CS100M Section 7

If you have any questions, **ask** your lab instructor or a consultant immediately! They are in the section to help you learn the material.

1 Determinant of a 3×3 matrix

Write a function **myDeterminant(x)**, where **x** is a 3×3 matrix. Use the following formula:

$$\det\left(\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}\right) = a\det\left(\begin{pmatrix} e & f \\ h & i \end{pmatrix}\right) - b\det\left(\begin{pmatrix} d & f \\ g & i \end{pmatrix}\right) + c\det\left(\begin{pmatrix} d & e \\ g & h \end{pmatrix}\right)$$

You may use the built-in function **det** to find the determinants of 2×2 matrices.

Vectorized code

MATLAB allows one to write vectorized code, or code that perform basic operations on multiple cells at the same time (with one statement). Consider code that we have written in the past involving vectors:

```
% Given vectors a, b of length 5
for k= 1:5
   c(k)= a(k) + b(k);
end
```

The above code *uses* vectors, but is not *vectorized code*, since the operations (addition, assignment) are performed on *individual cells*, one at a time. Another way to say this is that the *operands* are scalars (just one cell of a vector/matrix). Vectorized code perform operations (arithmetic, relational, logical, and assignment) on multiple cells in one statement:

```
% Given vectors a, b of length 5
c= a + b;
```

Notice that in the above vectorized code the operands are vectors. You can use vectorized code as long as the operands "line up," i.e., are vectors/matrices of the same dimensions. For multiplication, division, and exponentiation, you need to use a dot (.) in front of the operator. Examples:

```
% Given vectors a, b of length 5
d= a .* b;  % Result:  d(k) is a(k)*b(k)  for all k
f= a .^ b;  % Result:  f(k) is a(k)^b(k)  for all k
g= a == b;  % Result:  For all k,  g(k) is 1 if a(k)==b(k), otherwise g(k) is 0
```

What about using built-in functions? When you use a function that operates on a matrix, you do not know whether the code in that function are vectorized or not. We *do not* consider function calls to be vectorized code. For example, the statement `x= sum(v)` where `v` is a vector is *not* vectorized code—it just passes a vector to the function `sum`.

2 Back to DNA analysis

This problem should look somewhat familiar. This time, the goal is to create a solution using vectorized code. Write a function **countGGT(dna)** to return a count of the number of times the substring 'GGT' occurs in **dna**, an array of characters where each character can only be 'A', 'C', 'G', or 'T'. Assume that vector **dna** is longer than three.

Once you've complete your code for counting 'GGT', write a more general function **countPattern(dna,pat)** to return a count of the number of times string **pat** appears in **dna**. This function should also use vectorized code.

Which version (loops or vectorized code) is easier to write? Which version produces the most code?

Which version is fastest? The answer to this isn't clear without testing. For some patterns, the iterative version has the advantage of "giving up" quickly; often, after just two or three letters, it's clear that the pattern isn't going to match and the iterative version goes on to the next character position. The vectorized version always does work proportional to length(dna)*length(pattern); the iterative version can take less work.

3 Review exercises from last week's lab

Be sure that you complete/review last week's lab exercises. Ask if you have questions—don't just look at the solutions. Some of last weeks' questions are listed below:

4 Counting frequency

Write a MATLAB script that calls the function sequence from the previous question 1000 times for `m=10` and keep track of the lengths of the returned arrays. Specifically, your script should keep a vector `frequency` so that `frequency(f)` stores the number of arrays of length `f` that have been returned by function `sequence`.

5 A bit of DNA analysis

The four DNA nucleotides are represented by the letters A, C, G, and T. Write a function `findGGT(dna)` to return a vector of the locations where the substring GGT occurs in `dna`, an array of characters where each character can only be A, C, G, or T. A location where the substring GGT occurs is the index i for which the i-th position of `dna` is G and the following two vector components store the letters G and T.

6 A bit more DNA analysis

Write a function `findPattern(dna, pat)` to return a vector of the locations where the string `pat` appears in `dna`. As before, the strings contain only the characters A, C, G, and T. Assume that vector `dna` is longer than or equal in length to `pat`.