Homework 3

Problem 1: Warm up problems (15 points)

These questions are most similar to what I’d ask on an exam or on future quizzes.

1. Suppose $A$ is an $r \times s$ matrix. How many multiplication and addition operations are required for a general matrix vector product $Ax$.

2. Suppose $A$ is an $m \times n$ matrix and $D$ is an $m \times m$ diagonal matrix (that means that it has zeros everywhere except on the diagonal elements). Write down the simplest expression for the $ij$th entry of $DA$ you can. (This will end up being used in the future for some problems!)

3. This is just like the previous problem, but now do the same thing where $A$ is an $m \times n$ matrix and $D$ is an $n \times n$ diagonal matrix. Write down the simplest expression for the $ij$th entry of $AD$.

Problem 2: Implement MatMul

(If you wish, you may perform this task in a language besides Matlab that provides a built-in matrix-matrix multiplication routine – such as Python + numpy or julia, but you must have the same comparison.)

1. Implement the following function:

   ```matlab
   function C = matmul(A,B)
   % MATMUL Matrix-Matrix multiplication
   % C = matmul(A,B) returns the matrix-matrix product
   % C = A*B
   %
   using only scalar operations. (i.e. without any of Matlab’s built in matrix operations.)
   
   2. Report the output of your function for the following commands

   ```matlab
   matmul([1 2], [2; 1])
   matmul([1 2; -2 4; 0 3], [-1; 1])
   matmul([-1 1], [1 -2 0; 2 4 3])
   matmul(3, 5)
   t = pi/2; matmul([cos(t) -sin(t); sin(t) cos(t)], ...
                   [cos(t) sin(t); -sin(t) cos(t)]);
   ```

3. Compare the time and accuracy of your code to Matlab’s built in matrix-matrix operator on matrices of random normals with sizes ranging between 10 and 1000. To evaluate accuracy, use:
C = matmul(A,B)
D = A*B;
diff = norm(C-D,'fro');

To evaluate time, use:

timer = tic;
C = matmul(A,B);
time = toc(timer); % time contains the time elapsed in computing C

Use 50 different random samples for each time, and report the mean time for Matlab vs. your own code as one plot. (So on the x-axis we have problem size, and on the y axis, we have one line for the mean time over 50 trials of Matlab's built-in operation, and a second line we have the mean time over 50 trials of our function.)

Also prepare the same type of plot on the maximum difference over 50 trials between your code and Matlab's code to show the accuracy.

Use semilogy for your plots.

4. Write a new function:

function C = matmul2(A,B)
% MATMUL2 Faster Matrix-Matrix multiplication
%%
% C = matmul2(A,B) returns the matrix-matrix product
% C = A*B
%

That uses some of the ideas we talked about in class to make a faster matrix-matrix product. You may use Matlab's built-in matrix-matrix multiplication for up to 16-by-16 matrices. Or matrix-vector products up to \(256 - by - 1\). Compare the time and accuracy of your new \texttt{matmul2} code to your original.

Problem 3: Make Yoda Spin!
1. G&C Chapter 2 Problem 11
2. G&C Chapter 2 Problem 12

Problem 4: Make an image small

Consider the following problem. We have a 32 \times 32 pixel image. Each pixel is a real-valued number between 0 and 1. However, I want to show this on an iPhone and I only have a 16 \times 16 pixel area to show the image. In order to reduce the size of the image, I want to average groups of 4 pixels. In this problem, we'll create a Matlab program to do this.

Let's work through a smaller example first. For the 4 \times 4 image, represented here by a matrix-like array:

\[
\begin{bmatrix}
x_1 & x_2 & x_3 & x_4 \\
x_5 & x_6 & x_7 & x_8 \\
x_9 & x_{10} & x_{11} & x_{12} \\
x_{13} & x_{14} & x_{15} & x_{16}
\end{bmatrix}
\]

