PURDUE UNIVERSITY · CS 51400
NUMERICAL ANALYSIS

HOMEWORK
Patrick Talley
September 30, 2016

Please answer the following questions in complete sentences in submit the solution on Blackboard Fri. Sept. 30th by Midnight.

Homework 3

- 2016-09-27: Small clarifications based on Piazza questions.

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 $\boldsymbol{A}$ is an $r \times s$ matrix. How many multiplication and addition operations are required for a general matrix vector product $\boldsymbol{A}\mathbf{x}$.
2. Suppose $\boldsymbol{A}$ is an $m \times n$ matrix and $\boldsymbol{D}$ 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 ijth entry of $\boldsymbol{DA}$ 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 $\boldsymbol{A}$ is an $m \times n$ matrix and $\boldsymbol{D}$ is an $n \times n$ diagonal matrix. Write down the simplest expression for the ijth entry of $\boldsymbol{AD}$.

Solution

1. $$y_{ij} = \sum_{j=1}^{s} A_{ij} * x_j$$

   This is done r times to form an r x 1 matrix. This means addition and multiplication is done $r * s$ times.

2. $$DA = C \in \mathbb{R}^{m*n}$$
   $$C_{ij} = D_{ii}A_{ij}$$

3. $$AD = C \in \mathbb{R}^{m*n}$$
   $$C_{ij} = D_{jj}A_{ij}$$

Problem 2: Implement MatMul

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

1. Implement the following function:

```
"""
`matmul`
========

Compute Matrix-matrix multiplication or MatMul.

Functions
---------
- `C = matmul(A,B)` returns the matrix-matrix product of C=A*B
"""
function matmul(A,B)
end
```

using only scalar operations. (i.e. without any of Julia's built in matrix operations.)

2. Report the output of your function for the following commands

```
@show matmul([1 2], [2; 1])
@show matmul([1 2; -2 4; 0 3], [-1; 1])
@show matmul([-1 1], [1 -2 0; 2 4 3])
@show matmul(3, 5)
t = pi/2
@show 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 Julia'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 = vecnorm(C-D,);
```

To evaluate time, use:

```
dt = @elapsed C = matmul(A,B);
```

Use 50 different random samples for each size and report the mean time for Julia 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 Julia'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 Julia's code to show the accuracy.

4. Write a new function:

```
"""
`matmul2`
========

Compute Matrix-matrix multiplication or MatMul faster

Functions
---------
- `C = matmul(A,B)` returns the matrix-matrix product of C=A*B
"""
function matmul(A,B)
end
```

That uses some of the ideas we talked about in class to make a faster matrix-matrix product. You may use Julia'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 `matmul2` code to your original.

Solution

1.

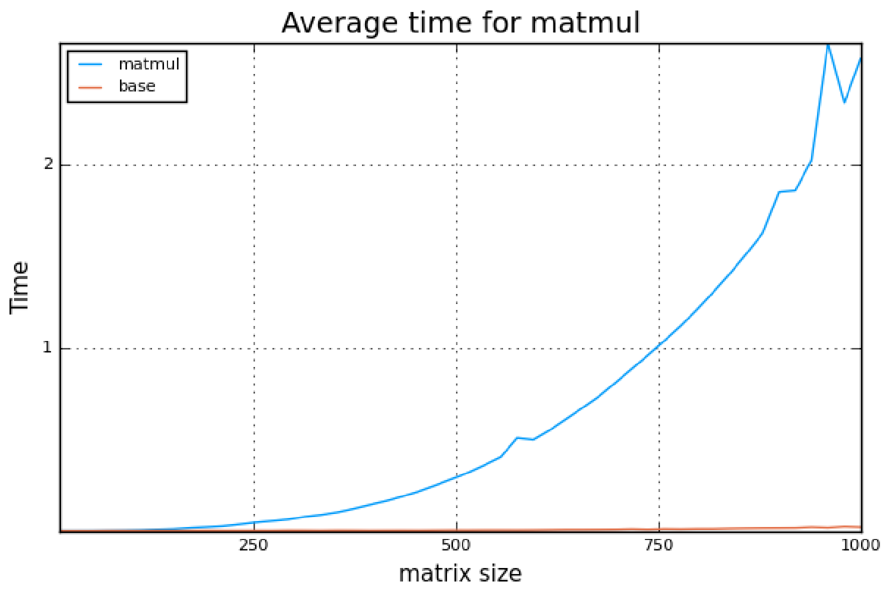
2. matmul([1 2],[2;1]) = [4.0]
   matmul([1 2;-2 4;0 3],[-1;1]) =

