

CMSC 203, Lab 11

Concepts tested by this program:

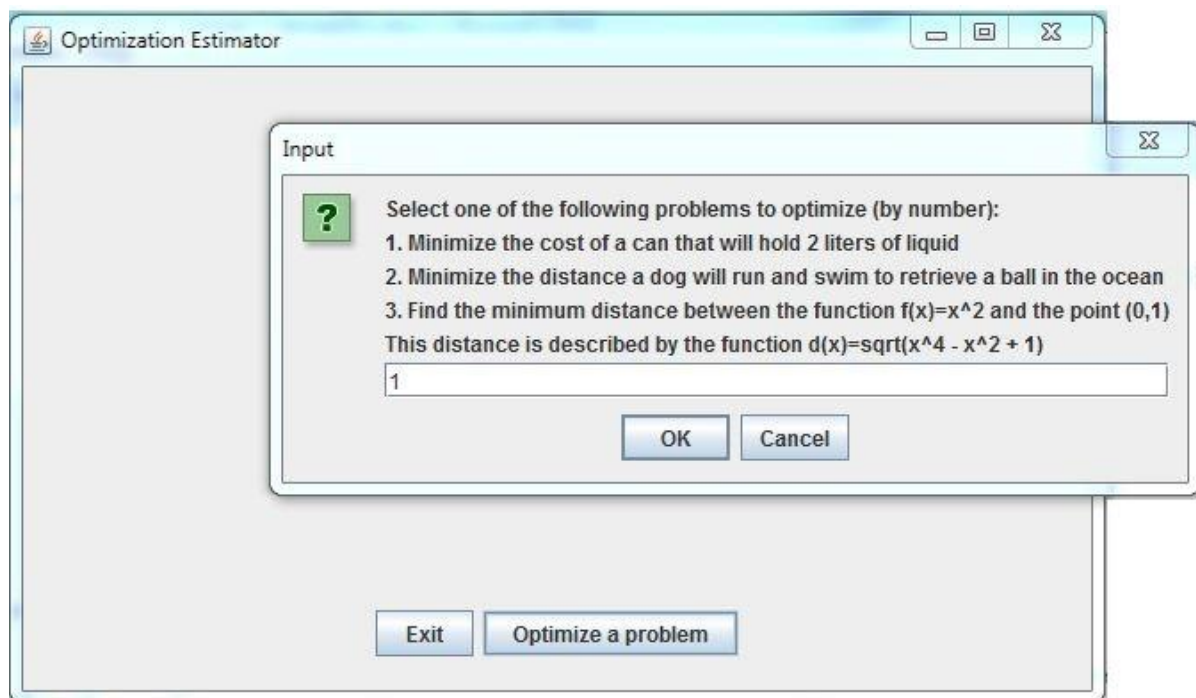
- Inheritance
- Interfaces
- Polymorphism
 - Abstract classes
 - Overriding methods
- JUnit Tests
- Optimization of functions by estimation

NOTE: This lab requires attention to detail. Follow the following instructions carefully.

In this project, you will extend the graphing application developed in Lab 8 to estimate the optimal solution(s) to three optimization problems presented in the MATH 181 YouTube videos by Dr Jason Lee entitled “Math 181 Section 4.6” (<https://www.youtube.com/playlist?list=PL94c6yqxunA3fz2AJ7TzMehUcb08JuJjz>). This application will replace functions #1, #2, and #3 from Lab 8 with the three functions developed in the videos for finding local extrema, and iterate over them from a left extent to a right extent to find the minimum or maximum. It will report the results, and continue on to graph the function using the same methods developed in the previous project. Based on good object-oriented programming practices, the function definitions can easily be changed, so design and programming tasks are focused on iterating through the function to find the local extrema.

