CS1115 Fall 2013 Project 1 Due Thursday September 12 at 11pm

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the project you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group. Each problem is worth 5 points. One point may be deducted for poor style. Your best six (of seven) assignments are counted.

Objectives

Completing this project will help you learn about MATLAB scripts, assignment statements, if-else statements, for-loops, while-loops, and some MATLAB built-in functions.

1 A Limited Cosine-Sine Table

Here are three known cosines:

$$\begin{aligned} \cos(0^o) &= 1 \\ \cos(60^o) &= 1/2 \\ \cos(72^o) &= 1/(1+\sqrt{5}). \end{aligned}$$

Using trigonometric identities it is possible to compute various sines and cosines without resorting to the built-in functions `sin` and `cos`. For example, here is a fragment that computes $\cos(132^o)$:

```
c60 = 1/2;
c72 = 1/(1 + sqrt(5));
s60 = sqrt(1 - c60^2);
s72 = sqrt(1 - c72^2);
c132 = c72*c60 - s72*s60;
```

Figure out how to compute $\cos(3^o)$ and $\sin(3^o)$ using trig identities. For this purpose you will find useful trigonometric identities on page 414 of FVL.

Write a script `P1` that displays x, $\cos(x^o)$ and $\sin(x^o)$ for $x = 0, 3, 6, ..., 87, 90$. For full credit your solution script must not make use of the built-in cosine and sine functions. The table should have three columns: x, $\cos(x^o)$, and $\sin(x^o)$. Use the `%3d` format for x and the `%20.15f` format when displaying the cosine and sine. The table should have a heading and look nice. Include comments in the script so that the reader can track your use of the trig identities. Submit your solution file `P1.m` to CMS.

2 Excellent Integers

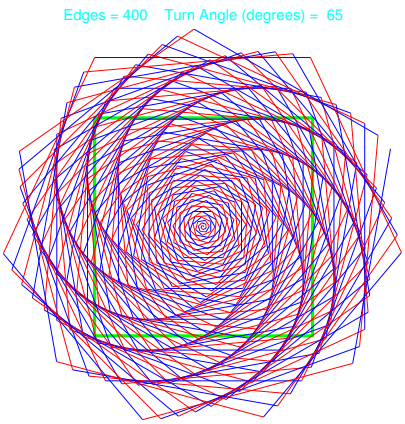
We say that a positive whole number x is an *excellent integer* if $x = 2^j 3^k$ for some choice of integers j and k. There are 20 excellent integers less than 100:

$1 = 2^0 \cdot 3^0$	$3 = 2^0 \cdot 3^1$	$9 = 2^0 \cdot 3^2$	$27 = 2^0 \cdot 3^3$ $81 = 2^0 \cdot 3^4$
$2 = 2^1 \cdot 3^0$	$6 = 2^1 \cdot 3^1$	$18 = 2^1 \cdot 3^2$	$54 = 2^1 \cdot 3^3$
$4 = 2^2 \cdot 3^0$	$12 = 2^2 \cdot 3^1$	$36 = 2^2 \cdot 3^2$	
$8 = 2^3 \cdot 3^0$	$24 = 2^3 \cdot 3^1$	$72 = 2^3 \cdot 3^2$	
$16 = 2^4 \cdot 3^0$	$48 = 2^4 \cdot 3^1$		
$32 = 2^5 \cdot 3^0$	$96 = 2^5 \cdot 3^1$		
$64 = 2^6 \cdot 3^0$			

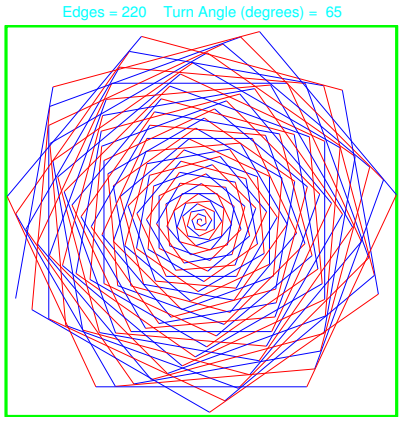
Write a script `P2` that solicits a positive integer M using `input` and nicely displays its value and the number of excellent integers that are less than or equal to M. (You do not have to print out the excellent integers.) To force issues, your solution must not involve nested loops, user-defined functions, or arrays. The built-in functions `log`, `floor`, and `ceil` are useful for this problem. Submit `P2.m` to CMS.

