ME 2450 Lab 04: Root Finding - Bracketing Methods

Objective

Write a computer program that calculates head loss in a section of a pipe.

References

- Optional Function Arguments in MatLab and Python, *OptionalArgs.pdf*.
- An Introduction to Function Handles, *FunctionHandles.pdf*.
- Lecture 05: Roots of Equations, Bracketing Methods, *Lecture05.pdf*.

Physical Model

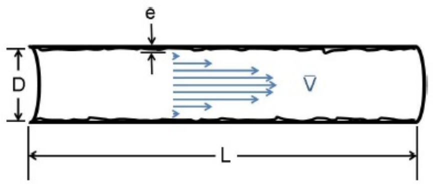


Figure 1: Pipe Flow Parameters

For this lab, you will be applying root-finding techniques to the analysis of fluid flow through a pipe (Figure 1). When analyzing such internal flow, we often want to calculate the head loss (a relative measure of pressure drop) in a section of pipe. One of the components of head loss (h_L) can be calculated using the formula:

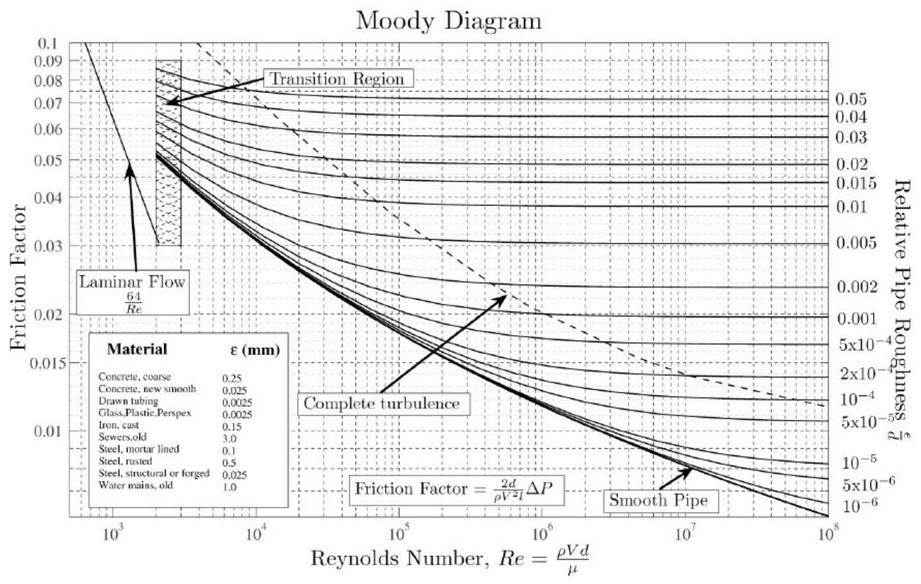
$$h_L = f\frac{L}{D}\frac{\bar{V}^2}{2g} \tag{1}$$

where L is the length of the pipe, D is the diameter, $\bar{V}$ is the average velocity of the flow in the pipe and f is the friction factor. An estimate for the friction factor can be found using the Colebrook equation:

$$\frac{1}{\sqrt{f}} = -2.0\log_{10}\left(\frac{e/D}{3.7} + \frac{2.51}{Re\sqrt{f}}\right) \tag{2}$$

where e is the roughness of the pipe and $Re = \frac{\rho\bar{V}D}{\mu}$ is the Reynolds number. ρ and μ are the density and viscosity of the fluid, respectively. The solutions to Equation 2 are plotted in Figure 2.

While it is possible to solve Equation 2 explicitly for f, see More [1], doing so is not easy or straightforward. It is therefore a candidate for numerical root-finding methods.



Material	ε (mm)
Concrete, coarse	0.25
Concrete, new smooth	0.025
Drawn tubing	0.0025
Glass,Plastic,Perspex	0.0025
Iron, cast	0.15
Sewers,old	3.0
Steel, mortar lined	0.1
Steel, rusted	0.5
Steel, structural or forged	0.025
Water mains, old	1.0

Figure 2: The Moody Diagram - Solutions to the Colebrook Equation [2]

Numerical Methods

Root Finding: The Bisection Method

The bisection method is a numerical root-finding technique that searches for roots of an equation by repeatedly decreasing the size of the interval in which the root exists. The bisection method is robust: it guarantees convergence, albeit slowly. To be a candidate for the bisection method, the function $f(x)$ must be continous on an interval $[a, b]$ that brackets the root, meaning that the signs of $f(a)$ and $f(b)$ must be opposite. By the intermediate value theorem, the function f will have at least one root in the interval (a, b).

