Alma Mater Studiorum - Università di Bologna

Relazione per il corso di Data Science

Liam Cavini
Semestre Invernale 2024/2025

1° Foglio, Algebra Lineare
08/10/2024

Esercizio 1 - Algebra delle matrici

(a) Per sommare le matrici si utilizza il seguente codice python (le librerie sono importate soltanto una volta, nei successivi codici presenti nella relazione è implicito che siano state importate):

```python
import numpy as np
A = np.array([[1,3],[5,7]])
B = np.array([[2,4],[6,8]])
print(A+B)
print(A-B)
```

Il cui output restituisce le matrici:

$$A + B = \begin{bmatrix} 3 & 7 \\ 11 & 15 \end{bmatrix}$$

$$A - B = \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}$$

(b) Per determinare il prodotto tra due matrici è stata usata l'apposita operazione di numpy:

```python
print(A@B)
print(B@A)
```

Il codice restituisce come output:

$$AB = \begin{bmatrix} 20 & 28 \\ 52 & 76 \end{bmatrix}$$

$$BA = \begin{bmatrix} 22 & 34 \\ 46 & 74 \end{bmatrix}$$

Si è quindi verificato che il prodotto tra matrici non commuta.

(c) Le matrici A e B sono invertibili, dato che hanno entrambe rango massimo. Dunque si può procedere col calcolo di A/B e B/A, che è stato portato a termine usando il seguente codice python:

```python
A_inv = np.linalg.inv(A)
B_inv = np.linalg.inv(B)
print(A@B_inv)
print(B@A_inv)
```

Che stampa i seguenti risultati:

$$A/B = \begin{bmatrix} 1.25 & -0.25 \\ 0.25 & 0.75 \end{bmatrix}$$

$$B/A = \begin{bmatrix} 0.75 & 0.25 \\ -0.25 & 1.25 \end{bmatrix}$$

(d) Lo Spettro degli autovalori della matrice $A(\epsilon)$ definita come:

$$A(\epsilon) = \begin{bmatrix} 101 - \epsilon & -90 - \epsilon \\ 110 & -98 \end{bmatrix}$$

è stato graficato con matplotlib usando il codice Python riportato sotto:

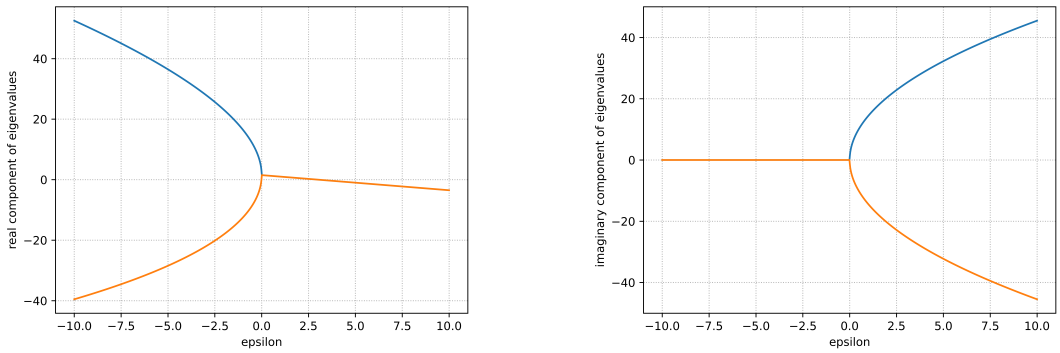

Figura 1: spettro degli autovalori di $A(\epsilon)$, in funzione di ϵ. A sinistra la componente reale dei due autovalori, a destra quella immaginaria.

```python
from matplotlib import pyplot as plt

# initializing arrays
A = np.array([[101,-90],[110,-98]])
B = np.array([[-1,-1],[0,0]])

#initializing array containing eigenvalues
eps = np.linspace(-10,10,1000)
y_real = np.zeros((len(eps), 2))
y_imag = np.zeros((len(eps), 2))

#filling eigenvalue arrays
for i, epsilon in enumerate(eps):
    y_real[i] = np.linalg.eig(A + epsilon*B)[0].real
    y_imag[i] = np.linalg.eig(A + epsilon*B)[0].imag

#creating graph with matplotlib
fig, ax = plt.subplots()

plt.xlabel('epsilon')
plt.ylabel('real component of eigenvalues')

ax.plot(eps,y_real[:,0])
ax.plot(eps,y_real[:,1])
ax.grid(True,ls='dotted')

fig, ax2 = plt.subplots()

plt.xlabel('epsilon')
plt.ylabel('imaginary component of eigenvalues')

ax2.plot(eps,y_imag[:,0])
ax2.plot(eps,y_imag[:,1])
ax2.grid(True,ls='dotted')
```

