CS1132 Fall 2012 Assignment 2

> Adhere to the Code of Academic Integrity. You may discuss background issues and general solution strategies with others and seek help from course staff, but the homework you submit must be the work of just you. When submitting your assignment, be careful to follow the instructions summarized in Section 4 of this document.

In this last assignment in the course, you will *design* the solution to the final problem posed. In problem 2, you need to think carefully about how to *decompose* the problem into subproblems—use subfunctions. The focus is for you to turn a problem statement written in English into a solution written as a MATLAB program. ***Read carefully*** and take some time to think about the *design*—don't just jump into coding immediately.

1 Bioinformatics Tools

This summer a paper was published in *Nature* as a summary of a six-year-long NIH project that was dedicated to the study of the human microbiome system. Both *New York Times* and *Wall Street Journal* have covered the exciting result recently in the bioinformatics field. The human microbiome project was considered to be a milestone in genome research and is showing great potential in assisting genome ID building, gene therapy, and medical research.

Basically, the human microbiome system as a microbial community includes bacteria, eukaryotes, and viruses. The way to identify those groups is to study the RNA/DNA sequence of nucleotides. The nucleotides we deal with in this assignment are adenosine (A), cytidine (C), guanosine (G), thymidine (T).

1.1 Palindrome-like sequence

We find a palindrome-type virus to be of great research interest. A palindrome is a sequence that reads the same backward as forward. This self-symmetric feature makes the virus sequence easy to target in the human microbiome system. Here is an example of the nucleotide sequence of a palindrome-type virus:

ATTTGCCGAGAGCCGTTTA

In practice, a palindrome-type virus may not be a "perfect" palindrome due to some defects in itself or system errors from the sequencing process. Therefore a palindrome-like sequence (not necessarily a perfect palindrome) likely encodes a palindrome-type virus. We define the number of different nucleotides between a palindrome-like sequence and a palindrome-type virus as the "mismatch." The following is an example of palindrome-like sequence where the mismatch is 1.

ATTTG<u>C</u>CGAGAGC<u>G</u>GTTTA

The mismatch tolerance q is the maximum mismatch allowed when classifying a palindrome-like sequence as a palindrome-type virus. Consider the following three palindrome-like sequences.

	Sequence	Mismatch
i	ATTTG<u>C</u>CGAGAGC<u>G</u>GTTTA	1
ii	ATTTG<u>GG</u>GAGAG<u>CC</u>GTTTA	2
iii	ATTTGCCGAGAGCCGTTTA	0

If the tolerance q is 2, all three sequences above will be classified as palindrome-type viruses. If $q = 1$, then only sequences i and iii will be classified as palindrome-type viruses. Implement the following function:

```
function misma = palindrome(str, q)
% misma: integer mismatch value indicating the difference between str and
%   a perfect palindrome.
% str: a dna sequence (string) containing UPPER CASE letters A, T, C, G
% q: mismatch tolerance
```

1.2 Virus positioning

A positioning tool is crucial in genome research for locating specific sequences, functioning much like a GPS on DNA a sequence map. Due to uncertainties in bioprocesses, a tolerance, q, is again defined to allow for a certain number of mismatches in this locator. Implement the following function:

```
function position = DNAGPS(vstr, hstr, q)
% position: vector of all the starting positions of vstr in hstr with a
%   tolerance of q
% vstr: a dna sequence (string) containing UPPER CASE letters A, T, C, G
% hstr: a dna sequence (string) containing UPPER CASE letters A, T, C, G.
%   hstr has a length greater than vstr.
% q: mismatch tolerance
```

Note that both `vstr` and `hstr` are strings, i.e., 1-d array of characters. For this function you can, but you don't have to, make use of function `palindrome` from §1.1.