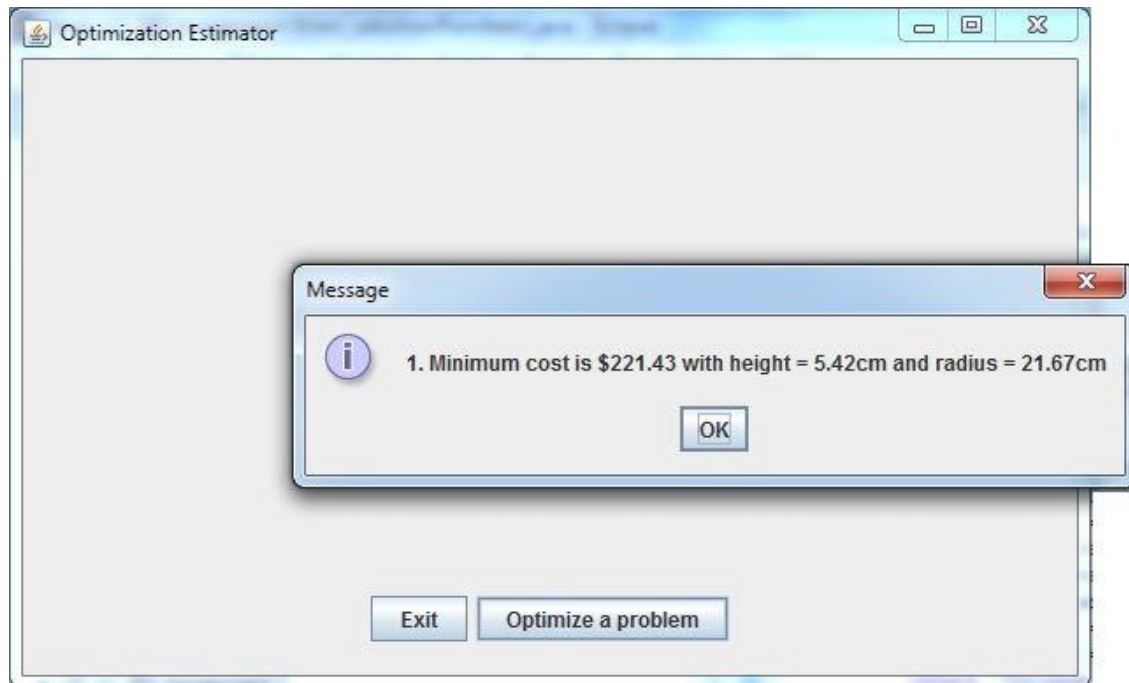
Operation

- When the user selects the “Optimize a problem” button, the application will display the three problems and ask the user to select one of them to optimize.

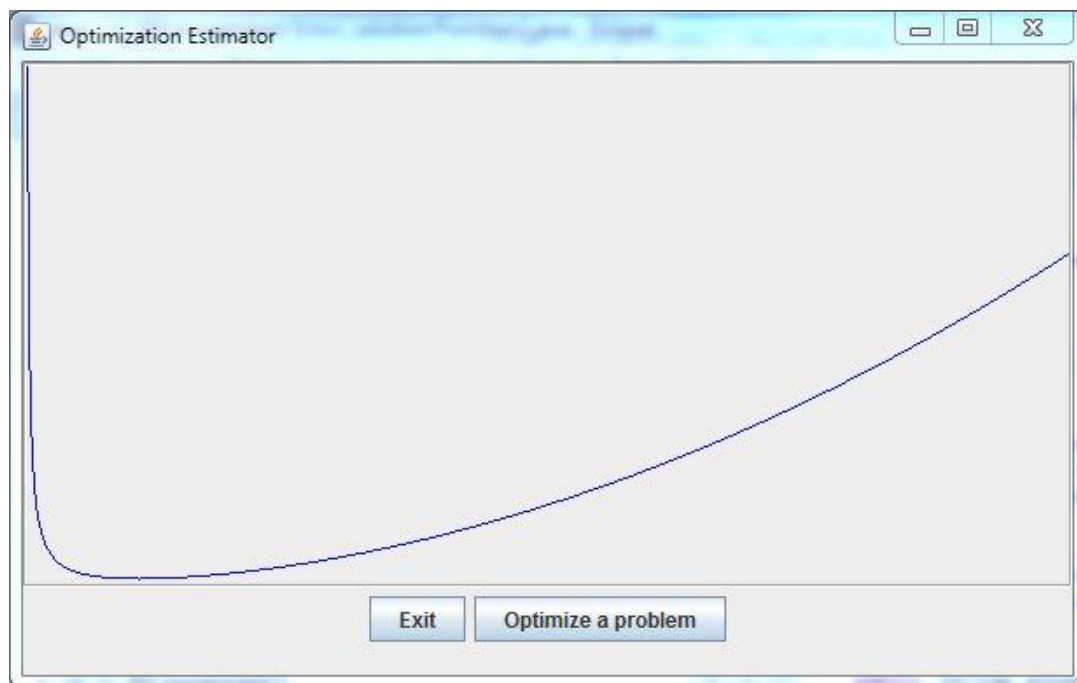


- The application will then ask the user for the left and right extents, which will be the range over which the function is computed.

