ENGI 1331: Exam 2 Review – Additional Practice Problems Fall 2020

Disclaimer: The problems given on the exam may not be similar to those found here. The problems below are designed only to provide additional practice with topics (for example, for and while loops) most likely to be covered on the exam.

Euclid's Algorithm

One of the oldest algorithms still in common use today is called the Euclidean Algorithm (named after the ancient Greek mathematician who first described it in c. 300 BCE). This algorithm provides an efficient method for determining the greatest common divisor (GCD) of two integers. The greatest common divisor is the largest integer that divides two numbers without a remainder. In this problem, you will create a function that can calculate the GCD of two integers using Euclid's Algorithm, and then apply that function in order to analyze a provided vector of integers.

To find the GCD of two integers a and b using the Euclidean Algorithm, we first find the remainder, r_1, of a and b. (In MATLAB we can use either the remainder function, **rem()**, or the modulus function, **mod()**, for this.)

$$\frac{a}{b} = q_1 \text{ rem } r_1$$

After this step, if $r_1 = 0$, then b is the GCD of a and b. Instead, if $r_1 > 0$, then we find the remainder of b and r_1.

$$\frac{b}{r_1} = q_2 \text{ rem } r_2$$

If $r_2 = 0$, then r_1 is the GCD of a and b. If $r_2 > 0$, then the algorithm continues by calculating the remainder of r_1 and r_2.

$$\frac{r_1}{r_2} = q_3 \text{ rem } r_3$$

If $r_3 > 0$, we continue this process until we find a remainder that equals 0.

$$\frac{r_2}{r_3} = q_4 \text{ rem } r_4$$

$$\frac{r_2}{r_3} = q_4 \text{ rem } r_4$$

$$\vdots$$

$$\frac{r_{n-2}}{r_{n-1}} = q_n \text{ rem } r_n$$

If $r_n = 0$, then r_{n-1} is the GCD of a and b.

Task 1:

Function

Create a function called **EuclidAlg** that calculates the greatest common divisor of two integers using the Euclidean Algorithm. Your will receive two integers as inputs, and will return their greatest common divisor.

Function Inputs:
1. The first integer, a
2. The second integer, b

Function Outputs:
1. The greatest common divisor of a and b

The function header should be formatted similarly to the following:

```
function out = EuclidAlg(in1, in2)
```

Remember you are free to use whatever variable names you want, but they must be listed in the same order as given in the input/output lists provided above.

Task 2:

Main Script

Now that we have implemented the Euclidean Algorithm with our function, **EuclidAlg**, we will use it to analyze a vector of integers. First, load the matrix of integers, **IntMatrix**, contained in **IntMatrix.mat**. Allow the user to select a column of this matrix. Include the number of columns of the matrix in the input prompt. If the user enters a number greater than the total number of columns, continue prompting the user until a valid column number is entered.

Task 3:

Main Script

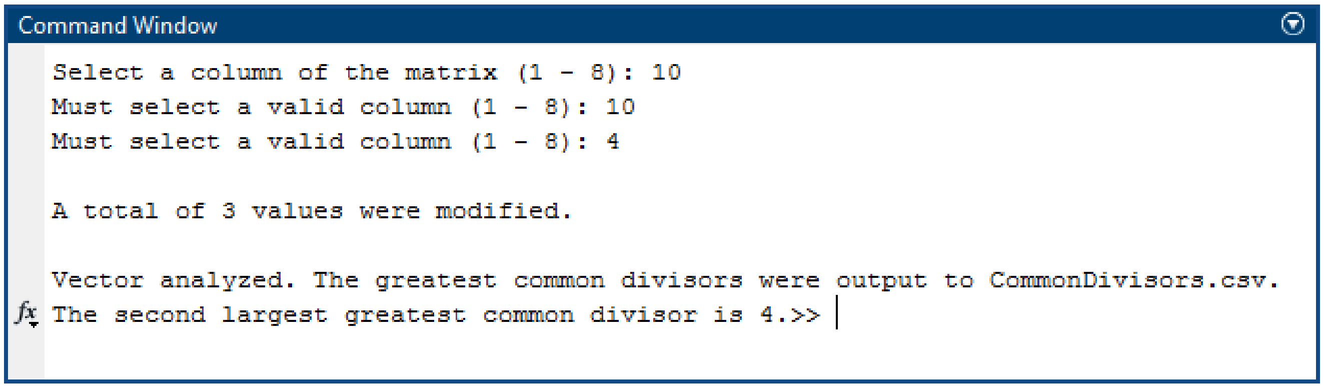
Since we are only considering values that positive (i.e. greater than zero), check to make sure that all values in the chosen column are positive. If a number is negative, replace it with its absolute value. If a number is zero, replace it with the sum of the elements before and after it. (You can assume that there are no zeros in either the first or last position the column vector.) Output to the command window the number of values that were modified.

