CS 100M: Project 1

Due: Thursday, February 3, 2005 at 6:00 PM

Files related to this assignment are available on the course web site http://www.cs.cornell.edu/courses/cs100M/2005sp/. When you submit your solutions for grading it must be done through the Course Management System (CMS). The deadline is firm. Each problem is worth the same amount of points. And for any problem, half the points are for style and half for correctness.

You must work either by yourself or with at most one other partner. You are allowed to discuss background issues with others, but the programs you submit must just be the work of you and your partner (if you have one). If something comes up and you are unclear about our Academic Integrity Policy (posted on the course website) contact a member of the course staff immediately.

P1A. (Medians of a Triangle)

A triangle has three medians; each extends from a vertex to the midpoint of the opposite side. In this problem you will graphically confirm that the three medians intersect at a common point which is the centroid of the triangle.

Recall that if the endpoints of a line segment are \((p, q)\) and \((r, s)\), then the coordinates of the midpoint \(M\) are given by

\[
M = \left( \frac{p + r}{2}, \frac{q + s}{2} \right).
\]

In other words, the \(x\) and \(y\) coordinates of the endpoints are averaged.

Now consider the triangle, whose vertices we denote by \(A = (a_1, a_2)\), \(B = (b_1, b_2)\), and \(C = (c_1, c_2)\). Thus, the median from vertex \(A\) connects \((a_1, a_2)\) with

\[
M_A = \left( \frac{b_1 + c_1}{2}, \frac{b_2 + c_2}{2} \right).
\]

(What are the endpoints of the other two medians?) It can be shown that the three midpoints intersect at the centroid \(F\) defined by

\[
F = \left( \frac{a_1 + b_1 + c_1}{3}, \frac{a_2 + b_2 + c_2}{3} \right).
\]

Here is how you can structure a MATLAB script that checks out this result:

Step 1. Create a figure window that can be used to display the triangle.

Step 2. Input the three vertices and display.

Step 3. Display the triangle by connecting the vertices.

Step 4. Compute the median point of each side.

Step 5. Display the medians in red.

Step 6. Compute the centroid and display it with a green asterisk.

To get started, copy the script P1A off the website and observe that it provides a template for the above six-step solution procedure. We’ve taken care of Step 1, i.e., the setting up of the figure window. You’ll learn more about close, figure, axis, hold, and other graphics-related commands later in the course.

In Step 2 use ginput, MATLAB’s command for mouse input. The fragment

\[
[p,q] = \text{ginput}(1) \\
\text{plot}(p,q,'o')
\]
will wait for you to click the mouse in the plot window. After you click, it will assign the \( x \) and \( y \) coordinate values of the clicked point to variables \( p \) and \( q \) respectively. The second command, \( \text{plot}(p,q,'o') \), puts a small circle at the clicked point. In Step 2, you have to repeat this pair of commands three times as there are three vertices to set up. You have to decide on the names of the variables that are used to store the \( x \) and \( y \) values. Mimicking the notation above, something like \([a_1,a_2] = \text{ginput}(1)\) would do just fine for storing the coordinate values of the vertex that you are regarding as vertex A. Note that in MATLAB you sometimes use square brackets and sometimes use parentheses. Follow the rules!

In Step 3 you need to know how to draw a line segment given that you know the endpoints. Suppose variables \( p, q, r, \) and \( s \) are initialized and that the command

\[
\text{plot}([p \ r],[q \ s])
\]

is executed. A line segment whose endpoints are \((p,q)\) and \((r,s)\) will be drawn. Thus, the fragment

\[
p = 1; \ q = 2; \ r = 3; \ s = 4; \text{plot}([p \ r],[q \ s])
\]

draws a line from \((1,2)\) to \((3,4)\). The rule that we are trying to convey here may best be remembered as \(\text{plot}([\text{firstX secondX}],[\text{firstY secondY}])\). Note (again!) that sometimes in MATLAB you use square brackets and sometimes parentheses.

In Step 4 you have to repeatedly use the midpoint formula given above. The assignment statement is how you do this. The fragment

\[
p = 1; \ q = 2; \ r = 3; \ s = 4; \ m1 = (p+r)/2; \ m2 = (q+s)/2;
\]

assigns to \(m1\) and \(m2\) the \(x\) and \(y\) coordinates of the midpoint of the line segment that connects \((1,2)\) and \((3,4)\). Pick good names for the variables you use to hold the midpoint values.

In Step 5 you have to draw three more line segments, i.e., the medians. We explained how to plot line segments above. Plotting with color requires that you name the color in the plot command. Thus,

\[
\text{plot}([p \ r],[q \ s],'r')
\]

will draw a red line segment from \((p,q)\) to \((r,s)\).

In the last step you must “highlight” the centroid. Do this by putting a green asterisk at that location. The command \(\text{plot}(p,q,'*g')\) displays such a mark at the position defined by the values in \(p\) and \(q\).

Submit your finished \(\text{P1A}\) script to CMS.

**P1B. (Newton’s Method for Square Roots)**

This problem is about Newton’s method for computing approximate square roots. While Newton used calculus to derive the algorithm, we will use a simpler derivation that is based on geometric intuition.

Suppose you are given a rectangle with area \(A\) and are asked to make it “more square”. It is understood that the area of your new rectangle should be the same. If you know how to compute square roots this is easy; just set the length and width of the new rectangle to be \(L_{\text{new}} = W_{\text{new}} = \sqrt{A}\). But suppose you don’t know how to compute square roots. Then what?

