

Laboratorio 1

Data Science

Pivi Riccardo

September 2025

1 Esercizio 1

1.1 Punto a

L'obiettivo dell'esercizio consisteva nel calcolare gli autovalori e gli autovettori complessi della matrice

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

e utilizzare tali risultati per determinare la matrice esponenziale e^{At} , con $t \in [0, 5]$. Infine, sono stati rappresentati graficamente gli elementi della matrice e^{At} in funzione di t .

Parte fondamentale del codice:

```
1 A = np.array([[ -1, 1], [-1, -1]])
2
3 autovalori, autovettori= np.linalg.eig(A)
4
5 print(autovalori)
6 print(autovettori)
```

```
1 D = np.array([[autovalori[0],0],[0, autovalori[1]]])
2 U = autovettori
3
4 def eAt(t):
5     Gt = np.zeros((2,2),dtype=complex)
6     for i in np.arange(2):
7         Gt[i,i] = cmath.exp(autovalori[i]*t)
8     return U @ Gt @ np.linalg.inv(U)
```

In conclusione gli autovettori e gli autovalori di A sono risultati essere rispettivamente: $(0.70710678, +0.70710678j)$, $(0.70710678, -0.70710678j)$ e $-1.+1.j$, $-1.-1.j$. Si riportano i grafici dei valori degli elementi della matrice esponenziale in funzione di t .

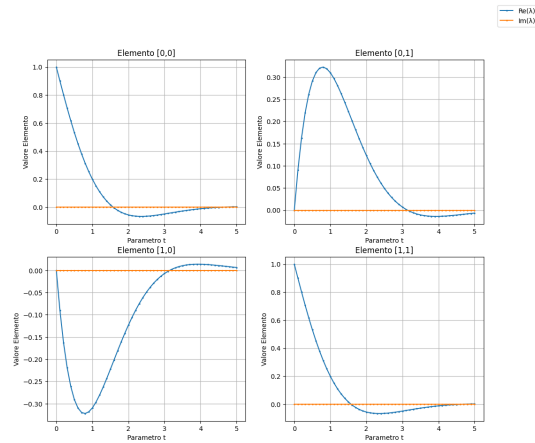


Figura 1: Valori degli elementi dell'esponenziale di matrice al variare del parametro t .

1.2 Punto b

L'obiettivo dell'esercizio è studiare la distribuzione degli autovalori complessi λ di matrici reali random $n \times n$ con elementi gaussiani standard. Si vuole verificare numericamente che, per piccoli valori di n , gli autovalori λ/\sqrt{n} si concentrano principalmente lungo l'asse reale, mentre per n grandi tendono a distribuirsi uniformemente nel disco unitario.

Parte fondamentale del codice:

```

1  Ns = np.array([2, 10, 50, 200]) # valori fissati di n
2  Nmat = 200 # numero di matrici generate per ciascun n
3
4  fig, axs = plt.subplots(2, 2, figsize=(10, 10))
5
6  for ax, n in zip(axs.ravel(), Ns):
7      alleig = []
8      for i in range(Nmat):
9          X = np.random.normal(loc=0.0, scale=1.0, size=(n, n))
10         eigv = np.linalg.eigvals(X)
11         scaled = eigv / np.sqrt(n)
12         alleig.extend(scaled)
13
14     alleig = np.array(alleig)
15
16     ax.scatter(alleig.real, alleig.imag, s=0.2, alpha=0.5, color="purple")

```

Si riportano i risultati grafici che confermano il risultato atteso:

