Homework 1 answers

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. Then do the same calculation by hand, justifying why the answer is correct.

Answer:

```
>> format rat
>> n = 3; A = [eye(n) ones(n,n)/n ; zeros(n,n) ones(n,n)/n]

A =

       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

>> A^300

ans =

       1              0              0             100            100            100
       0              1              0             100            100            100
       0              0              1             100            100            100
       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
```

Python

```python
import numpy as np
A = np.block([[np.eye(3),        np.ones((3,3))/3],
              [np.zeros((3,3)), np.ones((3,3))/3]])
np.linalg.matrix_power(A, 300)
```

Why? A is a block matrix:

$$A = \begin{pmatrix} I & B \\ 0 & B \end{pmatrix}$$ where I is the 3×3 identity matrix, $B = \frac{1}{3}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$.

$$\Rightarrow A^2 = \begin{pmatrix} I & I\cdot B + B^2 \\ 0 & B^2 \end{pmatrix}$$ Since $B^2 = B$, $A^2 = \begin{pmatrix} I & 2B \\ 0 & B \end{pmatrix}$.

Now let us prove by induction:

Claim: For $k = 1, 2, 3, \ldots,$ $A^k = \begin{pmatrix} I & kB \\ 0 & B \end{pmatrix}$.

Proof:

Base case $k=1$: ✓

Induction step: Assume $A^{k-1} = \begin{pmatrix} I & (k-1)B \\ 0 & B \end{pmatrix}$.

$$\begin{pmatrix} I & (k-1)B \end{pmatrix}$$

Induction step: Assume $A^{k-1} = \begin{pmatrix} I & (k-1)B \\ 0 & B \end{pmatrix}$.

$$\Rightarrow A^k = A \cdot A^{k-1} = \begin{pmatrix} I & B \\ 0 & B \end{pmatrix}\begin{pmatrix} I & (k-1)B \\ 0 & B \end{pmatrix}$$

$$= \begin{pmatrix} I & (k-1)B + B^2 \\ 0 & B^2 \end{pmatrix} = \begin{pmatrix} I & kB \\ 0 & B \end{pmatrix} \checkmark \square$$

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 ⓑ!)

a) $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$

$$A^{-1} = \frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

b) $A^{-1} = \begin{pmatrix} 1 & 2 & 0 \\ 1 & -1 & 3 \\ 1 & 0 & 3 \end{pmatrix} \Rightarrow A = \begin{pmatrix} 1 & 2 & -2 \\ 0 & -1 & 1 \\ -1/3 & -4/3 & 1 \end{pmatrix}$

c) $\begin{pmatrix} 5 & 15 & -1 \\ 10 & 0 & -1 \\ -1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 2 & -2 \\ 0 & -1 & 1 \\ -1/3 & -2/3 & 1 \end{pmatrix} = \begin{pmatrix} 16/3 & -13/3 & 4 \\ 31/3 & 62/3 & -21 \\ -4/3 & -8/3 & 3 \end{pmatrix}$

