

Tips and Tricks for Problem Set 7 (Fall 2005)

Topic: Hurricanes

Before you read this...

Read the problem set. Some terms mentioned in this document will only make sense once you've read the problem statement. You might want to look at this in conjunction with the solutions, in case anything said here is unclear. There are some neat features in the GUI so try them out and see.

Motivation

We started designing this problem set around the beginning of this hurricane season. There is no offense intended to anyone who has lost something as a result of the unfortunately high number of hurricanes and seemingly high number of landfalls that we've seen this year.

Hurricanes are under active study because their behavior is not accurately predicted. Although this is true of many weather phenomena, a single hurricane often affects an enormous area and several countries, disrupting economies and lives, and destroying property.

These facts make them a multimillion dollar issue, especially in the US where state governments may make evacuation orders based on predicted intensity and likely risk to people's lives. Whenever a hurricane hits and does not do the expected damage, or the hurricane does not land where it was forecast, businesses lose money and individuals are less likely to leave in the face of a similar threat.

What we're trying to do

We're taking one model of power dissipation by a hurricane and making a GUI that finds the power dissipated and models the hurricane power density visually. The hurricane is modeled as a circular, not spiral, system. We've provided a number of equations that lead up to finding the power dissipation. Look at the problem set for details.

The basic concept here is that the power dissipation is related to area and power density (power per unit area) is related to the radius. So we can think of power dissipation as being a function of x and y or r and θ . This would normally mean a 3-D plot and integration over the area but we have circular symmetry and that simplifies things greatly.

The symmetry means that the power dissipation per unit area is independent of θ . We did some fiddling with the original equations to get the one we've put in the problem statement for the power dissipated within a ring. Essentially we took the original equations and did an indefinite annular integration with a 3-dimensional trapezoidal rule. Remember that the trapezoidal rule is most accurate when smaller steps are taken. What they must now do is choose a step size for the radius that is "small enough" and sum the energy over all the rings to get the total.

Once they can perform the calculations, they'll need to draw the power density of the storm as concentric circles. By this time, they should have a method that calculates power density at a single radius. They'll have to figure out that we'll need a range of power densities and their corresponding radii, and that this is naturally a 2D array. We should poke them as far as we can in that direction without telling them the answer.

General Notes on what to expect

PS 5 and 6 involved a GUI. So far, they should be familiar with text fields, labels and buttons, among other things. They've used `ActionListener`. Everyone probably doesn't get it yet. Be patient. Refer them to the Swing Event model in lecture and tutorial. Go over examples if they're fuzzy. Draw pictures; use anything that works! Remind them that components like buttons don't **do** anything except look good in a layout and generate events. **They** must write the code that handles the events and actually do something.

They've seen 2D API but haven't used it much and this should present the greater challenge. Encourage them to use the notes as a resource and for hints/scrounging code. ☺ See the items below for questions we think they might ask based on the problem statement and most people's experience with GUIs.

Don't force them to use transforms if they don't understand them. Try it; if they seem to have a handle on it, go for it. In any case, have them take a piece of paper and draw what the image should look like and assign coordinates before they write code. Do encourage them to use branching/iteration wherever it makes sense, like drawing a bunch of concentric circles/rectangles of the same shape, etc.

They might be put off by the equations. If they've freaked out, try to get them to calm down. Use some of these notes ("Motivation" and "What we're trying to do") to explain how to think about the problem. Have them break the problem into pieces and solve the pieces separately.

Specific issues/Question/Problems

1. Does the program have to look exactly like the problem statement?

No, as long as it has the right inputs and displays. They are free to rearrange the layout as they please. They may use any colors they wish as long as rings are distinguishable. The power ranges are highly recommended because they give a nice banding pattern but they're not required. They should take a look and see what looks good. A general implementation would allow them to change the ranges quickly and easily; that's what they should be shooting for anyway.

2. How big is the hurricane?

There is no widely available literature on hurricane dimensions and wind speeds. Specifically, we needed the eye radius, the maximum wind speed and the physical size of the storm. While the first and the last may always be approximated from a visible plot, we decided to make an approximation for the storm's edge.

