Adrian Loekman
UID: 105785820
CEE/MAE M20
September 29, 2021

HOMEWORK 1

1. Evaluating Inputs

1.1. Introduction

For the first problem, the main goal is to create a script which users can input their names and a mathematical expression. The output would be the name in uppercase letters and evaluate the mathematical expression that the user has inputted. The key points in this problem are changing into capital letters and calculating expressions.

1.2. Model and Methods

The method that I developed is quite simple and straight forward. However, there's a complication when I coded the wrong class. So, I specifically used `'s'` for all the inputs because it clearly tells the computer that we want a char/string class to be stored in the workspace. The reason for this is that we will not be able to use *upper* and *eval* functions since they require the data to be a string. Next, the problem solving is using *upper* and *eval* functions. The upper should make all the all the text in variable 'name' turn into uppercase letters and *eval* should evaluate the mathematical expression based on the user's input. However, we would need to make the mathematical expression a string first because the entered expression must be printed in the output.

1.3. Calculations and Results

There are no calculations involved in this process apart from using *upper* and *eval* functions as described above. When a user inputs a name and a mathematical expression, we should get this output:

```
Enter your name: Josephine Bruin
Enter your mathematical expression: 3 + 5*8
Hello JOSEPHINE BRUIN, your expression 3 + 5*8 evaluates to 43
```

The output is based on the user inputs. Notice that the mathematical expression is printed along with the evaluation of the mathematical expression itself. We can do this using any known mathematical expressions that MATLAB can evaluate. Here's another example

```
Enter your name: adrian loekman
Enter your mathematical expression: 2+2^2*5
Hello ADRIAN LOEKMAN, your expression 2+2^2*5 evaluates to 22
```

The printed name would all be uppercase letters regardless of the input. Note that the first input must be alphabetical, and the second should be numeric in order for the script to run well.

1.4. Discussion

This problem allows us to learn and develop our MATLAB abilities regarding input and output, specifically if the input is string and we require an output with a different class (in this case is a double).
The takeaway from this problem is learning about how to solicit user input which will be evaluated and manipulated using *upper* and *eval* functions. This is a good example about evaluating data. I believe there will be much more to learn from this problem that will be implemented in the following weeks topics

2. Numerical Approximation

2.1. Introduction

The goal of this problem is to use two formulas to calculate the mass of an object moving at velocity v. The two formulas are special relativity and an approximation. Given the resting mass 1kg travelling at a speed 25% the speed of light (c), we can find the mass at that speed using both formulas. Once this is done, we can obtain the percent error of the approximation.

2.2. Model and Methods

The values given for this problem are the mass at rest and velocity. Since we know the ratio of the speed in terms of speed of light, we are able to calculate speed divided by the speed of light without knowing the value of speed of light. In other words, we will not have v and c as separate variables but instead use the variable v/c because this is already a known value. We can write the code in MATLAB like this:

```
v_c = 0.25;
```

since v/c = 25%*c/c then v/c is 25% or 0.25

After obtaining the mass using both methods, calculate the percent error using the formula given. It would be better to set a variable, such like 'diff', to calculate TrueValue-ApproxValue before putting the variable to calculate the percent error. Lastly, print the data using fprintf.

2.3. Calculations and Results

The program has a predetermined data, which is the mass at rest (1 kg) and the speed its travelling at (25% speed of light). We should get this output when the code is run:

```
The mass of the object using special relativity is: 1.032796 kg
The mass of the object using the approximation formula is: 1.031250 kg
The approximate error of these calculations is: 0.150%
```

As we look at the data using the two methods, the results are very similar one to another. In fact, the percent error is only 0.15%.

2.4. Discussion

The two methods give out two outputs that are very similar to one another. In this case, it presents a good approximation with respect to the special relativity formula presented. This conclusion is also derived from the percent error of the two calculations. The difference in the calculations may be caused by the formulas which in the special relativity has a square root in the denominator while the approximation uses a fraction which is multiplied by the ratio squared.

Comparing the two calculations enables us to verify whether the calculations are accurate and precise. This also refers to the code which may have errors due to the incorrect syntax. The 6 decimal places tells readers that the data is similar but different after the third decimal.

3. Digits of µ

3.1. Introduction

This problem is a user interactive code which enables users to input a desired whole number that would indicate how many decimals the value of pi is desired. In other words, after a user inputs a whole number, MATLAB will output the value of pi with the decimal number limited to the whole number inputted.

