol`. In all cases, appropriate values are `maxit = 12` and `ftol = 10^{-14}`, but of course you are free to try others.

The Newton code should perform iterations to generate a new iterate using $f(x)$ and $f'(x)$ at the previous iterate. An initial point $x_0$ must be provided. At iteration $k$ in the Newton code, print $k$, $x_k$, $f_k$, $x_k - x^*$, and the ratio $|x_k - x^*|/|x_{k-1} - x^*|$.

The secant code should generate a new iterate using the values of $f$ at two preceding iterates. Two starting points, $x_0$ and $x_1$, must be provided. At iteration $k$ in the secant code, print $k$, $x_k$, $f_k$, $x_k - x_{k-1}$, $f_k - f_{k-1}$, $x_k - x^*$, and the ratio $|x_k - x^*|/|x_{k-1} - x^*|$.

For each run of each method, please count and print the number of times that $f$ is evaluated before termination, including the evaluation of $f$ at the initial points.

Keep in mind that Matlab starts the numbering of arrays with 1, not with 0!

(b) Run the Newton code with three starting points:

(i) $x_0 = -2$;
(ii) $x_0 = 2$;
(iii) $x_0 = -0.96$.

In each case, comment on and explain the behavior of the Newton iterates: Are they converging? Do they appear to be converging quadratically? Are the iterates converging from one side of $x^*$? Are they converging monotonically, i.e., each iterate is closer to $x^*$ than the previous iterate?

(c) Run your secant code for four different sets of starting points:

(i) $x_0 = -2$ and $x_1 = -0.5$;
(ii) $x_0 = 0.5$ and $x_1 = 1.5$.
(iii) $x_0 = 0.5$ and $x_1 = 2$;
(iv) $x_0 = -0.5$ and $x_1 = 1.5$. Note that $f$ has the same sign at the starting points (iv).

For each pair, comment on and explain the behavior of the secant iterates. Are they converging? Does convergence appear to be superlinear? Do any iterates leave the interval of uncertainty? Are the iterates converging from one side of $x^*$? Are they converging monotonically, i.e., each iterate is closer to $x^*$ than the previous iterate?

For the last pair (iv) of initial points, comment on the effect (of any) of the fact that $f$ has the same sign at those two points.

(d) Run `fzero`, with three of the pairs of points from part (c)

(i) $x_0 = -2$ and $x_1 = -0.5$;
(ii) $x_0 = 0.5$ and $x_1 = 1.5$.
(iii) $x_0 = 0.5$ and $x_1 = 2$.

An explanation of how to call `fzero` can be found using `help` in Matlab and searching for `fzero`. In using `fzero` with two points, the user provides `x0`, which is an interval of the form `x0 = [a b]`, such that $f$ has opposite signs at $a$ and $b$. As discussed in class, the algorithm in `fzero` generates each new iterate using a combination of secant, inverse quadratic interpolation, bisection, and a step of $\delta$.

When calling `fzero` for this problem, you should print $x$ and $f(x)$ for every function call (to track what the algorithm is doing). The online documentation for `fzero` explains how to use an option that prints the ‘algorithm’ executed at every iteration, where the choices are ‘interpolation’ and ‘bisection’.
For each of the three pairs (i)–(iii) of initial points, comment on the behavior of \texttt{fzero} compared to the pure secant method.

You will need to run \texttt{fzero} in a different way for the two initial points in (iv) from part (c) ($x_0 = -0.5$ and $x_1 = 1.5$), since the function values at these points have the same sign. So please make two separate runs of \texttt{fzero} with the option of providing a single point (rather than an interval). Sample code for running \texttt{fzero} with a single initial point is given in the Matlab documentation for \texttt{fzero}.

Based on the results of running \texttt{fzero} with a single starting point, comment on its effectiveness in finding an initial interval of uncertainty. Can you think of a better one?

(e) Summarize and comment on the features of the three zero-finding methods that have been revealed by their performance on these examples. Discuss whether each method’s behavior matched your expectations from the theory, and please comment on any unexpected results.