Students must write a method that calculates the edge of the storm as a radius where $v \leq 7.5 \text{ ms}^{-1}$. See the solutions for how this was done. The basic idea is to start from R_0 and increment outwards until that condition is reached. The size of the increment is up to the student but encourage them to make it some fraction of R_0 that is probably “small enough”. Let them know that in general “too small” means more computation without meaningful increase in accuracy and “too large” means large inaccuracies. We’re not going to focus too much on this; once the output using the test data is similar, they’ll be fine.

The limits for the radius of maximum winds and the maximum wind speed come from general information on typical eye radii and wind speeds for well developed storms – tropical storms or hurricanes.

3. When calculating the power, how big a step size is “small enough”?

We don’t have a hard and fast answer for that. Our implementation simply focuses on the number of rings and uses a fixed number. This means that there may be lower accuracy with larger storms (larger maximum wind speeds). They may do things differently and make the number bigger for larger storms, etc. This is just one example of the things they need to think about when they design any program.

4. What do we mean by “Make sure that no plotting takes place if the user enters text, a ratio larger than 100% or a ratio equal to zero or less in this field.”? [There are similar instructions for the other input fields. See the problem statement for details]

They’ll have to use `Double.parseDouble()` in order to get the numeric values from the text fields. This method will throw an exception if the input is not numeric. Tell them it will cause an error. We don’t want to answer questions on exceptions right now. Also, storms aren’t defined as such below certain wind speeds – here we included tropical storms in the range – and most eye radii are more than 10km. Also, we don’t want to draw outside of our maximum radius – what would we draw? – and we don’t want to draw a storm of radius zero – that makes no sense whatsoever.

Our requirements take care of all these cases where the user might enter some input that is wrong or inappropriate. There is no need to be further restrictive. They can use dialog boxes - [`JOptionPane.showMessageDialog()`] - to deal with this. We’ve used a combination in the solutions. We used dialog boxes for the inappropriate values – too small/too large. For the other issue, we implemented the `DocumentListener` to listen to the fields and wrote methods to determine whether the values are numbers. If the fields contain text (letters), we disable the button so no computation can take place. If they have found and want to use other solutions (`Javax.swing.JSpinner`), that is okay as long as the way it is used makes sense.

5. Where to start?

First have them think about organization. What requirements are visual or involve using visual components? What functionality requires computation? Suggest that visual code go in

one or more classes and computational code in a separate class. Our implementation uses 5 classes, 4 of which involve visual components. Have them sketch out a little plan, thinking about what each class should be like, i.e. what are the methods and data members; what are the super classes and super interfaces?

6. How do we draw?

First off, refer them to the lecture and tutorial notes on 2D API. They should have some basic idea to start with. The power density will be circles. What order should the circles be drawn in? Let them think first about how to draw them centered at the origin, then think about moving them to the correct location. They can be centered anywhere as long as the entire plot is visible when the window is maximized.

If they have problems with the distance scale, get them to think about how long the scale should be on the screen. This depends on where they want to put it. It can be fixed length. But as the plot changes, the distance that the scale represents will change. How is this related to the scaling that they use to draw?

They also need to draw a power density scale. This one had colors. Refer them to the web – we searched for “rgb color” at Google and checked a couple sites for ideas, then tried them in Java. They have free reign here; we don’t expect them to reproduce our GUI down to the finest detail. If they want a titled border, refer them to the `BorderFactory` class (javadoc). Otherwise, they know how to draw `Strings` using the `Graphics2D` object. In drawing the rectangles, lead them through thinking about how to draw one rectangle and then translating it. Try to hint that if they need the same size rectangle equal distances apart, then maybe you can do that automatically, rather than manually drawing them at separate locations.

They don’t have to center the labels on this scale. If you want to give them hints on how to do this, see the solutions.

7. Why does my titled border move away from the edges of the panel?

In this case, the student probably modified a transform and did not reset it to it’s original state. Try to point this out and have them think about how to solve the problem.

8. Why doesn’t anything happen when I press “Enter” in a text field or while a button is highlighted?

In the first case, the field needs an `ActionListener` where the `actionPerformed()` method does the same thing that clicking the “Draw” button. They should search for a solution themselves using javadoc but we use the `JButton.doClick()` method. In the second case, the button would need a `KeyListener` that would check if the user pressed enter and then call the `JButton.doClick()` method.