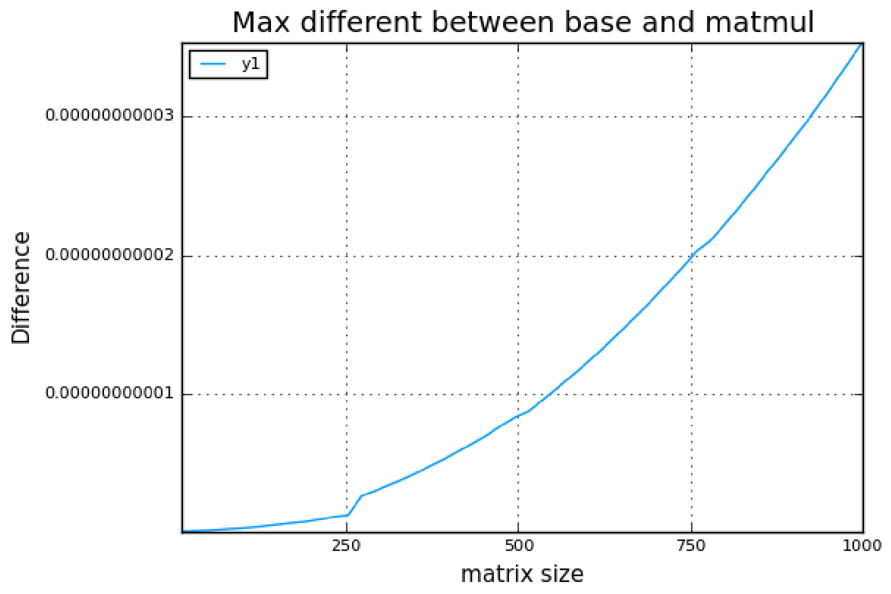
$$\begin{bmatrix} 1.0 \\ 6.0 \\ 3.0 \end{bmatrix}$$

   matmul([-1 1],[1 -2 0;2 4 3]) = [1.0 6.0 3.0]
   matmul(3,5) = [15.0]
   matmul([cos(t) -(sin(t));sin(t) cos(t)],[cos(t) sin(t);-(sin(t)) cos(t)]) =

$$\begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0] \end{bmatrix}$$

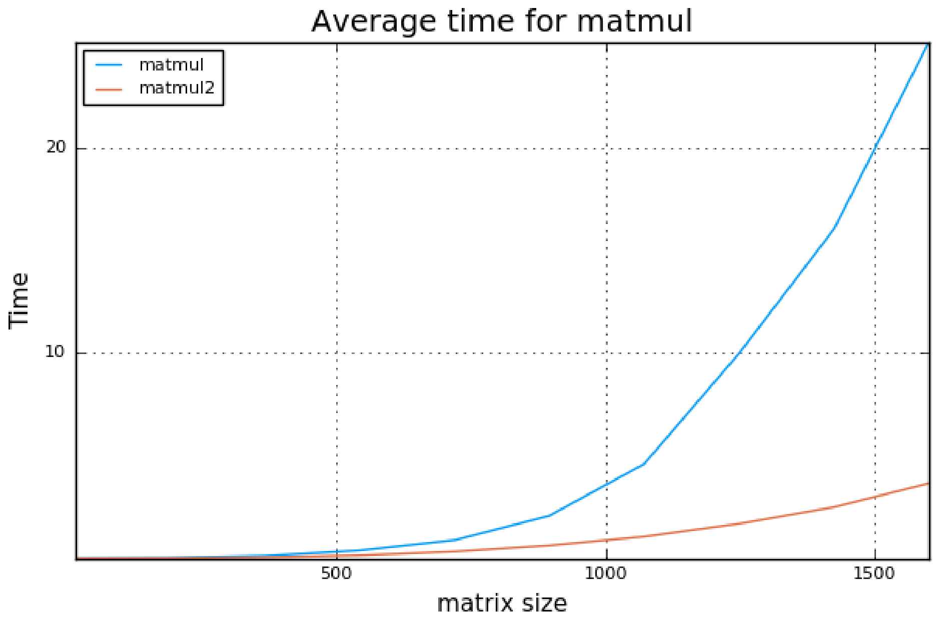
3.