I want to compute
We will solve this problem using a matrix-vector product.

1. Suppose I call:

   
   \[
   y_1 = \frac{x_1 + x_2 + x_5 + x_6}{4}, \quad y_2 = \frac{x_3 + x_4 + x_7 + x_8}{4}, \\
   y_3 = \frac{x_9 + x_{10} + x_{13} + x_{14}}{4}, \quad y_4 = \frac{x_{11} + x_{12} + x_{15} + x_{16}}{4}.
   \]

   Further, suppose we consider the vectors \(y \in \mathbb{R}^4\) and \(x \in \mathbb{R}^{16}\) to be the new image and old image, respectively. Write down the matrix \(A\) such that \(y = Ax\).

   In the remainder of the problem, we'll work through how to do this for a particular image in Matlab.


   ```matlab
   load smallicon.mat
   ```

   in Matlab. You should get a \(32 \times 32\) matrix \(X\). What is the sum of diagonal elements of \(X\)? (If you wish to use another programming language, feel free, but make sure you can do the same things in it. I’ve provided http://www.cs.purdue.edu/homes/dgleich/cs314-2014/homeworks/smallicon.txt as a text file for use in other languages.)

3. Matlab has a command for viewing a matrix as an image. Run the following commands:

   ```matlab
   imagesc(X)
   ```

   The result looks really weird, right? That’s because Matlab is making up a color for each pixel. We can tell it to use a greyscale colormap by executing:

   ```matlab
   colormap(gray)
   ```

   Save the resulting Matlab figure as an image to include in your homework.

4. In what follows, we’ll talk about two different types of indices. The image index of a pixel is a pair \((i, j)\) that identifies a row and column for the pixel in the image. The vector index of a pixel is the index of that pixel in a linear ordering of the image elements. For instance, above, pixel \((3, 2)\) has linear index 10. Also, pixel \((1, 4)\) has index 4. Matlab can help us built a map between pixel indices and linear or vector indices:

   ```matlab
   N = reshape(1:(4*4), 4, 4)';
   ```

   This creates the pixel index to linear index for the problem above because

   ```matlab
   N(1,4)
   N(3,2)
   ```

   return the appropriate entry.

   In your own words, explain what the `reshape` operation does. What happens if you omit the final transpose above and try:

   ```matlab
   N = reshape(1:(4*4), 4, 4);
   ```
5. Now we need to construct the matrix $A$ in order to reduce the size of a $32 \times 32$ image to a $16 \times 16$ image as we did in part 1. Suppose we call the output vector $y$ and the output image $Y$. I'm giving you the following template, that I hope you can fill in. Feel free to construct an $A$ that accomplishes our image reduction task any way you choose, but the following should provide some guidance.

\[
\begin{align*}
\text{NX} &= \text{<fill in>}; \quad \% \text{ the map between pixel indices and linear indices for X} \\
\text{NY} &= \text{<fill in>}; \quad \% \text{ the map between pixel indices and linear indices for Y} \\
\text{for } i=1:32 \\
\quad \text{for } j=1:32 \\
\quad\quad x_i &= \text{<fill in>}; \quad \% \text{ the index of the pixel } i,j \text{ in the vector } x \\
\quad\quad y_{ij} &= \text{<fill in>}; \quad \% \text{ the resulting location of pixel in the matrix } Y \\
\quad\quad y_i &= \text{<fill in>}; \quad \% \text{ the index of the linear pixel in the vector } y \\
\quad\quad A(y_i,x_i) &= 1/4; \quad \% \text{ place the entry of the matrix}
\end{align*}
\]

6. In order to use the matrix $A$ we created, we need to convert the matrix $X$ into a vector! The reshape operation helps here:

\[
x = \text{reshape}(X',32*32,1);
\]

We can now rescale the image $X$ by multiplying by $A$ and reorganizing back into a matrix $Y$.

\[
y = A*x;
Y = \text{reshape}(y,16,16)';
\]

Show the image of $Y$. Does that look correct?