- Then the application will compute the maximum or minimum, depending on the problem, and display the results.



- Then the application will graph the function being optimized (as in the previous project).



- The user may then re-optimize and graph the same or a different function, or exit.

Specifications

Data Element – (same as Lab 8)

Data Structure – Just as in Lab 8, the abstract class Function holds an abstract method `fnValue` that the programmer will implement for each sub-class function, and an implemented function `fnValueToPlot` (provided) that uses `fnValue` to transform the values to the display coordinate system.

The programmer will implement the three functions which are described in MATH 181 videos entitled “Math 181 Section 4.6”, each in a class that extends Function. The estimation approach will approximate local extrema programmatically, *avoiding* the need to deal with critical points, the closed interval method, first derivative tests, or second derivative tests. Note that this approach is not as precise as methods used in calculus.

Each function will have a toString() method that returns a description of the problem, preceded by its index, and an answerString(...) method tailored to each problem. The answerString() method will take the four values held by the Optimum instance, and display them as appropriate to the specific problem. Note that not all four values will always be displayed. For example, in problem #1, z is just a place-holder, and is not displayed.

Data Manager – the OptimizerManager class implements the methods of OptimizerManagerInterface, plus other helper methods as needed.

Just as in Lab 8, the OptimizerManager selects the correct instance of the function based on the integer chosen. It implements its toString method by calling the three function toString methods. It will receive the user-selected extents of x (left and right) and the grid width in pixels and compute the extents of f(x) (minimum and maximum) as in the previous project.

Then the OptimizerManager will use its optimize() method to iterate from the left extent to the right extent computing the value of the objective function at each point, and remember the minimum or maximum function value found. OptimizerManager will call the answerString() method for the selected function to provide the problem-specific solution.

Then the OptimizerManager will compute the value of the function within the coordinate system of the panel, just as in Lab 8. This value will be returned to the GUI to be plotted.

GUI Driver

- At start-up, a blank panel is displayed with an Exit Button and an Optimize button.
- When the Optimize button is selected, a JOptionPane prompts the user to choose one of the three problems.
- A JOptionPane is then used to select the left and right extents of x (i.e., the minimum and maximum values of x).
- The GUI will create and use a OptimizerManager object to iterate from left to right extents, and save the minimum (or maximum depending on the problem) value of the function. This will be the “extremum” reported in the answerString() method of the chosen function.
- The user will then be allowed to repeat the process for the same or another function, or exit.

TASK #0 – Study the code

Familiarize yourself with the code. Note how it is extended from Lab 8. Compile it and observe that there are errors in the OptimizerManager constructor, because none of the Function classes are implemented.

TASK #1 – Implement Problem #1

Create a subclass of the Function.java abstract class named Function1.java which implements the cost function shown below, as well as the other functions required by Function.java.

In MATH 181 (Calculus I) you studied how to optimize problems that could be expressed as a function of one variable. Three problems were presented in the MATH 181 **YouTube** videos by **Dr Jason Lee** entitled “Math 181 Section 4.6”. If you need to review these problems go to the link at:

<https://www.youtube.com/playlist?list=PL94c6yqxunA3fz2AJ7TzMehUcb08JuJjz>

The first problem is to find the radius and height of a cylindrical can that would hold 2 liters of fluid and minimize the cost of construction, according to different costs for the base and top versus the side. The following shows the derivation of cost as a function of radius (height is a function of radius also).