Max different between base and matmul
y1
Difference
0.00000000003
0.00000000002
0.00000000001
250
500
750
1000
matrix size

4.

5.


Average time for matmul
matmul
matmul2
Time
20
10
500
1000
1500
matrix size

Problem2

September 30, 2016

```
In [3]: """
        `C = matmul(A,B)` returns the matrix-matrix product of C=A*B
        function matmul(A,B)
        end
        """
        function matmul(A,B)
            if ndims(A) < 2
                A_rows = length(A)
                A_cols = 1
            else
                A_rows, A_cols = size(A)
            end
            if ndims(B) < 2
                B_rows = length(B)
                B_cols = 1
            else
                B_rows, B_cols = size(B)
            end

            if A_cols != B_rows
                return "Error: inner dimensions must agree"
            end

            C = zeros(A_rows, B_cols)
            for ii=1:A_rows
                for jj=1:B_cols
                    for kk=1:A_cols
                        C[ii,jj] += A[ii,kk]*B[kk,jj]
                    end
                end
            end

            return C
        end
```

```
Out[3]: matmul (generic function with 1 method)
```

```
In [3]: @show matmul([1 2], [2; 1])
        @show matmul([1 2; -2 4; 0 3], [-1; 1])
        @show matmul([-1 1], [1 -2 0; 2 4 3])
        @show matmul(3, 5)
        t = pi/2
        @show matmul([cos(t) -sin(t); sin(t) cos(t)],
            [cos(t) sin(t); -sin(t) cos(t)]);
```

```
matmul([1 2],[2;1]) = [4.0]
matmul([1 2;-2 4;0 3],[-1;1]) = [1.0
 6.0
 3.0]
matmul([-1 1],[1 -2 0;2 4 3]) = [1.0 6.0 3.0]
matmul(3,5) = [15.0]
matmul([cos(t) -(sin(t));sin(t) cos(t)],[cos(t) sin(t);-(sin(t)) cos(t)]) = [1.0 0.0
 0.0 1.0]
```

```
In [19]: nMats = 50
         nSamples = 50

         N = linspace(10, 1000, nMats)
         timediff = zeros(nMats, nSamples)
         dt = zeros(nMats, nSamples)
         dt2 = zeros(nMats, nSamples)
         diffSols = zeros(nMats, nSamples)

         for i=1:nMats
             M = round(Int,N[i])
             for j=1:nSamples
                 A = randn(M, M)
                 B = randn(M, M)
                 dt[i,j] = @elapsed C = matmul(A,B);
                 dt2[i,j] = @elapsed D = A*B;
                 diffSols[i,j] = vecnorm(C-D,)
                 timediff[i,j] = dt2[i,j]-dt[i,j]
             end
         end
```

```
In [28]: using Plots
         plot(N, mean(dt,2), xlabel= "matrix size", ylabel="Time", title="Average time for matmul",label
         plot!(N, mean(dt2,2),label="base")
```

```
In [29]: maxdiff = zeros(50,1)
         for i=1:50
             maxdiff[i], trash = findmax(diffSols[i,:])
         end
         using Plots
         plot(N, maxdiff, xlabel= "matrix size", ylabel="Difference", title="Max different between base
```

```
In [12]: """
         `matmul2`
         ========

         Compute Matrix-matrix multiplication or MatMul faster

         Functions
         ---------
         - `C = matmul2(A,B)` returns the matrix-matrix product of C=A*B
         """
         function matmul2(A,B)
             A_rows, A_cols = size(A)
             B_rows, B_cols = size(B)
             C = zeros(A_rows, B_cols)
```

```
    nAr= div(A_rows, 16)
    nBc= div(B_cols, 16)
    NAc = div(A_cols, 16)
    for ii=1:nAr
        I = (16*(ii-1)+1):(16(ii))
        for jj=1:nBc
            J = (16*(jj-1)+1):(16(jj))
            for kk=1:NAc
                K = (16*(kk-1)+1):(16(kk))
                C[I,J] += A[I,K]*B[K,J]
            end
        end
    end

    return C
end
```

```
Out[12]: matmul2 (generic function with 1 method)
```

In [16]:

```
nMats = 10
nSamples = 50

N = linspace(16, 1600, nMats)
timediff = zeros(nMats, nSamples)
dt = zeros(nMats, nSamples)
dt2 = zeros(nMats, nSamples)

for i=1:nMats
    M = round(Int,N[i])
    for j=1:nSamples
        A = randn(M, M)
        B = randn(M, M)
        dt[i,j] = @elapsed C = matmul(A,B);
        dt2[i,j] = @elapsed D = matmul2(A,B);
    end
end
```

In [23]:

```
using Plots
plot(N, mean(dt,2), xlabel= "matrix size", ylabel="Time", title="Average time for matmul",label
plot!(N, mean(dt2,2),label="matmul2")
savefig("problem2partD")
```

In [ ]:

Problem 3: Make Yoda Spin!

