CS100M/CIS121/EAS121 Spring 2004
Assignment 3: “Title TBA”

Due Wednesday, March 10, 11:59:00 PM

0. Introduction

0.1 Goals

This assignment will help you develop skills in:

- Using conditional and repetition statements.
- Using and designing functions.
- Storing values in arrays.
- Using the symbolic toolbox.
- Creating a larger, more complex program composed of subprograms.
- Solving for roots of equations.

0.2 Instructions

Be sure to read the *entire* assignment before answering the questions! Do the tasks in the following sections. You may work with one partner or by yourself for this assignment.

0.3 Submission

Follow the **Submission Format Requirements** at the CMS Info link on the course website. The last section describes what to submit on CMS.

0.4 Grading

All code that you submit must run without warnings and errors. Otherwise, you will receive a zero. This assignment’s weight counts as **1.5** times that of a typical assignment.

0.5 Academic Integrity

You must abide by the Code of Academic Integrity, which is provided for CS100M, the Department of Computer Science, and Cornell University on our course website. Refer to the link called Academic Integrity.

1. Motivation

Landscaping and construction often require structures to shore up and block portions of the ground from collapsing. Such structures are called ***sheet piles***. They protect workers from being crushed on construction sites. Figure 1 illustrates the cross-section of a sheet pile driven into sandy soil. You should also look around campus at the various construction projects! Various parameters effect the construction of sheet piles, such as loading and soil properties. Key parameters are listed below:

- P : force due to ground weight that pushes the pile to left and causes different pressure distributions to the left and right of the pile.
- γ : unit weight of soil (force/volume).
- ϕ : soil friction angle.
- L : height of sheet pile above ground surface (distance, measure as positive value)
- D : depth of sheet pile below ground surface (distance, measured as positive value)

There are also two key terms that relate to soil friction:

$$K_a = \tan^2\left(45° - \frac{\phi}{2}\right) \tag{1}$$

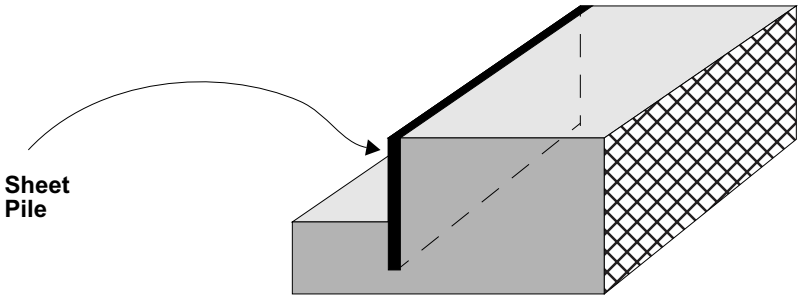
and

$$K_p = \tan^2\left(45° + \frac{\phi}{2}\right). \tag{2}$$

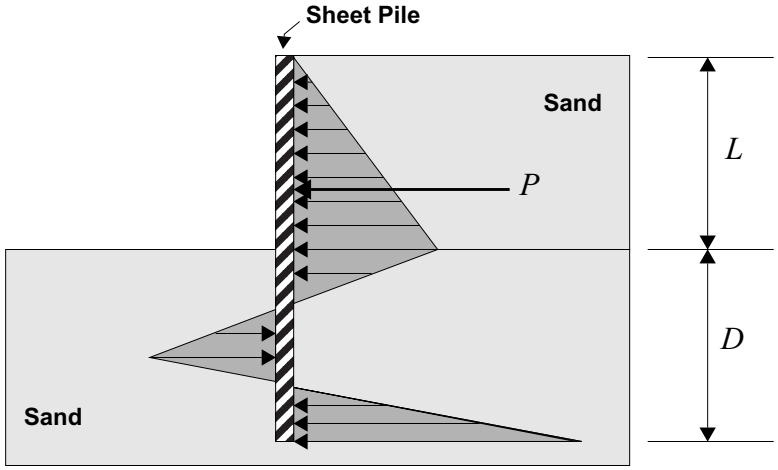
Equation 3 (below) shows the relationship of these parameters and terms:

$$D^4 - \frac{8P}{\gamma(K_p - K_a)}D^2 - \frac{12PL}{\gamma(K_p - K_a)}D - \left(\frac{2P}{\gamma(K_p - K_a)}\right)^2 = 0. \tag{3}$$

Given all other parameters from measurements at a job site, a designer can solve for D to produce the depth to which the sheet pile should be driven into the ground.



