ME 2450 Lab 5: Root Finding - Open Methods

Objectives

Write a program that uses the Newton-Raphson method to aid in designing a solar pond that achieves a specified density.

References

- Optional Function Arguments in MATLAB and Python, *OptionalArgs.pdf*.
- An Introduction to Function Handles, *FunctionHandles.pdf*.
- Lecture 06: Roots of Equations, Open Methods, *Lecture06.pdf*.
- Chapter 06, *Chapra, Numerical Methods for Engineers (7th Edition)*.

Physical Model

In class, we covered the main idea behind solar ponds in *Lecture 06, Slide 29*. A solar pond is defined as a large body of salt water that collects and stores solar thermal energy. The solar pond uses salt as a mean to suppress convection in the fluid. This allows lower layers of salt water in the pond to be heated to very high temperatures with minimal heat loss, since convection would cause heat loss.

The density, ρ, of the salt water in the gradient zone of a solar pond (Figure 1) is a function of the depth, z. For a solar pond, the density in the gradient zone is:

$$\rho(z) = \rho_0 \sqrt{1 + \tan^2\left(\frac{\pi}{4}\frac{z}{H}\right)} \tag{1}$$

where H is the height of the gradient zone and ρ_0 is the density of the water in the surface zone. Equation (1) gives you ρ when z is known, but we want to solve the inverse problem: finding how deep the pond needs to be to achieve a desired density (that is, find z when your desired ρ is known). We will use Newton-Raphson method to accomplish this.

Numerical Methods

Newton-Raphson Method

For this lab, you will write a Newton-Raphson function. As we have discussed in class, the Newton-Raphson method is based on the first-order Taylor series approximation of the function

$$f\left(x_{i+1}\right) = f\left(x_i\right) + f'\left(x_i\right)\left(x_{i+1} - x_i\right),$$

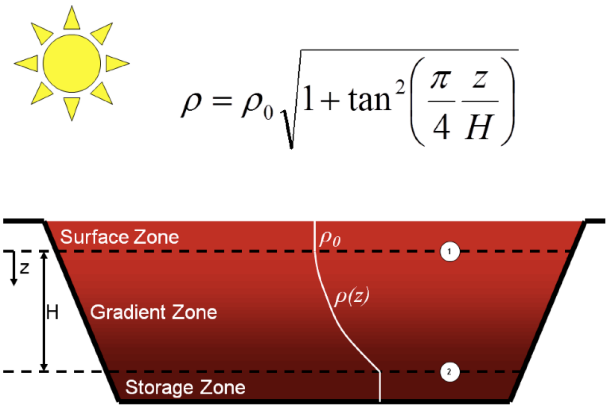

Figure 1: A Solar Pond. The density is constant in a Surface Zone of some given depth. There follows a Gradient Zone whose density varies with depth. Finally, the deepest part of the pond, the Storage Zone, has a constant, higher density. Some solar ponds' densities follow the equation given here, while others have a different relationship between density and depth.

and the assumption that the function value will be root at the next step (i.e., $f(x_{i+1}) = 0$). We then solve the Taylor Series approximation for the next estimate of the root:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{2}$$

This formula can also be understood graphically. Suppose that we have an estimate for a root, x_0, of a function. If we were to draw the tangent line (using the point-slope formula, $f(x) - f(x_0) = f'(x_0)(x - x_0)$) from the point $(x_0, f(x_0))$ until it intersects the x-axis (at the point $(x_1, 0)$), we would have a new estimate, x_1, for the root of the function. We could then find the point $(x_1, f(x_1))$ and the slope $f'(x_1)$ and draw the tangent line to find the point $(x_2, 0)$. If we then repeat the process, we could refine the estimate until we achieve some desired level of accuracy. This process is summarized in Algorithm 1.

Data: x_{root}: initial guess for root; ϵ: the error tolerance
Result: x_{root}: the converged root, if found.
while *maximum iterations not exceeded* **do**
  $dx = -\frac{f(x_{root})}{f'(x_{root})}$! increment in x
  $x_{root} = x_{root} + dx$! updated root
  if $|dx| \leq \epsilon$ **then**
    Exit loop
  end
end
return x_{root}

Algorithm 1: Newton-Raphson method algorithm.

The method is illustrated in Figure 2. In this figure, the function $y = x^3 - 4x^2 + 2x + 2$ is plotted. The root is determined to lie near the point $x_0 = 1$ and so this is chosen to begin the Newton-Raphson Method. The first four steps of the method are shown. The root estimate can be seen to converge to within 2% relative error after only two iterations. By the fourth iteration, the error is less than 10^{-5}%.

