# Homework 3: QR and Eigenproblems

CS 205A: Mathematical Methods for Robotics, Vision, and Graphics (Spring 2017)
Stanford University
Due Thursday, May 4, 11:59pm

**Textbook problems:** 5.11 (35 points), 6.10 (30 points)
**Programming Assignment:** SVD-based image compression (35 points)

In [ ]:
```
using Images # read and show images, more info here: https://github.com/JuliaImages/Images.jl
using JLD # used for File I/O, more info here: https://github.com/JuliaIO/JLD.jl
```

# Julia Programming Assignment (35 pts): ¶

In this programming assignment we will implement Singular Value Decomposition (SVD) for image compression. We will analyze the compression ratio and error of our implementation.

First, we load a 720 by 1280 jpeg image and convert it to gray scale.

In [ ]:
```
img = load("stanford_1280x720.jpg");
display(Gray.(img))
```

In [ ]:
```
# Get image as a raw matrix
raw = channelview(Gray.(img[:,:]));
```

# SVD compression (10 pts)

Let M be the matrix with the pixel informations stored as above, define the singular-value compression factor as

$$CompFact_{SVD} = \frac{n}{k}$$

where n is the number of singular values of M and k is the k largest singular values we use for compression.

Create a vanilla compression function to compress an image using SVD. For example, with a factor of 10, we only keep 1/10 of the singular values. Fille in the code portion below:

In [ ]:
```
# Arguments:
#   image: input image
#   factor: svd compression factor
# Output:
#   a recovered image after svd compression
function compress(image, factor)
    U,S,V=svd(image[:,:], thin=true);
    ###### Your Code Here ######

    ###### End of code ######
end
```

Create an array of images with svd compression factors of 360, 90, 20, and 10

In [ ]:
```
svd_compression_factors = [360, 90, 20, 10];
images = [compress(raw, svd_compression_factors[i]) for i=1:4];
```

Once your implementation is complete, run the following script and briefly explain your observation (2-3 sentences)

In [ ]:
```
for i=1:4
    display(Gray.(imresize(images, (180,320))))
end
```

In [ ]:
```
###### Your explanation ######

###### End of explanation ######
```

# Relative Compression Error (5 pts)

Define relative compression error to be:

$$\frac{vecnorm(M_{SVD} - M_{original})}{vecnorm(M_{original})}$$

Fill in the code below to compute the relative compression error.

In [ ]:
```
function compress_err(original_image, compressed_image)
    ###### Your Code Here ######

    ###### End of code ######
end
```

Plot relative compression error for svd compression factors between 2 and 20

In [ ]:
```
using PyPlot
N=linspace(2,20,19);
plot(N,map(x->compress_err(raw, compress(raw, x)), N));
xlabel("SVD Compression Factor")
ylabel("Relative Compression Error")
```

# Memory Compression Ratio (10 pts)

As you might have noticed above, the SVD decomposition requires storage of three matrices: U (left-singular vectors), S (diagonal matrix of the singular values), and V (right-singular vectors). Define the memory compression ratio as

$$CompFac_{memory} = \frac{\text{size of M}}{\text{size of SVD}}$$

Implement the following function that computes memory compression ratio (Hint: the matrix S is stored as an 1-D array)

In [ ]:
```
# Arguments:
#   image: input image
#   factor: svd compression factor
# Output:
#   memory compression ratio
function compute_memory_compression_ratio(image, factor)
    ###### Your Code Here ######

    ###### End of code ######
end
```

Plot relative error vs memory compression factor for svd compression factors between 2 and 20

In [ ]:
```
plot(map(x->compute_memory_compression_ratio(raw, x), N),map(x->compre
ss_err(raw, compress(raw, x)), N));
xlabel("Memory Compression Ratio")
ylabel("Relative Compression Error")
```

# Block Compression (10 pts)

One optimization we can do is to divide the original matrix into n by n blocks and implement compression using a threshold. The steps are as the following:

1. We divide the matrix into smaller blocks
2. For each block, we do a SVD factorization and discard singular values < epsilon for some small positive epsilon.

Implement the block version, and plot relative error vs memory compression ratio for block sizes of 20 by 20, 40 by 40 and 80 by 80. (Hint: you can reuse your code above)

In [ ]:
```
# Arguments:
#   image: input image
#   block_size: divide images into block_size by block_size blocks
#   threshold: singular value threshold
# Output
#   a recovered image after block svd compression
function block_compress(image, block_size, threshold)
    ###### Your Code Here ######

    ###### End of code ######
end

# Arguments:
#   image: input image
#   block_size: divide images into block_size by block_size blocks
#   threshold: singular value threshold
# Output:
#   memory compression ratio of block compression
function compute_block_memory_compression_ratio(image, block_size, thr
eshold)
    ###### Your Code Here ######

    ###### End of code ######
end
```

Plot relative error vs memory compression factor for svd compression factors between 2 and 20 for non-blocked version and threshold between 0.1 and sqrt(10) for block sizes of 20, 40, and 80. You should see improvements over the non-blocked version.

In [ ]:
```
N=2:20;
t=logspace(-1,0.5,10);
plot(map(x->compute_memory_compression_ratio(raw, x), N),map(x->compre
ss_err(raw, compress(raw, x)), N), label="Non-block");
plot(map(x->compute_block_memory_compression_ratio(raw, 40, x), t),map
(x->compress_err(raw, block_compress(raw, 20, x)), t), label="B_size=4
0");
plot(map(x->compute_block_memory_compression_ratio(raw, 40, x), t),map
(x->compress_err(raw, block_compress(raw, 40, x)), t), label="B_size=4
0");
plot(map(x->compute_block_memory_compression_ratio(raw, 80, x), t),map
(x->compress_err(raw, block_compress(raw, 80, x)), t), label="B_size=8
0");
legend();
xlabel("Memory Compression Ratio")
ylabel("Relative Compression Error")
```

## Bonus (5 pts)

Suppose we want to limit our relative error to be at most 10%, find the block size that gives the best memory compression factor. You may implement this with different compression factors for each block. Summarize your implementation and display the resulting image.

In [ ]:
```
###### Your Code Here ######

###### End of code ######
```