EGR 141: Intro to Functions

Summary: The goal of this lab is to help understand scripts, functions, and scope. You should not use any MATLAB commands or concepts that are discussed in future chapters and sections (no loops or if statements yet!)

- Each of the following problems should have a script and, possibly, a function associated with them.
- For each problem, the script file should be called something appropriate, such as *Lab3_1_yourName.m*. Alternatively, you can put all of your scripts and functions in one file, if desired. If you choose to do this instead, make sure everything is clearly labeled.
- Include any functions that you needed to create in order to complete the problem. Name them whatever is indicated in the problem.
- Inside your script, solve each of the given problems. In between each problem, type *pause;* Clearly indicate where the code for each problem begins by using a comment block. Start each new problem with a *clear .*
- If my example output "lines up nicely" then your output should as well.
- All output statements should output variables, not pre-computed constants.
- Note that example output for each problem is not necessarily correct output (I intentionally change numbers so my answers will not always match your answers).
- *Pay attention to what your input/output should be in each function.* There should be no fprintf or input statements within a function unless specifically asked for.

1. Make a function, called *areaPermTri* whose input consists of two 3×1 vectors x and y corresponding to the xy coordinates of three points in the xy plane. The output for the function is the area and perimeter (in that order) of the triangle made from those three points. Test your function on the following

   Triangle 1 vertices: $(0,0), (1,0), (0,1)$
   Triangle 2 vertices: $(0,0), (6,3), (6,5)$
   Triangle 3 vertices: $(5,5), (3,2), (2,6)$

```
Lab 3 - Triangle Area & Perimeter
 For a triangle with vertices
  (0,0)  (1,0)  (1,0)
 Area:      0.50
 Perimeter: 3.41
```

2. Create a function, called *matrixPrintNby2* which takes in two arguments. The first is a matrix of size $N \times 2$, and the second is a string (the *name*). The function returns nothing, but outputs the matrix "nicely" to the screen. The name should be as centered as possible with respect to the matrix and numbers should line up exactly as shown in the example. Test your function on each of the given names and sizes, using a matrix of random integers between -10 and 10 (inclusive).

   (a) ronSwanson, 3×2
   (b) leslieKnope, 5×2
   (c) aprilLudgate, 8×2
   (d) andyDwyer, 10×2

   Note that it is okay if your output isn't perfect for a 1x2 or a 2x2.

```
Lab 3 - Matrix Print Nx2
                [ -1  -4]
   alCzervik=[ -2   6]
                [ -1   6]
                [ -1 -10]
```

3. Define a function *addMultComp* that:

   - Take in two anonymous functions, $f(x)$ and $g(x)$
   - Returns the following three functions, in order: $f(x)+g(x)$, $f(x)g(x)$, and $f\left(g(x)\right)$

   Test your *addMultComp* using the functions $f(x)=x^2$ and $g(x)=\cos \pi x$ and the vector

