Linh Tang

CEE/MAE M20

UID: 205542275

Oct 24, 2020

# HOMEWORK 3

## 1. Golden Ratio

### 1.1 Introduction

The goal of this problem is using two different methods to calculate the Golden ratio. The first method is calculating the value of 2cos(36°) by using Taylor's series the uncertainty input *e*. The second method is calculating the ratio of two adjacent numbers in the Fibonacci sequence, and then display how many numbers the sequence takes to get the Golden Ratio.

### 1.2 Model and methods

- Calculating 2cos(36°):

  First, the script takes input for uncertainty by using input function. The input must be a positive value.

```
uncertainty = -1;
while (uncertainty < 0)
    uncertainty = input('Enter an uncertainty: ');
end
```

  After that, cos function is used to calculate the exact value of 2cos(36°) which is also golden ratio. In Matlab, cos function only takes radian as its argument, so calling deg2rad to convert 36° to radian. The exact value will be compared with the approximate value.

```
rad = deg2rad(36);
exact = 2*cos(rad);
```

The approximation of cos(36°) is obtained by using Taylor series method. The difference between the exact value and the approximate value is determined and then is compared with the uncertainty to break the loop and get the value for cos(36°).

$$\cos(x) = \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n}}{(2n)!}$$

```
err = 1;
n = 0;
sum = 0;
while(err >= uncertainty)

    cos = (-1)^n* rad^(2*n)/factorial(2*n);
    sum = sum + cos;
    approx = 2* sum;
    err = abs(approx - exact);
    n = n + 1;
end
```

- Finding two consecutive numbers giving the golden ratio in the Fibonacci sequence

Before the calculation, setting up the Fibonacci sequence is the first thing needs to be done. The first two numbers 0 and 1 are hard-coded; the next number is the sum of the two preceding numbers and so on. The golden ratio between two numbers is ϕ shown as the following:

$$\frac{a+b}{a} = \frac{a}{b} = \phi \longrightarrow \phi = 1 + \frac{1}{\phi} \longrightarrow \phi^2 - \phi - 1 = 0$$

$$\phi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ldots$$

The target error used in this part is calculated in part b (first method). The purpose of while loop is forming the Fibonacci sequence and calculate the ratio between two consecutive numbers; the if condition is used to break the loop when it find out the golden ratio.

```
actual_phi = (1 + sqrt(5))/2; %Exact value for golden ratio
error = 1;
targetError = err; % Using the erro in part b
i =0;
% Define first two numbers in the Fibonacci sequence
a = 0;
b = 1;
while(error >= targetError)
% Creating Fibonacci sequence
    c = a;
    a = b;
    b = c + a;
    % Calculating ratio
    phi = a/b;
    phi = 1+ phi;
    error = abs(actual_phi - phi);
    if (error < targetError)
        break;
    end
    i = i + 1; % Increment
end
```

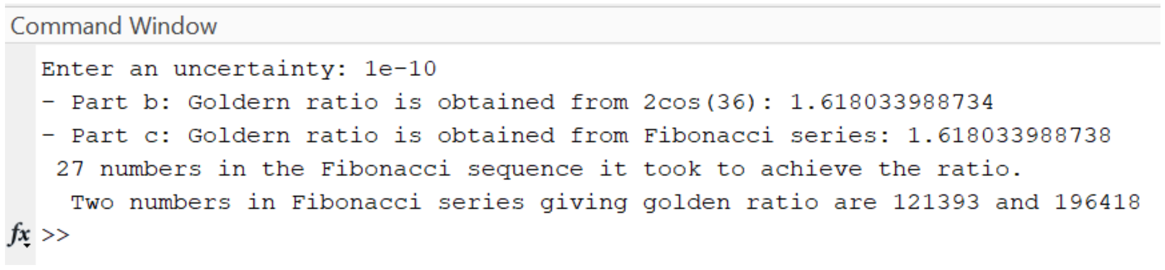
- Displaying the results:

  The fprintf function is called to display the results; the ratios are shown with 12 decimal places to see the difference between two methods. Since first two numbers are defined outside of the loop, adding two to the counter variable i of the loop, numbers the Fibonacci takes to achieve the golden ratio is determined.

```
fprintf ('- Part b: Goldern ratio is obtained from
2cos(36): %.12f\n', approx);
fprintf ('- Part c: Goldern ratio is obtained from
Fibonacci series: %.12f\n', phi);
fprintf (' %d numbers in the Fibonacci sequence it took to
achieve the ratio.\n ', i+2);
fprintf(' Two numbers in Fibonacci series giving golden
ratio are %d and %d\n', a, b);
```

## 1.3 Result

With the input for uncertainty is 1e-10, the golden ratio of  2cos (36°) is 1.618033988734, and the golden ratio of  two numbers 121393 and 196418 in the Fibonacci gives us the golden ratio 1.618033988738. The sequence takes 27 numbers to get the golden ratio.



```
Enter an uncertainty: 1e-10
- Part b: Goldern ratio is obtained from 2cos(36): 1.618033988734
- Part c: Goldern ratio is obtained from Fibonacci series: 1.618033988738
 27 numbers in the Fibonacci sequence it took to achieve the ratio.
  Two numbers in Fibonacci series giving golden ratio are 121393 and 196418
>>
```

1.4 Discussion

The most common mistake I encountered in this problem was infinite loops; it was because some variables did not get updated or were set up wrong, so the program were unable to exit the loops. Another mistake was about setting the Fibonacci sequence, I realized that it was easier to assign values for first two numbers since it did not follow the algorithm. Besides 2cos(36°), 2sin(54°), 1+ 2 sin(18°) also give us golden ratio; these can be tested in the future script.

## 2. Guess the number:

2.1 Introduction

This program generates a random number from 1 to 50, and then asks users to guess the number. The user has five tries to guess the number, otherwise the game will end and ask if the user want to start a new game.

2.2 Model and methods

This program needs nested while loops; one is to restart the game when the user want to play it again, and the on inside is for guessing numbers in five tries. The randi function is called to generate a random number from 1 to 50.

```
num = randi([1, 50]);
```

The outer while loop takes the answer “Y” for the question “Do you want to play again? (Y/N)” as its argument to continue the program. If the user enters any characters other than “y” or “Y”, the program will exit this while loop. In this one, a random number will be generated at the beginning, and getting the answer to restart the game at the end. The upper function is used when getting the answer, so that it does not matter if the user enters upper case or lower case; the character will be converted to upper case.

```
answer = 'Y';
while(answer == 'Y')
    num = randi([1, 50]);
    trials = 0;
    guess = -1;
    while
       …
    end
    answer = upper(input('Do you want to play again?(Y/N)
','s'));

end
```

The inner while loop lets the user guess the number and compares that number with the value the program randomly generates. The number of trials less than 5 is boolean argument for this while loop since trials is initialized as 0, thus this will give the user maximum 5 tries to guess the number. The trials variable gets updated for each time.

```
while (trials < 5)
        guess = input('Guess the number: ');
        …
        trials = trials + 1;
end
```

There are three if conditions to break this inner while loop. The first one is when the user enter an invalid input for guessing number; the number is not an integer and is out of range.

```
if(guess <1 || guess >50 || mod(guess,1)~= 0 ||
~isreal(guess))
      fprintf ('Invalid input.\n');
      break;
end
```

The second if condition is when the user guesses the number right; it will congrats the user and then break this loop.