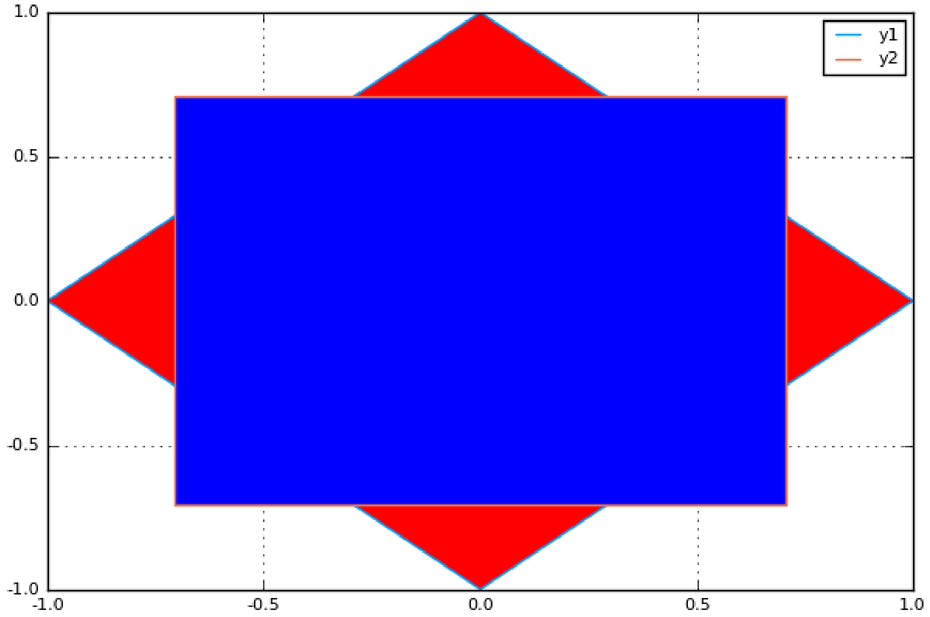
1. G&C Chapter 2 Problem 11. The Julia code for this example is

```
# create matrix whose columns contain the coordinates of
# each vertex.
U = [1.0 0 -1 0 1.0; 0 1 0 -1 0]

theta = pi/4.0

# Create a red unit square
# Note U(1,:) denotes the first row of U
plot(U[1,:]',U[2,:]',fill=(0,:red))

# Perform rotation.
R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
V = R*U;
# Plot the blue square
plot!(V[1,:]', V[2,:]',fill=(0,:blue))
```

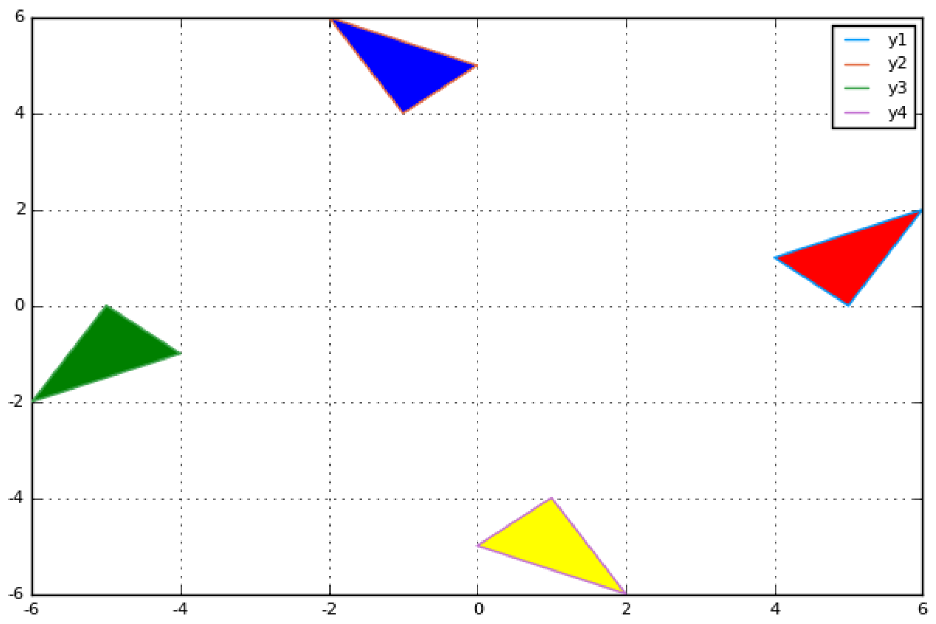
2. G&C Chapter 2 Problem 12 (not assigned)

```
**I really wanted to assign this one, but I still haven't
figured out how to translate the Matlab code into Julia
code. This one might get assigned in the future.**
```

