MTH 373/573:
Scientific Computing
Monsoon 2018

Homework 2

IIIT-Delhi
Kaushik Kalyanaraman

Due: September 6, 2018 by 11.55 pm **Total: 350 points.**

Before you start, please read General Submission Guidelines on Page 5.

Problem 1: Weighted norm of a vector (25 points)

Let A be a symmetric positive definite matrix, that is $A^T = A$ and $x^T A\, x > 0$ for all x. Show that $\|x\|_A = (x^T A\, x)^{1/2}$ satisfies all three properties of a vector norm as discussed in class.

Hint: You can assume (with or without proof) that A admits a decomposition of the form $A = BB^T$ where B is a nonsingular matrix. In addition, you may need to use Cauchy-Schwarz inequality.

Note 1: In contrast to the norm of a matrix induced from a vector norm, the above is referred to as a vector norm induced by a matrix!

Note 2: You should should not exhibit your proofs for *any* particular choice of vector or matrix sizes.

Problem 2: Transformation matrices (10 $\times$ 3 = 30 points)

Consider the vector

$$a = \begin{bmatrix} 3 \\ 5 \\ 12 \end{bmatrix}.$$

(a) Determine a 3×3 elementary (Gaussian) elimination matrix to eliminate the third component of a using its first component.

(b) Compute a 3×3 Householder transformation matrix that eliminates only the third component of a. Also specify α and the Householder vector v.

(c) Calculate a 3×3 Givens rotation matrix that annihilates the third component of a using its second component.

In each case, your answer should consist of:

- the 3×3 transformation matrix M,
- the product Ma,
- and any other information as requested.

Problem 3: Matrix inversion is stupid (25 points)

If A, B, and C are $n \times n$ matrices, with B and C nonsingular, and b is an n-vector, how will you implement the formula:

$$x = B^{-1}(2A + I)\,(C^{-1} + A)\,b$$

without explicitly computing any of the matrix inverses?

Problem 4: Gaussian elimination and partial pivoting

(30 + 30 + 25 × 3 + 15 = 150 points)

This is one of the few instances in this course where you will be asked to code an alogrithm. In general (and even in this problem), you will simply use a library function for solving linear systems.

(a) Write a Python function to implement Gaussian elimination with no pivoting. Your function definition should look like the following.

```
def ge_nop(A, b):
    # Your code for this function goes here
    return A, b
```

(b) Write a Python program to implement Gaussian elimination *with* partial pivoting.

```
def ge_pp(A, b):
    # Your code for this function goes here
    return A, b
```

(c) Test your functions using the following example matrices. Create each matrix below to be of size 100×100.

$\boldsymbol{A}_1$ a matrix with entries from the standard normal distribution (use `np.random.randn`)

$\boldsymbol{A}_2$ the matrix given by

$$(A_2)_{ij} = \begin{cases} 5 & (i+2) \bmod n = j, \\ 10^{-3} & \text{otherwise}, \end{cases} \qquad (i, j = 0, \ldots, n-1),$$

$\boldsymbol{A}_3$ the matrix given by

$$(A_3)_{ij} = \frac{1}{(1 + |(i+1) \bmod n - j|)^4}, \qquad (i, j = 0, \ldots, n-1).$$

For each of these matrices, generate three random vectors $x_i \in \mathbb{R}^n, i = 1, 2, 3$ as follows as your *true solutions*. Note that seeding the random number generator ensures "deterministic" random number generation!

```
np.random.seed(2018)
x_1, x_2, x_3 = np.random.rand(100, 3).T
```

Then, compute $b_i, i = 1, 2, 3$ as the matrix-vector product $b_i = A_i\, x_i, i = 1, 2, 3$ and solve each of the three resulting linear systems. Note that if you set up your matrices `A_1`, `A_2` and `A_3` as `np.array`'s, to perform the matrix-vector product, you will need to use `np.dot(A_1, x_1)`, for instance. `A_1 * x_1` will simply scale the columns of `A_1` by the entries in `x_1`. Try out both to see the differences yourself.