The actual value of the root is 1.311, which the method has identified to 3 significant digits at the second iteration.

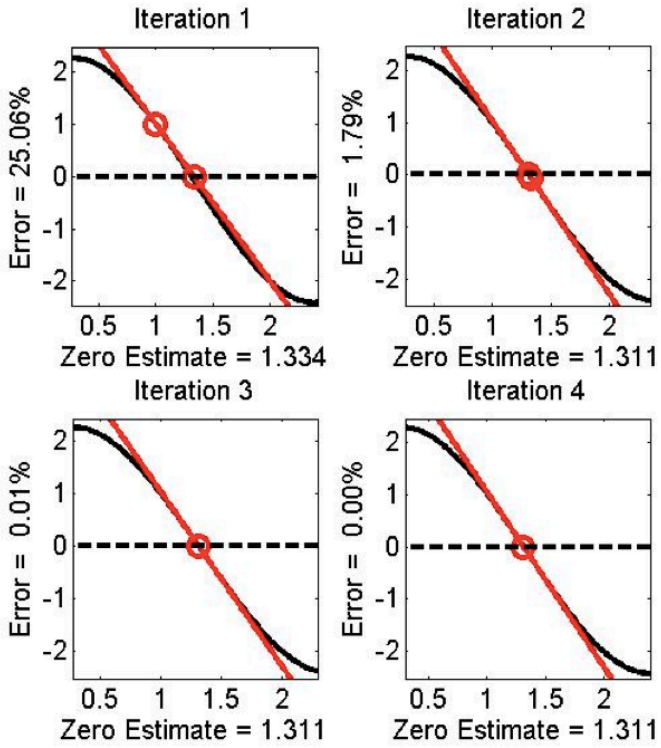

Figure 2: Using the Newton-Raphson method to find the root of $y = x^3 - 4x^2 + 2x + 2$. The black line is the function, while the red line is the tangent to the function. The red circles are the points (x_{root}, y_{root}) and $(x_{new}, 0)$. The relative percent error and new estimate for the root are given at each iteration.

Be aware that we are neglecting several things that could potentially cause a problem in more complicated situations, but which will simplify the programming for this lab. We are not going to verify that the derivative is not zero (or very close to zero), so if we are near a local maximum or minimum, our algorithm will most likely fail. We are not considering what to do if the root is near $x = 0$, in which case the percent relative error formula could create a problem, since it would try to divide by zero, or a very small number. We are also not going to consider the problem of multiple roots.

Lab Assignment

The driver script you will write, `find_pond_depth`, will depend on three functions:

- `newton`
- `solar_pond`
- `solar_pond_deriv` (analytical derivative) and `central_difference` (numerical derivative)

`find_pond_depth` will call `newton`, which will call `solar_pond` and either `solar_pond_deriv` or `central_difference`, depending on user input.

Newton-Raphson Method

Write a function `root = newton(fun, x0, fprime, tol, maxiter)` that computes the root of the function `fun` using Newton-Raphson method (Algorithm 1). The function shall be implemented in a file named `newton.[m,py]` (MATLAB or Python, respectively).

Input Arguments:

- `fun` (callable): Function for which you are to find a root.
- `x0` (scalar): Initial estimate of the root.
- `fprime` (callable, optional): The partial derivative of `fun` with respect to the independent variable. In our case, the independent variable is pond depth, z, so `fprime` should give the derivative of `fun` with respect to pond depth z.
- `tol`: (scalar, optional) Stopping tolerance.
- `maxiter`: (scalar, optional) Maximum number of iterations.

Output Arguments:

- `root` (scalar): final estimate of the root to within specified error tolerance.

