Laboratorio 1 - Data Science

Pivi Riccardo

Settembre 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:

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

autovalori, autovettori= np.linalg.eig(A)

print(autovalori)
print(autovettori)
```

```
D = np.array([[autovalori[0],0],[0, autovalori[1]]])
U = autovettori

def eAt(t):
    Gt = np.zeros((2,2),dtype=complex)
    for i in np.arange(2):
        Gt[i,i] = cmath.exp(autovalori[i]*t)
    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.

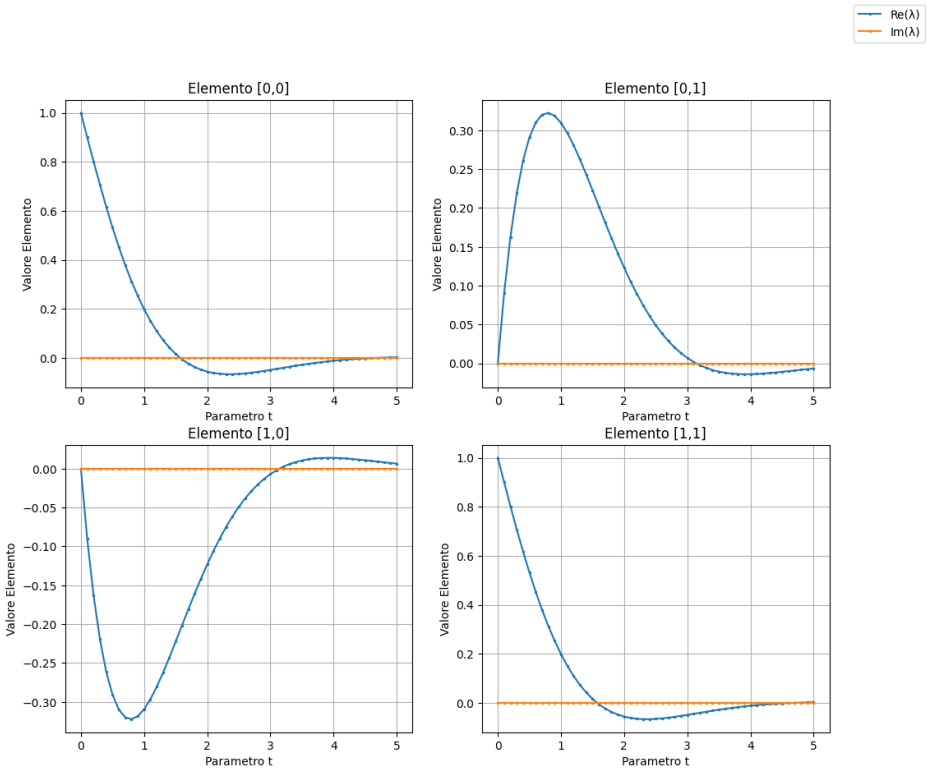


Figura 1: Valori degli elementi dell'esponenziale di matrice al variare del parametro t: in blu la parte reale in arancione la parte immaginaria.

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 $\lambda/\sqrt{n}$ si concentrano principalmente lungo l'asse reale, mentre per n grandi tendono a distribuirsi uniformemente nel disco unitario.

Parte fondamentale del codice:

```
Ns = np.array([2, 10, 50, 200]) # valori fissati di n
Nmat = 200 # numero di matrici generate per ciascun n

fig, axs = plt.subplots(2, 2, figsize=(10, 10))

for ax, n in zip(axs.ravel(), Ns):
    alleig = []
    for i in range(Nmat):
        X = np.random.normal(loc=0.0, scale=1.0, size=(n, n))
        eigv = np.linalg.eigvals(X)
        scaled = eigv / np.sqrt(n)
        alleig.extend(scaled)

    alleig = np.array(alleig)

    ax.scatter(alleig.real, alleig.imag, s=0.2, alpha=0.8, color="purple")
```

