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.
- Finally, 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 **`sparse_array_out`** that takes two input arguments and returns one output argument. Its first argument is a two-dimensional array of doubles, which it writes into a binary file. The name of that file is the second argument to the function. The file must have the following format. It starts with three **`uint32`** scalars specifying the number of rows of the array followed by the number of columns followed by the number of non-zero elements in the array. Then each non-zero element of the array is represented by two **`uint32`** scalars and a **`double`** scalar in the file in this order: its row index (**`uint32`**), its column index (**`uint32`**), and its value (**`double`**). Note that this file format is an efficient way to store a so-called "sparse array", which is by definition an array for which the overwhelming majority of its elements equal 0. The function's output argument is of type logical and equals false if there was a problem opening the file and true otherwise.
2. Write a function called **`sparse_array_in`** that reads a two-dimensional array of doubles from a binary file whose name is provided by the single input argument of the function. The format of the file is specified in the previous problem. The function returns the two-dimensional array that it reads from the file as an output argument. Note that if you call **`sparse_array_out`** with an array called **`A`** and then call **`sparse_array_in`** using the same filename and save the output of the function in variable **`B`**, then **`A`** and **`B`** must be identical. If there is a problem opening the file, the function returns an empty array.
3. Write a function called **`letter_counter`** that takes the name of a text file as input and returns the number of letters (i.e., any of the characters, a-to-z and A-to-Z) that the file contains. HINT: You can use the built-in function **`isletter`**. If there is a problem opening the file, the function returns -1.
4. Write a function called **`saddle`** that finds saddle points in the input matrix **`M`**. For the purposes of this problem, a saddle point is defined as an element whose value is greater than or equal to every element in its row, and less than or equal to every element in its column. Note that there may be more than one saddle point in **`M`**. Return a matrix **`indices`** that has exactly two columns. Each row of **`indices`** corresponds to one saddle point with the first element of the row containing the row index of the saddle point and the second column containing the column index. The saddle points are provided in **`indices`** in the same order they are located in **`M`** according to column-major ordering. If there is no saddle point in **`M`**, then **`indices`** is the empty array.

5. Write a function called **`prime_pairs`** that returns the smallest prime number **`p`** smaller than 100,000 such that **`(p + n)`** is also prime, where **`n`** is a scalar integer and is the sole input argument to the function. If no such number exists, then the function returns -1. You may use the built-in functions, **`primes`** and **`isprime`**. Note that an efficient solution to this problem, such as the one the grader uses, takes a fraction of a second, but depending on how you do it, yours may be significantly slower.

6. *** We start this problem with a tutorial on North America's most popular bowling game. In it contestants attempt to knock down as many as possible of ten pins that are standing at one end of an alley by rolling balls at them from the other end. A bowling game consists of ten "frames" for each player, and each player's frame except for the tenth consists of either one or two attempts to knock down the pins. After each player's frame is complete, all ten pins are set up again. The pins knocked down by one player have no effect on the other player. They could in fact compete on separate alleys.

   Frames are of different types. A frame in which all ten pins are knocked down by the first ball is a "strike", and after a strike the second ball of the frame is omitted. A frame in which all ten pins are knocked down with two balls (the first ball having left some standing) is a "spare". A frame with fewer than ten pins down is an "open". A spare in the tenth frame earns the bowler one "bonus" ball for that frame, which is simply an extra ball; a strike in the tenth earns two bonus balls, and if the first of these two extra balls knocks everything down, all ten pins are set up again for the second ball.

   A player's score is computed as follows. A strike counts 10 points plus the sum of the pins knocked down by the next two balls (which may include balls from the next two frames). A spare counts 10 points plus the pins knocked down by the next ball. An open counts the number of pins knocked down in that frame. A bonus ball counts the number of pins it knocked down. Consider this example game:

Frame	1		2		3		4		5		6		7		8		9		10		
Pins	9	1	0	10	10		10		6	2	7	3	8	2	10		9	0	9	1	10
Type	spare		spare		strike		strike		open		spare		spare		strike		open		spare + bonus		
Score	10		30		56		74		82		100		120		139		148		168		

   As another example, a "perfect" game, which is 12 strikes in a row, has the maximum possible score—300.

   Write a function called **`bowl`** that takes a vector of integers specifying the sequence of pins down after each ball and returns the final score. For example, for the frames in our example above, the input would be

   ```
   [9 1 0 10 10 10 6 2 7 3 8 2 10 9 0 9 1 10]
   ```

   and the output would be 168. If the input is not a valid sequence for a full game, the function returns -1.

7. *** Write a function called **`maxsubsum`** that takes a matrix **`A`** as an input, computes the sum of elements in each of its submatrices, and finds the submatrix that has the maximum sum. (For the purposes of this exercise, we define a submatrix as a matrix formed by a contiguous set of elements of the original matrix.) If there are more than one with the same maximum sum, the function can pick any one of them. Note that the entire matrix is considered to be a submatrix of itself, but because some elements of the matrix may be negative, it may not have the maximum sum. The function is defined like this:

   ```
   function [row,col,numrows,numcols,summa] = maxsubsum(A)
   ```

   where **`row`** and **`col`** specify the indexes of the top left corner of the submatrix with the maximum sum, **`numrows`** and **`numcols`** are its dimensions, and **`summa`** is the sum of its elements.

8. *** Write a function called **queen_check** that takes an 8-by-8 logical array called **board** as input. The variable **board** represents a chessboard where a **true** element means that there is a queen at the given position (unlike in chess, there can be up to 64 queens!), and **false** means that there is no piece at the given position. The function returns **true** if no two queens threaten each other and **false** otherwise. One queen threatens the other if they are on the same row, on the same column, or on the same diagonal.

9. *** Write a function called **roman2** that takes a string input representing an integer between 1 and 399 inclusive using Roman numerals and returns the Arabic equivalent as a **uint16**. If the input is illegal, or its value is larger than 399, **roman2** returns 0 instead. The rules for Roman numerals can be found here: http://en.wikipedia.org/wiki/Roman_numerals. Use the definition at the beginning of the page under the "Reading Roman Numerals" heading, but the function must use only the shortest possible Roman representation in each case. This rule eliminates four or more consecutive identical symbols, such as **IIII**, which equals **IV**, and **XXXX**, which equals **XL**, and it eliminates the use of subtractive notation followed by additive notation with the same symbol, such as **IXI**, which equals **X**.

10. *** Write a function called **bell** that returns the first **n** rows of the Bell triangle, where **n** is an input argument. For a precise definition, see http://en.wikipedia.org/wiki/Bell_triangle. The function must return an **n**-by-**n** array where the top left triangle contains the Bell triangle with each row of the Bell triangle positioned diagonally—bottom-left-to-upper-right—and the bottom right triangle contains only zeros. If **n** is not a positive integer, the function returns an empty array. As an illustration, the call **bell(7)** should return this array:

```
     1     2     5    15    52   203   877
     1     3    10    37   151   674     0
     2     7    27   114   523     0     0
     5    20    87   409     0     0     0
    15    67   322     0     0     0     0
    52   255     0     0     0     0     0
   203     0     0     0     0     0     0
```

Note that some of these problems were adapted from Dr. John Dalbey's computer science class at Cal Poly.