AN INTRODUCTION TO LINEAR ALGEBRA USING PYTHON

Summer 2021
Zoom Lecture: Tu: 2:00-4:00 p.m.
National Science Foundation (NSF) Center for Integrated Quantum Materials (CIQM), DMR -1231319
Dr. Steven L. Richardson (srichards22@comcast.net)
Professor Emeritus of Electrical Engineering, Department of Electrical and Computer Engineering, Howard University, Washington, DC
and
Faculty Associate in Applied Physics, John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA

PROBLEM SET XII
(due Tuesday, August 10, 2021)

Problem 1

As discussed in Lecture 12 please explicitly show that

$$\vec{a}_1 \cdot \left(\vec{b} - A\,\vec{x}^*\right) = 0$$

$$\vec{a}_2 \cdot \left(\vec{b} - A\,\vec{x}^*\right) = 0$$

can be recast as

$$\vec{a}_1^T \left(\vec{b} - A\,\vec{x}^*\right) = 0$$

$$\vec{a}_2^T \cdot \left(\vec{b} - A\,\vec{x}^*\right) = 0$$

which can finally be written as

$$A^T \left(\vec{b} - A\,\vec{x}^*\right) = \vec{0}$$

where

$$A = (\vec{v}_1\ \vec{v}_2)$$

As a hint use the case of square matrices of order 2.

Problem 2

Given the vectors

$$\vec{\mathbf{b}} = \begin{pmatrix} -1 \\ 6 \end{pmatrix}$$

$$\vec{\mathbf{a}} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

find

$$\boldsymbol{proj}_{\vec{a}}\,\vec{\boldsymbol{b}}$$

Now calculate the projection matrix $\boldsymbol{P}$ and verify your results.

Problem 3

Given the vectors

$$\vec{\mathbf{b}} = \begin{pmatrix} 2 \\ -1 \\ 3 \end{pmatrix}$$

$$\vec{\mathbf{a}} = \begin{pmatrix} 4 \\ -1 \\ 2 \end{pmatrix}$$

find

$$\boldsymbol{proj}_{\vec{a}}\,\vec{\boldsymbol{b}}$$

Now calculate the projection matrix $\boldsymbol{P}$ and verify your results.

Problem 4

Given the vector

$$\vec{\mathbf{b}} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

find the projection of $\vec{\boldsymbol{b}}$ along the line

$$\boldsymbol{2x - y = 0}$$

Problem 5

Given the vectors

$$\vec{\mathbf{b}} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\vec{\mathbf{a}} = \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix}$$

find the vector components of $\vec{\boldsymbol{b}}$ along $\vec{\boldsymbol{a}}$ and orthogonal to $\vec{\boldsymbol{a}}$.

Problem 6

Given the vectors

$$\vec{\mathbf{b}} = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 2 \end{pmatrix}$$

$$\vec{\mathbf{a}} = \begin{pmatrix} 4 \\ -4 \\ 2 \\ -2 \end{pmatrix}$$

find the vector components of $\vec{\boldsymbol{b}}$ along $\vec{\boldsymbol{a}}$ and orthogonal to $\vec{\boldsymbol{a}}$. Now you see the power of projections in linear algebra. Try solving this problem geometrically in $\boldsymbol{R}^4$. I wish you good luck!

Problem 7

Given the vectors

$$\vec{\mathbf{b}} = \begin{pmatrix} 1 \\ -2 \\ 4 \end{pmatrix}$$

$$\vec{\mathbf{a}} = \begin{pmatrix} 2 \\ 3 \\ 6 \end{pmatrix}$$

find the length of the projection of $\vec{\boldsymbol{b}}$ without finding the actual vector itself.

Problem 8

Given the vectors

$$\vec{\mathbf{b}} = \begin{pmatrix} 2 \\ 3 \\ 1 \\ 5 \end{pmatrix}$$

$$\vec{\mathbf{a}} = \begin{pmatrix} -4 \\ 2 \\ -2 \\ 3 \end{pmatrix}$$

find the length of the projection of $\vec{\boldsymbol{b}}$ without finding the actual vector itself.

Problem 9

Diagonalize the matrix $\boldsymbol{B}$

$$\mathbf{B} = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$$

and show that

$$\mathbf{B^k} = \begin{pmatrix} 1+3^k & 1-3^k \\ 1-3^k & 1+3^k \end{pmatrix}$$

Problem 10

Diagonalize the matrix $\boldsymbol{S}$

$$\mathbf{B} = \begin{pmatrix} 5 & 1 \\ 0 & 4 \end{pmatrix}$$

and show that

$$\mathbf{B^k} = \begin{pmatrix} 5^k & 5^k - 4^k \\ 0 & 4^k \end{pmatrix}$$

Problem 11

In Problems 9 and 10 what happens to the final results in the limit as k approaches infinity? You might think this happens to all matrices, but it really depends on the matrix elements. What do you think happens if all the matrix elements are each less than unity? Now here comes an interesting matrix $\boldsymbol{A}$ whose matrix elements are all each less than unity. For this case, consider the matrix $\boldsymbol{A^k}$. As k approaches infinity, the final behavior is quite peculiar as the matrix elements become final non-zero constants! This is an example of a **Markov matrix** where every column of the matrix adds up to unity and all matrix elements are positive. **Markov matrices** are quite useful in probability theory and for search engines like Google. Show exactly what is happening in this case and think about eigenvectors in discussing your final results.