Si riportano i risultati grafici che confermano il risultato atteso; ovvero che per piccoli n la distribuzione è sull'asse reale, mentre all'aumentare di n la distribuzione tende all'uniforme nel cerchio unitario:

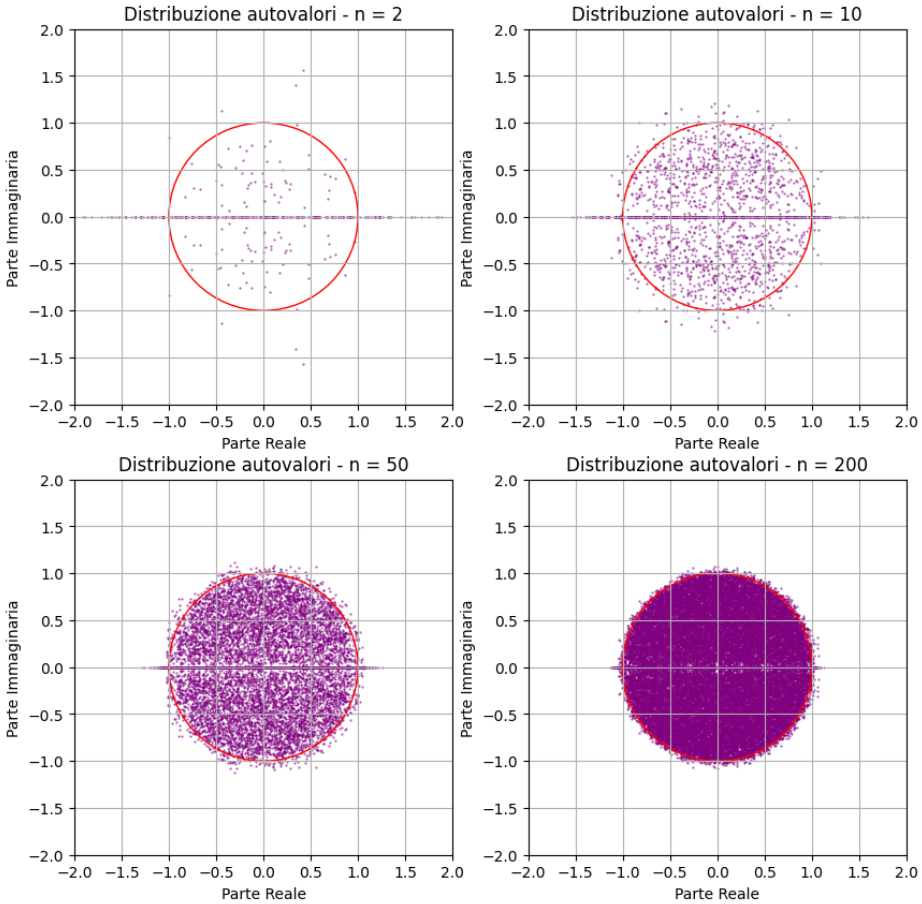


Figura 2: Distribuzione degli autovalori per vari n sul piano di Gauss. In rosso il cerchio unitario.

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:

```
def inertia_tensor(masses, positions):
    x = positions[:, 0]
    y = positions[:, 1]
    z = positions[:, 2]

    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)
    Ixz = -np.sum(masses * x * z)
    Iyz = -np.sum(masses * y * z)

    return np.array([[Ixx, Ixy, Ixz],
                     [Ixy, Iyy, Iyz],
                     [Ixz, Iyz, Izz]])
```

```
def clas(eigval, counters, tol=1e-1):
    eigval=np.sort(eigval)
    if np.allclose(eigval[0], eigval[1], atol=tol) and np.allclose(eigval[1], eigval[2], atol=tol):
        counters['sphere'] += 1
    elif np.allclose(eigval[0], eigval[1], atol=tol) and not np.allclose(eigval[1], eigval[2], atol=tol):
        counters['obl'] += 1
    elif np.allclose(eigval[1], eigval[2], atol=tol) and not np.allclose(eigval[0], eigval[1], atol=tol):
        counters['prol'] += 1
    else:
        counters['asim'] += 1
```

