Benjamin Asdell
CEE/MAE M20
UID: 005114992

Homework 1

October 15, 2020

1. I love M20

1. Introduction

   The objective of this problem is to develop a short script that takes in and manipulate user input to display a desired message. The main function is to register the first inputted phrase and flip the second phrase from left to right. The results are concatenated and printed to the command window, and the details of the program will be discussed.

2. Models and Methods

   The program starts by asking for two user inputs consecutively. The second phrase is then reversed using the `fliplr` function. After all the input gathering and string manipulation has completed, the resulting two strings are concatenated and printed to the command window using MATLAB's `fprintf` function.

3. Calculations and Results

   When the program is executed, the user inputs `I love` and `02M` to the respective input queries, and the command window displays the following output.

   ```
   Enter a phrase:
   I love
   Enter another phrase:
   02M
   I love M20
   ```

   Visibly, the second input is not an intelligible statement on its own, but the final output of `I love M20` does have a meaning.

4. Discussion

   This problem is extremely short and straightforward, with the solution being only four lines including the display function. The test phrase `I love M20` prints the intended output, and requires the user to input `02M`, the reverse string, for the second query. This program is more versatile than just this specific message, as long as the user inputs the second phrase in reverse format.

2. Ellipse Calculations

1. Introduction

   The goal of this problem is calculate or estimate the perimeter of an ellipse based on user-inputted axial lengths and mathematical equations given by the problem itself. All of the calculations can be considered estimates for ellipses except in the case of a circle, where some calculations may give an exact answer proportional to π.

2. Models and Methods

   The program starts by asking the user to input values for two variables: a and b. For the purposes of this program, which variable corresponds to which elliptical axis does not matter. The first calculation computes the 'departure from circlehood' of the ellipse based on the a and b values.

$$h = (\frac{a-b}{a+b})^2$$

   This `h` value is used in 5 of the 8 calculations in order to simplify the expressions and also provides some insight into how much the calculated estimates and the true perimeter will vary for the specific ellipse.

$$\begin{aligned}
P_1 &= \pi(a+b) & P_5 &= \pi(a+b)\left(1+\frac{3h}{10+\sqrt{4-3h}}\right) \\
P_2 &= \pi\sqrt{2(a^2+b^2)} & P_6 &= \pi(a+b)\frac{64-3h^2}{64-16h} \\
P_3 &= \pi\sqrt{2(a^2+b^2)-\frac{(a-b)^2}{2}} & P_7 &= \pi(a+b)\frac{256-48h-21h^2}{256-112h+3h^2} \\
P_4 &= \pi(a+b)\left(1+\frac{h}{8}\right)^2 & P_8 &= \pi(a+b)\left(\frac{3-\sqrt{1-h}}{2}\right).
\end{aligned}$$

Figure 1: The eight different perimeter calculation formulas. Notice how the h variable is present in five, as discussed above. For later reference in this problem, the equations will be referred to top to bottom, left to right.

3. Calculations and Results

   When the program is executed, the user inputs two numbers corresponding to the axial lengths of the ellipse in question. Here is a sample output from the command window when the user inputs a = 2 and b = 5:

```
Calculation Results

 21.99115           23.92566           22.97877           23.01254
 23.01311           23.01310           23.01311           23.05213
```

From looking at these values, it is evident that all of the estimates except the first produce very similar results when the a and b values are relatively similar. The first calculation produces a result that is somewhat within the grouping of the others but significantly lower. The next sample output describes the perimeter of an elongated ellipse where a = 40 and b = 2.

```
Calculation Results

131.94689          177.93732          156.63919          160.33115
160.76860          160.68662          160.74960          169.82108
```

From these results, it is becoming more clear that some of these approximations might always be under approximations (Eq.1, Eq.3) or over approximations (Eq.2, Eq.8). The other four are clearly more spread out than the previous sample, but still remain in a visible grouping. The final sample output is a circle, in this case a = b = 2.

```
Calculation Results

 12.56637           12.56637           12.56637           12.56637
 12.56637           12.56637           12.56637           12.56637
```

The results of this calculation are eight identical values (within significant digits). Note the spacing in the command window that is provided to improve readability, especially within MATLAB.