06a We're going to eliminate a variable. We are given that the can is supposed to contain 2 liters. There are 1000 cubic centimeters in a liter, so the volume of the can is going to be 2000 cubic centimeters. Since the volume is given by $\pi r^2 h$, we can set these equal to each other and solve for h :

$$\begin{aligned}\pi r^2 h &= 2000 \\ h &= \frac{2000}{\pi r^2}.\end{aligned}$$

Now we can go back to our cost function to substitute this h :

$$\begin{aligned}C &= 0.8\pi r^2 + 0.4\pi r h \\ &= 0.8\pi r^2 + 0.4\pi r \left(\frac{2000}{\pi r^2} \right) \\ &= 0.8\pi r^2 + \frac{800}{r}.\end{aligned}$$

Since this is the cost, we're trying to make it as small as possible. Therefore, the function we want the absolute minimum of is

$$C(r) = 0.8\pi r^2 + \frac{800}{r} \quad r > 0$$

So $C(r)$ is the expression of cost as a function of the radius. Create a subclass of the Function.java abstract class named Function1.java which implements the functions fnValue, toString, answerString, getXVal, getYVal, and getZVal, according to the following descriptions:

- fnValue(double x): returns a double which is the cost in terms of x, the specified radius. (Follow the examples in Lab 8 for this method).
- toString(): returns a string describing the problem, e.g., "Minimize the cost of a can that will hold 2 liters of liquid";
- answerString(double cost, double radius, double height, double z): returns a string that describes the result of the optimization. For problem 1, z is not used, so can be any value.
- getXVal(double x): returns x, the radius
- getYVal(double x): returns the height in terms of the radius, according to the formula derived above in eliminating the variable h.
- getZVal(double x): returns the double -1.0, but is not relevant to problem 1. This method is included only to conform to the abstract class Function.java, which requires it to be implemented.

In order to run the application with only Function1.java implemented, you comment out the following lines:

- two lines in OptimizerManager() constructor that attempt to instantiate Function2 and Function3
- two lines in OptimizerManager.toString() that deal with functions 2 and 3
- two lines in OptimizerManager.answerString(...) that deal with functions 2 and 3

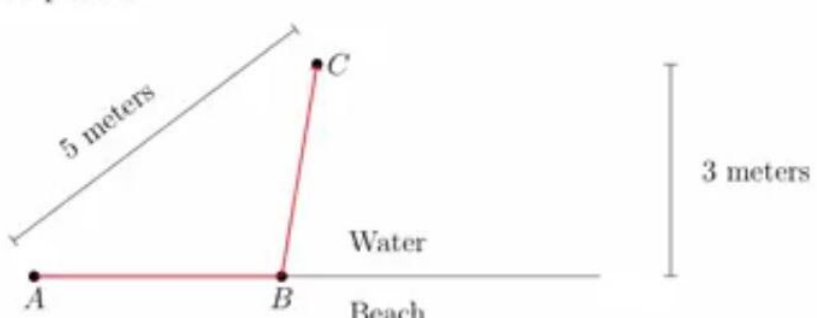
When running the application, be careful not to select anything other than choice #1 .

TASK #2 – Implement Problem #2

The second problem discussed in the above videos is to find the minimum speed with which a dog can run and swim to fetch the ball in the problem below, along with the distance the dog should run along the beach (AB) before jumping in the water.

Optimization

Example: Suppose you are playing fetch with your dog on the beach. You are standing on the shore of the water, and you throw a ball 5 meters, and it lands 3 meters away from the shore, as in this picture.

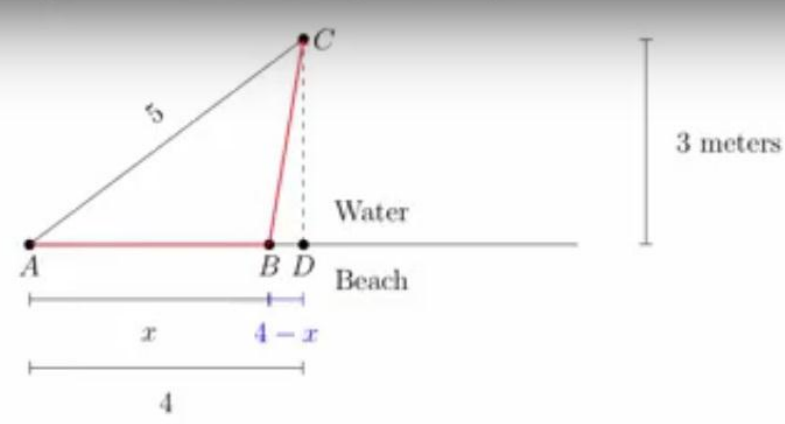


Path of the dog in red – he starts at A , runs along the shoreline to B , then swims to C

Suppose your dog swims at the rate of 0.5 meters per second, and runs at the rate of 3 meters per second. Suppose that it takes your dog 7.25 seconds to get to the ball. Could your dog have gotten there any faster?

