Programming Assignments

Introduction to Programming with MATLAB

Lesson 6

- 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 **`neighbor`** that takes as input a row vector called **`v`** and creates another row vector as output that contains the absolute values of the differences between neighboring elements of **`v`**. For example, if **`v == [1 2 4 7]`**, then the output of the function would be **`[1 2 3]`**. Notice that the length of the output vector is one less than that of the input. Check that the input **`v`** is indeed a vector and has at least two elements and return an empty array otherwise. You are not allowed to use the **`diff`** built-in function.

2. The function **`replace_me`** is defined like this: **`function w = replace_me(v,a,b,c)`**. The first input argument **`v`** is a vector, while **`a`**, **`b`**, and **`c`** are all scalars. The function replaces every element of **`v`** that is equal to **`a`** with **`b`** and **`c`**. For example, the command

   ```
   >> x = replace_me([1 2 3],2,4,5);
   ```

   makes **`x`** equal to **`[1 4 5 3]`**. If **`c`** is omitted, it replaces occurrences of **`a`** with two copies of **`b`**. If **`b`** is also omitted, it replaces each **`a`** with two zeros.

3. Write a function called **`halfsum`** that takes as input an at most two-dimensional matrix **`A`** and computes the sum of the elements of **`A`** that are in the diagonal or are to the right of it. The diagonal is defined as the set of those elements whose column and row indexes are the same. For example, if the input is **`[1 2 3; 4 5 6; 7 8 9]`**, then the function would return 26.

4. Write a function called **`large_elements`** that takes as input an array named **`X`** that is a matrix or a vector. The function identifies those elements of **`X`** that are greater than the sum of their two indexes. For example, if the element **`X(2,3)`** is 6, then that element would be identified because 6 is greater than 2 + 3. The output of the function gives the indexes of such elements found in row-major order. It is a matrix with exactly two columns. The first column contains the row indexes, while the second column contains the corresponding column indexes. For example, the statement **`indexes = large_elements([1 4; 5 2; 6 0]`**, will make **`indexes`** equal to **`[1 2; 2 1; 3 1]`**. If no such element exists, the function returns an empty array.

5. Write a function called **one_per_n** that returns the smallest positive integer n for which the sum 1 + 1/2 + 1/3 + … + 1/n, is greater than or equal to **x** where the scalar **x** is the input argument. Limit the maximum number n of terms in the sum to 10,000 and return **-1** if it is exceeded. (Note: if your program or the grader takes a long time, you may have created an infinite loop and need to hit Ctrl-C on your keyboard.)

6. Write a function called **approximate_pi** that uses the following approximation of π:

$$\pi = \sqrt{12}\sum_{k=0}^{\infty}\frac{(-3)^{-k}}{2k+1}$$

   Instead of going to infinity, the function stops at the smallest **k** for which the approximation differs from **pi** (i.e., the value returned MATLAB's built-in function) by no more than the positive scalar, **delta**, which is the only input argument. The first output of the function is the approximate value of π, while the second is **k**. (Note: if your program or the grader takes a long time, you may have created an infinite loop and need to hit Ctrl-C on your keyboard.)

7. Write a function called **separate_by_two** that takes a matrix **A** of positive integers as an input and returns two row vectors. The first one contains all the even elements of **A** and nothing else, while the second contains all the odd elements of **A** and nothing else, both arranged according to column-major order of **A**. You are not allowed to use for-loops or while-loops.

8. Write a function called **divvy** that takes a matrix **A** of integers greater than or equal to zero and a single positive integer **k** as its two inputs and returns a matrix **B** that has the same size as **A**. The elements of **B** are all divisible by **k**. If an element of **A** is divisible by **k**, then the corresponding element in **B** must have the same value. If an element of **A** is not divisible by **k**, then the corresponding element of **B** must be the product of the given element of **A** and **k**. You are not allowed to use any for-loops or while-loops. For example, the call **X = divvy([1 2 ; 3 4], 2)** would make **X** equal to **[2 2 ; 6 4]**.

9. *** Write a function called **square_wave** that computes the sum

$$\sum_{k=1}^{n}\frac{\sin\big((2k-1)t\big)}{(2k-1)}$$

   for each of 1001 values of t uniformly spaced from 0 to 4π inclusive. The input argument is a positive scalar integer **n**, and the output argument is a row vector of 1001 such sums—one sum for each value of t. You can test your function by calling it with **n** == 200 or greater and plotting the result, and you will see why the function is called "square_wave".

10. *** Write a function **myprime** that takes **n**, a positive integer, as an input and returns **true** if **n** is prime or returns **false** otherwise. Do not use the **isprime** or **primes** or **factor** built-in functions. Hint: you can use the **rem** or **fix** functions.