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


counters = {'sphere':0, 'obl':0, 'prol':0, 'asim':0}

DataSet = read("dataset.xyz", index=":")
for mol in DataSet:
    masses = mol.get_masses() # array delle masse della molecola
    positions = mol.get_positions() # array Nx3 delle coordinate
    print("Masse:", masses)
    print("Posizioni:", positions)
    com = np.average(positions, axis=0, weights=masses) #centro di massa
    print("Centro di massa:", com)
    pos_com = positions - com
    I = inertia_tensor(masses, pos_com)
    eigval, _ = np.linalg.eig(I)
    print(eigval)
    clas(eigval, counters)

print(counters)
```

Il risultato ottenuto è che il dataset di molecole contiene: 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

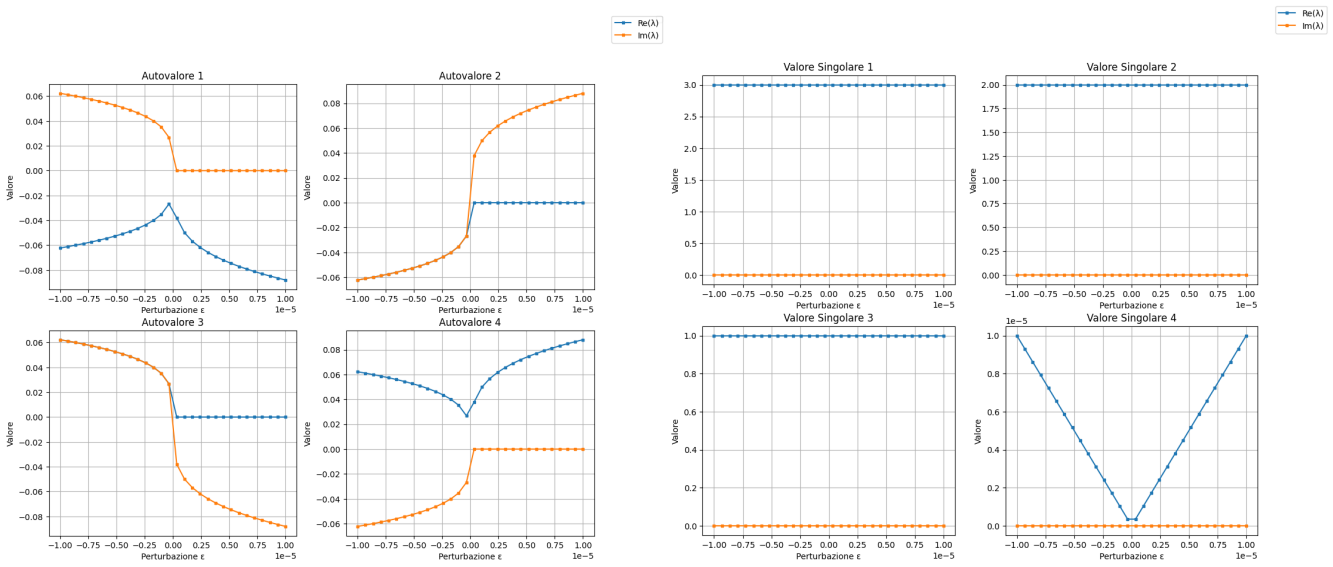
L'obiettivo dell'esericizio era lo studio degli autovalori e dei valori singolari della matrice A al variare di ε:

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

Parte fondamentale del codice:

```
delta = 0.00001
eps_vals=np.linspace(-delta, delta, 30)
alleigval=[]
allsingular=[]
for eps in eps_vals:
    A = np.array([[0, 1, 0, 0],[0, 0,2,0],[0,0,0,3],[eps,0,0,0]])
    U, S, Vh = np.linalg.svd(A, full_matrices=True)
    eigval,_=np.linalg.eig(A)
    alleigval.append(eigval)
    allsingular.append(S)
alleigval = np.array(alleigval)
allsingular=np.array(allsingular)

print(alleigval)
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]



(a) *(b)*

Figura 3: (a) Autovalori (b) Valori Singolari in funzione della perturbazione ε. In blu la parte reale in arancione la parte immaginaria.