Task 4:

Main Script

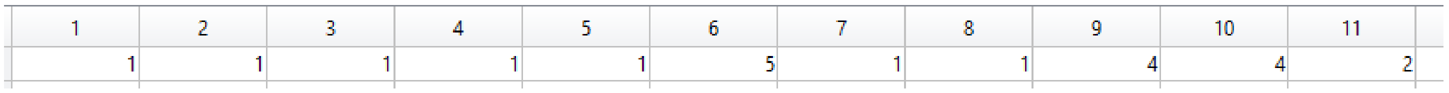
Using your function, **EuclidAlg**, calculate the greatest common divisor of every adjacent element of the chosen column vector, and then store these results as a row vector. Save this vector to **CommonDivisors.csv**. Finally, output the second highest value in this vector to the command window. **Hint:** There may be multiple occurrences of the highest value. If this is the case, how could you determine the second largest value?

Sample Output



```
Select a column of the matrix (1 - 8): 10
Must select a valid column (1 - 8): 10
Must select a valid column (1 - 8): 4

A total of 3 values were modified.

Vector analyzed. The greatest common divisors were output to CommonDivisors.csv.
The second largest greatest common divisor is 4.>>
```

CommonDivisors.csv



1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	1	5	1	1	4	4	2

A Not-So-Famous Sequence

There's been a good deal of ruckus made in recent years about the wide array of occurrences of the Fibonacci sequence found in nature. This particular sequence of integers, first known to Europeans through the work of the great Italian mathematician, Leonardo Fibonacci (c. 1170 – c. 1240-50), can be found all over the natural world, from the arrangement of the pines of a pine cone to the number of petals of a daisy. However, much less is known about the contributions of Fibonacci's older half-cousin, once removed, the mediocre mathematician Ignoramius "Velveeta" Fibonacho (c. 1130 - ?). Fibonacho also discovered an interesting sequence of integers, which for one reason or another hasn't quite caught on.

The so-called Fibonacho sequence can be described as follows: given two starting values, F_1 and F_2, if F_2 is odd, then $F_3 = F_2 + F_1$. However, if F_2 is even, then $F_3 = F_2 - F_1$. This leads to the following general formula:

$$F_n = \begin{cases} F_{n-1} + F_{n-2}\,, & \text{if } F_{n-1} \text{ is odd} \\ F_{n-1} - F_{n-2}, & \text{if } F_{n-1} \text{ is even} \end{cases}$$

Task 1:

Function

Create a function called **Fibonacho**, which, given a 1x2 vector containing the sequence's two starting values and the number of desired terms of the sequence, returns the Fibonacho sequence with the desired number of terms.

Function Inputs:
1. The two starting values of the sequence (a 1x2 vector)
2. The total number of terms of the sequence, n

Function Outputs:
1. A Fibonacho sequence containing n terms

The function header should be formatted similarly to the following:

```
function out = Fibonacho(in1, in2)
```

Remember you are free to use whatever variable names you want, but they must be listed in the same order as given in the input/output lists provided above.

Task 2:

Main Script

Now that we've completed the **Fibonacho** function, let's put it to good use. First, prompt the user to enter the starting values of the sequence as a 1x2 vector. If the user enters a vector that does not have the correct dimensions, display a warning and prompt the user to reenter the vector. If the user does not enter a correct vector after 3 attempts, produce an error and terminate the program. Then prompt the user to enter the desired number of elements of the sequence. This number should be greater than 2. If it is not, continue prompting the user until a valid value is entered.

Task 3:

Main Script

Use the **Fibonacho** function to construct the desired Fibonacho sequence. To analyze our sequence, we will calculate the average of every three adjacent values. Starting with the second element of the sequence, calculate the average of the current element, the previous element and the next element. If the average is not a whole number, replace the current element (i.e. center element) with the average rounded either up or down, according to these rules:

- If the center element is greater than the average, then round up.
- If the center element is less than the average, then round down.