Isometric View



Cross-Section

Figure 1: Sheet Pile

2. Mathematical Background

2.1 Terminology

A geotechnical engineer would need to solve Equation 3 for the value of D to determine pile depth as part of a larger design process. In general, the variable D acts as the ***independent variable*** because the value may change. As with any equation, substituting the *correct value* of the unknown independent variable (D in this case) causes the left-hand side to become zero. Such a value *satisfies* an equation and is called a ***root***. For example, you learned in high-school (maybe earlier?) about the general solution to $ax^2 + bx + c = 0$. As in the quadratic equation, some equations generate ***multiple roots*** that all satisfy the equation. So, the engineer must understand the underlying physical model to make sense of the numerical answers. But, what if the equation is more complicated, as in Equation 3? Section 3 reviews a variety of approaches that assist the task of finding roots.

2.2 Equality and Round-off Error

Before solving for roots, consider the following issue, faced in all numerical computations. When using floating-point numbers, computers rarely compute an exact value of an expression. Try computing `1.0/3.0`, for example. Usually, an arithmetic operation introduces a small amount of round-off error at a "distant" decimal place. Though minuscule, this error typically causes failure of an equality comparison. When comparing two floating-point numbers, use a ***tolerance***, a small numerical value below which you "don't mind" a difference. Many texts denote tolerance as ε and use the following equation to compare two quantities x and y for approximate equality:

$$|x - y| \leq \varepsilon . \tag{4}$$

For instance, the numbers 1.0001 and 1.0002 are essentially "equal" within a tolerance of 0.0001. Refer to Chapman, pp. 89–90, for further details.

2.3 Angles

Many computer languages require angles in terms of ***radians***. The formula for converting between degrees and radians is as follows:

$$\frac{degrees}{360^\circ} = \frac{radians}{2\pi} . \tag{5}$$

Consult `help deg2rad` for a built-in MATLAB function that performs the conversion.

3. Solution Techniques

Refer to Section 1 and Equation 1. We need to try various algorithms to solve for the sheet-pile depth D given a pre-determined amount of round-off error. This section provides an overview of some root-solving techniques that will help you.

3.1 Guessing

Yes, you could feasibly just keep guessing random values for the independent variable. Sooner or later (most likely later) you will find a root because the equation will produce a value within your tolerance. Obviously, this approach is wildly inefficient…you need an algorithm that applies a bit more control.

3.2 Experimental

Rather than near-random guessing, you could visually inspect the equation. How? Set the equation to a ***dependent variable***, as in

$$y = f(x). \tag{6}$$

The "body" of the equation is $f(x)$, where x is the independent variable. Values of x that generate values of $y = 0$ (within a tolerance) are the roots. So, you would pick a large interval (domain) of values of x and plot the corresponding values of y (range). Simply looking at the plot will tell you the approximate location of the root.

3.3 Exact

Sometimes you might be able to find an ***analytical solution*** to the equation, depending on the complexity of the equation. An analytical solution is an exact symbolic solution to a mathematical problem. In the case of a 4th-order polynomial, a rather lengthy formula exists for an exact solution.

3.4 Numerical Analysis

When you inspect the complexity of the exact formula, you will discover why a numerical approach is more practical than an analytical solution. Although the numerical answer will only approximate the analytical, or true, solution, all you need is a value within an acceptable tolerance. In fact, soil properties are relatively uncertain, so a very precise answer will not improve the solution. Another advantage of a numerical solutions is that you can run a program many times for different parameter values, i.e., conduct a ***parametric study***.

3.4.1 LHS/RHS Method

This section describes a brute-force numerical technique. Rather than relying on random guessing, the left-hand side/right-hand side (***LHS/RHS***) technique, as denoted by DIS, introduces more control and less randomness in the search for the root. LHS/RHS searches for a solution within an assumed interval, according to this algorithm:

- Pick a starting point (initial value of D) and direction to iterate.
- Substitute D into the equation and check if the resulting value meets the tolerance.
- If not, increment D until:
  – the tolerance is satisfied
  – the equation changes sign
  – an excessive amount of iterations has occurred
- If the equation changes sign before meeting the tolerance, the direction must reverse, using a smaller increment. Why? The increment was too large, forcing the analysis to miss the root.