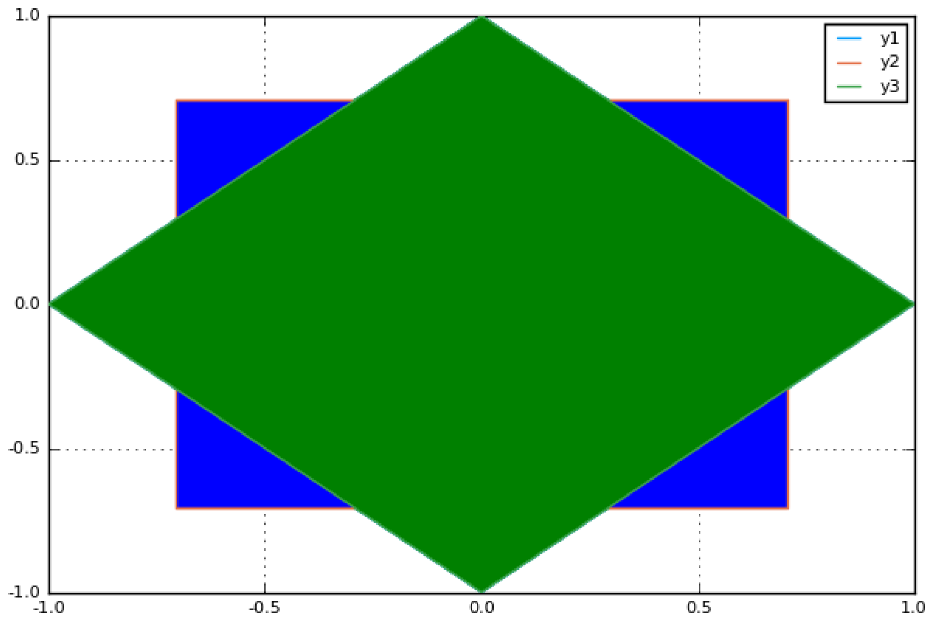
Solution



1.



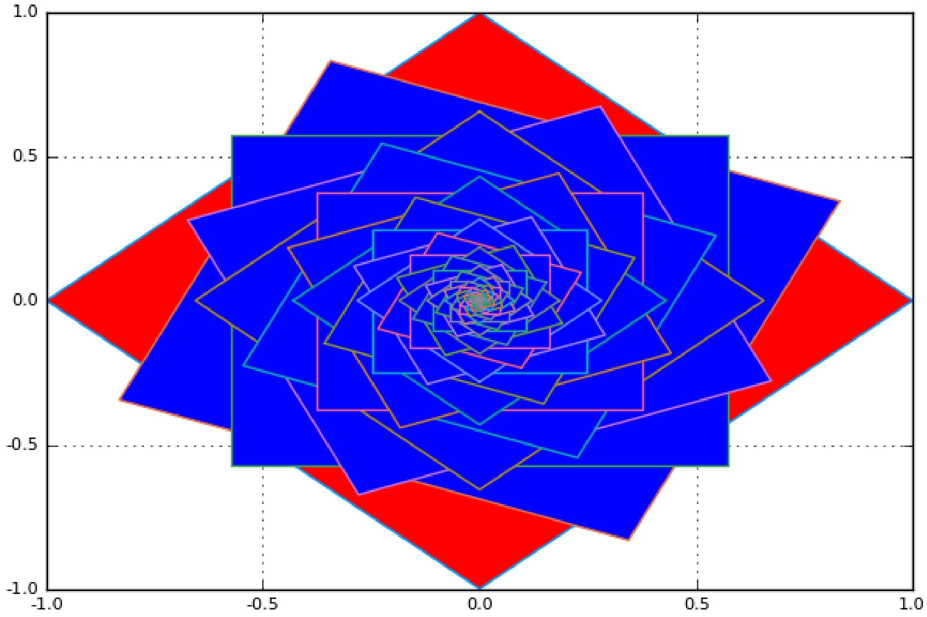
2.



