Homework 1

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

① For the matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 1 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \end{pmatrix}$$

determine A^{300}.

First do the calculation in Matlab/python. Then do the same calculation by hand, justifying why the answer is correct.

② a) Give a 2×2 matrix A that transforms

$(1,0)$ to (a,c)
and $(0,1)$ to (b,d).

(That is, $A\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}a\\c\end{pmatrix}$ and $A\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}b\\d\end{pmatrix}$.)

Assuming $ad \neq bc$, give a 2×2 matrix that transforms

(a,c) to $(1,0)$
and (b,d) to $(0,1)$.

b) Give a 3×3 matrix A that transforms

$(1,1,1)$ to $(1,0,0)$,
$(2,-1,0)$ to $(0,1,0)$
and $(0,3,3)$ to $(0,0,1)$.

(Feel free to use a computer, if that helps.)

c) Give a 3×3 matrix B that transforms

$(1,1,1)$ to $(5,10,-1)$
$(2,-1,0)$ to $(15,0,0)$
$(0,3,3)$ to $(-1,-1,1)$.

(Hint: Use part ⓑ!)

③ Sketch the three lines

$x + 2y = 2$

$$x - y = 2$$
$$y = 1$$

Are the equations solvable?
What happens if the right-hand sides are all 0?
Is there any nonzero choice for the right-hand sides that allows the three lines to intersect at the same point?

④ Describe the intersection of the three planes

$$u + v + w + z = 6,$$

$$u + w + z = 2, \text{ and}$$

$$u + w = 2,$$

all in four-dimensional space. Is it a line or a point or an empty set?

What is the intersection if the fourth plane $u = -1$ is included? Find a fourth equation that leaves us with no solution.

⑤ **How fast is Matlab/numpy?**

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 1000×1000 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

Python

```
import time
import numpy as np

trials = 10

n = 1000
A = np.random.randn(n, n)
```

```
A = randn(n, n);
B = randn(n, n);
starttime = cputime;
C = A * B;
endtime = cputime;
elapsedtime = endtime - starttime

% same code, but averaged over 10 trials
trials = 10;

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
```

CPU time in seconds

(to reduce noise)

```
import time
import numpy as np

trials = 10

n = 1000
A = np.random.randn(n, n)
B = np.random.randn(n, n)

start = time.clock()
for i in range(trials):
  C = A * B
end = time.clock()

averageelapsedtime = (end - start) / trials
averageelapsedtime
```

No! C = A.dot(B)

Your problem: Estimate the scaling 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 α and constant c, estimate values for α 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 $\beta = 2^\alpha$ times as long. Show your work!)

Based on your estimates for c and α, 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:

Matlab

```
% 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
```

python

```
import numpy as np
import scipy.sparse

n = 1000
A = scipy.sparse.lil_matrix((n,n))  # list-of-lists sparse format for
                                    #  incrementally building A
i = np.random.randint(n)
j = np.random.randint(n)
A[i,j] = np.random.randn()
# ...

# after creating the matrix, convert it to CSR format, for speed
A = A.tocsr()  # a faster sparse matrix format
```