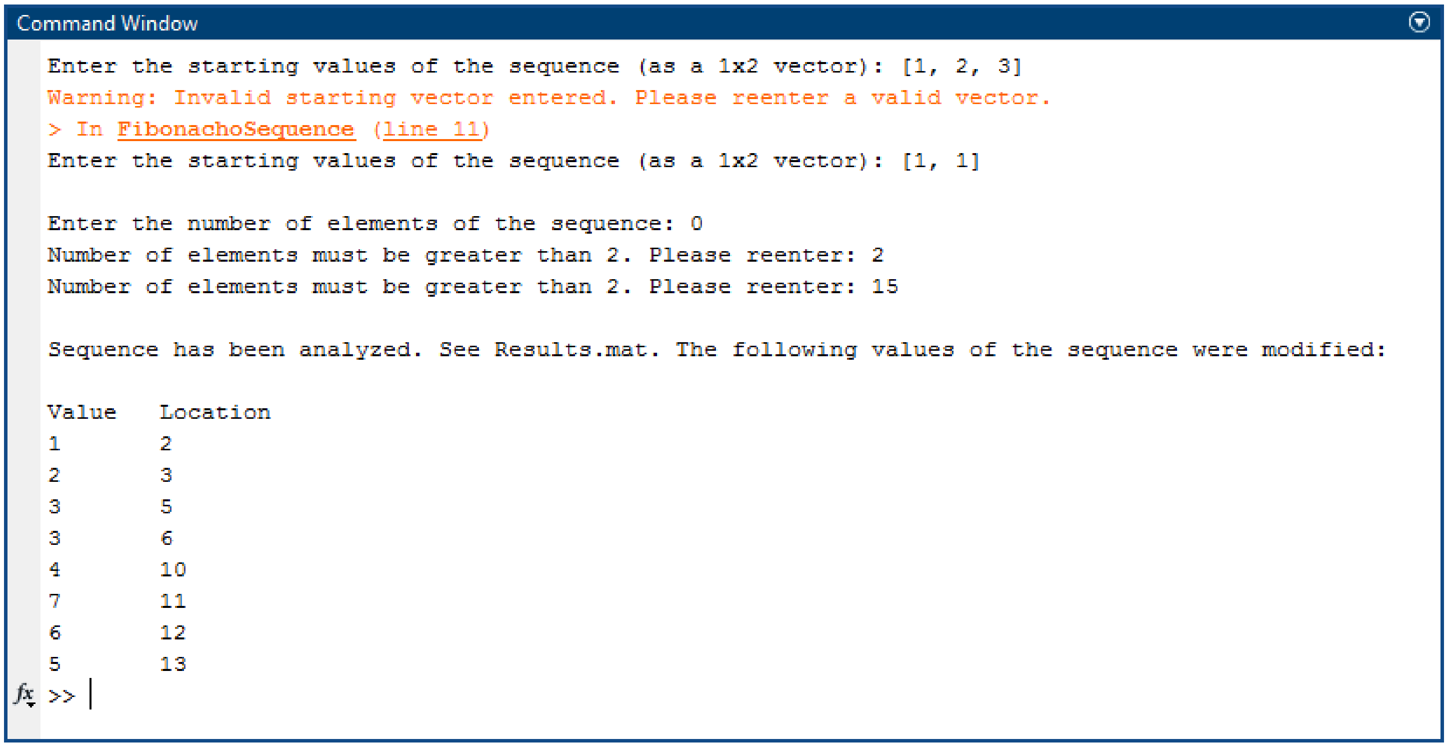
As you modify the sequence, keep track of the changes you make. Create a matrix that contains the modified value in column 1, and that value's location in column 2. Every time another element of the sequence is modified, a new row should be added to the matrix.

Task 4:

Main Script

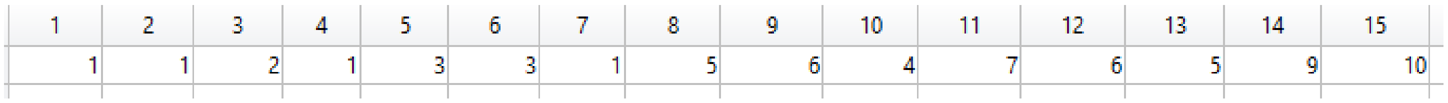
Save your modified sequence and the matrix of tracked changes to a .mat file named **Results.mat**. Finally, output the matrix of changes to the command window, formatted as a table. **Note:** The table should be produced using **fprintf()** within a loop. Use of the function **cell2table()** or any similar function is not allowed.

Sample Output

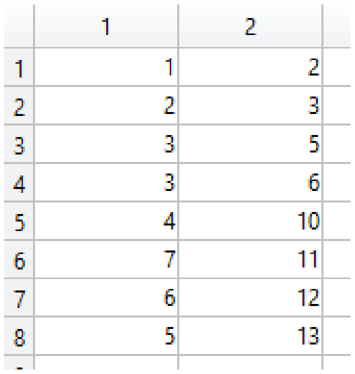

Command Window

```
Enter the starting values of the sequence (as a 1x2 vector): [1, 2, 3]
Warning: Invalid starting vector entered. Please reenter a valid vector.
> In FibonachoSequence (line 11)
Enter the starting values of the sequence (as a 1x2 vector): [1, 1]

Enter the number of elements of the sequence: 0
Number of elements must be greater than 2. Please reenter: 2
Number of elements must be greater than 2. Please reenter: 15

Sequence has been analyzed. See Results.mat. The following values of the sequence were modified:

Value   Location
1       2
2       3
3       5
3       6
4       10
7       11
6       12
5       13
>> 
```

Results.mat



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	1	3	3	1	5	6	4	7	6	5	9	10



	1	2
1	1	2
2	2	3
3	3	5
4	3	6
5	4	10
6	7	11
7	6	12
8	5	13