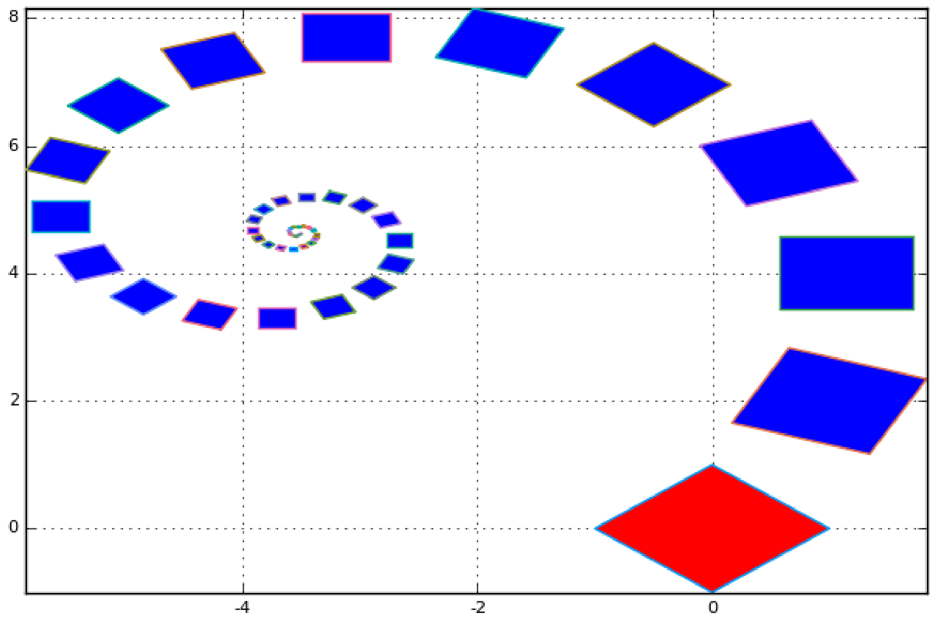
3.

$$R(\theta) * R(-\theta) = \begin{bmatrix} cos^2\theta + sin^2\theta & cos\theta sin\theta - cos\theta sin\theta \\ cos\theta sin\theta - cos\theta sin\theta & cos^2\theta + sin^2\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

4.



5.