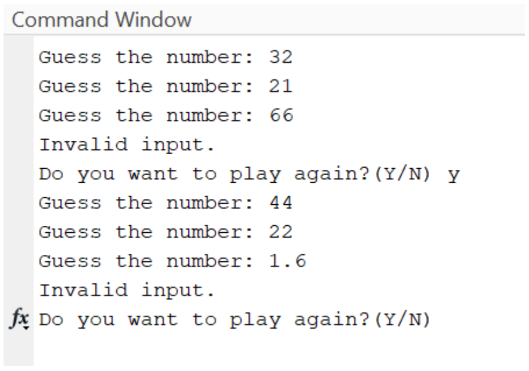
```
if (guess == num)
      fprintf('Nice guess, you found my number!\n');
      break;
end
```

The last if condition is when five tries run out, but the user does not get the correct number.

```
if(trials >= 5)
      fprintf('Out of tries, better luck next time\n')
      break;
end
```

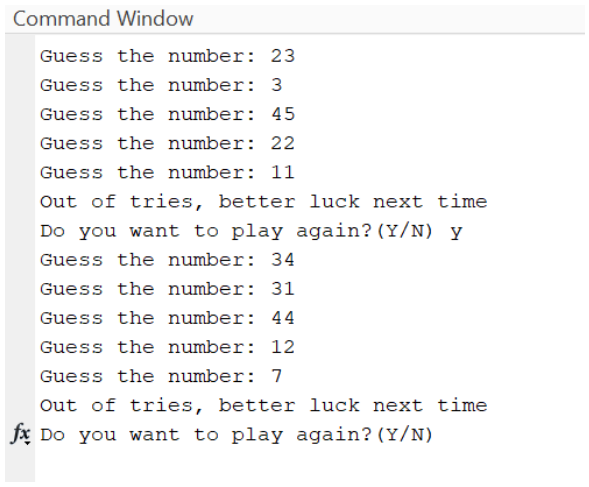
## 2.3 Result

- Test for validity of the input


Command Window

```
Guess the number: 32
Guess the number: 21
Guess the number: 66
Invalid input.
Do you want to play again?(Y/N) y
Guess the number: 44
Guess the number: 22
Guess the number: 1.6
Invalid input.
Do you want to play again?(Y/N)
```

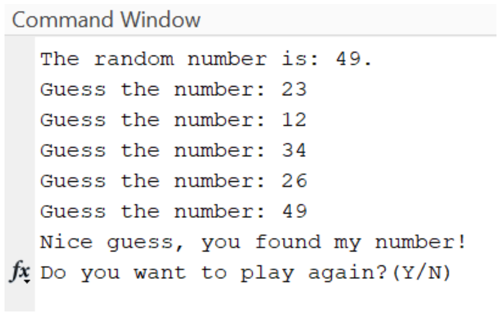
- When the user finishes 5 tries, but can not get the correct number



Command Window

```
Guess the number: 23
Guess the number: 3
Guess the number: 45
Guess the number: 22
Guess the number: 11
Out of tries, better luck next time
Do you want to play again?(Y/N) y
Guess the number: 34
Guess the number: 31
Guess the number: 44
Guess the number: 12
Guess the number: 7
Out of tries, better luck next time
fx Do you want to play again?(Y/N)
```

- When the user guesses the number right.



Command Window

```
The random number is: 49.
Guess the number: 23
Guess the number: 12
Guess the number: 34
Guess the number: 26
Guess the number: 49
Nice guess, you found my number!
fx Do you want to play again?(Y/N)
```

(The first line is for testing)

## 2.4 Discussion

In this problem, the challenge is we need to figure out the correct order for the codes, so the program can give the result as we expects; which one should be in the outer while loop and which one should be included in the inner. The number of tries needs to get updated before the last if condition (breaking the loop when it is greater than five). Since it is difficult to guess the number right in five tries, the program can be designed in the way giving the user an idea whether the number is too low or too high.

## 3. SIR Simulation of the Spread of Disease

### 3.1 Introduction

This script is simulating the spread of influenza in a school using forward Euler method, displaying a plot for number of infected students, and returning the maximum number of infected students. The program is done by using for loop, if condition, and some functions for plotting.

