Lab Assignment #4

Due 2/19/2016 at 4pm on bCourses

Some guidelines for successfully completing an E7 assignment:

- First convert the engineering problem into a function that can be effectively calculated.
  - The function should be written down using pencil and paper in the language of mathematics
- Program the function.
  - Translate the mathematical language into a programming language, i.e., the language understood by your computer.
  - Then run the program and test it to see if its outputs match those specified by the function. If they do, compute the function on the input data and check that your answers make sense and are reasonable. If they do not, look for any mistakes in your math or in your coding.

The assignment will be partially graded by an autograder which will check the values of the required variables, so it is important that your function names are exactly what they are supposed to be and the input and output arguments are in the correct order (names ARE case-sensitive). Instructions for submitting your assignment are included at the end of this document.

1. Interest

   The interest rate, r, on a principal, P_0, is a payment for allowing the bank to use an investor's money. Compound interest is accumulated according to the recursive formula $P_n = (1+r)P_{n-1}$, where P_n is the balance after n compounding periods (e.g. years), and P_{n-1} is the balance after the previous period. The interest rate r is given in decimal, e.g. 0.1, which is 10%. Write a **recursive** function with header
   `function [years]=mySavingsPlan(P0, rate, goal)`
   where `years` is the number of years it will require for an investor to reach the investment goal defined by `goal` while investing an original amount `P0` at the annual interest rate specified by the variable `rate`. For the purpose of this problem, consider the investment balance to change only once per year (i.e. the output `years` should be a whole number).
   Test Case:

   ```
   >> years = mySavingsPlan(2000, 0.15, 10000)

   years =

       12
   ```

2. Searching

   Write a function with the header
   `function [value, location] = myMin(P)`

where `P` is a class double (numeric) array. The output `value` is the minimum value in the array `P` and `location` is the index of the minimum value. Note that the minimum value could appear more than once. If this is the case, `location` should specify the indices of all occurrences of the minimum value. You may **NOT** use the built-in MATLAB commands `min` or `find` in your function.

Test Case 1:

```
>> P=[5, 4, 5, 2, 6, 7, 1, 8, 9];
>> [value, location] = myMin(P)

value =
      1

location =
      7
```

Test Case 2:

```
>> P = [2, 3, 5, 7, 2, 4, 3, 2];
>> [value, location] = myMin(P)

value =
     2

location =
     1   5   8
```

3. Rainfall totals

   Design a function that computes the total rainfall at a location using daily rainfall measurements entered by a user. The list may contain the number -999 indicating the end of the data from a given period. There may be negative numbers other than -999 in the list which represent erroneous measurements which should be neglected. Produce the total and average of the non-negative values in the list up to the first -999 (if there is a -999 value). Your function header should be
   `function [rainTot, rainAvg] = myRainfall(rainData)`
   where `rainData` is a double array of daily rainfall data, `rainTot` is the total rainfall in the period represented by either the entire array `rainData` or the period from the beginning until the first occurrence of -999. `rainAvg` is the average daily rainfall during this period (not counting days with neglected measurements). Test Case:

```
>> rainData = [2, 4, -5, 1, 7, -2, 8, -999, 4, 6, 4]
>> [rainTot, rainAvg] = myRainfall( rainData )

rainTot =
      22
```

```
rainAvg =
      4.4000

>> rainData = [2, 4, −5, 1, 7, −2, 8, 0, 4, 6, 4]
>> [rainTot, rainAvg] = myRainfall( rainData )

rainTot =
      36

rainAvg =
      4

>> rainData = [2, 4, −999, 1, 7, −2, 8, −999, 4, 6, 4]
>> [rainTot, rainAvg] = myRainfall( rainData )

rainTot =
      6

rainAvg =
      3
```

4. Random Walk Vacuum Robot
In this problem, we are going to study the displacement of a random walk vacuum cleaner robot that randomly wanders around a closed room.

- The room is represented by a square grid $[0, L]$ ft $\times[0, L]$ ft, so there are $(L+1)^2$ "grid points," each of size 1 ft^2 in the room. Each grid point is represented by (x, y) coordinates, where x and y are both integers.
- The robot cleans the room by randomly moving from one grid point to another. Whenever the robot stops on a grid point, the area around that grid point is cleaned. The robot also cleans as it moves, meaning that if it moves from $(x, y) = (1, 1)$ to $(x, y) = (1, 3)$, then the points defined by $(x, y) = (1, 1)$, $(1, 2)$, and $(1, 3)$ have now all been cleaned. The robot starts in the bottom left corner of the room ($(x, y) = (0, 0)$).
- The direction and length of each robot movement is determined by two random numbers, which are provided in the input argument `rmat`. The first movement is determined by the first row of `rmat`, the second movement is determined by the second row, and so on. The first number in each row determines the direction (see table below), and the second number in each row determines the length of each movement. If all rows of `rmat` have been used and the robot still has not cleaned the entire room, then the values of `rmat` are recycled, starting with the first row.

