APM120, Homework #1

Applied Linear Algebra and Big Data

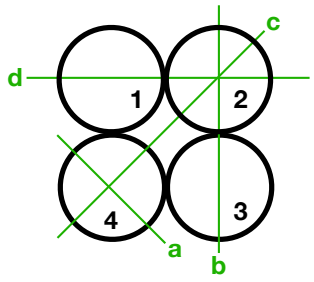
Last updated: Friday 27th January, 2023, 13:48

Assigned Jan 31, 2023, due Feb 7, 1:00 pm, via gradescope, **in pdf $\leq$ 20Mb, $\leq$ 30 pages.**

Linear equations

Show all steps in all calculations explicitly. Attach code used, well documented, and relevant plots and Matlab/python output, attaching code and figures *immediately following* the relevant question solution. A code printout is not a substitute for complete solutions, your solution should stand alone without the Matlab/python code or output. See needed python preliminaries at end of this HW[1]. It is fine to use Matlab/python unless a hand calculation (using only a simple calculator) is required *in orange*. For all questions, make sure you show all steps as if you are doing the problem by hand, and do not use library functions unless explicitly allowed in the question. Make sure you can do all calculations using no more than a hand-held calculator. See the end-note for guidelines and examples of hand-calculations.[2]

1. Medical tomography and linear equations. Consider the following 4 circular pixels representing a tissue, and the 4 lines representing X-rays through the tissue. Write the equations in matrix form for the density of the 4 pixels, for all drawn rays, assuming the rays go through the pixel centers so that $\Delta u = 1$. Given that the 4 rays a–d lead to intensity measurements of $I(s) = [1.78504; 0.398297; 0.624653; 0.462755]$, assume a decay coefficient $\mu = 1.5$ an initial intensity at the source of $I(0) = 8$, and solve for the 4 densities using Matlab/python. *A check:* the 3rd pixel density is 1.3.



2. Row and column geometric interpretations: Explain why the system of equations $\mathsf{A}\mathbf{x} = \mathbf{b}$ specified by the following matrix and RHS has no solutions, using **(A)** row interpretation; **(B)** column interpretations.

```
A=[ -4,  -7,  -12.4;          b=[ 1;
    8,  -7,  -17.2;               -1;
    2,  -1,  -2.8];               0];
```

3. Direct solution of $\mathsf{A}\mathbf{x} = \mathbf{b}$ using LU decomposition & matrix transformations. Consider

```
A=[ -2,  -1.33333,  0.666667;     b=[ 3;
    3,  -1,  0;                       9;
    6,  -5,  4];                      6];
```

   (a) Hand calculation: Derive the LU decomposition: write explicitly the transformation matrix L and the permutation matrix P at each step of the Gaussian

elimination, also deriving and writing the resulting U at each step. Exchange rows such that the current pivot is always the largest possible choice.

(b) Show that LU = PA: check your answer for L, U, and P using Matlab `[L,U,P]=lu(A)` or python `P,L,U=scipy.linalg.lu(A); P=P.T;` (note that python is solving for A = PLU, therefore the need to transpose P.)

(c) Hand calculation: Solve $\mathsf{L}\mathbf{c} = \mathsf{P}\mathbf{b}$ and then $\mathsf{U}\mathbf{x} = \mathbf{c}$; compare your solution for $\mathbf{x}$ to the ones obtained by Matlab or python using `x=A\b;` or `x=linalg.solve(A,b);`.

4. Iterative projection method (aka Kaczmarz method or algebraic reconstruction method): **(A)** Initialize the solution to $\mathsf{A}\mathbf{x} = \mathbf{b}$ with the zero vector. Perform the first set of iterations by hand. Perform another one by code or by hand, where each set corresponds to going over all equations. Start from the Kaczmarz_example.m/py demo. **(B)** Compare your solution after 1 and 2 sets of iterations to the exact one obtained by Matlab/python, by calculating the error which is the norm of the difference between the iterative solution and the exact solution. **(C)** Calculate the three angles between the different pairs of rows of the matrix A and discuss the convergence.

```
A=[ -9,  -5,  1;          b=[ 1;
    5,  -9,  -2;             -1;
    1,  -1,  10];            -1];
```

(D) ***Optional extra credit*** draw the three planes corresponding to the three equations (start from the code provided for the 3d geometric interpretation of linear equations) and superimpose the trajectory of the iterations in 3d.