By "bouncing" back and forth with smaller and smaller increments, the root will eventually be reached, assuming a smooth and continuous curve. On the other hand, if the user initially picked the wrong direction, the algorithm might continue forever because the root is in the other direction. So, the algorithm must decide what counts as *excessive iterations* and start over again. Of course, this technique leaves much to be desired. The next section presents a more refined technique.

3.4.2 Bisection Method

A still-somewhat-brute-force-but-not-as-brutal technique involves a bit more strategy. What happens to a function $f(x)$ when the independent variable x becomes a root? The function either touches or crosses the independent variable's axis. For a simple case, consider the line $y = x + 1$. Using $x = -1$ and $y = 0$, try the following:

- Pick a value $x > -1$. To the right of $x = -1$, y is positive.
- Pick a value $x < -1$. To the left of $x = -1$, y is negative.

Thus, tracking where a function changes sign helps locate a root, as shown in the LHS/RHS method. But, a more refined technique called the ***bisection method*** searches for a root by investigating sign changes within intervals. Refer to Figure 2 and the following algorithm:

- Pick a starting point, x_L, and an end point, x_R such that the interval between x_L and x_R contains the root. Yes, you do have to *guess* a proper interval.
- Compute the midpoint $x_M = \frac{x_L + x_R}{2}$.
- Using x_M, compute the equation $f(x_M)$.
- Check the tolerance of $f(x_M)$.
- Until $f(x_M)$ meets the tolerance:
  - Compute x_L, x_R, x_M, $f(x_L)$, $f(x_R)$, and $f(x_M)$.
  - If $f(x_M)f(x_R) < 0$, then x_M is to the left of the root: x_M becomes the new x_L.
  - If $f(x_M)f(x_L) < 0$, then x_M is to the right of the root: x_M becomes the new x_R.

What if the user picks an interval that does not contain a root? The algorithm might keep iterating forever unless the user supplies a stopping condition to prevent excessive iteration. Also, where does that initial interval come from? The next method bypasses this issue.

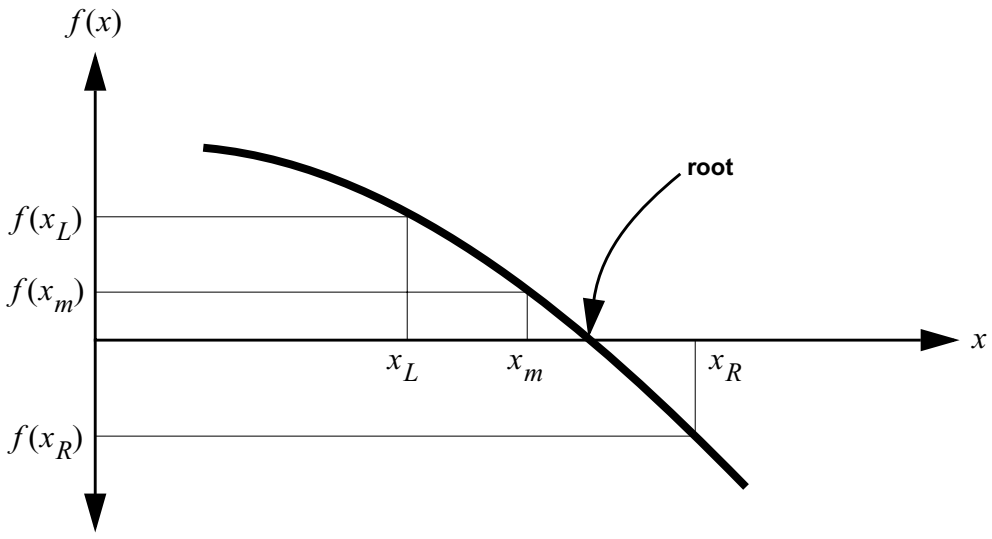


Figure 2: Bisection Method

3.4.3 Newton's Method

A more efficient method adapts the notion of a Taylor series. Given a point x_0 "close" to a root of $f(x)$, you can expand $f(x)$ as follows:

$$f(x) = f(x_0) + (x - x_0) f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) + \ldots . \tag{7}$$

Setting Equation 7 to zero implies that x is a root of $f(x)$. Leaving out all higher order derivatives gives an approximate equation:

$$0 \approx f(x_0) + (x - x_0) f'(x_0) . \tag{8}$$

Rearranging the terms in Equation 8 yields

$$\delta = x - x_0 = -\frac{f(x_0)}{f'(x_0)} . \tag{9}$$

Refer to Figure 3. You can use Equation 9 to implement this efficient algorithm

- Pick an initial value x_1.
- Calculate $f(x_1)$.
- While tolerance is not met, iterate as follows:
  – Compute the value of $f'(x)$.
  – Compute the new value of x (denoted x_2) using $x_2 \leftarrow x_1 - \frac{f(x_1)}{f'(x_1)}$
  – Compute the new value $f(x_2)$.

