MTH 373/573:
Scientific Computing

Homework 1

IIIT-Delhi, Monsoon 2022
Kaushik Kalyanaraman

Due by: September 26, 2022 (Monday) by 23:59 hours IST

Total: 75 points

Guidelines

This assignment should be answered individually.

- You are not allowed to discuss specifics of your solutions, solution strategies, or coding strategies with any other student in this class.
- Please consult with and carefully read through IIIT-Delhi's Academic Dishonesty Policy.

Please don't be a cheater!

- You shall submit your solutions via Google Classroom – there are no printed/handwritten/-paper submissions. Please type, take a picture, or scan your solutions and upload them as a single PDF file and only a single PDF file. You can find many websites that will merge your scanned images into a PDF file or, for the nerdy types, you can find a command line script to do this.
- You may post queries relating to any HW question to `#discussions` on the class's Slack channel.
- Start working on your solutions early and don't wait until close to due date.
- Each problem should have a clear and concise write-up.
- Please clearly show all steps involved in your solution.
- You will need to write a separate code for each computational problem and sometimes for some subproblems too. You should name each such file as `problem_`*n*`.py` where *n* is the problem number. For example, your file names could be `problem_5.py`, `problem_6a.py`, `problem_6b.py`, and so on.
- *Python tip:* You can import `Python` modules as follows:

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

- We shall only accept `*.py` `Python` files – no Jupyter Notebook submissions are allowed. Please write your code for the machine problems in a text editor and generate appropriate output via Terminal using `Python` (directly) or `IPython` (recommended).

Problem 1: Condition Number — 2 points

> **Background.** Suppose as a toy example that we want to find the solution $x_0 := b/a$ to the linear equation $ax = b$ for $a, x, b \in \mathbb{R}$. The absolute forward error of a potential solution x is given by $|x - x_0|$ while the absolute backward error is given by $|b - ax|$ (why?) in which setting $b = ax_0$ yields $|a(x - x_0)|$. So, when $|a| \gg 1$, the problem is well-conditioned since small values of the backward error $a(x - x_0)$ imply even smaller values of $x - x_0$. In contrast, when $|a| \ll 1$ the problem is ill-conditioned since even if $a(x - x_0)$ is small, the forward error $x - x_0 = 1/a \cdot a(x - x_0)$ may be large given the $1/a$ factor.

What are the absolute and relative condition numbers of this problem using the definition that it is the ratio of the absolute (respectively, relative) forward error to the absolute (relative) backward error?

Points breakdown: Each of the (correct!) solutions to the absolute and relative condition numbers is worth 1 point.

Problem 2: Condition Number for Function Evaluation — 12 points

What are the absolute and relative condition numbers of the following functions? For what values of x are they large?

(a) $(x-1)^\alpha$

(b) $\ln x$

(c) $x^{-1}e^x$

(d) $1/(1+x^{-1})$

Note: To answer "where is the condition number large", recall that $\tan x$ has a large condition number for x close to $\pm\pi/2$; in fact, the condition number is ∞ for $x = \pm\pi/2$.

Points breakdown: Each of the parts is worth 3 points; for a part, the split is 0.5 points each for correctly stating the absolute and relative condition numbers, and 2 points for correctly specifying the values of x for which this condition number is large.

Problem 3: Conditioning of Root Finding and Wilkinson's Polynomial — 5 points

In this problem, we shall explore the conditioning of the problem of root-finding which is as follows: given $f : \mathbb{R} \longrightarrow \mathbb{R}$ find x^* such that $f(x^*) = 0$. We shall assume that a x^* exists (although it may not be unique and this is not really a devastating problem for *well posedness*). Furthermore, we shall also suppose that $f(x)$ is *smooth*, that is, all its derivatives of any order exist and are continuous.

(a) Because of inaccuracies in how we express or evaluate $f(x)$, we might end up computing the roots of a perturbation of this function: $f(x) + \varepsilon p(x)$ where $p(x)$ is some arbitrary but fixed smooth function. As stated before for x^* the root of f, $f(x^*) = 0$. If $f'(x^*) \neq 0$, for small

ε, we can write a function $x(\varepsilon)$ such that $f(x(\varepsilon)) + \varepsilon p(x(\varepsilon)) = 0$, with $x(0) = x^*$. Assuming such a function exists and is differentiable, show that:

$$\left.\frac{dx}{d\varepsilon}\right|_{\varepsilon=0} = -\frac{p(x^*)}{f'(x^*)}.$$

Note: Although the description here may be a teeny bit daunting, all I'm asking is to simply use the chain rule in calculus.