I risultati sono mostrati in Figura 1.

(e) Per trovare i gli autovalori e autovettori della matrice

$$A = \begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix}$$

si è usato l'apposita funzione di numpy, **np.linalg.eig()**, trovando gli autovalori:

$$\lambda_1 = -1 + i$$
$$\lambda_2 = -1 - i$$

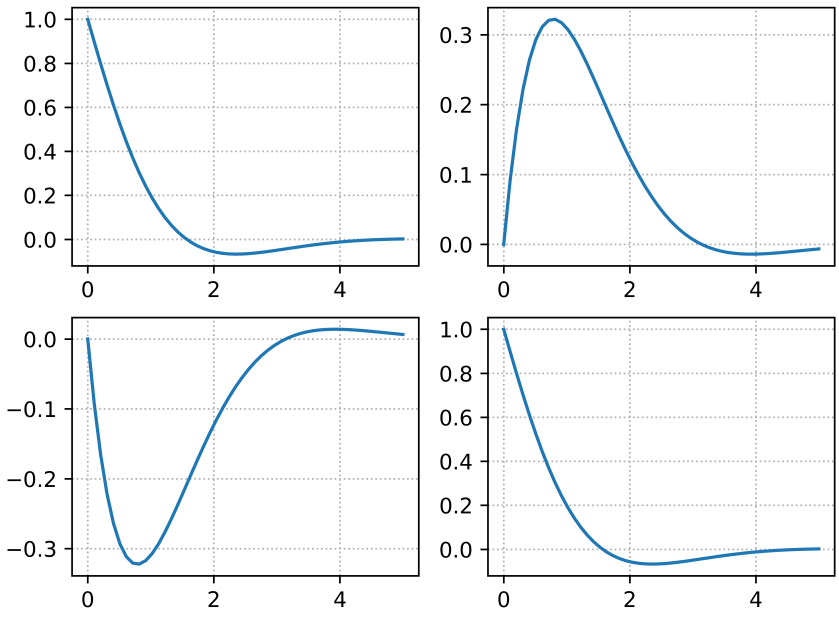

Figura 2: *I valori reali dei quattro elementi della matrice esponenziale e^{tA} in funzione di t. I valori immaginari non sono mostrati in quanto sempre nulli.*

e gli autovettori:

$$\begin{bmatrix} 0.70710678 \\ 0.70710678i \end{bmatrix}, \begin{bmatrix} 0.70710678 \\ -0.70710678i \end{bmatrix}$$

Per trovare l'esponenziale è stato necessario compiere un cambio di base, rendendo la matrice A diagonale, e calcolare l'esponenziale di questa, per poi tornare nella base originaria. In pratica il primo cambio di base è già stato compiuto nel calcolo degli autovettori. I risultati sono rappresentati in Figura 2. I valori immaginari risultano nulli, come atteso, dato che la matrice originaria è a valori reali. Il codice utilizzato è il seguente:

```
#initializing array
A = np.array([[-1,1],[-1,-1]])

#calculating eigenvalues and vectors
eigenvalues, eigenvectors = np.linalg.eig(A)
print("eigenvalues: ", eigenvalues)
print("eigenvectors: \n",eigenvectors)

#calculating inverse, used to find exponential matrix
eigenvec_inv = np.linalg.inv(eigenvectors)

t = np.linspace(0, 5)

y_real = np.zeros((2,2,len(t)))
y_imag = np.zeros((2,2,len(t)))

for i, t_element in enumerate(t):
    exp_diagonal = np.diag(np.exp(t_element*eigenvalues))
    #change of basis - we return to A's basis
    exp_matrix = eigenvectors @ exp_diagonal @ eigenvec_inv
    y_real[:,:,i] = exp_matrix.real
    y_imag[:,:,i] = exp_matrix.imag


fig, ax = plt.subplots(2,2)
```

```
26
27 #plotting real and imaginary in for plots
28 ax[0,0].plot(t,y_real[0,0])
29 ax[1,0].plot(t,y_real[1,0])
30 ax[0,1].plot(t,y_real[0,1])
31 ax[1,1].plot(t,y_real[1,1])
```