Concludiamo che la stabilità degli autovalori è presente solo per valori positivi di ε ed esclusivamente per una fra le due parti (reale o immaginaria). Inoltre possiamo notare che gli autovalori sono legati fra loro da relazioni di segno. Invece, per i valori singolari, la stabilità è presente per tutti eccetto l'ultimo, il quale potrebbe avere una dipendenza lineare dalla perturbazione.

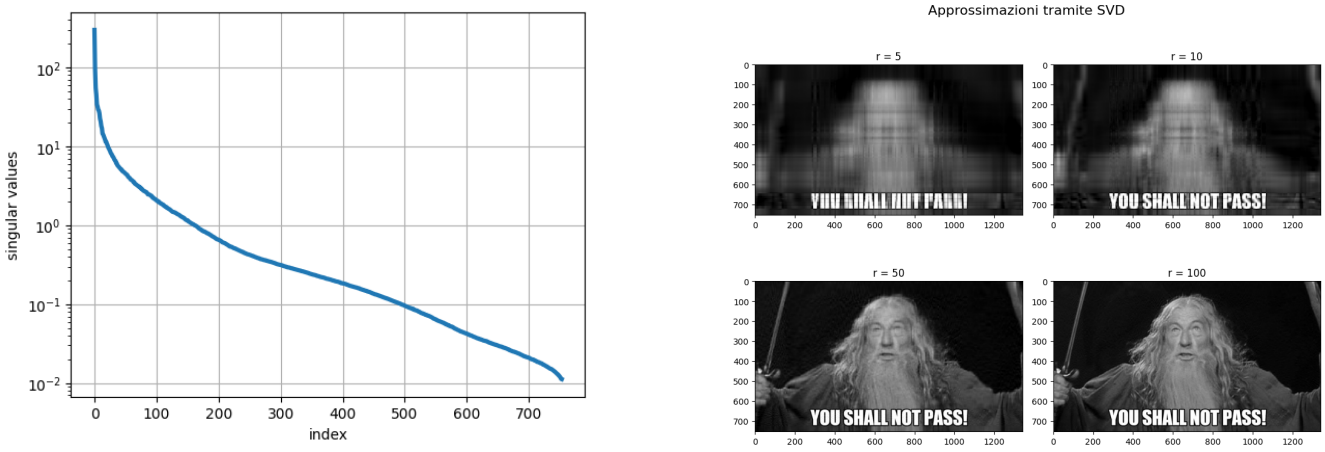
2.2 Punto b

L'obiettivo dell'esercizio consisteva 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:

```
U, S, Vh = np.linalg.svd(X, full_matrices=True)

fig, ax = plt.subplots()
plt.grid('True')

ax.set_xlabel('index')
ax.set_ylabel('singular values')

plt.semilogy()
ax.plot(S, lw=3)
```

```
r_val = np.array([5, 10, 50, 100])

fig, axs = plt.subplots(2, 2, figsize=(10, 8))

for ax, r in zip(axs.flat, r_val):
    U_tilde = U[:, :r]
    S_tilde = S[:r]
    Vh_tilde = Vh[:r, :]

    X_tilde = U_tilde @ np.diag(S_tilde) @ Vh_tilde

    im = ax.imshow(X_tilde, cmap="gray", vmin=0, vmax=1)
    ax.set_title(f"r = {r}")

fig.suptitle("Approssimazioni tramite SVD", fontsize=16)
plt.tight_layout()
plt.show()
```



(a) *(b)*

Figura 4: (a) Grafico dei valori singolari (b) Approssimazione tramite SVD

Concludiamo che è possibile, grazie alla SVD, di ricreare l'immagine con buona approssimazione visiva senza utilizzare la totalità dei valori singolari.