(b) Assume $f(x)$ is given by what's called as Wilkinson's polynomial:

$$f(x) \coloneqq (x-1)(x-2)(x-3)\cdots(x-20).$$

We can expand $f(x)$ and write it in the monomial basis as $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{20}x^{20}$, for appropriate choices of $a_0, \ldots, a_{20}$. If we now express the coefficient a_{19} inaccurately, we could use the model from the previous part with $p(x) \coloneqq x^{19}$ to predict how much root-finding will suffer. For these choices of $f(x)$ and $p(x)$, show that for the root computation $x^* = j$ (where j may be one of $1, 2, \ldots,$ or 20):

$$\left.\frac{dx}{d\varepsilon}\right|_{\varepsilon=0, x^*=j} = -\prod_{k\neq j}\frac{j}{j-k},$$

where k takes all the values $1, 2, \ldots, 20$ except j.

(c) Compare $\dfrac{dx}{d\varepsilon}$ from the previous part of $x^* = 1$ and $x^* = 20$. Which root is more stable to this perturbation?

Points breakdown: The three parts of this problem are worth 1 point, 2 points and 2 points ((1 + 1) points), respectively.

Problem 4: IEEE Double Precision Floating Point System — 6 points

(a) What does this code snippet do? Type it in an IPython terminal or Jupyter Notebook. Explain what you observe when you run it and why in no more than 2-3 sentences.

```
a = 1.
while a != 0:
    a /= 2
    print(a)
```

(b) What does this code snippet do? Type it in an IPython terminal or Jupyter Notebook. Explain what you observe when you run it and why in no more than 2-3 sentences.

```
a = 1.; eps = 1.; b = a + eps
while a != b:
    eps /= 2
    b = a + eps
print(eps)
```

(c) What does this code snippet do? Type it in an IPython terminal or Jupyter Notebook. Explain what you observe when you run it and why in no more than 2-3 sentences.

```
a = 1.
while a != inf:
    a *= 2
    print(a)
```

Important: The decimal point `.` after the `1` in the variable `a` is important in this part. For fun (and no explanation needed), set `a = 1` and observe what you obtain.

Note: There is no code or output submission for this problem. You will observe by running the code snippets at your end but only provide your explanations in your write up/submission.

Points breakdown: Each of the parts are worth 2 points in this question.

Problem 5: Understanding Conditioning and Rounding Error — 10 points

Using (a small code in) `Python`, using `np.tan` compute $\tan(x)$ for $x = \pi/4 + 2\pi \times 10^j$, $j = 0, 1, 2, \ldots, 20$ and print your results for each of the different arguments using (something like):

```
print(''(x, tan(x)) = (%1.16f, %1.16f)''%(x, tan(x)))
```

What is the relative condition number of this problem (considering x as the input)? Now, suppose that the only error that the computation here makes is in the representation of argument $x = \pi/4 + 2\pi \times 10^j$ because of rounding π to 16 decimal places – in other words, assume that the computation suffers from no rounding errors in addition (subtraction) and multiplication (division) and these are exactly done. (This is our *spherical cow*!) By what absolute amount will your computed input argument x differ from the exact one? Use this to explain the results you see for the computations of $\tan x$.

Note: You will turn in a single file named `problem_5.py` as part of your code submission. And provide your analysis in the PDF write up (typed or handwritten) submission.

Points breakdown: 3 points for submitting a working code, 3 points for the correct results and 4 points for a correct explanation.

Problem 6: Vector and Matrix Norms — 10 points

(a) Let $\|\cdot\|$ be a vector norm in $\mathbb{R}^m$, and assume that $A \in \mathbb{R}^{m\times n}$. Show that if $\text{rank}(A) = n$, then $\|x\|_A := \|Ax\|$ is a vector norm on $\mathbb{R}^n$.

Hint: This means that you have to verify that $\|x\|_A$ satisfies the three defining properties of norms. Triangle inequality property may look harder at first glance to show but you should be able to simply argue this from $\|\cdot\|$.