  For example, if the robot's current position is $(x, y) = (2, 3)$ and the next row in `rmat` is `1, 3`, then the robot's next position is $(x, y) = (2, 6)$ (if $L \geq 6$).

- If a step would cause the robot to hit a wall, then the robot should not take that step but instead go to the next row of `rmat`.

Write a function with header
`function [N, path] = RandomWalk(L, rmat)`
where `L` determines the size of the room as described above and `rmat` is a $M \times 2$ matrix of random integers 1 through 4, which will tell the robot where to move.

Your function will output `N`, the number of steps it takes for the robot to cover the entire room, as well as `path`, a $(N+1) \times 2$ matrix where each row contains (x, y) coordinates of every step. The first row of `path` will be `0, 0`, and the second row will have the (x, y) coordinates of the robot after the first step.

Number	Direction
1	up (+y)
2	down (-y)
3	right (+x)
4	left (-x)

Test Case:

```
>> rmat = [4 1;1 2;3 2;1 1;4 1; 2 1; 2 2; 3 1; 1 1 ;2 2];
>> [N, path] = RandomWalk(2,rmat)

N =
     8

path =
     0     0
     0     2
     2     2
     1     2
     1     1
     2     1
     2     2
     2     0
     1     0
```

```
plot(path(:,1), path(:,2), '-o')
```

You can experiment with the command `randi` to see how to generate your own matrix of random values. For instance, `randi([1 4], 100, 2)` will give a 100×2 matrix filled with randomly selected integers between 1 and 4. When testing your code, make sure that `rmat` is big enough that the robot doesn't get "stuck" in a repeating path that doesn't cover the entire room. The larger `L` you use, the more rows `rmat` should have. Reasonable values for `L` should range from about 5 to 10, along with a few hundred rows in `rmat`.

You may also wish to plot the path of the robot to confirm that it has covered all the squares in the room. You do not need to submit this plot. You can use the command shown in the test case above to plot the path using the function output.

5. Fibonacci and Golden Ratio

The Fibonacci Sequence can be defined by the recursive relationship $F_N = F_{N-1} + F_{N-2}$, where the first two numbers of the sequence by convention are $F_1 = 0$ and $F_2 = 1$. The sequence is defined for any positive integer N. The Fibonacci sequence can also be written using an iterative relationship. See Chapter 6 in the book for more information.

(a) Using **recursion**, write a function with header
`function [F] = FibRec(N)`
that returns the N^{th} number in the Fibonacci sequence, F_N. If the input `N` is not a positive integer (positive means that $N > 0$, and integer means the number has no fractional component), the output `F` should be the `char` array:
`N must be a positive integer`
If the class of the input `N` is not `double`, the output `F` should be the `char` array:
`N must be of class double`

(b) Using a `for` loop (no recursion), write a function with header
`function [F] = FibIter(N)`
that returns the N^{th} number in the Fibonacci sequence, F_N. If the input `N` is not a positive integer (see above), the output `F` should be the `char` array:
`N must be a positive integer`
If the class of the input `N` is not `double`, the output `F` should be the `char` array:
`N must be of class double`

Test Case:

```
>> fr = FibRec(12)

fr =
    89

>> fr = FibRec(-3)

fr =
    N must be a positive integer

>> fi = FibIter(12)

fi =
    89
```

```
>> fi = FibIter(0)

fi =
     N must be a positive integer
```

(c) Write a function to compute the ratio $r = \frac{F_N}{F_{N-1}}$. Your function should have the header
`function [r] = FibRatio(N)`
where `r` is the ratio defined above for the values of the Fibonacci Sequence defined by `N`. If `N` is a not an integer greater than or equal to 2, the output `r` should be the `char` array: `N must be an integer greater than 1`. Your function `FibRatio` should call one of the functions from earlier in this problem. Which one you use is up to you.
Test case:

```
>> r10 = FibRatio(10)

r10 =
      1.6190
```

You do not have to submit any answer to the following questions to complete this assignment, but you should think about the answers to improve your understanding: How does the computational speed of the iterative and recursive functions compare? How does the speed of each one change for $N = 5, 10, 50$? Why do you think that is? Compute `FibRatio(N)` for `N = 5, 10, 20, 50`, and `100`. What do you notice?

