Linh Tang

CEE/MAE M20

UID: 205542275

Oct 10, 2020

HOMEWORK 1

1. String manipulation

1.1 Introduction

The goal of this problem is to ask a user inputs name and UID, then print out a greeting with the name in uppercase and UID backwards.

1.2. Model and methods

The script first gets two string inputs for name and UID by using input function in MATLAB. For name, upper function is called to capitalize all characters for name no matter what the user inputs, lower or upper cases. Besides, UID input can be stored in backwards after getting input from the user by using fliplr function.

```
get_name = upper(input('Enter your name: ', 's'));
get_uid = fliplr(input('Enter your UID: ', 's'));
```

For two inputs, the user needs to enter nine numbers for UID, no letters in it. To check the validity of the input, str2double might be used if the string can be converted into numbers of not; length function is used to get the length of the input. Function isnan() is used to check if elements are NaN.

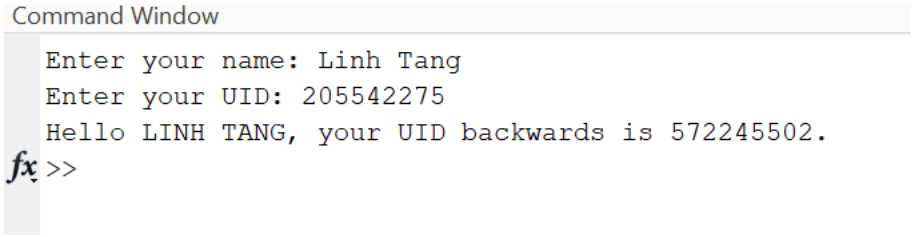
```
if (isnan(str2double (get_uid)))
      error ('Invalid input. Only numbers for UID.');
elseif (length(get_uid) ~= 9)
      error ('Invalid input. Enter 9 numbers for UID.');

end
```

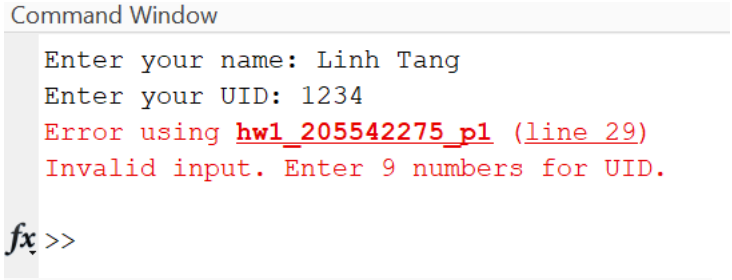
After getting all valid inputs for name and UID， a greeting sentence with the name and UID backwards is printed out to the command window by using fprintf function.

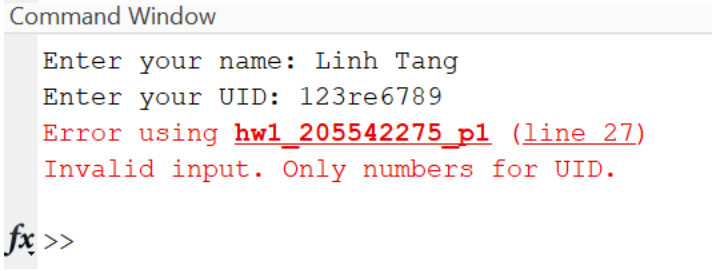
```
fprintf('Hello %s, your UID backwards is %s.\n', get_name, get_uid);
```

1.3 Results:



Test for UID validity





1.4 Discussion

The script works well in getting inputs and displaying expected sentence in the command window. From this script, more inputs can be obtained for a better script. Besides, checking validity for name and some other invalid inputs should be considered.

2. Sterling's approximation

2.1 Introduction

This problem is about calculate factorials by using Sterling's approximation method and determine the relative error. The percent error is determined by compare the approximation value with the exact value (obtained by using built-in Matlab function).