(f) Per verificare le proprietà dei autovalori delle matrici generate casualmente, è stato generato un ensamble di matrici di dimensione $n \times n$ per ogni n fino ad un certo valore nmax. Questo è stato impostato prima ad un valore basso (nmax $= 5$) per verificare le proprietà degli autovalori a bassi n, e successivamente ad un valore "alto" (nmax $= 50$) per osservare le proprietà ad n elevati.

Si è stimata la probabilità che un autovalore si trovi sulla retta reale, al variare di n, verificando che ad n bassi gli autovalori si dispongono principalmente sulla retta reale. I valori trovati sono stati riportati in Tabella 1.

n	prob(%)
1	100
2	71
3	57
4	49
5	43

Tabella 1: *Probabilità arrotondate alla seconda cifra di trovare un autovalore sulla retta reale, per i primi cinque valori di n. Il calcolo è stato compiuto su* 1e5 *campionamenti.*

Per valori elevati di n gli autovalori della matrice, se divisi per $\sqrt{n}$, tendono a essere distribuiti in maniera uniforme nel disco unitario del piano complesso. Questo comportamente è illustrato nella Figura 3.

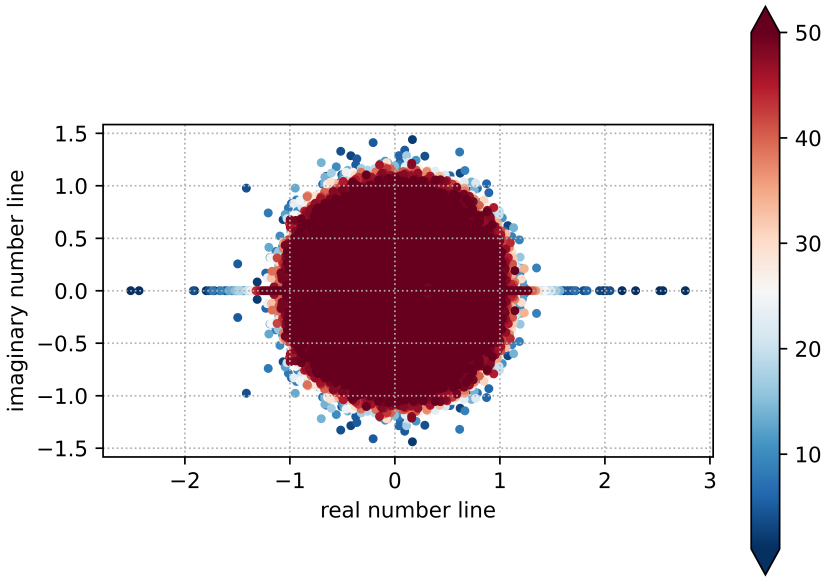

Figura 3: *La distribuzione di autovalori scalati di $\sqrt{n}$ nel piano complesso. Il colore indica la dimensione n della matrice quadrata da cui è stato estratto l'autovalore.*

Per visualizzare il comportamento degli autovalori più distanti dall'orgine, si è graficato l'autovalore con modulo massimo ottenuto in funzione di n (riscalato di $\sqrt{n}$), e la media degli autovalori con a modulo maggiore di uno, sempre in funzione di n. Il risultato si può osservare in Figura 4.

Si può anche ottenere lo stesso effetto di riscalare gli autovalori cambiando in maniera appropriata la varianza della gaussiana da cui si campionano gli elemnti della matrice.