3 Spiral in a Box

Download and run the script `P3.m` from CMS. It is a slight modification of the script `spiral` that you worked with in the lab:



Notice that a "box" is now part of the graphic display. We say that an edge is *bad* if one or both of its endpoints are strictly outside the box. In this assignment you are to modify `P3` so that spiral generation ends as soon as the first bad edge is encountered:



To do this problem for full credit, you must replace the `for`-loop that generates the "full" spiral with a `while`-loop that generates the "boxed" spiral. And just to be clear, the first bad edge is *not* displayed. One final detail, instead of displaying the value of `numEdges` in the figure window, your script should display the number of "boxed" edges. Submit your modified `P3.m` to CMS. Points off for poorly commented code.

4 Perimeter GUI

Download the files `PerimeterGUI.fig` and `PerimeterGUI.m` from CMS. Open `PerimeterGUI.fig` in GUIDE and observe that this (sloppy) GUI can report various estimates of the perimeter of the displayed ellipse. In particular, it can display the estimates P_1, P_2 and P_3 that can be found on page 14 of FVL. (a) By using the property editor, clean up the appearance of the GUI. In addition to resizing and aligning the components, play with color, fontsize, and font and the background color of the whole figure window. (Type `uisetfont` in the command window to see available fonts.) Add a title using Static Text and enclose the pop-up menu

with a nicely sized and colored panel. The goal here is simply to get experience with GUIDE. (b) Now let's modify the GUI so that it can display the perimeter estimates P_4, P_5, and P_6 given in FVL page 14. First, using the Property Editor, update the `'String'` property of the pop-up menu so that it includes formulas P_4, P_5, and P_6. Next, scroll through `PerimeterGUI.m` until you reach the line that begins with `function popupmenu1_Callback`. The commands that follow process the pop-up menu when a selection is made. By mimicking what you see, add code so that the GUI correctly processes pop-up menu requests for the P_4, P_5, and P_6 values. The goal is not to understand fully `PerimeterGUI.m` but to get experience modifying a small part of the code that hopefully makes sense.

Challenge 1: Average Distance to the Sun.

Suppose Ithaca is at (0,0) and that you take a 9-day trip visiting Syracuse (20,40) for 3 days, Rochester (-50,45) for 2 days, and Elmira (-20,-15) for 4 days. What is your average distance from Ithaca during your trip? A plausible answer to this question is

$$d_{ave} = \frac{3}{9}\sqrt{20^2+40^2} + \frac{2}{9}\sqrt{50^2+45^2} + \frac{4}{9}\sqrt{20^2+15^2},$$

i.e., a weighted average of distances where the weight associated with a given point is the fraction of time spent at that point. In this problem we use a similar idea to approximate the average distance of a planet to the Sun as it moves along its elliptical orbit. Kepler's "equal area" law will be taken into account.

We start with some math facts about the orbit itself. Assume that (a) the Sun is positioned at (0,0), (b) P is the minimum Sun-to-planet distance, and (c) A is the maximum Sun-to-planet distance. The orbit can then be described parametrically as follows:

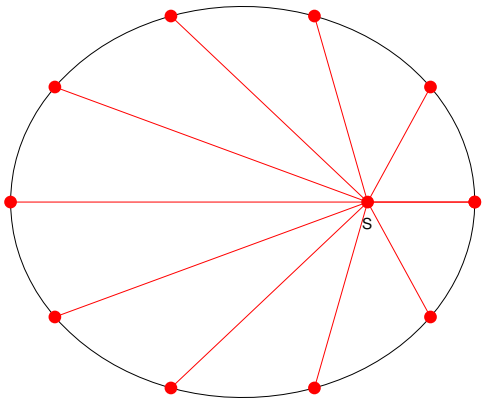
$$\begin{aligned} x(\tau) &= \left(\frac{P-A}{2}\right) + \left(\frac{P+A}{2}\right)\cos(\tau) \\ y(\tau) &= \sqrt{P\cdot A}\sin(\tau) \end{aligned}$$

where the parameter τ satisfies $0 \le \tau \le 2\pi$.