2.2 Model and method

The value of number n that is required for factorial calculation is obtained by asking the user inputs in the command window; Matlab's input function is called here.

```
n = input('Enter a value of n: ');
```

The approximation for n! is determined through the given Sterling's formula. The value here needs to be rounded to the nearest whole number; round() function is called to achieve that requirement.

```
nf_approx = round(sqrt(2*pi*n) * (n/exp(1))^n);
```

For the next step, the exact value for n! is obtained by using the built-in function factorial in Matlab.

```
nf_exact = factorial(n);
```

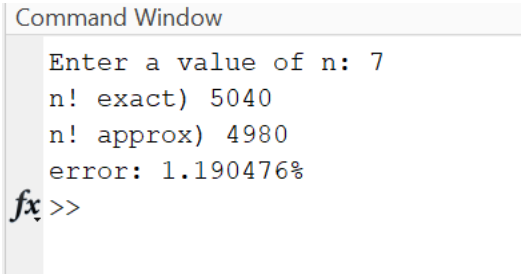
The relative error is determined through the formula for percent error based on calculated approximation and exact values.

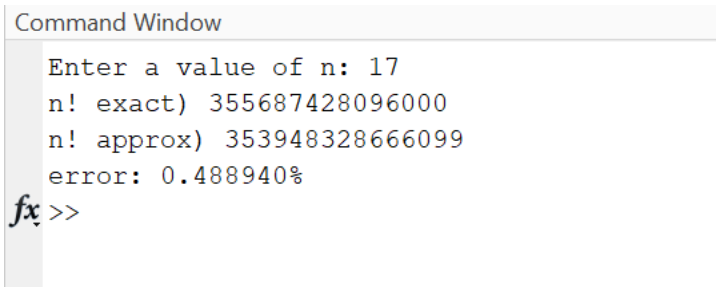
```
error = (nf_exact - nf_approx)/nf_exact *100;
```

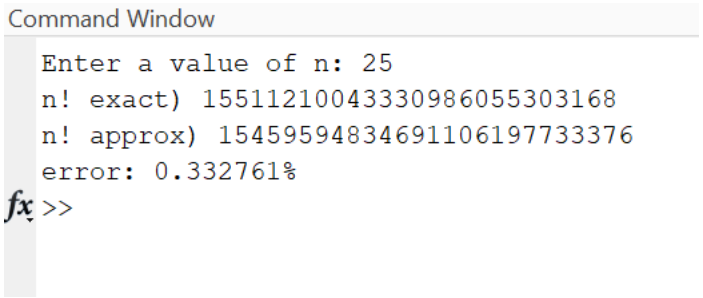
After getting all values, the result is displayed by using fprintf function.

```
fprintf('n! exact) %0f\n', nf_exact);
fprintf('n! approx) %0f\n', nf_approx);
fprintf('error: %.6f%%\n', error); % .6f to get 6 decimal places.
```

2.3 Result







2.4 Discussion

The script works as expected with different input numbers. In the future, this script can be developed by adding a loop to investigate the relative error for factorials of different numbers. From the results, the larger number n is, the less the relative error is.

3. Law of Sines and Cosines

3.1 Introduction

The goal of this problem is to find three angles α, β, γ (in degree) of the triangle shown in the problem by using the Law of Sines and Cosines, and then verify the sum of three angles whether 180 degree or not.

3.2 Model and method

At first, all given values of side lengths need to be stored into variables, respectively.

```
a = 10;
b = 20;
c = 25;
```

In part a, the Law of Cosines is used to compute cosine of the angle α. After that, acos function is used to get the inverse cosine of alpha, this is how alpha is obtained in radian. However, the problem requires alpha in degree, so rad2deg function is called to convert the unit.

$$a^2 = b^2 + c^2 - 2bc\cos(a)$$

⇨ $\cos(\alpha) = (b^2 + c^2 - a^2)/2bc$

Codes:

```
cos_a = (b^2 + c^2 - a^2)/(2*b*c);
alpha = acos(cos_a);
alpha_deg = rad2deg(alpha); % Convert the angle rad to degree

fprintf ('a)alpha is %.2f degree.\n', alpha_deg);
```

In part b, two angles β and γ are computed by using the Law od Sines:

$$\frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma}$$

⇨ $\sin(\beta) = b.\sin(\alpha)/a$; $\sin(\gamma) = c.\sin(\alpha)/a$

⇨

Similar to α calculation, asin function is used to get inverse sines of β and γ, and then using rad2deg to get correct units.

```
% Calculate sin(beta) and beta
sin_b = b*sin(alpha)/a;
beta = asin(sin_b); % Using inverse sine to get beta in rad
beta_deg = rad2deg(beta); % Convert the angle rad to degree

% Calculate sin(gamma) and gamma
sin_g = c*sin(alpha)/a;
gamma = asin(sin_g); % Using inverse sine to get gamma in
rad
gamma_deg = rad2deg(gamma); % Convert the angle rad to
degree
```

