1. Two-dimensional interpolation

When you enlarge an image, you are actually adding data points among the existing data (pixels). How do you get the additional data points? One way is to interpolate from the neighboring points—take the average value. First, consider a simple case of one-dimensional interpolation, we add a data point between neighboring pairs of existing data points by taking the simple average. For example,

\[
\begin{array}{c c c}
2.0 & 1.0 & 2.0 \\
\end{array}
\]

becomes

\[
\begin{array}{c c c c c c c}
2.0 & 1.5 & 1.0 & 1.0 & 1.0 & 1.5 & 2.0 \\
\end{array}
\]

In 2-d interpolation, work with one dimension at a time. For example, given a matrix

\[
\begin{array}{c c c}
2.0 & 1.0 & 2.0 \\
6.0 & 4.0 & 3.0 \\
5.0 & 5.0 & 4.0 \\
\end{array}
\]

First we can add a column between two neighboring columns, so the matrix becomes 3 \times 7:

\[
\begin{array}{c c c c c c c}
2.0 & 1.5 & 1.0 & 1.0 & 1.0 & 1.5 & 2.0 \\
6.0 & 5.5 & 5.0 & 4.5 & 4.0 & 3.5 & 3.0 \\
5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 4.5 & 4.0 \\
\end{array}
\]

Then add a row between neighboring rows, so the final matrix will be 5 \times 7:

\[
\begin{array}{c c c c c c c}
2.0 & 1.5 & 1.0 & 1.0 & 1.0 & 1.5 & 2.0 \\
4.0 & 3.5 & 3.0 & 2.8 & 2.5 & 2.5 & 2.5 \\
6.0 & 5.5 & 5.0 & 4.5 & 4.0 & 3.5 & 3.0 \\
5.5 & 5.2 & 5.0 & 4.8 & 4.5 & 4.0 & 3.5 \\
5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 4.5 & 4.0 \\
\end{array}
\]

Write two versions of the following function \texttt{Interpolate2D}: (a) use non-vectorized code; (b) use vectorized code (work with whole rows, or whole columns, one at a time). Do not use built-in function \texttt{linspace}.

\begin{verbatim}
function newM = Interpolate2D(M)
% Perform 2-d interpolation on the real-valued data in nr-by-nc matrix M.
% The interpolated data are added between existing data points so newM is
% (2*nr-1)-by-(2*nc-1). Use the simple average as the interpolated value.
\end{verbatim}
2. Cell array vs. vector

You already know that a vector is a collection of simple data. For example, you can have a vector of numbers (each component stores a single number) or a vector of characters (each component stores a single character). In a cell array, each cell can store an item that may be more complex than just a number or a character.

```matlab
v = rand(1,4) % a VECTOR of length four, each cell stores ONE number
v(3) % Notice that you use PARENTHESES to access a cell in a VECTOR

c = cell(1,4) % Use built-in function CELL to create a CELL ARRAY. Note that its "class" in the Workspace pane is "cell." Right now each cell is empty, therefore the screen output shows four empty vectors.

c{2} = v % Put a VECTOR in the 2nd cell of the CELL ARRAY. Notice that we use CURLY BRACKETS to access a cell in a CELL ARRAY.

c(3) = 1 % You get an error message: Must use curly brackets to access a cell in a CELL ARRAY; parentheses are for VECTORS.

c{2} % Display what is in cell 2 of CELL ARRAY c: a vector!

% So how do you display, say, the fourth value in the VECTOR in the 2nd cell of CELL ARRAY c?
c{2}(4) % Once again, use curly brackets for the index of the CELL ARRAY; use parentheses for the index of the of VECTOR.

% Now we put other things in the cell array. Note that one can put different types of things in a CELL ARRAY. This is not possible in a VECTOR, whose cells must store the same (simple) type of data.
c{1} = 'cat'
c{3} = 10
c{4} = ones(2,1)

% An alternate way to create a cell array is to specify all the contents inside CURLY BRACKETS using spaces, commas, or semi-colons as the separator:
d = {'cat'; 10; v; ones(2,1)} % A cell array of four cells
length(d) % The length function works for cell arrays as well.
```

3. Deck of cards

During lecture we developed a function CardDeck that returns a 1-d cell array of strings, each string representing one card in a standard deck of cards. Now write the following function:

```matlab
function DispCards(ca, p, q)
% Display the contents in cells p through q of cell array ca.
% ca is a 1-d cell array of strings. Assume that p and q are valid indices of ca.
```