Introduction to Computers and Engineering Problem Solving

Spring 2005

Problem Set 8: Ship masts, and river flows

Due: 12 noon, Session 29

Problem 1: Sailboat masts

Wind forces on the mast of a sailboat vary as shown in Figure 1 below. The sail is roughly triangular and is attached to the mast at all points $z = 0$ to $z = 30$. The force is lower near the top of the mast because sail area is less. The force is at a maximum in the lower portion of the mast, but goes to zero at the bottom of the mast, which is also the bottom of the sail and also due to boundary effects of the sailboat hull.

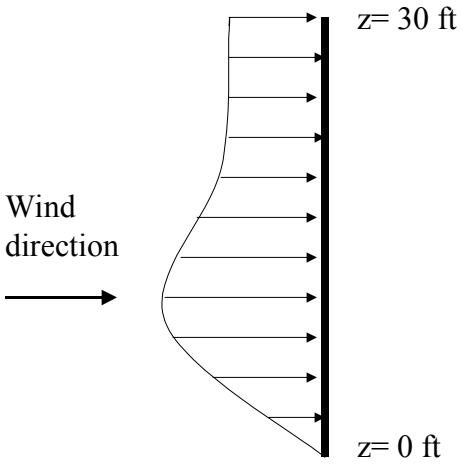

Figure 1: Variation of wind force vs. height

The total force F on the mast can be represented as:

$$F = \int_0^{30} 200\left(\frac{z}{z+5}\right)e^{-2z/30}dz \qquad (1)$$

Equation (1) may also be written as:

$$F = \int_0^{30} f(z)dz \tag{2}$$

We can also find the effective height of the net force on the mast as:

$$d = \frac{\int_0^{30} zf(z)dz}{\int_0^{30} f(z)dz} \tag{3}$$

Based on the lecture notes and code examples in class, write a program to find F and d. Your code must create objects that implement the `MathFunction` interface, pass them to general integration methods, etc.

For simplicity, there are no user inputs. Solve equations (1) and (3) using rectangular, trapezoidal and Simpson's rules:

1. Use `Integration.rect()` as the rectangular method.
2. Use `Trapezoid.trapzd()` as the trapezoid method. See class `Trapezoid` from the lecture notes for an example of how to use `trapzd()` correctly.
3. Use `Simpson.qsimp()` as the Simpson's method.

Use 2^{15} (32,768) intervals for the rectangular and trapezoid methods (set n = 0, 1…, 14 when you call `trapzd`). Simpson's method will use approximately this number of intervals as it finds the integral to a tolerance of 10^{-15}, or 14-15 digits of precision. The method `qsimp` reports the number of recursions n that it uses; the number of intervals is 2^n.

Place, in comments at the top of your `main` method, how many digits of precision the rectangular and trapezoid methods gave. The Simpson result is accurate to 14 or 15 places. (See where the other results diverge from the Simpson result.)

The approximate results, to check your code, are F= 1480.6 lbs and d= 13.05 ft.

Problem 2: River flow

Rivers can be modeled as open channels with flow (see Figure 2),

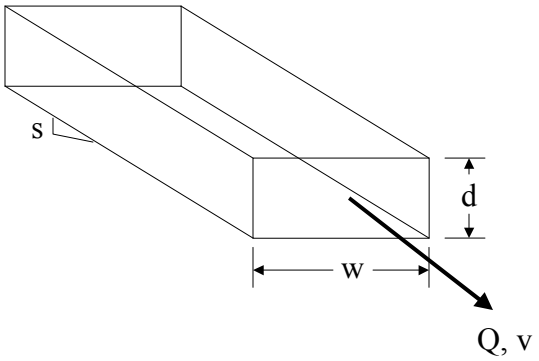

Figure 2: Flow in an open channel

where Q = river flow (m^3/sec) of water
w = width of river channel (m)
d = depth of water in river (m)
s = slope of river bottom (m/m, dimensionless)
v = velocity of water (m/sec)

When a volume of water per unit time Q flows downhill in a channel with a given slope s, we can compute the depth d and velocity v of the water as follows:

Flow conservation requires that:

$$Q = vA = vdw \tag{1}$$

where $A = dw$ (the cross-sectional area of flow)

There is an empirical formula (the Manning equation), based on conservation of momentum, that predicts velocity as a function of the river channel size and slope:

$$v = \frac{\sqrt{s}}{n}\left(\frac{dw}{d+2w}\right)^{2/3} \tag{2}$$

where n is a roughness coefficient (dimensionless)

By substituting equation (2) into (1) we obtain:

$$Q = \frac{\sqrt{s}}{n}\left(\frac{(dw)^{5/3}}{(d+2w)^{2/3}}\right) \tag{3}$$

To solve for the depth d, we rewrite equation (3) as:

$$f(d) = \frac{\sqrt{s}}{n}\left(\frac{(dw)^{5/3}}{(d+2w)^{2/3}}\right) - Q = 0 \tag{4}$$

and solve for its root. We then obtain the river velocity v, once d is known, from equation (1).

