Introduction

This assignment is a timing exercise for 3 implementations of convolution with MatLab’s acceleration feature being both 'on' and 'off'. First, you need to read the *lena.jpg* image as an RGB image, convert it to a YIQ image and then use the Y images (now a grayvalue file) to perform your convolutions. The following MatLab code will read the *lena.jpg* image and compute the Y grayvalue image:

```matlab
close all
RGB=imread('lena.jpg');
figure
imshow(RGB,[]);
title('Original RGB lena.jpg');
YIQ=rgb2ntsc(RGB);
Y=squeeze(YIQ(:,:,1));
```

For *lena.jpg*, Y has $512 \times 512$ grayvalues.

Convolution is to be done using a $3 \times 3$ mask of all $1/9$’s i.e. 

```
mask=ones(3,3,'double')/9.
```

Convolution can be thought of as point-wise multiplication and summing. Given a mask, *mask*, and an image Y each convolved pixel can be computed as:

```matlab
temp3(i,j)=0;
for a=-1:1
  for b=-1:1
    temp3(i,j)=temp3(i,j)+mask(a+2,b+2)*Y(i+a,j+b);
  end % b
end % a
```
where \texttt{temp3} is a temporary array of the same size as \texttt{Y} and \texttt{mask} is \texttt{ones(3,3,'double')/9}. So we multiply each pixel in the $3 \times 3$ neighbourhood centered at pixel $(i,j)$ by their corresponding mask values. This, effectively, computes a weighted average of the pixels in that neighbourhood.

You are to implement convolution via \texttt{imfilter}, via \texttt{vectorization} and via nested loops doing array multiplications and summing. These 3 functions will be run with both \texttt{feature('accel','on')} and \texttt{feature('accel','on')} to see the effect on the timing. You are to write 3 nested MatLab functions, called \texttt{method1}, \texttt{method2} and \texttt{method3}, to be called from your main MatLab function \texttt{ass2_2016_pgm} (all 4 functions stored in MatLab file \texttt{ass2_2016_pgm.m}, which you have to submit via owl).

Some details:

1. Function \texttt{[time1(II,JJ),temp1]=method1(II,Y)} does convolution trivially and efficiently using \texttt{imfilter} as:

\[
\texttt{temp1} = \texttt{imfilter(Y,mask,'same','symmetric','conv')};
\]

\texttt{II} is the index (from 1 to 2) that tells if acceleration is to be on or off. \texttt{Y} is the grayvalue \texttt{lena} image. \texttt{JJ} is 1 to \texttt{NUMBER_RUNS}. The value of \texttt{NUMBER_RUNS} can be 1 (to debug your code and display images) or 10 to obtain the final results (without displaying images). \texttt{time1} is a 2D array containing the individual timing measurements for \texttt{NUMBER_RUNS} runs and for acceleration being on or off (see below). \texttt{mask} is \texttt{ones(3,3,'double')/9}.

2. Function \texttt{[time2(II,JJ),temp2]=method2(II,Y)} does convolution by vectorization. This uses point by point multiplication of elements on 2D arrays.

Consider the 1D case, where we have 2 equal sized vectors, \texttt{x} and \texttt{y}. We want to compute the element by element product \texttt{x(i)*y(i+1)}. To do this we use \texttt{circshift(y,-1)} to shift the values of \texttt{y} to the left by 1. Wraparound occurs in that \texttt{y(1)} becomes \texttt{y(end)}. Now the product can be computed as \texttt{x.*y}.

In 2D we have to sum a pixel at \texttt{Y(i,j)} with the pixels at its 8 neighbours \texttt{Y(i-1,j+1)}, \texttt{Y(i-1,j)}, \texttt{Y(i-1,j-1)}, \texttt{Y(i,j+1)}, \texttt{Y(i,j-1)}, \texttt{Y(i+1,j+1)}, \texttt{Y(i+1,j)}, and \texttt{Y(i+1,j-1)}. We need to line up the neighbouring image pixels in eight additional image arrays using
\( \text{circshift}(Y, [a \ b]) \), where \( a \) and \( b \) are 0, 1 or -1. Then we can do vectorization by simple summing, diving the result by 9 and saving that result in temp2. This removes the need for explicitly using the mask array.