For a given even integer N we define the orbit points $Q_0, Q_1, \ldots, Q_N$ as follows

$$Q_k = (x_k, y_k), \qquad x_k = x(\tau_k), \qquad y_k = y(\tau_k), \qquad \tau_k = \frac{2\pi k}{N}.$$

Note that $Q_0 = Q_N$. The orbit points points allow us to partition the ellipse into sectors, e.g.,



Let sector k be defined by the Sun, P_{k-1}, and P_k. Let α_k be its area and let ρ_k be the time it takes for the planet to move from P_{k-1} to P_k. A consequence of Kepler's equal-area law is that

$$\frac{\alpha_k}{\alpha_1+\cdots+\alpha_N} = \frac{\rho_k}{\rho_1+\cdots+\rho_N}.$$

To make sense of this, note that $\mathbf{p} = \rho_1 + \cdots + \rho_N$ is the period of revolution and $\mathbf{a} = \alpha_1 + \cdots + \alpha_N$ is the total area enclosed by the orbit. Thus,

$$\frac{\rho_k}{\mathbf{p}} \;=\; \frac{\alpha_k}{\mathbf{a}},$$

i.e., the fraction of time that the planet spends "sweeping out" a sector is exactly same as the ratio of the sector's area to the total area.

For a given N, we define the average distance between the planet and the Sun by

$$d_{ave}(N) \;=\; \sum_{k=1}^{N} \left(\frac{\alpha_k}{\mathbf{a}}\right)\left(\frac{d_{k-1}+d_k}{2}\right)$$

where

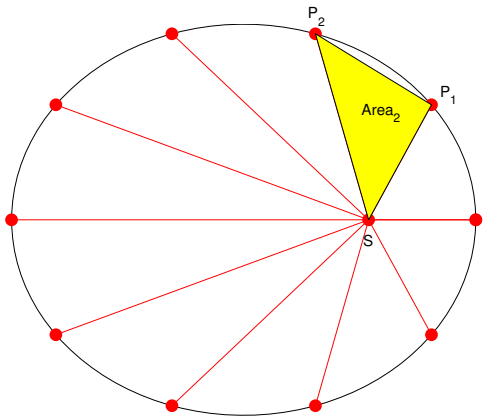
$$d_k \;=\; \sqrt{x_k^2 + y_k^2}.$$

This is analogous to our upstate NY trip calculation. When the planet "visits" the kth sector the fraction of time that it spends there is $\alpha_k/\mathbf{a}$ and we define its distance to the Sun by the average of d_{k-1} (the distance from P_{k-1} to the Sun) and d_k (the distance from P_k to the Sun).

Although the area enclosed by the orbit has a simple formula,

$$\mathbf{a} \;=\; \pi\left(\frac{P+A}{2}\right)\sqrt{P\cdot A}$$

the sector area α_k does not. However, we can approximate α_k by the area of the triangle defined by the Sun, P_{k-1} and P_k:



Write a script `C1` that solicits P and A using `input` and computes the value of

$$\tilde{d}_{ave}(N) \;=\; \sum_{k=1}^{N} \left(\frac{\text{Area}_k}{\mathbf{a}}\right)\left(\frac{d_{k-1}+d_k}{2}\right)$$

A formula for triangle area can be found on page 413 of FVL. As to the value of N, it would seem that $\tilde{d}_{ave}(N)$ should converge to a limit as $N \to \infty$. Experiment and select a value that you think would satisfy a user who wants accuracy through the third decimal place. You may assume that the inputs P and A satisfy $1 <= P \leq A \leq 1000$.

Your script should neatly output the value of P, A, N, $\tilde{d}_{ave}(N)$, and $(P+A)/2$. (FYI, $(P+A)/2$ is the average distance formula used in Kepler's Third Law). To force issues, your script should not use arrays or user-defined functions. Submit your implementation of `C1` to CMS. **You may not work in groups on any challenge problem.**

Interesting choices for (P, A): Mercury = (46,70), Venus = (107,109), Earth = (147,152), Mars = (207,249), Jupiter = (741,817), Saturn (1354,1513), and Halley's comet = (88,5265). These distances are in 10^6km.