(b) For a matrix $A \in \mathbb{R}^{m \times n}$, recall that the Frobenius norm is defined as:

$$\|A\|_F := \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right)^2.$$

Verify that this satisfies the three properties of norms.

Hint: Again, simply be "clever" with the argument for triangle inequality.

(c) Let $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$. Show that if $E = uv^T$ then $\|E\|_F = \|E\|_2 = \|u\|_2 \|v\|_2$.

Hint: Here you can either use the submultiplicative property of the matrix norm to initially argue $\|uv^T x\|_2 \leq \|u\|_2 \|v\|_2 \|x\|_2$. (However, if your tastes are a little bit "purer", you can also arrive at this using Cauchy-Schwarz inequality!) You will now need to simply plug this and complete the argument for $\|uv^T\|_2$. It's merely an algebraic problem to then check that $\|uv^T\|_F = \|u\|_2 \|v\|_2$.

Points breakdown: 3 points for each of parts (a) and (b), and 4 points for part (c).

Problem 7: Numerical Computation of Euler's Constant — 10 points

Background. The *harmonic series* $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots$ is a classical infinite sum that diverges. In addition, it is known that the n^{th} partial sum $\sum_{k=1}^{n} \frac{1}{k}$ of this series approaches ∞ at the same rate as $\ln(n)$. The limiting difference of this n^{th} partial sum of the harmonic series and the $\ln(n)$ is called the *Euler's constant* denoted as γ:

$$\gamma = \lim_{n \to \infty} \left[\left(\sum_{k=1}^{n} \frac{1}{k} \right) - \ln(n) \right] \approx 0.5772156649015328. \tag{6.1}$$

It is also known (DeTemple, Duane W. "A Quicker Convergence to Euler's Constant." The American Mathematical Monthly 100, no. 5 (1993): 468–70. https://doi.org/10.2307/2324300) that γ can also be represented by:

$$\gamma = \lim_{n \to \infty} \left[\left(\sum_{k=1}^{n} \frac{1}{k} \right) - \ln\left(n + \frac{1}{2} \right) \right]. \tag{6.2}$$

Write two programs that uses $m = 1, 2, \ldots, 5000$ to estimate Euler's constant using equations (6.1) and (6.2). Print intermediate answers at every 100 steps, and verify that the second formula indeed converges more rapidly than the first.

Note: You will turn in two files named `problem_7.1.py` and `problem_7.2.py` as part of your code submission. And provide your output from each program in the PDF submission.

Points breakdown: 5 points for each part.

Problem 8: Gaussian Elimination and Partial Pivoting — 20 points

(a) Write a function to implement Gaussian elimination with no pivoting as in algorithm 1.

(b) Write a function to implement Gaussian elimination with partial pivoting as in algorithm 2.

(c) Test your code. For each of the examples as stated below, report the various measurements as below in a table. Use relative measurements as necessary, and 2-norm for all appropriate norms and condition numbers. You can use functions available from `NumPy` or `SciPy` for these purposes.

- The condition number (`np.linalg.cond`),
- the error from un-pivoted solve from your function in part (a),
- the residual from un-pivoted solve from your function in part (a),
- the error from partially-pivoted solve from your function in part (b),
- the residual from partially-pivoted solve from your function in part (b),
- the error from `np.linalg.solve`,
- the residual from `np.linalg.solve`.

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

For each matrix below, write 2 or 3 sentences on your observations with regard to conditioning, necessity of pivoting, and reasons for your observed results.

Use the following matrices of sizes $n = 10, 20, 30, 40$ to test your code:

1. A *random* matrix of size $n \times n$ with entries uniformly sampled from $[0, 1)$ (use `np.random.random_sample`).
2. The Hilbert matrix given by:

$$a_{ij} = \frac{1}{i + j - 1} \qquad (i, j = 1, \ldots, n).$$

3. The matrix given by

$$A_n = \begin{bmatrix} 1 & & & \cdots & 1 \\ -1 & 1 & & \cdots & 1 \\ -1 & -1 & 1 & \cdots & 1 \\ \vdots & & & \ddots & \vdots \\ -1 & -1 & -1 & \cdots & 1 \end{bmatrix}$$

For each case, let $x^\star = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}^T \in \mathbb{R}^n$ be the vector of all 1s of size n. Now, compute b as the matrix-vector product $A\,x^\star$ and solve the linear system $A\,x = b$.