5. **(A)** Write code to create a sparse 100×200 matrix A in Matlab/python from the following nonzero elements, then use the appropriate function to calculate the number of its nonzero elements:

i	30	43	25	13	81	62
j	27	84	142	54	9	108
a_{ij}	0.6	0.9	-0.5	2.8	1.8	-1.8

(B) Sparse matrices: run the following Matlab/python code, explain very briefly each of the commands, and discuss the resulting printout:

```
% Matlab
N=2150; rng(2023); A=rand(N,N);
A=(A'+A)/2; A(abs(A)<0.97)=0;
A(A~=0)=(A(A~=0)-mean(A(A~=0)))*1000;
Asp=sparse(A); nnz(Asp)
figure(1); clf;
plot(Asp(Asp~=0),'.')
tic; X=A*A*A*A*A*A+A; toc;
tic; Xsp=Asp*Asp*Asp*Asp*Asp*Asp+Asp; toc;
```

```
# python:
import time; from scipy import sparse
N=2150; np.random.seed(2023);
A=np.random.random((N,N)); A=(A.T+A)/2;
A[np.abs(A)<0.97]=0;
A[A!=0]=(A[A!=0]-np.mean(A[A!=0]))*1000;
Asp=scipy.sparse.coo_matrix(A);
print(np.count_nonzero(A));
plt.figure(1); plt.clf();
plt.plot(A[A!=0],'.',markersize=1);
t=time.time(); X=A@A@A@A@A+A;
elapsed=time.time()-t; print(elapsed);
t=time.time(); Xsp=Asp@Asp@Asp@Asp@Asp+A;
elapsed=time.time()-t; print(elapsed);
```

6. **(A)** Calculate the solution to the following equation with and without the added small "noise" $\delta\mathbf{b}$. **(B)** Explain the reason for the sensitivity to the noise term. **(C)** ***Optional extra credit*** plot the three planes and the intersection point.

```
A=[ 4.51383,  4.81534,  1.60189;   delta_b=[ -0.122189;    b=[ -6.70581;
    2.57472,  3.69404,  1.5812;               0.155624;        -4.52604;
    5.28217,  -0.091319,  -0.476303];         0.0291701];      -3.94298];
```

7. Solve the following system using Gaussian elimination with and without pivoting, assuming a 3-digit accuracy. Plot the two lines corresponding to the two equations, is the matrix near-singular? What is the source of the sensitivity to noise?

$$\mathsf{A} = \begin{pmatrix} 0.0003 & 1 \\ 3 & 1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}.$$

8. ***Optional extra credit*** **MapReduce:** read in the notes and attempt this:

   (a) Describe how you would apply the MapReduce algorithm to calculate the *number of mutual friends* of each possible pair of users in a social network using the MapReduce algorithm as implemented in Matlab/ python. Explain explicitly what is done in each of the three stages of MAP, SHUFFLE, REDUCE.

   (b) Modify the python or Matlab mutual friend-finding code on the course web page here; MapReduce in python requires an external utility, "Hadoop", see notes with installation and running instructions, and the python flight delay example given by the codes `find_mutual_friends_*` here to calculate the *number of mutual friends* of each possible pair of users.

* [What's the point of ***optional extra credit challenge problems: apart from the fun of doing them, they may bring the total score of this HW assignment up to 110%, making up for problems you may have missed in this or other HW assignments...]

Python preliminaries & notes

1 python commands within the HW assume you have first used the followings: `import numpy as np; from numpy import linalg; import scipy as scipy; from scipy import linalg; import matplotlib.pyplot as plt; import matplotlib;`
Input a matrix A, column vector **b** and row vector **c** into python in the form
`A=np.array([[a11,a12,a13],[a21,a22,a23],[a31,a32,a33]]; b=np.array([[b1],[b2],[b3]]); c=np.array([[c1,c2,c3]]);` or convert Matlab arrays given in HW directly to python arrays using, e.g., `A=np.array(np.matrixlib.defmatrix.matrix('[1 2 3; 4 5 6]'));`

2 **Hand calculations**, in which you are asked to use only a simple hand-held calculator, are required only when we want to make sure you understand exactly what each step of an algorithm is doing. These also prepare you for the quizzes that involve similar hand calculations. **How much work to show?** Just don't use scratch paper, show all steps that you actually use for the hand calculations, but no more. Trust the graders to be reasonable, they were students in the course last year. **Examples: (i)** If you are asked, *not* in orange, to calculate the LU decomposition of a matrix, you may use Matlab/python as in `A(2,:)=A(2,:)-A(1,:)*(A(2,1)/A(1,1))` etc, but you may not use a library routine as in `[L,U,P]=lu(A)` except for checking your results. **(ii)** If required to calculate by hand the element $C_{2,3}$ of a matrix product C = AB, you need to explicitly multiply using a hand calculator the second row of A with the third column of B. You may not use Matlab/python to calculate `C=A*B` and then take the needed element from that product, nor to calculate `C23=A(2,:)*B(:,3)`.