CS1132 Fall 2011 Assignment 2

Adhere to the Code of Academic Integrity. You may discuss background issues and general strategies with others and seek help from course staff, but the implementations that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is never OK for you to see or hear another student's code and it is never OK to copy code from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

Note: In this last homework in the course, you will *design* the solutions to the problems posed. Minimal specifications are given so that you will design the appropriate program structure, including any subfunctions, yourself. 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 Finding Genes

A four-letter alphabet, A, C, T, and G, is used to represent the four nucleotides, Adenine, Cytosine, Thymine, and Guanine, in the DNA of living organisms. There is one (long) sequence of these nucleotides, or bases, for each chromosome, and the set of sequences for all the chromosomes in an organism is called the genome. The genomic sequences for many organisms are now known, including a human genome, which is a sequence of about three *billion* characters!

A gene is a substring of a genome that codes for a specific function in an organism. It is a chain of *codons*, each of which is a triplet of bases that encodes one amino acid (e.g., the codon CAA is the amino acid Glutamine, which plays a role in regulating the acid-base balance in the kidney). In a sequence of bases, the *start codon* ATG marks the beginning of a gene, and one of three *stop codons*, TAG, TAA, or TGA, marks the end of a gene (but the start and stop codons themselves are not part of the gene). One of the first steps in analyzing a genome is to identify its genes.

You will write a program to find all the genes in a partial genomic sequence. Specifically, you will write a function `findGenes` such that the statement

```
n= findGenes('data.txt', 'result.txt')
```

reads a DNA data file called `data.txt` in the current directory, identifies all the genes in the sequence in the data file, writes the genes to a file called `result.txt`, and stores in variable `n` the number of genes found. *This is the only specification* in addition to the file formats that will be given below following the example. You will design the entire program and implement it following good programming style. Specifically, your gene finding algorithm must be clear—encapsulate specific, detailed tasks as subfunctions so that the structure of your gene finding algorithm is easy to understand.

1.1 Example

Consider the following string, which is not a real genomic sequence (at least not yet!). The sequence

AATGCCCATGGGGTATGTGTGAATGACATGCACTAACATGAAGTTGTAGGTTTAG

encodes four genes, which are underlined below:

AATGCCCATGGGGTATGTGTGAATGACATG<u>CAC</u>TAACATGAAGTTGTAGGTTTAG
AATGCCCATG<u>GGGTATGTG</u>TGAATGACATGCACTAACATGAAGTTGTAGGTTTAG
AATGCCCATGGGGTATGTGTGAATGACATGCACTAACATG<u>AAGTTG</u>TAGGTTTAG
AATGCCCATGGGGTATG<u>TGTGAA</u>TGACATGCACTAACATGAAGTTGTAGGTTTAG

Note the following features:

- Each gene is preceded by the start codon and is followed by a stop codon.
- Each gene has a non-zero length that is a multiple of 3 (since a codon is a triplet of bases).

- It is possible for genes to "overlap" since there are three "reading frames" within which one can identify codons (again because a codon is a triplet). The first "frame" starts at the beginning of the sequence. Below, the first 36 bases of the example sequence are shown, grouped in triplets with zero offset:
  AAT—GCC—CAT—GGG—GTA—TGT—GTG—AAT—GAC—ATG—CAC—TAA
  With this frame, we find the gene CAC. Next we show the same 36 bases with a frame offset of 1:
  A—ATG—CCC—ATG—GGG—TAT—GTG—TGA—ATG—ACA—TGC—ACT—AA
  With this second frame, we find the gene GGGTATGTG. Finally we show the 36 bases with a frame offset of 2:
  AA—TGC—CCA—TGG—GGT—ATG—TGT—GAA—TGA—CAT—GCA—CTA—A
  With this third frame, we find the gene TGTGAA.

1.2 File Format

The data files `humanC5_?.txt` where `?` is 1, 2, or 3, are portions of the human chromosome 5 nucleotide sequence downloaded from the data bank of the European Bioinformatics Institute. The sequences in the files are approximately 100, 1000, and 10000 bases long, respectively. The files contain ASCII characters (plain text). Each file begins with a line of text that identifies the sequence but *is not* part of the nucleotide sequence. The remaining lines in the file contain the sequence and are of varying lengths. Your code should work with the given files—do not modify the files in any way!

Your program writes the resulting gene data file in the following way: (1) write an ASCII (plain text) file; (2) each line corresponds to one gene and is exactly the length of the gene—no extra spaces; (3) there are as many lines as there are genes; and (4) the file is named as specified by the function call to `findGenes`.

1.3 Hints

- Break down the problem! There are three main tasks: find genes given a string, read a file, and write a file. Don't try to do everything at once.
- Creating a vector of characters is a good intermediate step between reading data and finding genes. Similarly, creating a cell array of genes is a reasonable intermediate step before writing the final data file.
- You can use the example sequence above in developing your gene finding algorithm. One of the three given data files doesn't encode a gene—be sure that your program ends appropriately and not in error. If a gene isn't found, `findGenes` returns the value zero and it is up to you whether to write an empty file or not write a file at all.
- Be sure to work through Module 2 Part 4 before starting to code; it will save you time.

2 Tic-tac-toe

Implement the game tic-tac-toe using MATLAB's graphic functions. In this game, two players take turns picking cells in a 3×3 board. The player who gets all 3 cells of a row, column, or diagonal wins. If there are no more free cells and neither player has won, the game ends in a tie.

You have the freedom to structure the code as you see fit, but you should define helper functions (subfunctions) as needed to make your solution modular—do not submit one long giant function. The helper functions must be saved in the same file containing the main function for the game; name your file `tictactoe.m`. The main function `tictactoe` has neither input nor output parameters, i.e., the function header is simply `function tictactoe()` .

Here are the requirements:

- The interaction happens only in one figure window.
- Display the name of the game and the 3x3 board.

- At each step display whose turn it is (i.e., player 1 or player 2)
- To choose a cell on the board, the players click in one of the cells (*hint*: check out function `ginput`). If the cell is already taken, ask the player to choose again. Once the player chooses an empty cell, display an X or an O, (depending on whose turn it was) in the cell.
- The game must stop once a player wins; display an appropriate message.
- If the game ends in a tie display an appropriate message.

The example code given in Module 2 Part 3 Items 1-3 demonstrate many of the graphics commands that you will need. Be sure to read them.

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.
Note that although all of these 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 submission!

- Δ Comment your code! If any of your functions is not properly commented, regarding function purpose and input/output parameters, you will be asked to resubmit.
- Δ Suppress all unnecessary output by placing semicolons (;) after assignment statements. 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 this by running each script several times in a row. Before each test run, you should 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 `findGenes` and `tictactoe.m` to CMS in the submission area corresponding to Assignment 2.
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 are need to resubmit (i.e., given a score of 0 or 1), fix all the problems and go back to Step 1! Otherwise you are done with this assignment. Hurrah!