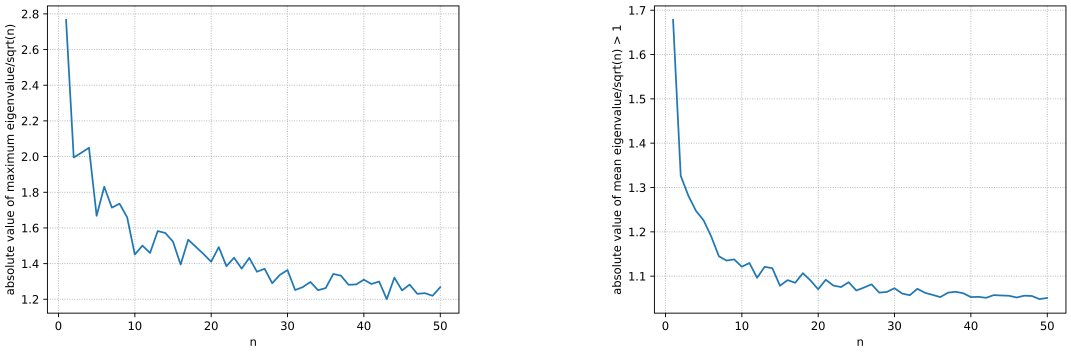

Figura 4: *A sinistra gli autovalori (divisi per $\sqrt{n}$) con modulo massimo in funzione di n, a destra la media degli autovalori (sempre divisi per $\sqrt{n}$) con modulo maggiore di 1 in funzione di n.*

(g) Il seguente codice python è stato utilizzato per trovare gli autovalori e autovettori delle matrici di inerzia:

```python
from ase import Atoms
from ase.io import write, read

def principal_axis(x,y,z, masses):

    # calculating moment of inertia:
    Ixx = np.sum(masses*(y**2+z**2))
    Iyy = np.sum(masses*(x**2+z**2))
    Izz = np.sum(masses*(x**2+y**2))
    Ixy = - np.sum(masses*x*y)
    Iyz = - np.sum(masses*y*z)
    Ixz = - np.sum(masses*x*z)


    inertia_tensor = np.array([[Ixx, Ixy, Ixz], [Ixy, Iyy, Iyz], [Ixz, Iyz, Izz]])


    eigenvalues, eigenvectors = np.linalg.eig(inertia_tensor)

    return (eigenvalues, eigenvectors)

#change of coordinates to center of mass as origin
def center_of_mass_coordinates(x,y,z,masses):

    #finding center of mass
    cmx = np.sum(masses*x)/np.sum(masses)
    cmy = np.sum(masses*y)/np.sum(masses)
    cmz = np.sum(masses*z)/np.sum(masses)

    #calculating new coordinates
    x = x-cmx
    y = y-cmy
    z = z-cmz

    return (x,y,z)

#lists cointaining all eigenvalues and eigenvectors
eigenvalues = []
eigenvectors = []

```

```
DataSet = read("dataset.xyz",index=':')
for i, molecule in enumerate(DataSet):

    positions = molecule.get_positions()
    masses = molecule.get_masses()

    x = positions[:,0]
    y = positions[:,1]
    z = positions[:,2]

    #shifting origin to center of mass
    x,y,z = center_of_mass_coordinates(x,y,z, masses)

    #calculating eigenvalues of I and eigenvectors (principal axis)
    eigval, eigvect = principal_axis(x,y,z,masses)

    eigenvalues.append(eigval.tolist())
    eigenvectors.append(eigvect.tolist())

#counting symmetric molecules
dim_1 = []
dim_2 = []
symm_axis = []
symm_total = []

tolerance = 1

for i, eigval in enumerate(np.array(eigenvalues)):
    #checking if all elements are distinct
    eigval = np.sort(eigval)

    #if all values are disctinct
    if (not np.isclose(eigval[0], eigval[1], atol=tolerance)) and (not np.isclose(
    eigval[1], eigval[2], atol=tolerance)) :
        #if one value is zero
        if np.isclose(eigval[0], 0, atol=tolerance):
            dim_2.extend([i])

    #if at least one value is equal
    else:
        #if at least one value is zero
        if np.isclose(eigval[0], 0, atol=tolerance):
            #if two values are zero (and two are equal)
            if np.isclose(eigval[1], 0, atol=tolerance):
                dim_1.extend([i])
            #if one value is zero (and two are equal)
            else:
                dim_2.extend([i])
                symm_axis.extend([i])
        #if no value is zero
        else:
            #if all values are equal (none is zero, no single atoms in the dataset)
            if np.isclose(eigval[0], eigval[1], atol=tolerance) and np.isclose(eigval
    [1], eigval[2], atol=tolerance):
                symm_total.extend([i])
            #if two values are equal, and none is zero
            else:
                symm_axis.extend([i])




print("symmetric axis:\n", symm_axis)
print("symmetric total:\n", symm_total)
print("2 dimensional:\n", dim_2)
print("1 dimensional:\n", dim_1)
```