Note: You will turn in two files named `problem_8a.py` and `problem_8b.py` as part of your code submission. And provide your output from each program in the PDF submission.

Points breakdown: 5 points for parts (a) and (b), and 10 points for part (c).

Inputs : A, b
Output : x

1 **for** $k = 1,\ k \leq n-1,\ k$++ **do**
2 **for** $i = k+1,\ i \leq n,\ i$++ **do**
3 $a_{\text{temp}} \longleftarrow a_{ik}/a_{kk}$
4 $a_{ik} \longleftarrow a_{\text{temp}}$
5 **for** $j = k+1,\ k \leq n,\ j$++ **do**
6 $a_{ij} \longleftarrow a_{ij} - a_{\text{temp}}a_{kj}$
7 **end**
8 $b_i \longleftarrow b_i - a_{\text{temp}}b_k$
9 **end**
10 **end**
11 $x_n \longleftarrow b_n/a_{nn}$
12 **for** $i = n-1,\ i \geq 1,\ i$- - **do**
13 $sum \longleftarrow b_i$
14 **for** $j = i+1,\ j \leq n;\ j$++ **do**
15 $sum \longleftarrow sum - a_{ij}x_j$
16 **end**
17 $x_i \longleftarrow sum/a_{ii}$
18 **end**

Algorithm 1: Gaussian elimination without partial pivoting

Inputs : A, b
Output : x

1 **for** $i = 1,\ i \leq n,\ i$++ **do**
2 $\quad \ell_i \longleftarrow i, \quad s_{\max} \longleftarrow 0$
3 $\quad$ **for** $j = 1,\ j \leq n,\ j$++ **do**
4 $\quad\quad s_{\max} \longleftarrow \max\{s_{\max}, |a_{ij}|\}$
5 $\quad$ **end**
6 $\quad s_i \longleftarrow s_{\max}$
7 **end**
8 **for** $k = 1,\ k \leq n-1,\ k$++ **do**
9 $\quad r_{\max} \longleftarrow 0$
10 $\quad$ **for** $i = k,\ i \leq n,\ i$++ **do**
11 $\quad\quad r \longleftarrow |a_{\ell_i,k}/s_{\ell_i}|$
12 $\quad\quad$ **if** $r > r_{max}$ **then**
13 $\quad\quad\quad r_{\max} \longleftarrow r, \quad j \longleftarrow i$
14 $\quad\quad$ **end**
15 $\quad$ **end**
16 $\quad \ell_{\text{temp}} \longleftarrow \ell_k, \quad \ell_k \longleftarrow \ell_j, \quad \ell_j \longleftarrow \ell_{\text{temp}}$
17 $\quad$ **for** $i = k+1,\ i \leq n,\ i$++ **do**
18 $\quad\quad a_{\text{mult}} \longleftarrow a_{\ell_i,k}/a_{\ell_k,k}$
19 $\quad\quad a_{\ell_i,k} \longleftarrow a_{\text{mult}}$
20 $\quad\quad$ **for** $j = k+1,\ j \leq n,\ j$++ **do**
21 $\quad\quad\quad a_{\ell_i,j} \longleftarrow a_{\ell_i,j} - a_{\text{mult}} a_{\ell_k,j}$
22 $\quad\quad$ **end**
23 $\quad$ **end**
24 **end**
25 **for** $k = 1,\ k \leq n-1,\ k$++ **do**
26 $\quad$ **for** $i = k+1,\ i \leq n,\ i$++ **do**
27 $\quad\quad b_{\ell_i} \longleftarrow b_{\ell_i} - a_{\ell_i,k} b_{\ell_k}$
28 $\quad$ **end**
29 **end**
30 $x_n \longleftarrow b_{\ell_n}/a_{\ell_n,n}$
31 **for** $i = n-1,\ i \geq 1,\ i$− **do**
32 $\quad sum \longleftarrow b_{\ell_i}$
33 $\quad$ **for** $j = i+1,\ j \leq n,\ j$++ **do**
34 $\quad\quad sum \longleftarrow sum - a_{\ell_i,j} x_j$
35 $\quad$ **end**
36 $\quad x_i \longleftarrow sum/a_{\ell_i,i}$
37 **end**

Algorithm 2: Gaussian elimination with partial pivoting