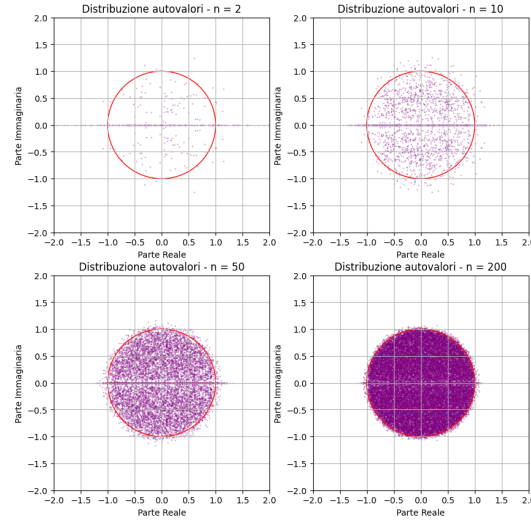


Figura 2: Distribuzione degli autovalori per vari n

Inoltre si potrebbe evitare di riscalare gli autovalori, modificando la varianza della distribuzione gaussiana degli elementi di matrice. In particolare, scegliendo $X_{ij} \sim \mathcal{N}(0, 1/n)$, il raggio spettrale si mantiene dell'ordine dell'unità e gli autovalori risultano distribuiti direttamente nel disco unitario.

1.3 Punto c

L'obiettivo dell'esercizio è calcolare i momenti principali di inerzia di un insieme di molecole, a partire dalle loro masse e posizioni atomiche, dopo averle riportate nel sistema di riferimento del centro di massa. Si diagonalizza quindi il tensore di inerzia per ottenere i valori $I_a \leq I_b \leq I_c$, che permettono di classificare le molecole come sferiche, oblate, prolate o asimmetriche, e di determinare il numero di molecole appartenenti a ciascuna categoria.

Parte fondamentale del codice:

```

1 def inertia_tensor(masses, positions):
2     x = positions[:, 0]
3     y = positions[:, 1]
4     z = positions[:, 2]
5
6     Ixx = np.sum(masses * (y**2 + z**2))
7     Iyy = np.sum(masses * (x**2 + z**2))
8     Izz = np.sum(masses * (x**2 + y**2))
9     Ixy = -np.sum(masses * x * y)
10    Ixz = -np.sum(masses * x * z)
11    Iyz = -np.sum(masses * y * z)
12
13    return np.array([[Ixx, Ixy, Ixz],
14                    [Ixy, Iyy, Iyz],
15                    [Ixz, Iyz, Izz]])
16
17 def clas(eigval, counters, tol=1e-1):
18     eigval=np.sort(eigval)
19     if np.allclose(eigval[0], eigval[1], atol=tol) and np.allclose(eigval[1], eigval[2], atol=tol):
20         counters['sphere'] += 1
21     elif np.allclose(eigval[0], eigval[1], atol=tol) and not np.allclose(eigval[1], eigval[2], atol=tol):
22         counters['obl'] += 1
23     elif np.allclose(eigval[1], eigval[2], atol=tol) and not np.allclose(eigval[0], eigval[1], atol=tol):
24         counters['prol'] += 1
25     else:
26         counters['asim'] += 1

```

```

1 from ase import Atoms
2 from ase.io import write, read
3
4 counters = {'sphere':0, 'obl':0, 'prol':0, 'asim':0}
5
6 DataSet = read("dataset.xyz", index=":")
7 for mol in DataSet:
8

```

```

9 masses = mol.get_masses() # array delle masse della molecola
10 positions = mol.get_positions() # array Nx3 delle coordinate
11 print("Masse:", masses)
12 print("Posizioni:", positions)
13 com = np.average(positions, axis=0, weights=masses) #centro di massa
14 print("Centro_di_massa:", com)
15 pos_com = positions - com
16 I = inertia_tensor(masses, pos_com)
17 eigval, _ = np.linalg.eig(I)
18 print(eigval)
19 clas(eigval, counters)
20
21 print(counters)

```

Il risultato è che nel dataset di molecole sono presenti: 2 molecole sferiche, 4 oblate, 13 prolate, 7146 asimmetriche. Importante è sottolineare la tolleranza alla prima cifra decimale usata per categorizzare le molecole.