4. Discussion

The three sample inputs shown in the above section clearly show the functionality and limit of these eight approximations to the perimeter of a given ellipse. Using relatively close a and b values will produce a grouping of results from the calculations, most likely due to the geometry of the ellipse being near-circular. Increasing the difference between a and b creates a very oblate ellipse, which in turn produces a series of results that are not nearly as grouped. As stated before, Eq.1 and Eq.3 seem to be under approximations while Eq.2 and Eq.8 appear to be over approximations. It can be concluded that as the ellipse becomes more oblate (as `h` increases), these trends will continue and possibly the four calculations that remain grouped in the second sample will also diverge further.
The third sample input is a circle, and as expected, produces eight accurate and precise calculations of the perimeter. From the geometrical definition, $P = 2\pi r$. Using a = b = r = 2, the expected perimeter is consistent with the eight identical outputs of the program.
The final comment on this program involves the output format, especially in regards to output comparison. I chose to truncate the output values at five decimal places because I found it provided the most ability to differentiate the results without skewing the comparison due to MATLAB's inherent computational error. To clarify, the error resulting from MATLAB's inability to compute exactly when using irrational numbers produces a statistical error if one was to compare the results to the `true` value of the perimeter. However, at very similar values of a and b, the difference in the results are not visible unless extended past four decimal places. Therefore, I believe five decimal places best balances the error and comparison aspect of the output.

5. Improvement

The most glaring flaw in my solution to this problem revolves around error handling and input sanitation. The formulas for the calculation all require the a and b values to be numerical values. In addition, certain combinations of a and b that produce a denominator of zero in the calculation of `h` will also produce a divide by zero error. These issues can be resolved by sanitizing the user input inside boolean if blocks and the use of MATLAB's `isscalar()` and `isnumeric()` functions to validate that the user input is actually a scalar, numerical value. Secondly, the denominator `a+b` can be calculated before `h` in order to verify it will return a nonzero value.

1 3. Trigonometric Calculation

1. Introduction

   The purpose of this problem is to creatively use trigonometric identities and other mathematical formulas related to cosine and sine in order to approximately calculate the values of uncommon angles. The program itself will sequentially apply trigonometric identities to given values and store the result of each step in a new variable. Finally, the desired values of cos(11°) and sin(11°) will be printed to the command window display.

2. Models and Methods

   All three of the given values and identity from the assignment specifications are used in the solution, in addition to the below identities given in Appendix B of the textbook.

$$sin^2(x) + cos^2(x) = 1, 0 \leq x \leq 90^\circ \tag{1}$$
$$cos(60^\circ) = 0.5 \tag{2}$$
$$cos(16^\circ) = 0.96126169593 \tag{3}$$
$$cos(-x) = cos(x) \tag{4}$$
$$sin(-x) = -sin(x) \tag{5}$$
$$cos(\frac{x}{2}) = \sqrt{\frac{1 + cos(x)}{2}} \tag{6}$$
$$cos(x + \phi) = cos(x)cos(\phi) - sin(x)sin(\phi) \tag{7}$$

   My solution to this problem involved trigonometric manipulation in order to achieve the desired angle of 11°. I recognized that cos(60°) could be reduced to cos(15°) and cos(16°) could be reduced to cos(4°) by repeatedly using Eq.6. I also figured that by using Eq.1, I could calculate sin(15°) and sin(4°) using

$$sin(x) = \sqrt{1 - cos^2(x)}$$

   Then, Eq.7 can be solved for cos(11°) after using Eq. 5 and Eq.6 to find the remaining necessary values. The final step once again uses Eq.1 and the result from Eq.7 to solve for sin(11°).

3. Calculations and Results

   When the program is executed, the following is printed to the command window:

```
cos(11°) =  0.98163
sin(11°) =  0.19081
```

Checking these results with a calculator will verify that the program prints the correct output. Note the spacing in the command window that is provided to improve readability, especially within MATLAB.

4. Discussion

   The main complexity of this problem stems from being able to intuitively work with trigonometric identities in order to solve for cosine and sine of 11°.
   There are many possible combinations of identities that would achieve the same result, but I chose this specific method because I believe it completes the task in the least or near the least amount of steps. In particular, the program uses half angle formula (Eq.6) at the beginning and never has to use it after the initial angle reduction step.
   Theoretically, any angle could be evaluated using a variation of this program as long as it is less than the 90° bounding parameter set by cosine's half angle formula. To achieve this, more calculations would be needed to find the path of trigonometric manipulation for the given angle.