1.3 Human microbiome mark

Now you will work with real human microbiome sequencing data and collect information on some viruses. Implement this function:

```
function microb = HMM(sfile, L, q)
% Read sfile and store length L palindrome-type virus data in cell array microb.
% sfile names the genome data file.  The file contains multiple lines of
%   data in plain text.
% L: the length of the palindrome-type virus of interest
% q: mismatch tolerance
% microb: n-by-2 cell array where n is the number of different length L
%   palindrome-type viruses given the mismatch tolerance q.  If no length L
%   palindrome-type virus is found then microb is empty cell array.
% Make effective use of functions palindrome and DNAGPS.
```

For example, if we have the following sequence in a data file

ATTTGATAGATTAGGACCCGCGCGTACGTAATGATTAG

and we are looking for perfect ($q = 0$) palindrome viruses of length 6, then function `HMM` should return the following cell array:

```
{'GATTAG', [9 33];  'GTAATG', [28]}
```

Note that the same virus, allowing for the tolerance q, should not appear in multiple rows of the returned cell array. Here is a more extreme example: suppose the genome data is 'GAGAGAGAT' and the input arguments are `L`=3 and `q`=1. (`q` is the mismatch tolerance for both function `palindrome` and function `DNAGPS`.) The returned cell array should be

```
{'GAG', [1 3 5 7];  'AGA', [2 4 6]}
```

and not

```
{'GAG', [1 3 5 7];  'AGA', [2 4 6];  'GAG', [1 3 5 7];  'AGA', [2 4 6]; ...
 'GAG', [1 3 5 7];  'AGA', [2 4 6];  'GAT', [1 3 5 7] }
```

nor

```
{'GAG', [1];  'AGA', [2];  'GAG', [3];  'AGA', [4]; ...
 'GAG', [5];  'AGA', [6];  'GAT', [7] }
```

Hint: It may be helpful to implement a subfunction that determines whether an integer k is in a numeric vector v.

Two data files are provided: `DNAsample1.txt` is a short data file suitable for program development and `DNAsample2.txt` is a bigger file. These files are parts of the original 77MB file from the government open source database `http://www.ncbi.nlm.nih.gov/projects/genome/guide/human/`. Your code must work with the file format specified here—do not alter the data files in any way. The data files for this function are plain text files containing both header (comment) lines and data lines. Each header line begins with the character '`>`'; data lines are made up of the upper case letters '`A`', '`C`', '`G`', and '`T`' only. *There may be any number of header lines above the data lines.* The first few lines in `DNAsample1.txt` are shown below.

```
>gi|157713538|ref|AC_000149.1| Homo sapiens chromosome 17,
>alternate assembly HuRef, whole genome shotgun sequence
>Note: lines 89-105 from original file
CTGCTCATTTAGGAGTCGTTCAATGTAACAAGGATCTTGGTCTTCACAGAGACCCTAAACGCGTGTGGCC
CAGGGCCATGTGGATGCCATCAGCCACTAGGTGGCAGCAGGGGCCCTTTGGATTTGAACAATGCAGCTCA
```

1.4 Palindrome-type virus database

Implement the following function:

```
function vgen = writeVdata(sfile, vfile, L, q)
% Write length L palindrome-type virus data to a plain text file.
% sfile names the genome data file.  The file contains multiple lines of
%   data in plain text.
% vfile names the virus data file to be written.  First line of the file is
%     Length <L>
%   where <L> is the length of the virus of interest. Each subsequent line is
%     <sequence><tab><pos1><tab><pos2><tab> ... <posk> ... <tab><posG>
%   where <sequence> is the palindrome-type virus sequence, <posk> is the
%   position of the kth appearance of the sequence in the genome data, and
%   G is the number of times (generations) that the virus appears in the
%   genome data.
% L: the length of the palindrome-type virus of interest
% q: mismatch tolerance
% vgen: the maximum number of appearances (generations) of Length L
%   palindrome-type viruses
% Make effective use of function HMM.
```