Problem3

September 29, 2016

```
In [41]: using Plots
         # create matrix whose columns contain the coordinates of
         # each vertex.
         U = [1.0 0 -1 0 1.0; 0 1 0 -1 0]

         theta = pi/4.0

         # Create a red unit square
         # Note U(1,:) denotes the first row of U
         plot(U[1,:]',U[2,:]',fill=(0,:red))

         # Perform rotation.
         R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
         V = R*U;
         # Plot the blue square
         plot!(V[1,:]', V[2,:]',fill=(0,:blue))
         savefig("problem3")
```

```
In [42]: using Plots
         # create matrix whose columns contain the coordinates of
         # each vertex.
         U = [5 6 4 5; 0 2 1 0]

         theta = pi/2.0

         # Create a red triangle
         # Note U(1,:) denotes the first row of U
         plot(U[1,:]',U[2,:]',fill=(0,:red))

         # Perform rotation.
         R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
         V = R*U;
         # Plot the blue square
         plot!(V[1,:]', V[2,:]',fill=(0,:blue))

         # Perform rotation.
         W = R*V;
         # Plot the blue square
         plot!(W[1,:]', W[2,:]',fill=(0,:green))

         # Perform rotation.
         Z = R*W;
         # Plot the blue square
```

```
        plot!(Z[1,:]', Z[2,:]',fill=(0,:yellow))
        savefig("problem3partA")
```

```
In [43]: using Plots
         # create matrix whose columns contain the coordinates of
         # each vertex.
         U = [1.0 0 -1 0 1.0; 0 1 0 -1 0]

         theta = pi/4.0

         # Create a red unit square
         # Note U(1,:) denotes the first row of U
         plot(U[1,:]',U[2,:]',fill=(0,:red))


         # Perform rotation.
         R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
         V = R*U;
         # Plot the blue square
         plot!(V[1,:]', V[2,:]',fill=(0,:blue))

         # Perform counter rotation.
         R = [cos(-theta) -sin(-theta); sin(-theta) cos(-theta)];
         W = R*V;
         # Plot the blue square
         plot!(W[1,:]', W[2,:]',fill=(0,:green))
         savefig("problem3partB")
```

```
In [44]: using Plots
         # create matrix whose columns contain the coordinates of
         # each vertex.
         U = [1.0 0 -1 0 1.0; 0 1 0 -1 0]

         theta = pi/3.0

         # Create a red unit square
         # Note U(1,:) denotes the first row of U
         plot(U[1,:]',U[2,:]',fill=(0,:red))


         # Perform rotation.
         R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
         V = R*U;
         # Plot the blue square
         plot!(V[1,:]', V[2,:]',fill=(0,:blue))

         # Perform counter rotation.
         R = [cos(-theta) -sin(-theta); sin(-theta) cos(-theta)];
         W = R*V;
         # Plot the blue square
         plot!(W[1,:]', W[2,:]',fill=(0,:green))
         savefig("problem3partC")
```

```
In [17]: theta = pi/4.0
         @show [cos(theta) -sin(theta); sin(theta) cos(theta)] * [cos(-theta) -sin(-theta); sin(-theta)
         theta = pi/3.0
         @show [cos(theta) -sin(theta); sin(theta) cos(theta)] * [cos(-theta) -sin(-theta); sin(-theta)

[cos(theta) -(sin(theta));sin(theta) cos(theta)] * [cos(-theta) -(sin(-theta));sin(-theta) cos(-theta)]
 0.0 1.0]
[cos(theta) -(sin(theta));sin(theta) cos(theta)] * [cos(-theta) -(sin(-theta));sin(-theta) cos(-theta)]
 0.0 1.0]

Out[17]: 2x2 Array{Float64,2}:
          1.0  0.0
          0.0  1.0

In [45]: using Plots
         # create matrix whose columns contain the coordinates of
         # each vertex.
         U = [1.0 0 -1 0 1.0; 0 1 0 -1 0]

         theta = pi/8.0

         # Create a red unit square
         # Note U(1,:) denotes the first row of U
         plot(U[1,:]',U[2,:]',legend=false,fill=(0,:red))

         N=50
         for i=1:N
             # Perform rotation.
             R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
             U = 0.9*R*U;
             # Plot the blue square
             plot!(U[1,:]', U[2,:]',fill=(0,:blue))
         end
         savefig("problem3partC")

In [46]: using Plots
         using FileIO
         using ImageMagick


         # create matrix whose columns contain the coordinates of
         # each vertex.
         U = [1.0 0 -1 0 1.0; 0 1 0 -1 0]

         theta = pi/8.0

         # Create a red unit square
         # Note U(1,:) denotes the first row of U
         plot(U[1,:]',U[2,:]',legend=false, fill=(0,:red))

         N=50
         for i=1:N
             # Perform rotation.
             R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
             U = 0.9*R*U + [1 1 1 1 1; 2 2 2 2 2];
             # Plot the blue square
             plot!(U[1,:]', U[2,:]',fill=(0,:blue))
```

```
        end
        savefig("problem3partD")
```

```
In [ ]:
```

Problem 4: Make an image small

Consider the following problem. We have a 32×32 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 16×16 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 Julia program to do this

Let's work through a smaller example first. For the 4×4 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

$$\begin{bmatrix} (x_1 + x_2 + x_5 + x_6)/4 & (x_3 + x_4 + x_7 + x_8)/4 \\ (x_9 + x_{10} + x_{13} + x_{14})/4 & (x_{11} + x_{12} + x_{15} + x_{16})/4 \end{bmatrix}.$$

We will solve this problem using a matrix-vector product.

1. Suppose I call:

   $$y_1 = (x_1 + x_2 + x_5 + x_6)/4 \qquad y_2 = (x_3 + x_4 + x_7 + x_8)/4$$
   $$y_3 = (x_9 + x_{10} + x_{13} + x_{14})/4 \quad y_4 = (x_{11} + x_{12} + x_{15} + x_{16})/4.$$

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

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

2. Download http://www.cs.purdue.edu/homes/dgleich/cs314-2016/homeworks/smallicon.csv and type

   ```
   download("http://www.cs.purdue.edu/homes/dgleich/cs314-2016/homeworks/smallicon.csv","smallicon.csv
   X = readcsv("smallicon.csv")
   ```

   in Julia. You should get a 32×32 matrix $\boldsymbol{X}$. What is the sum of diagonal elements of $\boldsymbol{X}$?