To see how \( \text{circshift} \) works in 2D consider the 2D array \( A \) having values:

\[
A = \begin{pmatrix}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15 \\
16 & 17 & 18 & 19 & 20 \\
21 & 22 & 23 & 24 & 25
\end{pmatrix}
\]

We can shift up and down as:

\[
\text{circshift}(A, [-1 \ 0]) = \begin{pmatrix}
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15 \\
16 & 17 & 18 & 19 & 20 \\
21 & 22 & 23 & 24 & 25 \\
1 & 2 & 3 & 4 & 5
\end{pmatrix}
\quad \text{and} \quad
\text{circshift}(A, [1 \ 0]) = \begin{pmatrix}
21 & 22 & 23 & 24 & 25 \\
1 & 2 & 3 & 4 & 5 \\
11 & 12 & 13 & 14 & 15 \\
16 & 17 & 18 & 19 & 20
\end{pmatrix}.
\]

Note the “wraparound” as the top row of \( A \) becomes the bottom row of \( A \) for \( \text{circshift}(A, [-1 \ 0]) \) while the bottom row of \( A \) becomes the top row of \( A \) for \( \text{circshift}(A, [1 \ 0]) \). Similarly, \( \text{circshift}(A, [0 \ -1]) \) and \( \text{circshift}(A, [0 \ 1]) \) shift \( A \) to the left of the right respectively, again with wraparound. Lastly we note that we can combine horizontal and vertical shifts. For example, \( \text{circshift}(A, [1 \ 1]) \) and \( \text{circshift}(A, [-1 \ 1]) \) produce:

\[
\text{circshift}(A, [1 \ 1]) = \begin{pmatrix}
25 & 21 & 22 & 23 & 24 \\
5 & 1 & 2 & 3 & 4 \\
10 & 6 & 7 & 8 & 9 \\
15 & 11 & 12 & 13 & 14 \\
20 & 16 & 17 & 18 & 19
\end{pmatrix}
\quad \text{and} \quad
\text{circshift}(A, [-1 \ -1]) = \begin{pmatrix}
10 & 6 & 7 & 8 & 9 \\
15 & 11 & 12 & 13 & 14 \\
20 & 16 & 17 & 18 & 19 \\
25 & 21 & 22 & 23 & 24 \\
5 & 1 & 2 & 3 & 4
\end{pmatrix}.
\]
where \( A \) has been shifted to the right and down and \( A \) has been shifted up and to the left respectively. Both results again show wraparound.

3. Function \([\text{time3}(\text{II},\text{JJ}), \text{temp3}] = \text{method3}(\text{II},\text{Y})\) does convolution via array multiplication and summing and requires a temporary array the same size as the image (either preallocated if you want JIT compilation to occur) or allocated element by element as needed (if you don’t want JIT computation to occur). So note that acceleration and JIT should go hand in hand. The input and output parameters are similar to the other 2 functions above.

Convolution at pixel \((i,j)\) is done multiplying the 8 neighbouring pixel values around \((i,j)\) by the corresponding values of mask and summing the result and storing it at \((i,j)\). For pixel \((i,j)\) the calculation can be done as above. Note that \(Y(i+a,j+b)\) requires that the smallest value of \(i\) or \(j\) to be 2 (as \(a\) and \(b\) can be -1 and we don’t want the array indices to go out of bounds). Similarly, the largest values of \(i\) or \(j\) can be 511 at most (as \(a\) or \(b\) can be 1 and we don’t want the array indices to go out of bounds). Note that you should not use magic constants like 512 as the dimension of the image in your program. Instead use \text{size}(Y,1)\) or \text{size}(Y,2)\) as appropriate (this makes your code more general).

Use \text{feature}('accel','on')\) to turn acceleration on and \text{feature}('accel','off')\) to turn acceleration off for all 3 functions depending on the value of input parameter \(\text{II}\).