③ Sketch the three lines

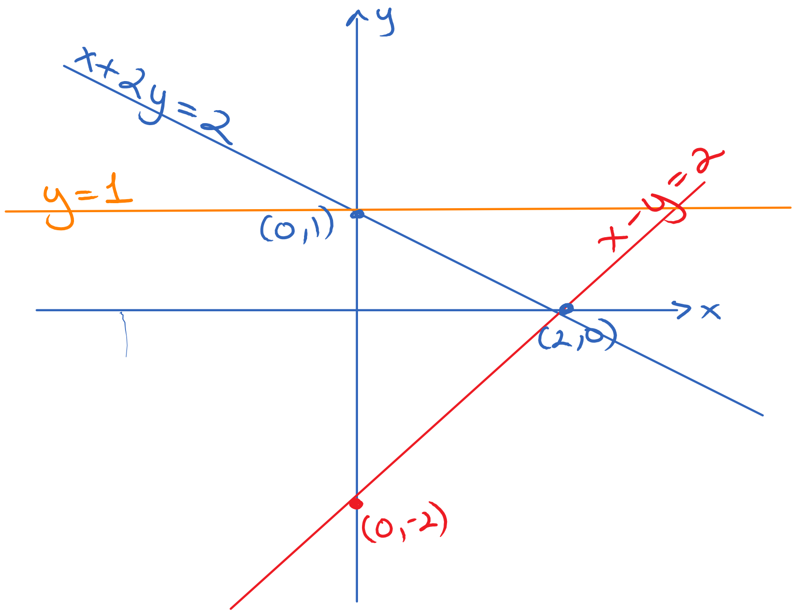
③ Sketch the three lines

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

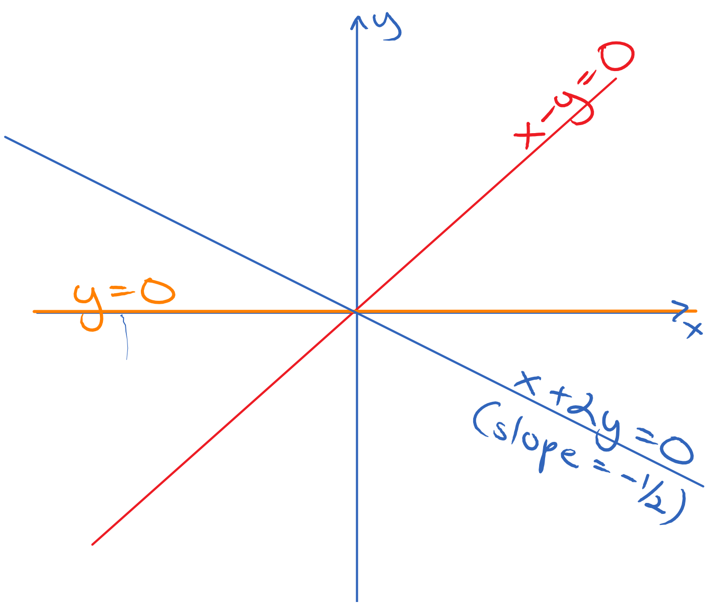
Are the equations solvable?

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?



Since the three lines don't intersect, the equations are not solvable.



Now there is a unique solution $x = y = 0$

($\vec{0}$ solves any homogeneous system of equations)

④ Describe the intersection of the three planes

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

$u+w+z=2$, and

$u+w=2$,

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

Answer: Use Gaussian elimination to find their intersection:

$$\begin{array}{c} -1 \\ -1 \end{array}\left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 6 \\ 1 & 0 & 1 & 1 & 2 \\ 1 & 0 & 1 & 0 & 2 \end{array}\right) \longrightarrow \left(\begin{array}{cccc|c} 0 & 1 & 0 & 0 & 4 \\ 1 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right)$$

$\Rightarrow$ the solution is

$v=4$, $z=0$

$u=2-w$, where w is free

This is a line

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

If $u=-1$ is added, the solution is a point:

$$\boxed{(u,v,w,z)=(-1,4,3,0)}$$

$u+w=10$ will give no solution, for example

⑤ **How fast is Matlab?**

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

```
% 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
```

Python

```python
import time
import numpy as np

n = 2000
trials = 10

totaltime = 0
for _ in range(trials):
  A = np.random.randn(n, n)
  B = np.random.randn(n, n)
  start = time.perf_counter()
  C = A.dot(B)   # or np.dot(A,B)
                 # NOT A * B!
  totaltime += time.perf_counter() - start

averagetime = totaltime / trials
averagetime
```

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 $8 = 2^{\alpha}$ 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.)

Answer: There are lots of ways of doing this, to find c and α to fit observed running times. We'll talk more about fitting curves later.

One simple solution to find α is to simply take the ratio of the average CPU time for n=1000 and n=2000, ie.,

$$\frac{t_{2000}}{t_{1000}} = \frac{\cancel{c}\cdot 2000^\alpha}{\cancel{c}\cdot 1000^\alpha} = 2^\alpha$$

$$\Rightarrow \alpha = \log_2\left(\frac{t_{2000}}{t_{1000}}\right)$$

$$c = \frac{t_{1000}}{1000^\alpha}$$

Since 24 hours = 86,400 seconds, solving

$$86400 = c \cdot n^\alpha$$

$$\Rightarrow n = \left(\frac{86400}{c}\right)^{1/\alpha}$$

My Matlab code:

```
n = 1000;
totaltime = 0;
trials = 10;
for i = 1:trials
  A = randn(n, n);
  B = randn(n, n);
  starttime = cputime;
  C = A * B;
  endtime = cputime;
  totaltime = totaltime + (endtime - starttime);
end
t1000 = totaltime / trials
```

```
n = 2000;
totaltime = 0;
trials = 10;
for i = 1:trials
  A = randn(n, n);
  B = randn(n, n);
  starttime = cputime;
  C = A * B;
  endtime = cputime;
  totaltime = totaltime + (endtime - starttime);
end
t2000 = totaltime / trials
```

```
alpha = log2(t2000/t1000)
c = t1000 / 1000^alpha
n24 = (86400 / c)^(1/alpha)
```

```
alpha =

   2.9069


c =

   2.3211e-10


n24 =

   1.0292e+05
```

$\approx 100{,}000$

Your answers will vary!

b) Solving a system of linear equations:

b) Solving a system of linear equations:
Repeat part a, but for solving n random linear equations in n variables.