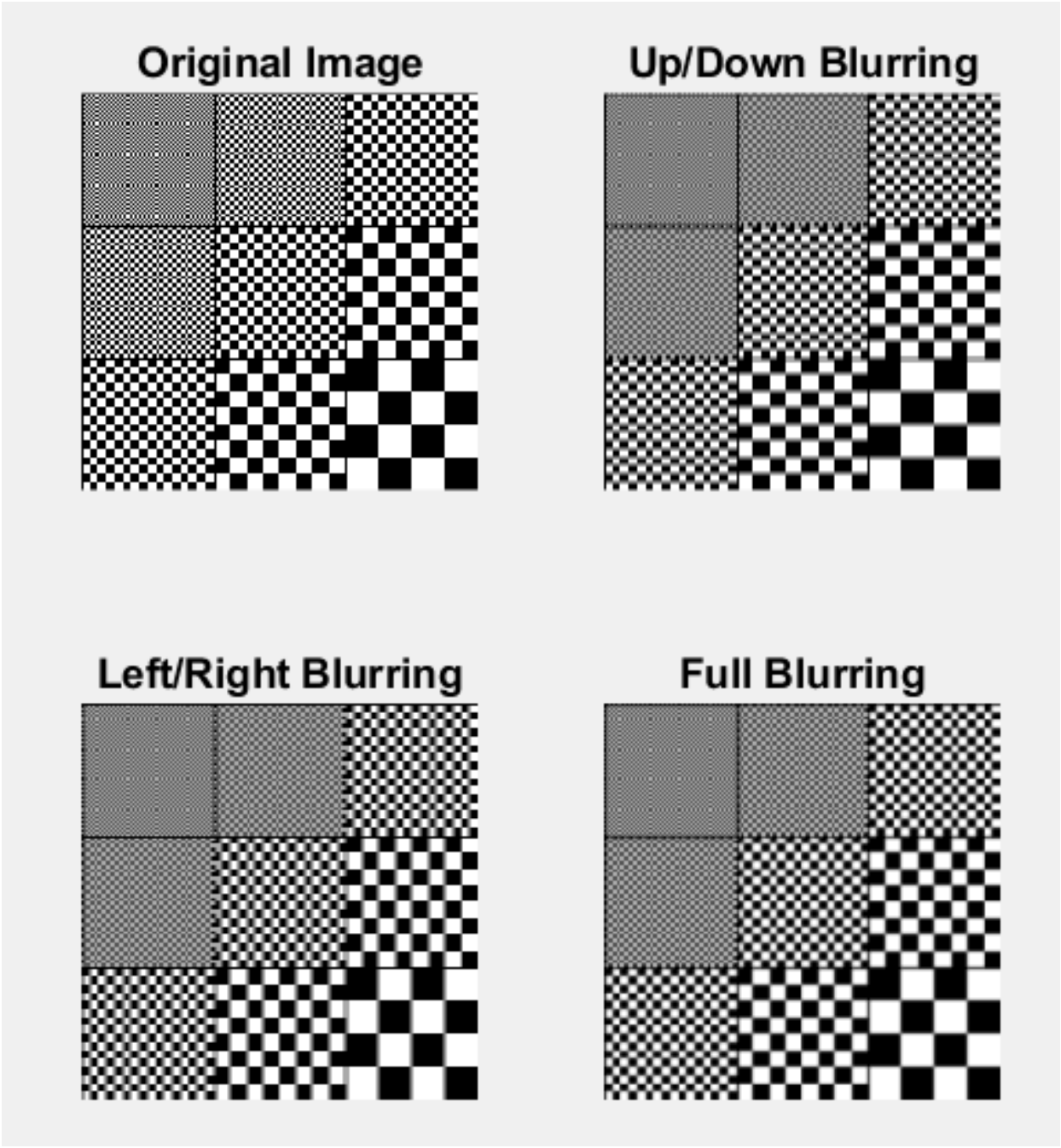
   $$x=\left[0,\ \frac{1}{4},\ \frac{1}{2},\ \frac{3}{4},\ 1\right].$$

   Output x, $f(x)$, $g(x)$ and the three results of your function in a nicely formatted table. As usual, make sure your example lines up exactly like the example.

```
Lab 3 - Function Functions
 f(x) = x^3 and g(x) = sec(pi/4*x)
         x:  0.00   0.25   0.50   0.75   1.00
      f(x):  0.00   0.02   0.13   0.42   1.00
      g(x):  1.00   1.02   1.08   1.20   1.41
 f(x)+g(x):  1.00   1.04   1.21   1.62   2.41
 f(x)*g(x):  0.00   0.02   0.14   0.51   1.41
   f(g(x)):  1.00   1.06   1.27   1.74   2.83
```

4. You are given a script in the file *plotCheckerboardAndBlurs.m*. Inside the script, there is code to create a checkerboard image, pass the matrix from the image into a function, *aBlur*, and then plot the original image along with three blurred images. Your job is to create the function, *aBlur*. The input consists of a matrix containing grayscale values of each pixel in an image (values between 0 and 1. 0 = black, 1 = white). The output is three matrices, in this order:

   - *ud*: A matrix that does up/down blurring. Each pixel is averaged with the one(s) above and/or below it. Pixels on a boundary (top/bottom) are averaged with only one neighbor. Middle pixels are the average of themselves, the one above it, and the one below it.
   - *lr*: A matrix that does left/right blurring. Each pixel is averaged with the one(s) to the left and/or right of it. Same rules as *ud* with respect to boundaries.
   - *all*: A matrix that is the average of the *ud* and *lr* matrices, pixelwise (no neighbor blurring, just average of the two matrices).


Original Image
Up/Down Blurring
Left/Right Blurring
Full Blurring