Vectors, vectorized code, and matrices

1 New vectors from old ones

Write a function \texttt{interleave} that receives as input two vectors \( v \) and \( w \) of the same length and returns a new vector \texttt{interleaved} consisting of all the elements of \( v \) and \( w \) in alternate order.

Example: If \( v \) is the vector \([1 \ 1 \ 1 \ 1]\) and \( w \) is the vector \([5 \ 6 \ 7 \ 8 \ 9]\) then \texttt{interleave}(v, w) gives \([1 \ 5 \ 1 \ 6 \ 1 \ 7 \ 1 \ 8 \ 1 \ 9]\).

\begin{verbatim}
function interleaved = interleave( v, w )
    % Post: interleaved stores individual elements of v and w in alternate order 
    % Pre: length(v)==length(w)
    interleaved = zeros(2*length(v),1);
    for i = 1:length(v)
        interleaved(2*i-1) = v(i);
        interleaved(2*i) = w(i);
    end
end
\end{verbatim}

2 Faster DNA analysis through vectorizing

Write a function \texttt{isPatternAt} that receives as input a DNA sequence \texttt{str}, a pattern \texttt{pat}, and an index \texttt{j} and returns 1 or 0 (i.e., \texttt{true} or \texttt{false}) depending on whether or not the pattern \texttt{pat} appears in \texttt{str} starting at position \texttt{j}.

Example: If \texttt{str} contains \texttt{'TGACTAC'} and \texttt{pat} contains \texttt{'ACT'}, then \texttt{isPatternAt(str, pat, 2)} gives 0 because the length 3 subsequence of \texttt{str} starting at position 2 is \texttt{'GAC'}, which is not the same as \texttt{pat}. On the other hand, \texttt{isPatternAt(str, pat, 3)} gives 1, and \texttt{isPatternAt(str, pat, 6)} gives 0 because the subsequence starting at position 6 is (at most) \texttt{'AC'}, which is only 2 characters long and therefore does not match \texttt{pat}.

Write two versions of the function. One using a loop \textit{without} vectorized code and another one using vectorized code. The vectorized version should contain no loops at all. In fact, it is possible to write a vectorized version consisting of one or two lines.

\begin{verbatim}
function result = isPatternAt(str, pat, j)
    % Post: result= {pat appears in str starting at position j}, true or false.
    % VECTORIZED VERSION
    % Pre: length(str) >= j >= length(pat). str and pat stores letters in upper case.
    result = length(str) >= j+length(pat)-1 && ... 
    sum( str( j : j+length(pat)-1 ) == pat ) == length(pat);
end
\end{verbatim}
function result = isPatternAt(str, pat, j)
% Post: result = true if pat appears in str starting at position j, false otherwise.
% Pre: length(str) >= j >= length(pat). str and pat stores letters in upper case.
if length(str) >= j+length(pat)-1
    i = 0; result = true;
    while result && i<length(pat)
        i = i + 1;
        if str(j+i-1) ~= pat(i)
            result = false;
        end
    end
else
    result = false;
end

3 Calculating “pair sums” of a vector

Write a function that takes as input a vector v and returns the vector of pair sums of v. A pair sum is the sum of two neighboring elements in a vector. E.g., if v = [2 3 6 1], then the pair sum vector is [5 9 7]. Note that the pair sum vector is one cell shorter than v. Use vectorized code (in this case, no loops!). With vectorized code one line of code will do the job! Hint: draw a picture to help visualize the situation. A cryptic hint: what is the “picture” of long addition you learned years ago in grade school?

function sums = pairSum(v)
% Post: sums = pair sums of vector v
% Pre: length(v) >= 2

    sums = v(1:end-1) + v(2:end);

4 Playing with matrices a little

Write a function that takes as input a square matrix M and computes the sum of the entries on the diagonal starting at the lower left corner and ending at the upper right corner.

function sumd = sumDiag(M)
% Post: sumd = sum of the entries on the 'secondary' diagonal of M
% Pre: M is square and is at least 1-by-1 in size

    sumd = 0;
    for i=1:length(M)
        sumd = sumd + M(i, length(M)-i+1);
    end