3.2. Model and Method

The strategy in this problem is to input a whole number. This will be stored in the workspace as double. Then, sprintf is used to convert the data in the workspace so it can be printed into the command window. This code is used to convert user's input into the desired number of decimals in the value pi:

```
text = sprintf('%.*f',n,pi);
```

the variable text will be used in the output with the help of fprintf. `'%.*f'` is used to determine the fixed point notation (or the number of decimals) where the * is going to determine the number of decimals from the user's input. Pi is the data the problem will evaluate.

The last part is to print the text with the sentence stated in the problem. This includes the number of digits inputted by the user and the value of pi to those digits of decimals.

```
fprintf('Pi to %d digits is %s\n', n, text);
```

Notice that n is double and text is char. This is because the code with the variable text converts the value of pi into a char to facilitate the user input.

3.3. Calculations and Results

When the program is executed, the user will be prompted to input the number of digits to print. This is how many decimals in the value of pi.

```
number of digits to print:  8
Pi to 8 digits is 3.14159265
```

We can look that there are exactly 8 digits of decimals in the value of pi. However, when we input a large number such as 60, we get a rather odd output.

```
number of digits to print:  60
Pi to 60 digits is
3.141592653589793115997963468544185161590576171875000000000000
00
```

We can see that after the 48th digit, we get zeros instead of the other digits of pi. Also, referring to the website https://oeis.org/A000796, we get different values of decimals after the 15th digit.

3.4. Discussion

The input and output work well with the code and gives out the value of pi with the decimals from the user's input. In this code, the value of pi becomes wrong after the 16th digit of the decimal. Then, after the 48th digit, the decimals turn into 0.

I believe this is due to the double precision of the command 'pi'. This limits the precision of the value of pi due to the method the CPU uses to calculate pi. After some research, there is a function called 'vpa' that can be used to increase the precision of a value to several digits desired. This way we can get the precise value of pi. In other words, there is a limit to precision of a calculation that is done by MATLAB. Thus, we need to account how many digits of data are precise until it becomes wrong.

4. Heat Transfer

4.1. Introduction

The problem presented in this course is to solve an unknown from a set of known data using the formula shown. In the 3 sections, we are given different knowns. Thus, we will need to rearrange the equations according to the unknown we will find.

4.2. Model and Method

In order to determine the right equation to use for each section, we will need to look at what data is given by each section. We have 1 constant k = 0.45.

Section a is straight forward, no rearranging needed. We can plug in all values stated in the section and get the value of T_f.

```
T_f_a = T_s_a + ((T_0_a - T_s_a)*exp(-k*t_a));
```

This equation above is the equation to solve for section a.

Section b needs rearranging. The process is quite simple. In this section, we would need to find the value of t. So, the main goal of the equation is to isolate t. This is quite easy as we would only need to utilize mathematical knowledge about natural logarithm and exponentials. After that is done, we can easily plug in the values from the data and get the value of t. But, notice that we need to express time in seconds, so we will need to convert the time from hours to seconds. Also, the temperature drops 10 C from 100 C, so it is 90 C.

```
t_b = (log(T_f_b-T_s_b)-log(T_0_b-T_s_b))/(-k);
t_b_seconds  = t_b * 60;
```

The equation for section b needs 'log' which is the MATLAB code for natural logarithm (ln). We multiply the time by 60 to convert the time to seconds.

Section c is like section b because it needs rearranging in order to isolate thermal chamber temperature. After rearranging we can easily find the answer, which is the temperature of the thermal chamber. The rearrangement also involves using natural logarithm to eliminate the exponential shown on the equation. After the rearrangement, we can formulate the equation in MATLAB have the syntax: `T_s_c = (T_f_c - T_0_c * exp(-k*t_c))/(1-exp(-k*t_c));`

The syntax exp is used to evaluate e values in MATLAB.

Lastly, the outputs will be printed using text formatting %f in order to maintain the decimals as stated on the problem.

4.3. Calculations and Results

Each section has different inputs. For section A, the inputs are the initial temperature, thermal chamber temperature, and time. For section B, they are initial temperature, thermal chamber temperature, and final temperature. For section C, they are initial temperature, final temperature, and time. The output from the inputs and through calculations are below:

```
(a) The final temperature of the object is 44.44302 degrees Celsius
(b) The time needed for the object is 19 seconds
(c) The temperature of the thermal chamber is 29.62691 degrees Celsius
```

4.4. Discussion

The conclusion from this problem is we can use MATLAB as a high-tech calculator which is user friendly. We can calculate any equations with variables easily if we can identify each variable correctly.

The takeaway lessons include refreshing the skills of rearranging an equation. Then, finding the unknown from a set of known data. The MATLAB code is quite simple because the rearranging is done by hand. These are then converted into MATLAB syntax. From this problem, I wonder whether I need to rearrange the equations. I believe there is a way which we can get the answers to each section while only using the equations stated on the problem.