Final Programming Assignments

Introduction to Programming with MATLAB

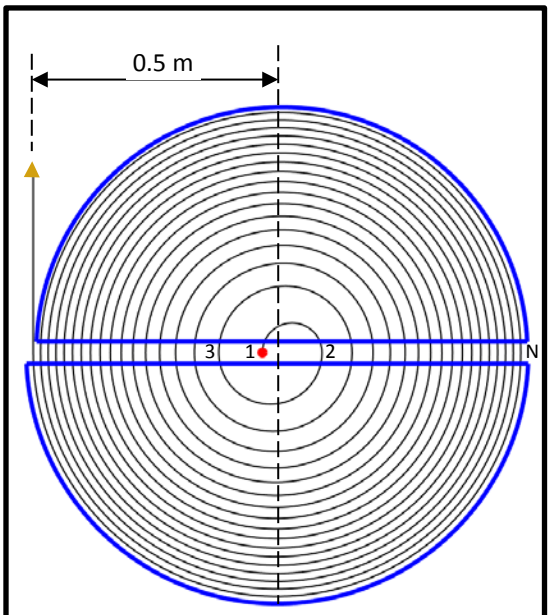
- Unless otherwise indicated, you may assume that each function will be given the correct number of inputs and that those inputs have the correct dimensions. For example, if the input is stated to be three row vectors of four elements each, your function is not required to determine whether the input consists of three two-dimensional arrays, each with one row and four columns.
- Unless otherwise indicated, your function should not print anything to the Command Window, but your function will not be counted incorrect if it does.
- Note that you are not required to use the suggested names of input variables and output variables, but you must use the specified function names.
- Also, read the instructions on the web page on how to test your functions with the auto-grader program provided, and what to submit to Coursera to get credit.
- Note that starred problems, marked by ***, are harder than usual, so do not get discouraged if you have difficulty solving them.

1. Write a function called **`digit_counter`** that takes the name of a text file as input and returns the number of digits (i.e., any of the characters, 0-to-9) that the file contains. If there is a problem opening the file, the function returns -1. **WARNING**: if you use the 'w' flag with **`fopen`**, as opposed to 'r', you will overwrite the file. The grader uses your own m files to check your function, so be careful!
2. Write a function called **`day_counter`** that returns the number of Mondays that fell on the first day of the month in a given year between 1776 and 2016 inclusive where the requested year is the only input to your function and it is a positive integer scalar. Note that a leap year occurs on any year evenly divisible by 4, but not on a century unless it is divisible by 400. In a leap year, February has 29 days. You are not allowed to use the **`datenum`** built-in function. (Inspired by Project Euler.)
3. Write function called **`huge_add`** that adds together two positive integers of any length specified as strings using decimal notation. The single output argument is the result and it is a string as well. The inputs and output must contain digits only; no commas, spaces or any other characters are allowed. If any of these assumptions are violated by the input, the function returns the number -1.
4. 2520 is the smallest number that can be divided by each of the numbers from 1 to 10 without any remainder. Write a function called **`smallest_multiple`** that returns a **`uint64`**, the smallest positive number that is evenly divisible by all of the numbers from 1 to n where n is a positive integer scalar and is the only input argument of the function. If the result would be greater than what can be represented as a **`uint64`**, the function returns 0. (Inspired by Project Euler.)
5. *** Write a function called **`maxproduct`** that takes a matrix **`A`** and a positive integer scalar **`n`** as inputs and computes the largest product of **`n`** adjacent elements in the same direction in **`A`**. That is, we are looking for products of consecutive elements in the same row, column, diagonal or reverse diagonal. The function must return an **`n`**-by-**2** matrix containing the row and column indexes ordered first by row and then by column. If no such product exists, the function returns the empty array. For example, valid outputs for a max product of 3 neighbors in four different matrices might be [2 2; 2 3; 2 4] or [1 1; 2 1; 3 1] or [3 5; 4 4; 5 3] or [4 2; 5 3; 6 4]. If there are multiple products with the same maximum value, return the first one you find. (Inspired by Project Euler.)

6. *** If the numbers 1 to 5 are written out in words: one, two, three, four, five, then there are 3 + 3 + 5 + 4 + 4 = 19 letters used in total. Write a function called **`number2letters`** that returns the number of letters needed to write down the number n in words. For example, 342 (three hundred forty two) contains 20 letters. Notice that we do not count spaces, nor do we use hyphens. The only input to the function is n, a positive integer smaller than 1000, but you do not need to check this. (Inspired by Project Euler.)

7. *** Write a function called **`circular_primes`** that finds the number of circular prime numbers smaller than **`n`**, where **`n`** is a positive integer scalar input argument. For example, the number, 197, is a circular prime because all rotations of its digits: 197, 971, and 719, are themselves prime. For instance, there are thirteen such primes below 100: 2, 3, 5, 7, 11, 13, 17, 31, 37, 71, 73, 79, and 97. It is important to emphasize that rotation means circular permutation not all possible permutations.

8. *** Write a function that is called like this **`[E,N] = cyclotron(V)`**. All arguments are scalars. The input argument is the voltage applied to a cyclotron (figure), which is a device that accelerates subatomic particles—in this case, positively charged isotopes of hydrogen, called "deuterons"—which spiral outward in a clockwise direction. The cyclotron rapidly alternates the sign of the voltage difference **`V`** in units of volts between two "D"-shaped vacuum chambers (blue outlines), which are placed within a strong uniform magnetic field (not shown but perpendicular to the page). The deuteron is accelerated only as it is leaving one "D" and entering the other. While the deuteron is inside a given "D", it moves at a constant speed, and the magnetic field causes it to move on a semicircle. Each deuteron moves as follows (check the numbers in the figure): (1) It originates from a source (red dot) located at a distance s_0 to the left of the center of the cyclotron, is accelerated vertically into the upper "D" and then moves on a semi-circle of radius r_1 . (2) It leaves the upper "D" and is accelerated vertically downward into the lower "D" where moves with a larger radius r_2. (3) It leaves the lower "D" and is accelerated vertically into the upper "D", etc, moving with ever increasing radii r_n until (N) it is accelerated for the final time as it leaves the upper "D" and enters the lower "D", follows a semicircle of radius r_N , and emerges from the cyclotron at the left. The formulas for the radii are as follows: $r_1 = \sqrt{mV/(qB^2)}$, m = deuteron mass $= 3.344 \times 10^{-27}$ kg, q = deuteron charge $= 1.603 \times 10^{-19}$ coulomb, and B = magnetic field strength $= 1.600$ tesla. For $n \geq 2$, $r_n = \sqrt{r_{n-1}^2 + 2mV/(qB^2)}$. These expressions give the radii in units of meters, and $s_0 = r_1/2$. The deuteron escapes through a window at the left that is placed so the particle cannot leave until it is more than 0.500 m to the left of the center of the cyclotron. The gap between the "D"s is exaggerated in the figure, has no effect, and can be assumed to be of zero width. The function returns energy **`E`** of the deuteron when it escapes in units of million electron volts (MeV), which equals $V\mathbf{N} \times 10^{-6}$, and the number **`N`** of times the deuteron enters the "D"s. HINT: Notice that the centers of the semicircles ≠ the center of the cyclotron.



Cyclotron. A charged particle emerges from a source near the center (red), spirals out while accelerating to high speed, and emerges at the left.

Here is a sample run:

```
>> [E, N] = cyclotron(4.8e5)  % depicted in figure
E =
   16.32
N =
   34
```