Based on the lecture notes and code examples in class, write a program to accept inputs for *Q, w, n* and *s*, and solve for *d* and *v*.

Unlike past problem sets, you must check your inputs for validity, and handle the exceptions thrown if invalid input is entered.

Solve equation (4) using bisection and Newton's method. Use `RootFinder.rtbis()` as the bisection method. Use `Newton.newt()` as the Newton's method. Remember that the classes that implement interfaces `MathFunction` and `MathFunction2` may have constructors, other data members, etc.; create constructors that take *Q, w, n* and *s* as arguments so that *f(d)* and *f'(d)* can be computed. Use a tolerance of 10^{-8} or better.

For bisection, use initial guesses based on physical reasoning. The minimum depth of a river is obvious; use a reasonable guess based on the maximum river depth you can imagine. `Newton.newt()` takes a pair of values as its input and averages them for the initial guess; you may use the same initial guesses as for bisection.

Output *Q, w, s, n, d* and *v*. Modify `Newton.newt()` and `RootFinder.rtbis()` to output (using `System.out.println` for simplicity) the number of iterations to find the root if successful. (Note the tradeoff between the efficiency of Newton's method and the robustness of bisection.)

Last, invoke *f(d)* from one of your objects that implement `MathFunction` or `MathFunction2`, with the solution value *d*, to ensure that *f(d)* is truly zero at that point. Output *f(d)* at the solution value *d* as a check.

The first derivative of equation (4), which is needed in Newton's method, is:

$$f'(d) = \frac{w^{5/3}\sqrt{s}}{3n}\left[5\left(\frac{d}{d+2w}\right)^{2/3} - 2\left(\frac{d}{d+2w}\right)^{5/3}\right] \qquad (5)$$

Example: If Q = 5 m^3/sec, w = 20m, n = 0.03 and s = 0.0002, then d = 0.91 m and v = 0.275 m/sec approximately.

Extra Credit

In addition to your solution above, you may implement a Swing GUI for this problem set and get up to 40 extra credit points. **You must first complete and submit the entire non-GUI solution as outlined above.** Do not attempt to develop your GUI until you have completed the normal assignment since it will be graded separately. You should understand that a GUI solution often requires changes to the rest of your code. Therefore we require you to develop your GUI solution in a separate directory (folder), copying all the files you need. When you submit your solution, first submit your original solution. Then upload your extra credit solution as a second .zip file. Both versions should contain all the files needed to compile and run your solution.

You can get extra credit on only one homework from problem sets 8 to 10. If you don't have time do to the extra credit this time, you still have the opportunity to do so in the future.

You can receive extra credit only for a GUI for the river flow problem from this homework. There is no user input for the sailboat mast problem, so it's not appropriate for extra credit.

You are free to design the GUI as you wish; the following items are suggestions only.

For the river flow problem, allow the user to enter the input parameters, and show the outputs graphically, roughly as in the figure above. Shade the rectangle blue up to the depth d of the river; possibly change the shade of blue as the velocity changes. Show some roughness in the river bottom that changes as parameter n changes.

You will receive a maximum of 20 points of extra credit if you just modify the solutions to homework 6-7 in a straightforward way for this problem. You need to add one or two new aspects to the solution, such as using color, showing roughness, etc.

Turn In

1. Place a comment with your full name, MIT server username, section, TA name and assignment number at the beginning of all `.java` files in your solution.
2. Place all of the files in your solution in a single zip file.
    a. Do not turn in electronic or printed copies of compiled byte code (`.class` files) or backup source code (`.java~` files)
    b. Do not turn in printed copies of your solution.
3. Submit this single zip file.

4. Your solution is due at noon. Your uploaded files should have a timestamp of no later than noon on the due date.

Penalties

- 30% off if you turn in your problem set after Friday noon but before noon on the following Monday. You have one no-penalty late submission per term for a turn-in after Friday noon and before Monday noon.
- No credit if you turn in your problem set after noon on the following Monday.