# Homework 2

Thursday, September 3, 2015 9:30 AM

Due in class September $10^{th}$

Note: For full credit, show your work!
You are welcome to discuss the problems with others, but write up your own solutions.

1. Which of the following sets span $\mathbb{R}^3$?
   - (a) $\{(1,1,1)\}$
   - (b) $\{(1,0,0),(0,0,1),(1,0,1)\}$
   - (c) $\{(1,0,0),(0,1,0),(0,0,1),(1,1,1)\}$
   - (d) $\{(1,2,1),(2,0,-1),(4,4,1)\}$
   - (e) $\{(1,2,1),(2,0,-1),(4,4,0)\}$

2. What is the span of the union $\{3\times 3 \text{ symmetric matrices, i.e. } A=A^T\} \cup \{3\times 3 \text{ upper-triangular matrices}\}$?

3. Let $S=\{s_1,\dots,s_r\}$, $T=\{s_1,\dots,s_r,s_{r+1}\}$ be two sets of vectors from the same vector space.
   When is $\mathrm{Span}(S)=\mathrm{Span}(T)$?

4. <u>How fast is Matlab?</u>
   A useful skill is to be able to estimate how long a calculation is going to take before starting it. (For example, you can determine how large a problem you can solve without buying a new computer!)

a) Matrix multiplication:

Here is some crude Matlab/Octave code for timing the multiplication of two random $1000 \times 1000$ matrices:

```
% time to multiply two random 1000x1000 matrices
n = 1000;
A = randn(n, n);
B = randn(n, n);
starttime = cputime;
C = A * B;
endtime = cputime;
elapsedtime = endtime - starttime     <- CPU time in seconds

% same code, but averaged over 10 trials
trials = 10;                (to reduce noise)

n = 1000;
A = randn(n, n);
B = randn(n, n);

starttime = cputime;
for i = 1:trials
  C = A * B;
end
endtime = cputime;

averageelapsedtime = (endtime - starttime)/trials
```

<u>Your problem</u>: Estimate the <u>scaling</u> of the running time for matrix multiplication, as $n$ increases. That is, if the running time is $\approx c \cdot n^\alpha$ for some exponent $\alpha$ and constant $c$, estimate values for $\alpha$ and $c$.

(Don't worry about being too precise, but try to get something reasonable. For example, if $\alpha = 3$, then running the above code with $n=2000$ should take $8 = 2^\alpha$ times as long. Show your work!)

Based on your estimates for $c$ and $\alpha$, for how large an $n$ can your computer multiply two random $n \times n$ matrices in one day (24 hours)?

(Again, don't worry about being too precise, or about what happens when your computer runs out of memory.)

b) Solving a system of linear equations:

Repeat part a, but for solving n random linear equations in n variables.

c) Solving a sparse system of linear equations:

Repeat part a, but for solving

$$A\vec{x} = \vec{b},$$

where $\vec{b}$ is a random vector of length n, and A is an n×n matrix with 10n random nonzero entries in random positions.

This code might be helpful:

```
% code for generating a random n x n matrix with one random non-zero
  entry in a random position
n = 1000;
A = sparse(n, n);        % create an all-0 sparse matrix
i = randi(n);            % randi(n) returns a random integer from 1 to n
j = randi(n);
A(i, j) = randn(1, 1);   % set the i,j entry of A to a random value
```

⑤ A blurry camera:

Let's pretend you have a 64×64 pixel grayscale camera. As if that isn't bad enough, the camera's sensor is blurry; some of the light from each pixel spreads into the neighboring pixels.

More precisely, let $I_{x,y}$ be the amount of light that hits pixel $(x,y)$. Here x and y are both from 1 to 64.

The camera records

$$C_{x,y} = \frac{1}{2} \times I_{x,y} + \frac{1}{8} \times \left( \begin{matrix} I_{x-1,y} + I_{x+1,y} \\ + I_{x,y-1} + I_{x,y+1} \end{matrix} \right).$$

That is, it gets 1/2 times what you want, but also 1/8 times the light from the neighboring pixels.

(**Note**: If $(x,y)$ is on the boundary, then there will be fewer than four neighboring pixels. For example,

$$C_{1,1} = 1/2 \times I_{1,1} + 1/8 \times (I_{1,2} + I_{2,1})$$

because the corner pixel (1,1) has only two neighbors

$$C_{1,1} = \tfrac{1}{2} \times I_{1,1} + \tfrac{1}{8} \times (I_{1,2} + I_{2,1})$$

(because the corner pixel (1, 1) has only two neighbors.)

Here's a picture that your camera took:

By setting up and solving $64^2$ equations in the $64^2$ variables $I_{1,1}, \ldots, I_{64,64}$, **recover the correct image $I$.**

**Technical notes:**

The blurred image file is available as "blurryimage.mat". To load it, run the commands

```
load('blurryimage.mat');
imshow(blurryimage);
```

this should display the blurry image

To turn it into a vector of length $n^2$, you can use

```
b = reshape(blurryimage, n^2, 1);
```

where $n = 64$.

Then run the following code:

```
A = sparse(n^2, n^2);
%% fill in here commands to set up the sparse matrix A,
   each row containing an equation for one of the pixels
x = A \ b;
recoveredimage = reshape(x, n, n);
imshow(recoveredimage);
save('recoveredimage.mat', 'recoveredimage');
```

As in class, the following function, for converting a pair of indices $(x, y)$ to one index from 1 to $n^2$, should be helpful:

```
% returns an integer from 1 to n^2; x, y should be
  from 1 to n
function j = xytoj(x, y, n)
  j = 1 + (x-1) + n * (y-1);
end
```

<u>Deliverable</u>: Turn in all your code and the recoveredimage.mat file (by email to the TA).