L'ultima parte del codice ha lo scopo di trovare, confrontando tra loro gli autovalori del tensore di inerzia, quali molecole hanno certe simmetrie, e quali risiedono nel piano. È stato trovato che 6 molecole sono bidimensionali, tra cui C2H2, C3NH, C4H2, C5NH, 31 hanno simmetrie discrete lungo un asse

(per cui hanno 2 autovalori del tensore uguali), tra cui C3H4 e C4H6, e 2 hanno simmetrie discrete lungo più di un asse, cioè tutti gli autovalori uguali, che sono CH4 (che ha geometria tetraedrica) e C5H12.

Esercizio 2 - Decomposizione SVD

(a) Per costruire la SVD in python delle seguenti matrici:

$$A = \begin{bmatrix} 2 & -1 \\ 2 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 7 & 1 \\ 0 & 0 \\ 5 & 5 \end{bmatrix}$$

Per determinare la SVD è stata usata la apposita funzione numpy:

```
A = np.array([[2,-1],[2,2]])
B = np.array([[7,1],[0,0],[5,5]])
U, S, Vh = np.linalg.svd(A)
print("SVD of A\n", U)
print(S)
print(Vh)

U, S, Vh = np.linalg.svd(B)
print("SVD of B\n", U)
print(S)
print(Vh)
```

Trovando le scomposizioni:

$$A = \begin{bmatrix} -0.4472136 & -0.89442719 \\ -0.89442719 & 0.4472136 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -0.70710678 & 0.70710678 & 0 \\ 0 & 0 & -1 \\ -0.70710678 & -0.70710678 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} -0.4472136 & -0.89442719 \\ -0.89442719 & 0.4472136 \end{bmatrix} \begin{bmatrix} 9.48683298 & 0 \\ 0 & 3.16227766 \end{bmatrix} \begin{bmatrix} -0.89442719 & -0.4472136 \\ 0.4472136 & -0.89442719 \end{bmatrix}$$

(b) Si è proceduto come nell'esercizo precedente:

```
A = np.array([[0,1,0,0],[0,0,2,0],[0,0,0,3],[0,0,0,0]])
U, S, Vh = np.linalg.svd(A)
print("SVD of A\n", U)
print(S)
print(Vh)
```

trovando:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix}$$

(c) Notando che:

$$A \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & \epsilon \end{bmatrix}$$

abbiamo che banalmente gli autovalori (che sono anche i valori singolari in questo caso) sono 1,2,3,ϵ.

(d) Per risolvere l'esericizio si è usato il fatto che i primi k (dove k è il rango della matrice A) vettori colonna di U (la matrice sinistra nella scomposizione SVD) generano la immagine di A, mentre gli altri vettori colonna generano il ker di A^T. Queste considerazioni hanno portato alla seguente soluzione:

```
A = np.array([[1, 2],[3, 4],[1, 1],[1, -1]])
b = np.array([1, 2, 3, 4])

U, s, Vh = np.linalg.svd(A)
U_inv = np.linalg.inv(U)

b_coeff = U_inv @ b

rank = np.sum(s > 1e-10)
U_col =  U[:, :rank]
U_null = U[:, rank:]


b_hat = U_col @ b_coeff[:rank]
z = U_null @ b_coeff[rank:]

print(b)
print(b_hat+z)
```