For an acceptable function, f, defined on the interval $[a, b]$ containing at least one root (solution to $f(x) = 0$), the bisection method works by determining if the root is contained in the lower half of the interval $[a, (a+b)/2]$ or the upper $[(a+b)/2, b]$. If the root is contained in the lower half of the interval, that is if the signs of $f((a+b)/2)$ and $f(b)$ are the same, the interval is redefined as $[a, (a+b)/2]$. Likewise, if the root is contained in the upper half, the interval is redefined as $[(a+b)/2, b]$.

This process is continued until the interval is sufficiently small, $|(b-a)|/2 < tol$, or such that the function evaluates to near zero at the root estimate, $f((a+b)/2) < tol$, or where $\epsilon_a < tol$ (see Equation (3)). A user-defined tolerance can be used to define the tol value in each case.

$$\epsilon_a = \left| \frac{c_{\text{new}} - c_{\text{old}}}{c_{\text{new}}} \right| \cdot 100\% \tag{3}$$

Once one of the above conditions is met, then the `bisection` `while` loop should be terminated.

The process is illustrated in Figure 3. In this figure, the bisection method is used to find the root of the function $f(x) = 6x^3 - 5x^2 + 7x - 2$. The root is determined to lie in the interval $[0, 1]$ and so these limits are chosen to begin the bisection method. The first step is to find the midpoint ($x = 0.5$). The sign of the function values at the end points and the midpoint are found. Since $f(0)$ is negative, while $f(0.5)$ and $f(1)$ are positive, the next interval is chosen to be $(0, 0.5)$. This interval is plotted, and the process is repeated. The first four steps of the Bisection Method are shown. The root estimate can be seen to converge toward the actual value of $\frac{1}{3}$.

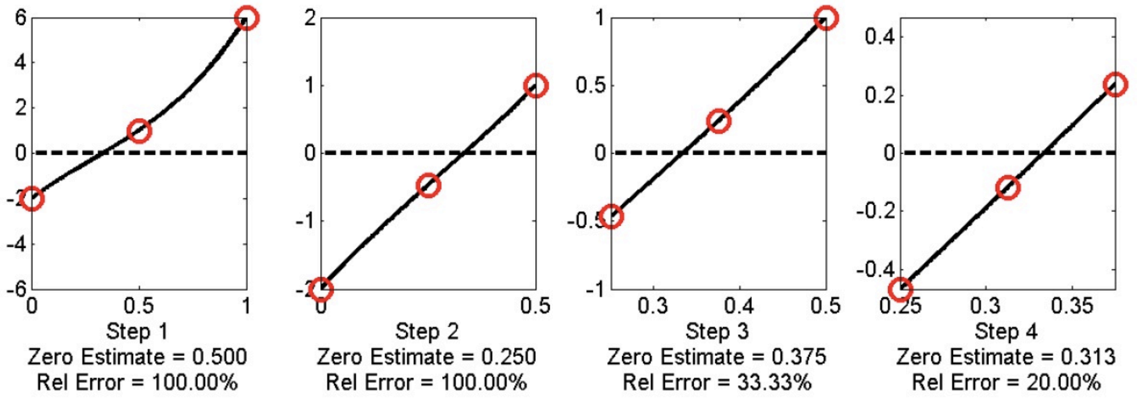


Figure 3: The bisection method used to find the root of $f(x) = 6x^3 - 5x^2 + 7x - 2$ in the interval $(0, 1)$.

The bisection method is implemented as shown in the following pseudocode:

```
Data: $[a, b]$: initial interval, bracketing the root
Data: $tol$: the tolerance
Result: $c$: the root, if found
while iters < maxiter do
    iters = iters + 1
    Calculate the new midpoint
    $c = \frac{a+b}{2}$
    if iters>1 then
        Calculate approximate normalized error:
        $\epsilon_a = \frac{(c - c_{old})}{c}$
        if $|(b - a)|/2 < tol$ or $|f(c)| \leq tol$ or $|\epsilon_a| \leq tol$ then
            Exit loop
        end
    end
    if Sign of $f(c)$ is same as the sign of $f(a)$ then
        Set $a = c$
    else
        Set $b = c$
    end
    $c_{old} = c$
end
return $c$
```

The bisection pseudocode can also be found in Figure 5.10 of the textbook and in the Lecture06 slides.