$$\mathbf{A} = \begin{pmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{pmatrix}$$

Problem 12

The $\boldsymbol{n^{th}}$ power of rotation through $\boldsymbol{\theta}$ is a rotation through $\boldsymbol{n\,\theta}$. Using matrix diagonalization and Euler's formula prove this gem of a formula

$$\mathbf{R^n} = \begin{pmatrix} \cos\boldsymbol{\theta} & -\sin\boldsymbol{\theta} \\ \sin\boldsymbol{\theta} & \cos\boldsymbol{\theta} \end{pmatrix}^{\boldsymbol{n}} = \begin{pmatrix} \cos\boldsymbol{n\,\theta} & -\sin\boldsymbol{n\,\theta} \\ \sin\boldsymbol{n\,\theta} & \cos\boldsymbol{n\,\theta} \end{pmatrix}$$

Python Exercise 12

1. Up until now every time in this course that you have created a matrix and its matrix elements everything vanishes once you shut down your Jupyter notebook. We need to learn how to save matrices in Python as files and how to read and write general files in Python.

First of all how do we write a file?

with open("greet.txt", "w") as f:#This command opens the file "greet.txt" if it exists and creates it if it does not exist. The "w" command tells Python to write data into the file.

f.write ("Hello, world!")#This command tells Python to write the string "Hello, world!" into the file"greet.txt".

Next how do we read a file?

with open("greet.txt", "r") as f:#This command opens the file "greet.txt" to read from it.

sample =f.read#This command tells Python to load the contents of the file "greet.txt" into the variable "sample"

print(sample)

Practice these commands yourself by creating and reading your own files. Might I suggest the nice little inexpensive book by Mark Myers,"A Smarter Way to Learn Python"? It deals with learning Python from a non-scientific viewpoint (i.e. no linear algebra, no calculus, no scientific computing, period). A much more comprehensive book by Andrew Bird *et al.* is entitled "The Python Workshop: A New, Interactive Approach to Learning Python".

2. We introduced the **while** loop back in Problem Set VII. Here we introduce the **for** loop in Python. What is the difference? Consult on-line resources in Python to learn more about the differences between these two important kinds of loops in Python. Experiment using this script with random matrices of your choice.

Here is how you create a file in Python and write to it and save the data:

import numpy as np

A = np.array([[1, 2, 3], [4, 5, 6],[7, 8, 9]], np.int32)#Look up the meaning of np.int32

np.savetxt("test.txt", A)#write content of matrix A in file "text.txt"

B = np.loadtxt("test.txt")#now read file "text.txt" and load it into a new matrix B

print(B)

3. Here is another example of how to create a file in Python and write to it and save the data. Look up the details of any new commands or syntax from on-line Python resources. Vary and play with some of the commands to make sure you understand what is going on here.

```
f = open(“log.txt”,“w”)
from datetime import datetime
import time
for i in range(0,10):
    print(datetime.now().strftime(“%Y%m%d_%H:%M:%S - ”),i)
    f.write(datetime.now().strftime(“%Y%m%d_%H:%M:%S - ”))
    time.sleep(1)
    f.write(str(i))
    f.write(“_n”)
```

4. Here is another example of a matrix that does something when it operates on a vector. It is called a shear matrix. Can you understand why? Create several examples in Python to visualize what is going on here. Can you invent other types of shear matrices in $\boldsymbol{R^2}$?

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

5. Use Python to experiment with large **Markov** and **non-Markov** matrices as discussed in Problem 11 in the limit as k gets larger for $\boldsymbol{A^k}$.

6. We spent most of our time in Python using the **NumPy** library and very little time using the **SymPy** library. Actually **SymPy** does most of what **NumPy** can do, but the syntax is different. Use on-line resources to investigate this.

7. Here is a nice little Python script which you should be able to understand now. it also tests the speed of direct matrix multiplication. This is a rough version of the script and it can be optimized. That way you will learn something! Experiment with this script.

```
import numpy as np
import scipy.linalg as la
import time
P = np.array([[1,1],[1,-1]])
print(P)
D=np.diag((3,1))
print(D)
M = P@D@la.inv(P)
print(M)
evals,evecs = la.eig(M)
print(evals)
Pinv =la.inv(P)
k=200
start =time.time()
result = M.copy()
for _ in range(1,k):
    result = result @ M
end =time.time()
print(result)
print(end-start)
```

8. Now let us refer back to Problem 7. Edit it to perform the same calculation using matrix diagonalization and compare the run times. You should discover something interesting here! Vary the matrix sizes and the value of **k**.

You will need some additional code for this problem: **P @ D**k @ Pinv**

9. Here are some interesting concluding remarks:

9.1 All symmetric matrices are diagonalizable.

9.2 Distinct eigenvalues always lead to distinct eigenvectors.

9.3 Distinct eigenvalues always lead to linearly independent eigenvectors.

9.4 A square matrix of order **n** is diagonalizable if and only if it has **n** linearly independent eigenvectors.

9.5 Eigenvalues which are not distinct will lead to eigenvectors which may or may not be linearly independent.

You should think carefully why all of these statements are true. You can consult the references in Problem Set XIII to check your thinking!