For each of your matrices in this part, report the following in a table using *2-norm* measurements for all vectors involved, and *relative errors* or *residuals* as appropriate.

(i) Condition number of the matrix (use `np.linalg.cond`)

(ii) Error in your un-pivoted solve using your function in part (a)

(iii) Residual from your un-pivoted solve

(iv) Error from your partially-pivoted solve in part (b)

(v) Residual from your partially-pivoted solve

(vi) Error from `np.linalg.solve`

(vii) Residual from `np.linalg.solve`

Also report if any of your solves fail. You should try to write your code so that it prints the above table automatically, without user interaction.

(d) For each of the three instances of the linear system solution, write one sentence each on what you observe with regard to conditioning of the matrix, need (or not) for pivoting, and reasons for your observed results. (The open-ended nature of this part is by design. I will carefully examine each of your solutions to ensure you are supplying your own thoughts.)

Python tips: You may find the following NumPy functions useful for this problem: `np.array`, `np.dot`, `np.random.rand`, `np.linalg.norm`, `np.roll`, `np.fill_diagonal`.

Concept note: This idea of first generating a solution and then using it to create a right hand side is referred to as the *method of manufactured solutions*.

Problem 5: Scaling a linear system (50 points)

Multiplying both sides of a linear system $A\,x = b$ by a nonsingular diagonal matrix D to obtain a new system $DA\,x = D\,b$ simply rescales rows of the system. In theory, this does not change the solution. Such scaling, however, *does* affect the condition number of the matrix and choice of pivots in Gaussian elimination. As a result, it may affect accuracy of the solution in finite-precision arithmetic.

Using a linear system with the following code to set up A:

```
n = 100
np.random.seed(1729)
A = np.random.randn(n, n)
```

Using `np.ones`, set up the solution vector $x = \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$, and generate the corresponding right hand side $b = A\,x$. Experiment with the following scaling matrices D to see what effect they have on the condition number of the matrix DA, and the solution given by `numpy.linalg.solve` for solving the linear system $DA\,x = D\,b$.

(i) No scaling, that is, $D = I$, the identity matrix. (Feel free to use your judgement to either implement D as I, or solve an unscaled linear system.)

(ii) `D = np.diag(2 * np.ones(n))`

(iii) `D = np.diag(np.linspace(1, 100, 100))`

(iv) `D = np.diag(np.linspace(1, 10000, 100))`

(v) `D = np.diag(2**-np.arange(-n//2, n//2, dtype=np.float64))`

Report the 2-norm *relative residuals*, the *relative error*, and the *condition number* given by the various scalings. Which of the these scalings gives a very poor accuracy? Is the residual still small in this case? How does the condition number correlate with your observations?

Problem 6: Perturbed linear systems

(10 + 25 + 25 + 10 = 70 points)

In this problem, we will computationally study the linear system sensitivity analysis that we performed in class. Let A be a matrix of size 512×512 which has 2 as each diagonal entry, and -1 as each superdiagonal and subdiagonal entry as shown below.

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}_{512 \times 512}$$

As with other problems on this HW, we will measure all vector norms and matrix condition numbers in the 2-norm.

(a) Set up matrix A and find its condition number using `np.linalg.cond`. You are encouraged to explore and use `np.diag` for setting up the matrix in your Python code.

(b) Setup 10 random *unit* vectors all of length 512. (Think about how you can make any given vector to be of unit length.) We will abuse notation a little bit and refer to each one of them as simply b. For each b, setup 10 sets of 10 random perturbation vectors δb with the following stipulation. Each set of 10 random vectors should have norms $0.01, 0.02, \ldots, 0.1$.

Sanity checks: There are a total of 100 perturbation vectors in this problem therefore. Make sure all your random perturbation vectors are all different. Think again about how you can do this without examing each 512 length vector which is impossible!