3. Show the image

   ```
   using Images
   grayim(X)
   ```

   Or write it to a file

   ```
   using FileIO
   using ImageMagick
   save("icon1.png",X)
   ```

   But that picture doesn't look right, does it? To get full points, make sure you adjust the figure so that it looks correct.!

   Show your final code and the image you get.

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, in the small little example,

pixel (3,2) has linear index 10. Also, pixel (1,4) has index 4. Julia can help us built a map between pixel indices and linear or vector indices:

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

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

```
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:

```
N = reshape(1:(4*4), 4, 4)
```

instead?

5. Now we need to construct the matrix $\boldsymbol{A}$ in order to reduce the size of a 32×32 image to a 16×16 image as we did in part 1.
   Suppose we call the output vector $\mathbf{y}$ and the output image $\boldsymbol{Y}$. I'm giving you the following template, that I hope you can fill in. Feel free to construct an $\boldsymbol{A}$ that accomplishes our image reduction task any way you choose, but the following should provide some guidance.

```
NX = <fill in> # the map between pixel indices and linear indices for X
NY = <fill in> # the map between pixel indices and linear indices for Y
A = <fill in>  # the matrix we are trying to build
for i=1:32
    for j=1:32
        xi = <fill in> # the index of the pixel i,j in the vector x
        yi = <fill in> # hint, div(i,k) is integer division!

        A[yi,xi] = 1/4 # place the entry of the matrix
    end
end
```

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

```
x = reshape(X',32*32,1)
```

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

```
y = A*x
Y = reshape(y,16,16)'
```

   Show the image of $\boldsymbol{Y}$. Does that look correct?
   (Hint, apply the same correction you did before!)

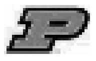
Solution

1.

2.

$$24.36862745098039$$

3.



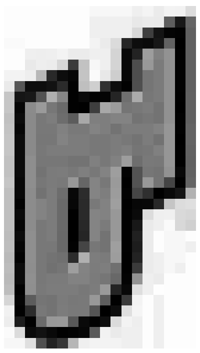
4. Reshape will take the vector from 1-16 and place it into a 4x4 matrix by going down the columsn. If you ignore the transpose the first row would be 1 5 9 13 instead of 1 2 3 4.

5.

Problem4

September 30, 2016

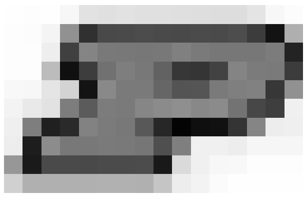
```
In [19]: using DataFrames
         #download("http://www.cs.purdue.edu/homes/dgleich/cs314-2016/homeworks/smallicon.csv","smallic
         X = readcsv("smallicon.csv");
         sum(diag(X))

Out[19]: 24.36862745098039

In [6]: using Images
        grayim(X)

Out[6]:
```



```
In [10]: using Images
         using FileIO
         using ImageMagick
         grayim(X')
         save("problem4partA.png", grayim(X'))

In [26]: N = reshape(1:(4*4), 4, 4)'

Out[26]: 4x4 Array{Int64,2}:
           1   2   3   4
           5   6   7   8
           9  10  11  12
          13  14  15  16

In [24]: N = reshape(1:(4*4), 4, 4)

Out[24]: 4x4 Array{Int64,2}:
          1  5   9  13
          2  6  10  14
          3  7  11  15
          4  8  12  16
```

```
In [18]: MY = Dict(1 => [1 2 33 34])
         k = 1;
         for i=2:(16*16)
             if MY[i-1][2] == k*32
                 MY[i] = MY[i-1] +  34
                 k+=2
             else
                 MY[i] = MY[i-1] + 2
             end
         end

         A = zeros(16*16,32*32)

         for j=1:(16*16)
             ind = MY[j]
             A[j, ind] = 1/4
         end
         x = reshape(X',32*32,1)
         y = A*x
         Y = reshape(y,16,16)'

         using Images
         grayim(Y')
```

Out[18]:



```
In [20]: using Images
         using FileIO
         using ImageMagick
         save("problem4partD.png", grayim(Y'))

In [ ]:
```