2 Esercizio 2

2.1 Punto a

L'obiettivo dell'esercizio era lo studio degli autovalori e dei valori singolari della matrice A al variare di ε :

$$A(\varepsilon) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ \varepsilon & 0 & 0 & 0 \end{bmatrix},$$

Parte fondamentale del codice:

```

1 delta = 0.00001
2 eps_vals=np.linspace(-delta, delta, 30)
3 alleigval=[]
4 allsingular=[]
5 for eps in eps_vals:
6     A = np.array([[0, 1, 0, 0],[0, 0,2,0],[0,0,0,3],[eps,0,0,0]])
7     U, S, Vh = np.linalg.svd(A, full_matrices=True)
8     eigval,_=np.linalg.eig(A)
9     alleigval.append(eigval)
10    allsingular.append(S)
11 alleigval = np.array(alleigval)
12 allsingular=np.array(allsingular)
13
14 print(alleigval)
15 print(allsingular)

```

Si riportano i grafici dello studio di parte reale ed immaginaria degli autovalori e dei valori singolari della matrice A al variare di ε fra [-0.00001; 0.00001]

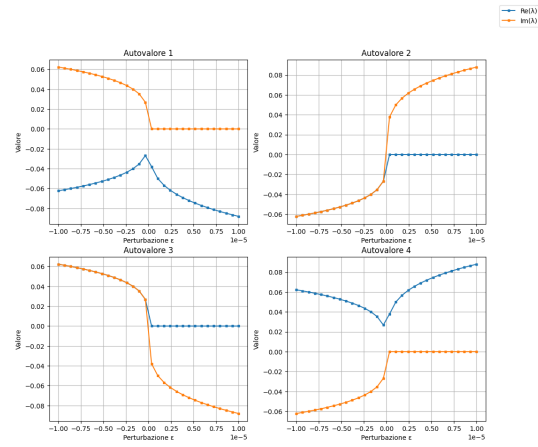


Figura 3

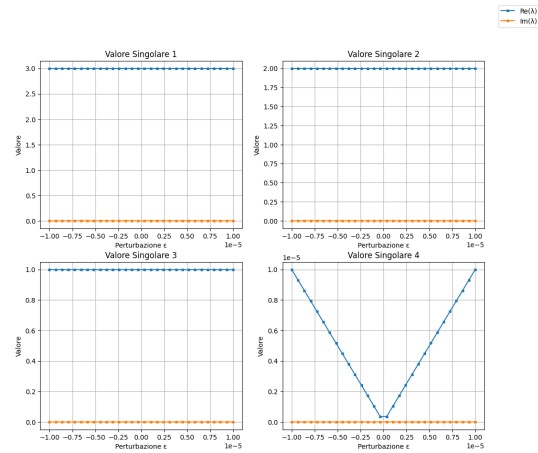


Figura 4

Concludiamo che

2.2 Punto b

L'obiettivo dell'esercizio consiste nell'usare la scomposizione SVD di una matrice per ricreare un'immagine a vari gradi di accuratezza. Inoltre, di analizzare i valori singolari della foto scelta.

Parte fondamentale del codice:

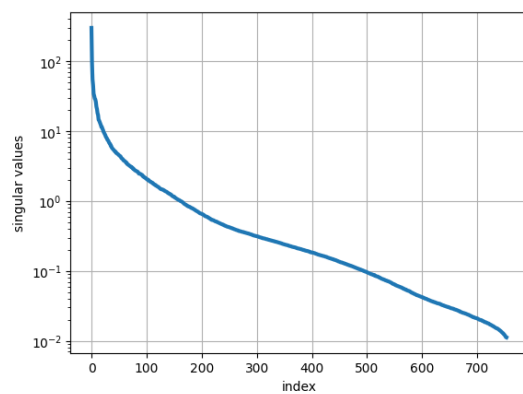


Figura 5

Approssimazioni tramite SVD

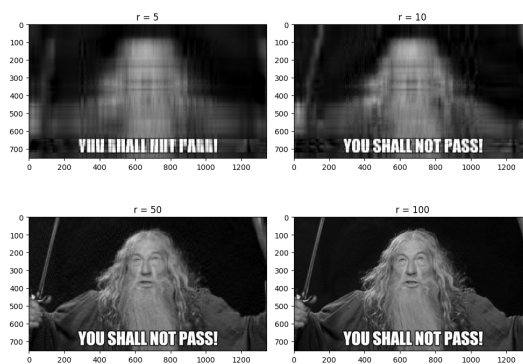


Figura 6