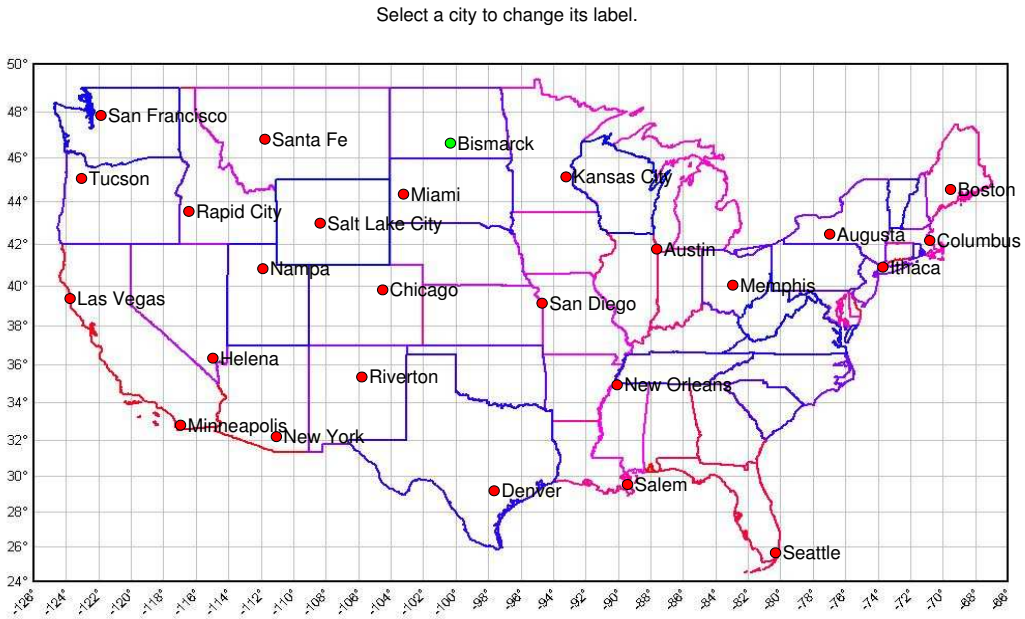
Again using the genome data 'GAGAGAGAT' as an example with input arguments `L`=3 and `q`=1, the output data file should contain the following lines of plain text

```
Length 3
GAG  1    3    5    7
AGA  2    4    6
```

and the return variable `vgen` should have the value 4. Note the correspondence between the output data file and the cell array returned by function `HMM` in §1.3!

2 Geography Game

Create an interactive "geography" game that allows users to learn the locations of major US cities. At the start of the game a USA map is displayed with small circles marking the locations of a number of cities. These cities are labeled but the labels may be wrong! The city marker is red if its label is wrong and green if the label is correct. The game starts up with a figure that looks something like the diagram below.



The goal of the game is to assign a correct label to every city marker. This will be achieved by repeatedly selecting a pair of cities with labels that should be swapped. A city is selected by a mouse click. When a first city is selected the marker turns yellow. After the player selects the second city, the labels are swapped. If a city's newly swapped label is still incorrect, its marker grows bigger (and is red)! If a city's newly swapped label is correct, then the marker turns green and goes back to its original size. Write a function `learnCityNames` that implements this educational game!

```
function learnCityNames(positions, trueNames)
% An interactive game to learn the names of major cities in the USA
% positions: x and y coordinates of n cities in an n-by-2 matrix
% trueNames: cell array of length n containing the names of n cities
% So positions(i,1) and positions(i,2) are the x and y coordinates of
%   the city whose name is trueNames{i}, where i<=n and n=length(trueNames).
%   These coordinates work with the given map USA.JPG.
```

At the start of the game, markers are put on the map based on the coordinates in `positions` to indicate the location of each city. Each marker is labeled with a *randomly* chosen name from `trueNames`. Then the game allows the user to swap the labels of the different cities by clicking on the figure. A click outside the white map box indicates that a user wants to leave the game. Throughout the game, messages appear at the top of the figure (title area) to give instructions or indicate the progress to the user. If all cities are correctly labeled then the game ends and a congratulatory message is displayed as the title.

2.1 Detailed specifications & instructions

- The given functions `usaData1` and `usaData2` return the data that you will need to run `learnCityNames`. `usaData1` contains the data of only five cities, so this is suitable for initial development of your code. `usaData2` gives you more cities to play with. Here's how you run the game:

  ```
  [positions, trueNames] = usaData1();  % Get the game data
  learnCityNames(positions, trueNames)  % Start the game
  ```

- Use the given map `USA.JPG`. The following code fragment will display it in a figure window:

```
clf      % Clear figure
hold on  % Hold subsequent plot commands to current axes
USA=imread('USA.JPG');  % USA is an array of numbers
USA = flipdim(USA, 1);  % Reverse the vertical coordinates: Usually in images the
                        %   origin (0,0) is the top left corner, therefore we need
                        %   to flip the data to use our Cartesian coordinates.
[n,m]=size(USA);  % Map dimensions:  x-coordinates in [1,m] and y-coordinates in [1,n]
image(USA);       % Display the map in a figure window
axis off          % Hide default axes

%%%% Add your code here to draw the cities and labels %%%%

hold off % Subsequent plot commands appear on new axes
```