Based on the figure of the triangle, γ is an obtuse angle while α, β are acute angles. Therefore, gamma calculation requires one more step to figure out a correct value since the Law of Sines give more than value; γ and 180 – γ have the same value of sine.

```
if(gamma_deg < 90)
    final_g = 180 - gamma_deg;
end
fprintf ('b)beta is %.2f degree.\n  gamma is %.2f
degree.\n', beta_deg, final_g);
```

In part c, sum of three angles is calculated by using simple arithmetic operation (+) to add them together.

```
sum = alpha_deg + beta_deg + final_g;
fprintf('c)The sum of three angles is %.0f degree.\n',sum);
```

3.3 Results

Command Window

```
a)alpha is 22.33 degree.
b)beta is 49.46 degree.
  gamma is 108.21 degree.
c)The sum of three angles is 180 degree.
fx >>
```

3.4 Discussion

The point that the law of sines might give two values for gamma is usually missed out; therefore, incorrect value of gamma was obtained at the first trial. Instead of using the Law of Since, gamma can be calculated, based on the fact that the sum of three angles of a triangle is equal to 180 degree. By this way, part c will be combined to part b. However, the good point of this script when separating part b and part c is showing that the law of sines does not give only one value.

4. Oblate Spheroid

4.1 Introduction

This problem is about surface area calculation of an oblate Spheroid. The program asks the user inputs values for equatorial radius and polar radius, and then gives out the approximation and the exact value for the surface are.

4.2 Model and method

Before the calculation, the values for equatorial radius and polar radius are collected by calling input function, and then storing values to r1 and r2, respectively.

```
r1 = input('Enter the value for equatorial radius (r1): ');
r2 = input('Enter the value for polar radius (r2): ');
```

The exact surface area is calculated by using the formula:

$$\gamma = \arccos\left(\frac{r_2}{r_1}\right)$$

$$A(r_1, r_2) = 2\pi\left(r_1^2 + \frac{r_2^2}{\sin(\gamma)}\log\left(\frac{\cos(\gamma)}{1-\sin(\gamma)}\right)\right),$$

Codes:

```
gamma = acos(r2/r1);

% Define terms for the formula
sin_g = sin(gamma);
cos_g = cos(gamma);
log_term = log(cos_g/(1 - sin_g));

A_exact = 2*pi*(r1^2 + (r2^2/sin_g)* log_term);
```

The approximation for the surface area is calculated by using:

$$A(r_1, r_2) \approx 4\pi\left(\frac{r_1+r_2}{2}\right)^2$$

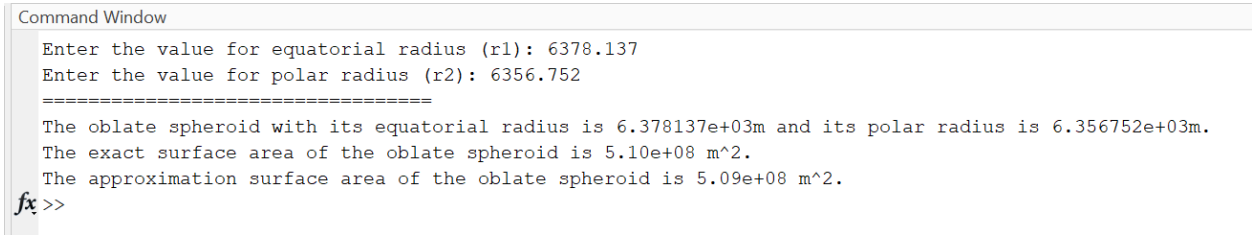
```
A_approx = 4*pi*((r1 + r2)/2)^2;
```

After getting all the values, the approximation and exact values of the surface area is displayed with two decimal places.

```
fprintf('The exact surface area of the oblate spheroid is %.2e
m^2.\n',A_exact);
fprintf('The approximation surface area of the oblate spheroid is
%.2e m^2.\n',A_approx);
```

4.3 Results

The script is tested by calculating the surface area of the Earth, with the equatorial radius is 6378.137m and polar radius is 6356.752m.



4.4 Discussion

In this script, two radiuses need to be entered in the right order since the exact formula only works when polar radius(r2) is less than the equational radius(r1). To improve the script, a section of codes might be added to check if r2 < r1; if not, a function is needed to swap two values.