### 3.2 Model and methods

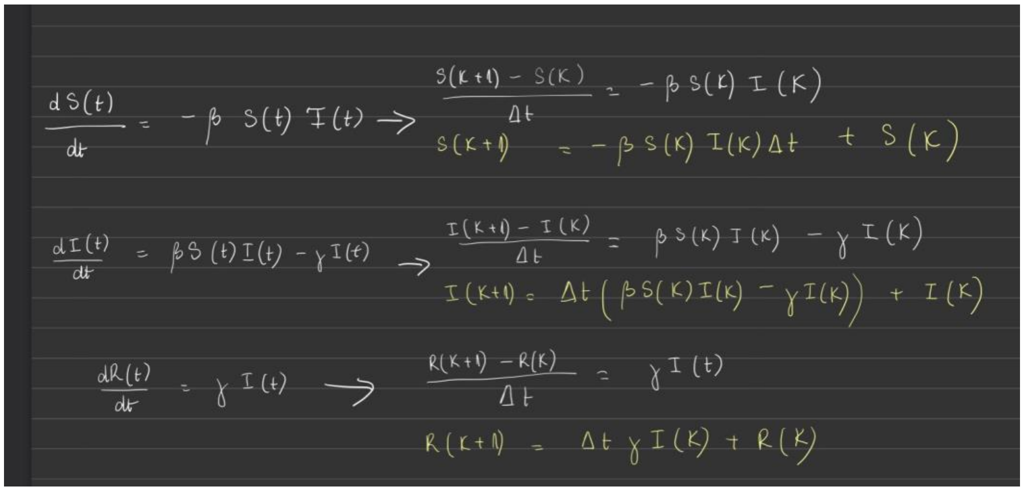
The first thing needs to be done is initializing variables and setting up values for constants which are used in the equations.

```
I_0 = 1;
S_0 = 700;
R_0 = 0;
beta = 0.0026;
gamma = 0.5;
```

The number of time steps is calculated using the initial time (ti), final time(tf) and the time step size (dt). Ceil() function is called to round the result to the nearest integer that greater or equal to that one.

```
ti = 0;
tf = 20;
dt = 0.1;
nt = ceil((tf-ti)/dt);
```

After that, the discretized governing equations are derived from the given SIR equations by using Forward Euler method.



$$\frac{dS(t)}{dt} = -\beta\, S(t)\, I(t) \rightarrow \frac{S(K+1) - S(K)}{\Delta t} = -\beta\, S(K)\, I(K)$$

$$S(K+1) = -\beta\, S(K)\, I(K)\, \Delta t + S(K)$$

$$\frac{dI(t)}{dt} = \beta S(t) I(t) - \gamma I(t) \rightarrow \frac{I(K+1) - I(K)}{\Delta t} = \beta\, S(K)\, I(K) - \gamma\, I(K)$$

$$I(K+1) = \Delta t \left( \beta S(K) I(K) - \gamma I(K) \right) + I(K)$$

$$\frac{dR(t)}{dt} = \gamma\, I(t) \rightarrow \frac{R(K+1) - R(K)}{\Delta t} = \gamma\, I(t)$$

$$R(K+1) = \Delta t\, \gamma\, I(K) + R(K)$$

```
S_k1 = - beta * S_k * I_k * dt + S_k;
I_k1 = (beta * S_k * I_k - gamma * I_k)*dt + I_k;
R_k1 = gamma*I_k*dt + R_k;
```

These equations are put in a for loop to get the results for all the time steps (nt). The for loop starts from 1 to nt – 1 since the calculations will return the values for the next time step by using the current values. After each calculation, new values now become old values before getting into another iteration.

```
for k = 1:nt-1
    % Forward Euler method
    S_k1 = - beta * S_k * I_k * dt + S_k;
    I_k1 = (beta * S_k * I_k - gamma * I_k)*dt + I_k;
    R_k1 = gamma*I_k*dt + R_k;
    % Updating the new values to the old values
    S_k = S_k1;
    I_k = I_k1;
    R_k = R_k1;
end
```