(c) Let x be the solution of $A\,x = b$, and let $x + \delta x$ the solution of $A(x + \delta x) = b + \delta b$. In other words, using `np.linalg.solve`, compute for each b the corresponding x and for each δb the corresponding δx by solving the appropriate linear systems. (*Sanity check:* You will thus have 100 $(b, \delta b, x, \delta x)$ tuples of 512 length vectors.)

For each b and δb pair, plot:

$$\left(\text{cond}(A)\, \frac{\|\delta b\|}{\|b\|}, \frac{\|\delta x\|}{\|x\|} \right).$$

Thus, you will be plotting 100 points. Use `'ko'` in your plot command `plt.plot(..., 'ko')` to ensure that only a black dot is printed for each point and various points are *not* joined by lines. You can use `plt.figure` to start your plotting. Note that $\text{cond}(A)\|\delta b\|/\|b\|$ is the x-component and $\|\delta x\|/\|x\|$ is the y-component of each data point above.

(d) Can there ever be a point on the plot for which the vertical coordinate (that is, $\|\delta x\|/\|x\|$) is larger than the horizontal coordinate (that is, $\text{cond}(A)\, \|\delta b\|/\|b\|$)? Why or why not? Provide a one or two sentence explanation.

Optional bonus challenge (100 points): Can you find a vector b and a perturbation vector δb such that $\|\delta x\|/\|x\|$ is *equal to* $\text{cond}(A)\, \|\delta b\|/\|b\|$?

Notes: This challenge is the first of several bonus problems you will encounter in this course. This is *not* part of the total points for this HW. You are *not* required to answer this problem unless you want these 100 extra points. You will *not* be able to find a solution for this online or anywhere else. Finally, I will *not* provide a solution to this problem when solutions are released. This is a *hard* problem!

General Submission Guidelines

- **This assignment should be answered individually.**
- **You will be penalized for copying or any plagiarism with an automatic zero.**
- **Start working on your solutions early and don't wait until close to due date.**
- The points for each problem is roughly indicative of effort needed for answering that problem. For instance, 50 points can be taken to mean about 50 minutes of *focussed work*. This can include reading and thinking if the problem is theoretical. For machine problems, this will include time spent in coding, debugging and producing output assuming some Python competency. Your mileage may vary!
- IIIT-Delhi academic policies on honesty and integrity apply to all HWs. This includes not copying from one another, from the internet, a book, or any other online or offline source. A repeat offense will be reported to academic administration.
- You can ask questions about concepts, ideas and basics of coding on the Piazza discussion group for the class. You can also ask the instructor questions during office hours. You should however provide your own solutions.
- Any technical question pertaining to the HW emailed to me will be ignored.
- If you discuss or read secondary sources (other than class notes), please list all your discussion partners and/or secondary sources in your writeup. Failure to do so will constitute violation of honor code.
- Each problem should have a clear and concise write-up.
- Please clearly show all steps involved in theoretical problems.
- The answer to all theoretical problems, and output, figures and analyses for computer problems should be submitted in a PDF file or turned in handwritten.
- Handwritten submissions can be turned in during class, in person in my office or under my office door before the submission deadline. I will be present in my office during and for 5 minutes after every submission deadline to collect submissions.
- No late submission will be allowed unless I explicitly permit it.
- All Python files for computer problems should be submitted through Backpack.
- However, if your Python code generates an output figure or table, please provide all such results in a single PDF file along with your code submission.
- You will need to write a separate Python code for each problem and sometimes for each subproblem as well. You should name each such file as `problem_`n`.py` where n is the problem number. For example, your files could be named `problem_3a.py`, `problem_3b.py` and `problem_4.py`.
- You should import Python modules as follows:

```
from __future__ import division
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import numpy.linalg as npla
import scipy.linalg as spla
```

Every code you write will have one or more of these import statements.