This assignment requires you to use \text{tic} and \text{toc} to perform timing measuring for the above 3 convolution strategies using acceleration on and off. Use 3 arrays:

\begin{verbatim}
NUMBER_RUNS=10;
time1=zeros(2,NUMBER_RUNS,'double');
time2=zeros(2,NUMBER_RUNS,'double');
time3=zeros(2,NUMBER_RUNS,'double');
\end{verbatim}

to keep track of each time measurement for each of the 3 strategies. The row index of these 3 arrays are for whether acceleration has been turned on or turned off. So one run through the 3 strategies for acceleration being both on and off will yield 6 timing measures. The
constant \texttt{NUMBER\_RUNS} can be 1 when you are debugging your program and want to display your convolved images but should be 10 for the final run of your program.

After all runs have been made, use \texttt{mean} and \texttt{std} to compute the mean and standard deviation for each related set of timing measurements. For example, \texttt{mean(squeeze(time1(1,:)))} would compute the average time for acceleration on for method 1 (convolution by \texttt{imfilter}) while \texttt{mean(squeeze(time3(2,:)))} would give the average time for acceleration off for method 3 (convolution by loops). Similarly, \texttt{std(squeeze(time1(1,:)))} and \texttt{std(squeeze(time3(2,:)))} would compute the respective standard deviation values.

Your program should display one set of blurred images by the 6 strategies (you should get the same very mild blurring results no matter which method you are using) but for the 10 run it is better to have figure output suppressed, otherwise you will have a cluttered display areas with 60+ images). The phrase “mild blurring” means just that, you may not see the blurring effect except in image areas with fine detail because the blurring mask is so small.

A nested loop for the number of acceleration types (2) and number of runs (\texttt{NUMBER\_RUNS}) might be the best way to run your program and collect the statistics. The mean and standard deviation values would be printed after this loop. For these values for acceleration on and off, compute the speedup factor for the 3 strategies. For example,

\[
\texttt{mean(squeeze(time3(2,:)))/mean(squeeze(time3(1,:)))}
\]

computes the speedup factor for convolution via loops when acceleration is both on and off. If acceleration is indeed a good idea, that the speedup factors should be \( > 1 \).

A possible strategy for writing this program is:

1. Code your convolution by \texttt{imfilter} solution first as this is dead simple. Turn acceleration on and off in this function depending on the II value. Note that at the left and right and top and bottom borders \texttt{imfilter} uses reflection of pixel rows and columns. Verify \texttt{temp1} correctness qualitatively by looking at the detail on the hat.

2. Code your convolution by vectorization second. Again, turn acceleration on and off in this function depending on the II value. Eight uses of \texttt{circshift} are required and then a simple addition of those arrays and the \texttt{Y} array. Don’t forget the divide this result by

9. Do you get the same result in \texttt{temp2} as in \texttt{temp1} using \texttt{imfilter} (except at the left
and right and top and bottom borders, where vectorization uses wraparound to compute these values).

3. Code your convolution by for loop third. Remember to perform array preallocation for the new image (in temp3) to get JIT acceleration when the value of II indicates this. Do not perform array preallocation when the value of II indicates acceleration is off. You should get the same image result as by the other 2 strategies (except at the left and right and top and bottom borders, where all pixels have the value 0).

4. Introduce a nested loop, with the first index, (II), going from 1 to 2 and the second index, (JJ), going from 1 to NUMBER_RUNS. Add code to collect the statistics at each set of indices using the time arrays.

5. After the nested loops are finished compute the mean and standard deviation values and print them. Print out the speedup factors as well.

Use print statements to help with your debugging.

Finally, note that your results may differ from your colleagues’ results and from the professor’s results as you most likely will be using a different OS system, a different machine and a different version of MatLab. JIT compilation is a (deliberately) undocumented feature of MatLab and is constantly changing. The MatLab command feature may or may not be working correctly. You may obtain results that are non-intuitive!!! Just make sure you calculations are correct and all will be ok.