Suppose \(L\) and \(W\) specify the length and width of the original rectangle. Our intuition tells us that \(\sqrt{A}\) is in between \(L\) and \(W\). So let’s take the average of \(L\) and \(W = A/L\) and think of that as a “better” length:

\[
L_{\text{better}} = \frac{1}{2} \left( L + \frac{A}{L} \right)
\]

Refer to this as the “update formula.” Sure enough, if you redraw the rectangle with length \(L_{\text{better}}\) and width \(W_{\text{better}} = A/L_{\text{better}}\), then you will find that it is more square. For example, a 1-by-2 rectangle becomes a \((3/2)\)-by\((4/3)\) rectangle.

We can obviously repeat the process making our improved rectangle even more “square-like”. Thus, our \((3/2)\)-by\((4/3)\) rectangle can be reshaped as \((17/12)\)-by\(-(24/17)\) which is approximately 1.4167-by-1.4118. It seems that as we iterate the lengths are converging to \(\sqrt{2} \approx 1.41421356237310\).

In this problem you will write a MATLAB script \(\text{P1B}\) that computes approximate square roots using this idea. Structure your script as follows:
Step 1. Use input to assign to a variable $A$ the value of the number whose square root we desire. For reasons explained below, we want $A$ to satisfy $1 \leq A \leq 4$, so the message that is part of the input command should mention this.

Step 2. We need to come up with the “initial rectangle”. Note that $f(x) = 1 + (x - 1)/3$ interpolates the square root function at $x = 1$ and $x = 4$. Thus, we can use this function to approximate $\sqrt{x}$ for $x$ in the interval $[1, 4]$. So your script should assign the value $f(A)$ to a variable $L$. As an example, if $A = 3$, then $L$ will be assigned the value $1 + (3 - 1)/3 = 5/3$.

Step 3. Improve $L$ using the update formula. Assign the improved value to $L_1$.

Step 4. Improve $L_1$ using the update formula. Assign the improved value to $L_2$.

Step 5. Improve $L_2$ using the update formula. Assign the improved value to $L_3$.

Step 6. Improve $L_3$ using the update formula. Assign the improved value to $L_4$.

So far we have described the computations, but not their display. We want your script to display the sequence of (ever improving) lengths and their error. In particular, it should produce 5-line table with headings similar to what is illustrated here for the input $A = 3$:

<table>
<thead>
<tr>
<th>Length</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.666666666666667</td>
<td>6.54e-002</td>
</tr>
<tr>
<td>1.733333333333333</td>
<td>1.28e-003</td>
</tr>
<tr>
<td>1.732051282051282</td>
<td>4.74e-007</td>
</tr>
<tr>
<td>1.732050807568942</td>
<td>6.51e-014</td>
</tr>
<tr>
<td>1.732050807568877</td>
<td>0.00e+000</td>
</tr>
</tbody>
</table>

A pair of disp commands can take care of the heading—you’ll have to play with the spaces in
disp(’ Length      Error’)
so that they (more or less) align with the values in the columns. These statements should be placed just after the command where the input of $A$ takes place.

The error for a given length value $L$ is simply abs($L - \sqrt{A}$). Note in the example above that the lengths are displayed to 15 decimal places and the error in scientific notation with 2 decimal places. So a statement like

fprintf(’ %18.15f %6.2e ’, ______ ,abs( ______ - sqrt(A)));

should come after each computation in Steps 2-6. (Fill in the blank with the appropriate variable name.)

Submit your final script P1B to CMS.

For your information, the square root function in MATLAB and in other programming languages works by applying the update formula three or four times. Our restriction that $1 \leq A \leq 4$ is not really a restriction. Any positive number $x$ can be uniquely written in the form $x = A \cdot 4^n$ with $1 \leq A < 4$. Since $\sqrt{x} = \sqrt{A} \cdot 2^n$ the square root problem boils down to finding square roots of numbers in the interval $[1, 4]$.

P1C. (Random Colors)

In this problem you modify a script that generates a logarithmic spiral so that (a) it does not generate spirals that are “too big” and (b) a given edge is drawn red or black depending upon a simulated weighted coin toss.

Copy the script P1C from the website and run it. It prompts you for the turn angle and the number of edges and generates a logarithmic spiral. You are to modify this script in two places. All your changes should be within the two “modification zones” that are clearly delineated by comments in the script.

If the input value of NumEdges is larger than 1000, then the message “NumEdges too large, reset to 1000” should be printed and the value 1000 should be assigned to NumEdges. This guards against spirals that might take too long to generate.
The second change has to do with the drawing of an edge. Instead of always drawing the $k$th edge black, it should be drawn red with probability $k/N$ and drawn black with probability $1 - k/N$ where $N$ is input value of `NumEdges`. Note that `NumEdges` might be revised downwards. Thus, it is important for your script to “remember” the initial value of `NumEdges` before any changes are made.

To simulate the “weighted” coin toss that determines the color of the edge, use the MATLAB `rand` function. An example illustrates how this built-in function works. If $p$ is a variable whose value is between 0 and 1, then the fragment

```matlab
if rand <= p
    disp('A')
else
    disp('B')
end
```

prints “A” with probability $p$ and prints “B” with probability $1 - p$. For example, if $p = .7$ and the above fragment is executed 1,000,000 times, then we would expect to see ”A” printed about 700,000 times.

Submit your modified P1C script via the CMS system.

Postscript

You will find that some of the solution programs in this assignment are rather tedious to write, involving sequences of commands that “almost” look alike. This tedium has a purpose: it motivates the need for language constructs that can specify repetition and which can permit the manipulation of subscripted variables. Loops and arrays do just that, and you will learn about them in the next few weeks.