Requirements:

1. Your function should check whether the initial guess is a root. If it is, it should return the guess as the correct answer.
2. If the root is not determined to within the desired tolerance in the maximum number of allowed iterations, an error should be issued (use the `error` function in MATLAB or raise a `RuntimeError` in Python).
3. In the Newton-Raphson step, the increment in the root is $-f(x)/f'(x)$. Your could should check the special case that $f'(x) = 0$. If $f'(x) = 0$ your function should emit a warning and return the current value of the root.
4. The inputs `fprime`, `tol`, and `maxiter` must be implemented as optional arguments. `fprime` should default to `'none'` (in MATLAB) or `None` (in Python) if user input is not provided. `tol` should default to $1e-6$. `maxiter` should default to 50.
5. If the `fprime` parameter is set to `'none'` (either by the user or by default), the function derivative should be approximated by the central difference method. The file `central_difference.[m,py]` implements the central difference approximation and is provided to you for approximating the derivative. To use the `central_difference` approximation, if needed, include the following near the beginning of the `newton` function:

   - in MATLAB

     ```
     if ~isa(fprime, 'function_handle')
         fprime = @(x) central_difference(func, x, 1e-4);
     end
     ```

     Be sure that `central_difference.m` is in the same directory as `newton.m`.

   - in Python

```
if fprime is None:
    fprime = lambda x: central_difference(func, x, 1e-4)
```

Be sure to include the line

```
from central_difference import central_difference
```

near the top of `newton.py`.

Solar Pond

Write a function `fd = solar_pond(depth, density, density_top, zone_height)` that implements the function you're trying to find the root of (cast Equation (1) as a root-finding problem). The function should be implemented in a file named `solar_pond.[m,py]` (MATLAB or Python, respectively).

Input Arguments:

- `depth` (scalar): depth in the gradient zone, measured from the top of the gradient zone (z).
- `density` (scalar): the density for which you're trying to find the corresponding depth.
- `density_top` (scalar): density at the top of the gradient zone (ρ_0).
- `zone_height` (scalar): height of the gradient zone (H).

Output Arguments:

- `fd` (scalar): the value of the function you're trying to find the root of, at the current root guess.

Note:

- Make sure you understand for which function you are finding the root. (Hint: You are **not** trying to find the root of Equation (1) itself, but a certain form of Equation (1).)

Solar Pond Derivative

The Newton-Raphson method also requires evaluations of the derivative in each iteration. Write a function `fd = solar_pond_deriv(depth, density_top, zone_height)` that implements the derivative of the function you're trying to find the root of. The inputs will be identical to those of `solar_pond`, except that `density_top` is not needed. `solar_pond_deriv` returns the value of the derivative of our expression at a given value of z.

If `newton` is passed a value of `None` (Python) or `'none'` (MATLAB) for the `fprime` function handle, the derivative must be approximated by central difference. `central_difference` performs this approximation. The code for `central_difference` will be provided to you on Canvas, and no alterations need to be made to it.

Find Pond Depth

Write a driver script `find_pond_depth` that determines the depth z in a solar pond at which a given density occurs. The function shall be implemented in a file named `find_pond_depth.[m,py]` (MATLAB or Python, respectively).

Requirements:

`find_pond_depth` should:

1. initialize all necessary input parameters for `newton`, `solar_pond`, and `solar_pond_deriv`.
2. establish function handles for both `solar_pond` and `solar_pond_deriv` and pass them (along with the appropriate inputs) to `newton`.
3. call `newton` to find the depth corresponding to a density of 1200kg/m^3 for $\rho_0 = 1100\text{kg/m}^3$ and $H = 5\text{m}$.
4. after calling `newton`, plug the root back into `solar_pond` and print the output. What value should this output be? Make sure it is what you expect
5. print the density and depth you find, clearly labelled, to the command window.
6. It is always a good idea to look at a plot of the function for which you are trying to find a root to graphically determine a good starting point. The Objectives section at the start of this document describes the plot you are required to make in more detail.

Scoring of Lab Exercises

- ☐ (1 point) Plot the function you're trying to find the root of, in order to help you identify a good initial guess of the root. Give your plot a title, and label the vertical and horizontal axes. Turn this plot in with the rest of your code.
- ☐ (3 points) Write a function `newton` that finds the roots of a nonlinear equation using Newton's method. The function is to be implemented in MATLAB or Python.
- ☐ (2 points) Write a function `solar_pond` that, along with either `solar_pond_deriv` or `central_difference` (provided on Canvas) will be fed to `newton` as input in order to find the depth at which a given density is found in a solar pond.
- ☐ (3 points) Write a driver script called `find_pond_depth` that defines all necessary input parameters and function handles, and calls `newton` in order to compute the depth where the desired density, 1200kg/m^3, occurs.
- ☐ (1 point) How can you verify that your answer is correct? And does it make physical sense? Discuss this in a couple short sentences. Also, comment on any differences you see between your answer calculated with `solar_pond_deriv` as the derivative vs. your answer calculated with `central_difference`. Turn your writing in with all the other code used in this lab.