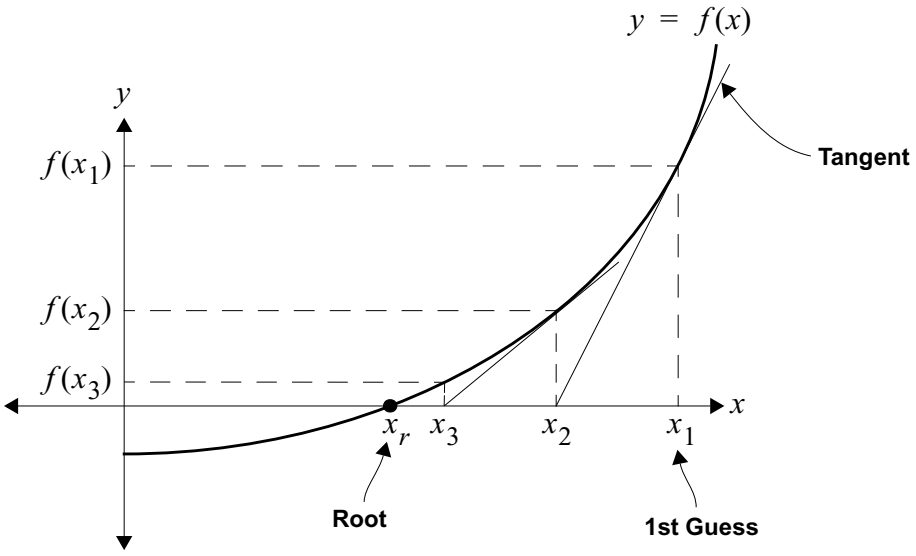


Figure 3: Newton's Method

The approach may "meet tolerance" by either checking the function or the relative change $x_2 - x_1$. Why check $x_2 - x_1$? As the independent variable approaches the root, each new increment shrinks. But, when the function is "steep," checking tolerance with function values $f(x_2) - f(x_1)$ is better. As a safe guard, the algorithm should also check for excessive iteration.

4. Problem/Tasks

Your job is to write code that implements numerical techniques that find the necessary depth to drive a sheet pile given a set of parameters. We have also included other tasks and suggestions that relate to the development of your solutions.

4.1 Run Our Code!

To assist with your development, we have provided a ***P-Code*** version of the entire solution for you to test. Before working on your own code, run the program and try different inputs, both legal and illegal, to learn how your program is required to work. All you need to do is make sure you set the path/current directory, unzip the p-code solution, and enter **`a3`** at the command prompt in MATLAB.

4.2 Data

We are including various functions that perform user input and obtaining parameter values. The function **`getParams`** will return the data we are assuming for this assignment: $\gamma = 18$ kN/m^3, $\phi = 30°$, $L = 3$ m, $P = 30$ kN/m, error tolerance of $\varepsilon = 0.01$, and maximum iterations = 1000. Because **`getParams`** will return multiple values, you would include the following statement in your solutions for each approach:

```
[GAMMA PHI LENGTH FORCE EPS MAXITERS] = getParams ;
```

Note that **`getParams`** takes no input arguments and that some tasks that you develop (below) do not use all of the values.

4.3 Guessing Method

Write a script called **`guessmethod`** that guesses just one pile depth and reports whether or not the guess is correct. This script will call a function called **`evalPileDepthEqn`** that we have provided.

4.4 Experimental Method

Create a script called **`manualmethod`** that performs the experimental approach. This script will produce a plot of the equation for a range of pile depths with these steps:

- Obtain parameters from **`getParams`**.
- Prompt the user for the lowest and highest value of D to use.
- Plot the pile depth equation for 100 increments in the specified interval for D.
- Prompt the user to try a new set of parameters.