Answer: Just as above, heres my code:

```
n = 1000;
totaltime = 0;
trials = 10;
for i = 1:trials
  A = randn(n, n);
  b = randn(n, 1);
  starttime = cputime;
  x = A \ b;
  endtime = cputime;
  totaltime = totaltime + (endtime - starttime);
end
t1000 = totaltime / trials
```

```
n = 2000;
totaltime = 0;
trials = 10;
for i = 1:trials
  A = randn(n, n);
  b = randn(n, 1);
  starttime = cputime;
  x = A \ b;
  endtime = cputime;
  totaltime = totaltime + (endtime - starttime);
end
t2000 = totaltime / trials
```

```
alpha = log2(t2000/t1000)
c = t1000 / 1000^alpha
n24 = (86400 / c)^(1/alpha)
```

```
alpha =

    2.4829


c =

   2.9529e-09


n24 =

   2.6516e+05
```

≈ 265,000

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
```

Answer: As before,

```
n = 1000;
numnonzeroentries = 10 * n;
totaltime = 0;
trials = 30;
for i = 1:trials
  A = sparse(n, n);
  for k = 1:numnonzeroentries
    i = randi(n);   j = randi(n);
    A(i, j) = randn(1,1);
  end
  b = randn(n, 1);
  starttime = cputime;
  x = A \ b;
  endtime = cputime;
  totaltime = totaltime + (endtime - starttime);
end
t1000 = totaltime / trials
```

```
n = 2000;
numnonzeroentries = 10 * n;
totaltime = 0;
trials = 30;
for i = 1:trials
  A = sparse(n, n);
  for k = 1:numnonzeroentries
    i = randi(n);   j = randi(n);
    A(i, j) = randn(1,1);
  end
  b = randn(n, 1);
  starttime = cputime;
  x = A \ b;
  endtime = cputime;
  totaltime = totaltime + (endtime - starttime);
end
t2000 = totaltime / trials
```

```
alpha = log2(t2000/t1000)
c = t1000 / 1000^alpha
n24 = (86400 / c)^(1/alpha)
```

```
alpha =

    2.6274


c =

   8.3090e-10


n24 =

   2.1627e+05
```

≈ 220,000

You may have noticed that the running time is highly variable. Running for more trials would help a little bit, but probably a different problem would give a more useful timing result.

Python:

a

```
import time
import numpy as np

n = 2000
trials = 10

totaltime = 0
for _ in range(trials):
  A = np.random.randn(n, n)
  B = np.random.randn(n, n)
  start = time.perf_counter()
  C = A.dot(B)  # or np.dot(A,B)
                # NOT A * B!
  totaltime += time.perf_counter() - start

averagetime = totaltime / trials
averagetime
```

b

```
import time
import numpy as np

n = 1000
trials = 10

totaltime = 0
for _ in range(trials):
  A = np.random.randn(n, n)
  x = np.random.randn(n)
  b = A.dot(x)
  start = time.perf_counter()
  np.linalg.solve(A, b)
  totaltime += time.perf_counter() - start

averagetime = totaltime / trials
averagetime
```

c

```
import time
import numpy as np
import scipy.sparse
from scipy.sparse.linalg import spsolve, bicg

n = 500
```

```python
import time
import numpy as np
import scipy.sparse
from scipy.sparse.linalg import spsolve, bicg

n = 500

total = 0
trials = 10
for _ in range(trials):
  sparsity = 10
  rows = np.random.randint(n, size=sparsity*n)
  cols = np.random.randint(n, size=sparsity*n)
  data = np.random.randn(sparsity*n)
  A = scipy.sparse.csr_matrix((data, (rows,cols)), shape=(n,n))
  #A += 1000* scipy.sparse.eye(n)  ## uncomment this to improve the condition no
  #A = A.toarray()                 ## uncomment this to use dense matrix

  x = np.random.randn(n,1)
  b = A.dot(x)
  start = time.perf_counter()
  c = bicg(A,b)
  end = time.perf_counter()
  total += end - start
total
```