The number of iterations, n, required to obtain a desired approximate error, $E_{a,d}$, can also be computed:

$$n = \frac{\log\left((b^0 - a^0)/E_{a,d}\right)}{\log 2} \tag{4}$$

See Section 5.2 of the textbook and Lecture 5 for further details, if needed.

Lab Assignment

Bisection Method

Write a function `c = bisection(fun, a, b, [optional arguments])` that computes the root, `c`, of a function, `fun`. The `bisection` function should be implemented in a file named `bisection.[m,py]` (MatLab or Python, respectively).

Input Arguments:

- `fun` (callable): Function handle for which you are to find a root. The function, `fun(x)`, must return a scalar.
- `a`, `b` (scalar): initial limits. The interval `[a b]` must bracket (at least) one root.
- `tol` (scalar): the desired approximate normalized tolerance.
- `maxits` (scalar): the maximum number of iterations that can be completed if the defined tolerance is not achieved.

Output Arguments:

- `c` (scalar): The estimated root.

Optional Input Arguments:

- `tol` (scalar): Stopping tolerance. Defaults to `1e-6` if not provided.
- `maxiter` (int): Maximum number of iterations. Defaults to `10` if not provided.
- `plot_output` (boolean): Graphical output display flag. If `true` (`True` in python), plot the updated guess at each iteration. Defaults to `false` (`False` in python) if not provided.

Additional Requirements:

1. In the bisection function, three of the input parameters are optional, with sensible defaults set if the parameter is not specified by the user. See "Optional Function Arguments in MatLab and Python" in *OptionalArgs.pdf* for an explanation of how optional arguments with defaults are handled in MatLab or Python.
2. The graphical output you should display is similar to that shown in Figure 3. At the very least, it should plot the three points $(a, f(a))$, $(c, f(c))$, and $(b, f(b))$ at each iteration and all plots should be appropriately labeled.
3. Your function should also verify that the value of the function `fun` at the limits `a` and `b` is such that these limits bracket a root (i.e., if $f(a) > 0$ then $f(b) < 0$, or vice versa). If the limits do not bracket a root, an error should be issued (use the `error` function in MatLab or raise a `ValueError` in Python).

4. If the root is not determined to be 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).

Colebrook equation

Write a function, `colebrook_equation(f,e,D,Re)`, that is used by `bisection` to estimate the friction factor. The `colebrook_equation` function should be implemented in a file named `colebrook_equation.[m,py]` (MatLab or Python, respectively).

- `f`: The friction factor which is a variable as described by the function handle written in the driver script.
- `e`: Roughness of the pipe
- `D`: Pipe diameter
- `Re`: Reynolds number

Inputs:

Variable	Value	Units
e	0.0002	[m]
D	0.25	[m]
ρ	1000	[kg/m^3]
μ	6e-4	[Pa/s]
V	10	[m/s]

Outputs:

- Evaluated value of the Colebrook equation.

Calculating Head Loss

A function, `calculate_head_loss`, must be written to compute the head loss. The script `calculate_head_loss.[m,py]`, which is the driver script, will be covered in class. The script implements (1) to determine head loss. The friction factor, modeled by (2), is determined using `bisection` function.

Inputs:

Variable	Value	Units
L	110	[m]

References

[1] Ajinkya A. More, *Analytical solutions for the Colebrook and White equation and for pressure drop in ideal gas flow in pipes*, Journal of Chemical Engineering Science, **61** (2006) 5515-5519.

[2] http://en.wikipedia.org/wiki/Moody_chart

Scoring of Lab Exercises

☐ (6 points) Write a `colebrook_equation` function that implements the Colebrook equation, Equation (2).

☐ (12 points) Write a `bisection` function that finds the roots of a nonlinear equation meeting all the above requirements.

☐ (4 points) Write and run `calculate_head_loss` and report to your TA the resultant head loss.

☐ (5 points) Compute the expected number of bisection iterations to achieve $E_{a,d} = 1\%$ using Equation (4) and compare with the number of iterations required by your `bisection` function.

☐ (3 points) Using Figure 2, explain if the obtained root for the friction factor is reasonable.