- At the start of the game, display the map of USA, draw the city markers according to the data but label them *randomly* using the names in the cell array of city names. Each city name should appear only once. The markers should be colored as described earlier.
- Ask the user (using the plot title) to select a city whose label should be swapped. Mark the selected node as described earlier. The statement `[a,b]= ginput(1)` stores the location of a user mouse click in variables `a` (x-coordinate) and `b` (y-coordinate).
- Ask the user to select another city to be swapped and redraw the map with the swapped labels and markers according to the rules described earlier. If the user does not (successfully) select the second city to be swapped but makes a mouse click on the map, deselect the first city.

  Throughout the game, refresh the plot title with appropriate instructions or messages. E.g., "New York was selected, select another city.", "New York and San Diego have just been swapped. Select a city to be swapped.", ..., etc.
- Continue the game (go back to asking the user to select a city for swapping) until one of two situations occur:
  - All cities are correctly labeled. Show as the plot title "Congratulations you labeled all cities correctly!"
  - The user clicks outside of the white box on the figure. Show as the plot title "You chose to quit the game!"

2.2 Hints

- *Encapsulate pieces of your code in subfunctions!* This will make things much easier. Really. (Subfunctions are simply functions written in the same file as the main function, `learnCityNames` in this case.) Consider writing subfunctions for each of the following tasks:
  - "Randomize" the order of the city names in a cell array. Built-in function `randperm` may be useful.
  - Draw (refresh) the figure with the given positions for the cities and the current labels. In fact, think about putting (most of) the above code we showed for graphics in a subfunction.
  - Check if the current labels are correct
  - Update marker radii. To make things easier, we provide function `DrawDisk` for drawing colored disks. In the example figure shown, the marker disk radius is 5.
  - Determine whether a position clicked is inside the map box
  - Determine whether a position clicked is close enough to any of the cities to consider that mouse click to have selected a city. 5 to 10 unit distance from the center of the marker may be a reasonable tolerance.

- Take a close look at (and experiment with) the example in Module 2 Part 3 to get a better feel for how to use graphic commands and interactivity.
- The "Graphics example files" section in Module 2 Part 3 shows commonly used controls. "Text Alignment" is especially relevant.
- Remember that you can read the documentation on MATLAB commands by typing `doc` <*command_name*> in the Command Window.
- To refresh a figure, use the command `clf` followed by `hold on` to make sure that subsequent plot commands add to the current set of axes. (As shown in the given fragment above.)

3 Self-Check List

The following is a list of the minimum *necessary* criteria that your assignment must meet in order to be considered *satisfactory*. Failure to satisfy any of these conditions will result in an immediate request to resubmit your assignment. Save yourself and the graders time and effort by going over it before submitting your assignment for the first time. Although all these criteria are necessary, meeting all of them might still not be *sufficient* to consider your submission satisfactory. We cannot list everything that could be possibly wrong with any particular assignment!

- Δ Comment your code! Make sure your functions are properly commented, regarding function purpose and input/output parameters.
- Δ Suppress all unnecessary output by placing semicolons (;) appropriately. At the same time, make sure that all output that your program intentionally produces is formatted in a user-friendly way.
- Δ Make sure your functions names are *exactly* the ones we have specified, *including* case.
- Δ Check that the number and order of input and output parameters for each of the functions match exactly the specifications we have given.
- Δ Test each one of your functions independently, whenever possible, or write short scripts to test them.
- Δ Check that your scripts do not crash (i.e., end unexpectedly with an error message) or run into infinite loops. Check your script several times in a row. Before each test run, type the commands `clear all; close all;` to delete all variables in the workspace and close all figure windows.

4 Submission instructions

1. Upload files `palindrome.m`, `DNAGPS.m`, `HMM.m`, `writeVdata.m`, and `learnCityNames.m` to CMS in the submission area corresponding to Assignment 2 in CMS.
2. Please do not make another submission until you have received and read the grader's comments.
3. Wait for the grader's comments and be patient.
4. Read the grader's comments carefully and think for a while.
5. If you need to resubmit, fix all the problems and go back to Step 1! *Be sure to select the* ***Regrade Request*** *option.* Otherwise you are done with this assignment. Well done!