(d) Actually, $r = \frac{F_N}{F_{N-1}}$ converges to a limit called the Golden Ratio whose exact value is $\phi = (1 + \sqrt{5})/2$. Write a function with header
`function [N] = FibApprox(e)`
where N is the smallest integer such that $|\phi - \frac{F_N}{F_{N-1}}| \leq e$.
Test Case:

```
>> N=FibApprox(1e-6)

N =
      18

>> N = FibApprox(1e-8)

N =
      22
```

6. Number guessing game

This game begins with player one saying something like, "I'm thinking of an integer between forty and sixty." Following a guess by player two, player one responds ‘higher’, ‘lower’, or ‘correct!’ as might be the case. Supposing that N is the number of possible values (here, twenty-one, since we will consider inclusive intervals), then at most $\lfloor \log_2 N \rfloor + 1$ questions should be necessary to determine the number, since each question can halve the search space.

Suppose player one selects a number between 1 and 30 to be the “secret number”. Since player one tells player two whether a guess is too low, too high, or correct, player two can start off by guessing in the middle, 15. If the secret number is less than 15, then because player two knows that 15 is too high, player two can eliminate all the numbers from 15 to 30 from further consideration. If the secret number is greater than 15, then player two can eliminate 1 through 15. Either way, player two can eliminate about half the numbers. On the next guess, player two again eliminates half of the remaining numbers, then keeps going, always eliminating half of the remaining numbers until the number is guessed. We call this halving approach a binary search algorithm, and no matter which number from 1 to 30 is the secret number, player two should be able to find the number in at most 5 guesses with this technique (for the range 1 to 30).

The function `player1` simulates the actions of player one and has been made available to you to help you familiarize yourself with the game. The function has two input arguments, `nmin` and `nmax` which define the range of possible values. To play the game with the range described above, call the function with the command `player1(40, 60)` and enter your guesses when prompted. The only output argument of `player1` is `N_guess`, which is the number of guesses it took to guess the secret number.

Your job is to write a **recursive** function that simulates the actions of player two in this game. Since the function `player1` requires user interaction in order to run, two different functions, `player1initialize` and `player1response` have also been provided. Note that these functions have been encrypted so they cannot be read or edited, but they can be run just like a regular function.

The function `player1initialize` has header
`function [secret] = player1initialize(nmin, nmax)` where `nmin` and `nmax` define the range for the game (just like in `player1`). The output `secret` is a randomly selected secret number from within the specified range, stored in an encrypted form which can be read by `player1response`.

The function `player1response` has header
`function [msg] = player1response(guess, secret)` where `guess` is a `double` representing the guess. The input argument `secret` is an encrypted secret number created by `player1initialize`. The output `msg` is a `char` array with three potential values: `’higher’`, `’lower’`, `’correct’`, which indicate the relationship between the secret number and the guessed number. Note that when `player1response` is called it will also print a message to the screen telling you if your guess is too high, low or correct, to help you keep track of what is going on; `msg` will only have the three possible values listed above.

The function you write will have the header
`function [N_guess] = player2(nmin, nmax, N_guess, secret)`
The inputs `nmin` and `nmax` are integers of class `double` which define the range of the

game just like before. The input argument `N_guess` represents the number of guesses which have taken place. Hint: when you call the function from the command window, the input `N_guess` will always be `0` since no guesses have been used yet. Only when you make recursive calls will it be different. The input argument `secret` should be taken directly from the output of `player1initialize`. The output argument `N_guess` is the number of guesses which was required to guess the secret number.

Test Cases:

```
>> N_guess = player2(10, 20, 0, player1initialize(10, 20) )
Guess a number between 10 and 20
the secret number is higher than your guess of 15
the secret number is higher than your guess of 18
you guessed the secret number, which is 19

N_guess =
     3


>> N_guess = player2(0, 30, 0, player1initialize(0, 30) )
Guess a number between 0 and 30
the secret number is lower than your guess of 15
the secret number is lower than your guess of 8
the secret number is higher than your guess of 4
the secret number is lower than your guess of 6
you guessed the secret number, which is 5

N_guess =
     5
```

Note that the statements printed to the screen are a result of calls to `player1response` and do not need to be created by you in your `player2` function. Also note that because the secret number is selected randomly in `player1initialize`, your results won't exactly match the test cases.

Submission Instructions

Your submission should include the following function files in a single zip file named FIRSTNAME_LASTNAME.zip

- mySavingsPlan.m
- myMin.m
- myRainfall.m
- RandomWalk.m

- FibRec.m
- FibIter.m
- FibRatio.m
- FibApprox.m
- player2.m

Don't forget that the function headers you use must appear *exactly* as they do in this problem set. So, make sure the variables have the right capitalization, the functions have the correct name, and the inputs and outputs are in the correct order.