Note that this script does not find or indicate a solution!

4.5 Exact Method (Optional)

For 10 bonus points, use MATLAB to solve the 4th order polynomial analytically and produce the correct root. If you choose to attempt this portion, call your script/function **`exactmethod`**.

4.6 LHS/RHS Method

Create a script M-File called **lhsrhsmethod** that solves for the depth, using the algorithm in Section 3.4.1. Your script will do the following:

- Obtain parameters from **getParams**.
- Sets an initial increment of 0.01.
- Prompts the user for a legal initial value (non-negative real number) and legal direction (**pos** or **neg**). You must ask for each input separately, as shown in our solution. Keep prompting the user if they enter illegal values.
- Use the LHS/RHS method to find a value of depth that satisfies Equation 3. Remember to use **EPS** to test the tolerance.
- Ensure that the program produces the correct result by using **MAXITERS** to determine if the user made poor choices of initial direction and the starting point.
- If original direction is **neg** and D becomes less or equal than 0, change the direction to **pos**.
- Reverse direction and divide your increment by 10 if the pile depth equation changes sign.
- Output the solution, value of the equation for the solution, and number of iterations to find the solution.

4.7 Bisection Method

Create a script M-File called **bisectionmethod** that solves for the depth. Follow the algorithm in Section 3.4.2. Your script will do the following:

- Obtain parameters from **getParams**.
- Prompts the user for an initial interval. The user will be prompted for legal values of each end point of the interval. If the interval is illegal (contains negative value or end points reversed), the program will re-prompt the user.
- Use the Bisection method to find a value of depth that satisfies Equation 3. Remember to use **EPS** to test the tolerance.
- Ensure that the program produces the correct result by picking a new interval if **MAXITERS** is exceeded.
- Output the solution, value of the equation for the solution, and number of iterations to find the solution.

4.8 Newton's Method

Create a script M-File called **newtonsmethod** that solves for the depth. Follow the algorithm in Section 3.4.3. Your script will do the following:

- Obtain parameters from **getParams**.
- Use Newton's method to find a value of depth that satisfies Equation 3. Remember to use **EPS** to test the tolerance.
- Ensure that the program produces the correct result by picking a new starting point if **MAXITERS** is exceeded or a negative root is found.
- Output the solution, value of the equation for the solution, and number of iterations to find the solution.

4.9 "Main Program"

Ensure that your scripts may be called by the supplied scripts **a3**, which has been provided for you. It automates the calling of all the scripts in one convenient package! You do not need to write anything for this task.

4.10 Feature List

If you find that you need additional functions to clarify your code and/or reduce redundancy, feel free to write them as needed. Include a `README.txt` file that explains what you have included. If you do not feel you have included more than what this document specifies, be sure to that fact as well.

Sometimes students wish add additional features that exceed the specifications of the assignment, as in using GUIs. We allow such enhancements as long as they do not interfere with the specifications and expectations of the assignment. In such cases, we have occasionally awarded bonus points for exceptional work. Note that bonus points to not raise the core-point total of the assignment.

4.11 Discussion

In a file called `discussion.txt`, answer the following questions:

1. How closely do the results match from each of the approaches?
2. Which technique is most efficient? Why?
3. How do the LHS/RHS and bisection methods compare to algorithms discussed for number guessing, which has been discussed in lecture? Compare and contrast these problems and their algorithms.

4.12 Advice

We suggest that you follow this advice:

- Develop all algorithms on paper first!
- Try to figure out various test cases before programming. Test along the way.
- Supply dummy input parameters to each technique so that you can trace its code.
- Plan out how each module is called. Refer to `a3`.
- Create empty versions of each script and function that include only comments and perhaps simple dummy return/output values.
- Write and test each ***module*** (individual script with specific purpose) *individually.* Confirm that the module fits into the overall program before working on the next module.

5. Submitting Your Work

Submit a zip file called `a3.zip` that contains these files:

- `guessmethod.m`
- `manualmethod.m`
- `exactmethod.m` (if you chose to do it)
- `lhsrhsmethod.m`
- `bisectionmethod.m`
- `newtonsmethod.m`
- `README.txt`
- `discussion.txt`
- all other functions/scripts that you have developed.

Do not submit files that we have provided! We will supply our own versions! If you modify our supplied files, chances are that your solution will not work.