Using the Pythagorean Theorem, we can sketch out the dog's path, and assign the variable x to the distance it runs along the beach.

06C Solution: Let's put some additional things into the picture.



We can see that the distance from A to D is 4, by the Pythagorean Theorem. We denote the distance from A to B by x . We mark that as an unknown because the dog can choose where the point B is – the point where the dog makes the change from running to swimming.

We're now going to write down an expression for the total amount of time the dog takes to reach the ball, as a function of x . The value of x is allowed to be between 0 and 4.

We'll use the formula

$$\text{Time} = \frac{\text{Distance}}{\text{Speed}}$$

to figure out how long it takes the dog to get to his goal.

Using the time-distance formula and the known speeds on land and in the water, we find a formula for the time, Q , it takes for the dog to reach the ball as a function of the distance x it runs along the beach.

The distance from B to C is $\sqrt{3^2 + (4-x)^2}$, so the total time is

$$\begin{aligned} Q(x) &= \frac{x}{3} + \frac{\sqrt{3^2 + (4-x)^2}}{0.5} \\ &= \frac{x}{3} + 2\sqrt{x^2 - 8x + 25}. \end{aligned}$$

We're looking for the absolute minimum of this function on the interval $[0, 4]$

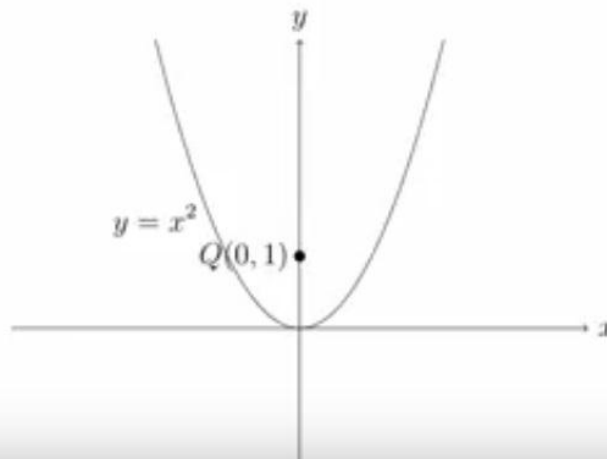
Create a subclass of the Function.java abstract class named Function2.java which implements the six functions described in Task #1. In this case, $\text{fnValue}(\text{double } x)$: should return a double which is the time required in terms of x .

TASK #3 – Implement Problem #3

The third problem discussed in the above videos is to find the closest point on a parabola to a given point.

Optimization

Example: Find the closest point P on the parabola $y = x^2$ to the point $Q(0, 1)$.



The formula for the distance between an arbitrary point P on the parabola to the point Q is derived as:

Solution: We let $P(x, y)$ be an arbitrary point on the parabola. Since $y = x^2$, we can say that this point is $P(x, x^2)$. The distance formula says that the distance between the points (x_1, y_1) and (x_2, y_2) is

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Therefore, the distance between P and Q is

$$\begin{aligned} D &= \sqrt{(x - 0)^2 + (x^2 - 1)^2} \\ &= \sqrt{x^2 + x^4 - 2x^2 + 1} \\ &= \sqrt{x^4 - x^2 + 1} \end{aligned}$$

We're looking for an absolute minimum of this function on the domain $(-\infty, \infty)$. Note that there must be an absolute minimum, because the geometry of the problem definitely shows that there must be an absolute minimum.

As above, create a subclass of the Function.java abstract class named Function3.java which implements the six functions described in Task #1. In this case, fnValue(double x): should return a double which is the distance between P and Q, according to the above formula for D.

Note that the graph of the distance function is not a parabola, because D is not a quadratic formula. It does make sense, however, in that there are two points at which the distance D will be minimized, for a negative and a positive x.

Turn in the following:

The corrected java src directory in a compressed (zip or tar) file