The following if – condition is performed to find the maximum of infected students. The Imax gets updated whenever the program finds the calculating Ik is greater than the current maximum.

```
if(I_k > I_max)

    I_max = I_k;

end
```

To get the plot, scatter function is used to plot values for I over time. The purpose of "hold on" is telling Matlab to wait for adding other points to the current axes instead of closing the figure.

```
scatter(ti + dt*k,I_k,'filled','b');
hold on
```

The following functions are used to name the title, axes, and enable the grid.

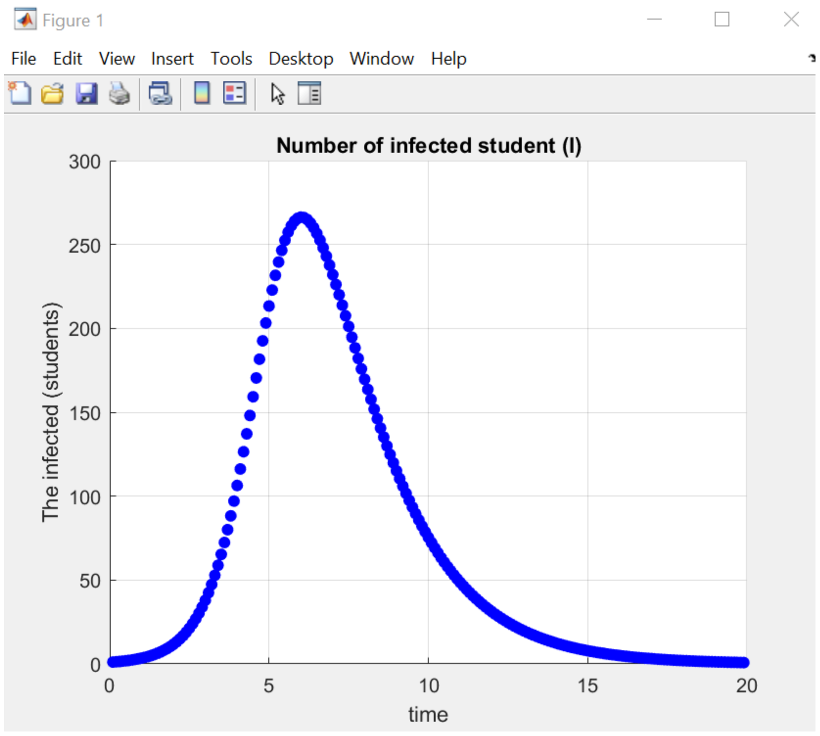
```
xlabel ('time');
ylabel ('The infected (students)');
title ('Number of infected student (I)');
grid on
```

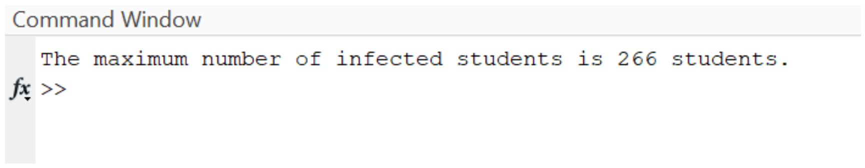
To round the maximum infected students, round() function is called to round the result to the nearest integer.

```
fprintf('The maximum number of infected students is
%d students.\n', round(I_max));
```

## 3.3 Results

The maximum number of infected students is 266. students.





Command Window

```
The maximum number of infected students is 266 students.
>>
```

3.4 Discussion

In this problem, the most important part is getting the discretized governing equations correct. For the future script, plots for all three variables S, R, I might be displayed at the same time to get a clearer trend for the SIR model. Array can be used to store all the values, so plots can be done outside the for loop. To find the maximum values, the built in function max() can be used to derive the maximum value from the plot