Lab 3 : Interpolazione, approssimazione e SVD

1) Approssimazione di un set di dati tramite Minimi Quadrati

Sia $\{(x_i, y_i)\}_{i=1}^{N}$ un set di dati, che devono essere approssimati da un polinomio

$$p(x) = \alpha_0 + \alpha_1 x + \cdots + \alpha_n x^n$$

**di grado $n \in \mathbb{N}$ fissato. **

Si definisce una matrice

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^n \end{bmatrix}$$

E i vettori

$$\alpha = \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_n \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ \vdots \\ y_N \end{bmatrix}$$

\\

Reimpostando il problema con la formulazione ai minimi quadrati e risolvendo quindi il problema

$$\min_{\alpha} ||A\alpha - y||_2^2$$

**si ottengono i coefficenti α che definiscono in modo univoco il polinomio interpolante $p(x)$. **

- **Calcolare il polinomio di grado $n = 5$ che approssimi i seguenti dati:** $\{(1.0, 1.18), (1.2, 1.26), (1.4, 1.23), (1.6, 1.37), (1.8, 1.37), (2.0, 1.45), (2.2, 1.42), (2.4, 1.46), (2.6, 1.53), (2.8, 1.59), (3.0, 1.50)\}$

- **Risolvere il problema ai minimi quadrati sia con le equazioni normali che con la SVD.**
- **Valutare graficamente i polinomi di approssimazione e confrontare gli errori commessi dai due metodi sul set di punti.**

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:

```
n = 5 # Grado del polinomio approssimante

x = np.array([1, 1.2, 1.4, 1.6, 1.8, 2, 2.2, 2.4, 2.6, 2.8, 3])
y = np.array([1.18, 1.26, 1.23, 1.37, 1.37, 1.45, 1.42, 1.46, 1.53, 1.59, 1.5])

print('Shape of x:', x.shape)
print('Shape of y:', y.shape, "\n")

N = x.size # Numero dei dati

A = np.zeros((N, n+1))

for i in range(n+1):
  A[:, i] = # TODO

print("A = \n", A)
```

```
Shape of x: (11,)
Shape of y: (11,)

A =
 [[  1.          1.          1.          1.          1.          1.     ]
 [  1.          1.2         1.44        1.728       2.0736      2.48832]
 [  1.          1.4         1.96        2.744       3.8416      5.37824]
 [  1.          1.6         2.56        4.096       6.5536     10.48576]
 [  1.          1.8         3.24        5.832      10.4976     18.89568]
 [  1.          2.          4.          8.         16.         32.     ]
 [  1.          2.2         4.84       10.648      23.4256     51.53632]
 [  1.          2.4         5.76       13.824      33.1776     79.62624]
 [  1.          2.6         6.76       17.576      45.6976    118.81376]
 [  1.          2.8         7.84       21.952      61.4656    172.10368]
 [  1.          3.          9.         27.         81.        243.     ]]
```

Risoluzione tramite equazioni normali

Il problema ai minimi quadrati

$$\min_{\alpha} ||A\alpha - y||_2^2$$

può essere risolto col metodo delle equazioni normali, ossia osservando che il problema di minimo può essere riscritto come:

$$A^T A\alpha = A^T y$$

Risolvendo questo sistema lineare (ad esempio con fattorizzazione di Cholesky o con metodi iterativi) si ottiene il vettore degli α che corrisponde ai coefficenti del polinomio approssimante.

In [ ]:

```
import scipy.linalg
import scipy.linalg.decomp_lu as LUdec

# Per chiarezza, calcoliamo la matrice del sistema e il termine noto a parte
ATA = # TODO
ATy = # TODO

lu, piv = LUdec.lu_factor(ATA)
```

```python
print('LU = \n', lu)
print('piv = ', piv)

alpha_normali = LUdec.lu_solve((lu, piv), ATy)
```

```
LU = 
 [[ 7.35328000e+02  1.95059392e+03  5.27874688e+03  1.45060095e+04
   4.03459360e+04  1.13308832e+05]
 [ 3.87218765e-01 -1.99785686e+01 -9.34359274e+01 -3.38252200e+02
  -1.11669401e+03 -3.52936988e+03]
 [ 2.99186213e-02  4.98488209e-01  3.04387901e+00  1.93477282e+01
   8.48920156e+01  3.19899161e+02]
 [ 1.55576831e-01  9.37725871e-01  5.56715647e-01  2.11588465e-01
   1.74616541e+00  9.26911097e+00]
 [ 1.49593107e-02  3.59362103e-01  9.89171005e-01 -8.65322309e-01
   2.11931056e-02  2.13563205e-01]
 [ 6.58209670e-02  7.00249264e-01  8.89987853e-01  8.03632732e-01
  -3.10453039e-01 -3.53025415e-04]]
piv =  [5 4 4 3 5 5]
```

Risoluzione tramite SVD

Consideriamo la decomposizione SVD della matrice A

$$A = USV^T$$

Con $U \in \mathbb{R}^{N \times N}$ **e** $V^T \in \mathbb{R}^{n \times n}$ **matrici unitarie e** $S \in \mathbb{R}^{N \times n}$ **diagonale. **

Le equazioni normali diventano:

$$A^T A\alpha = A^T y$$
$$\iff$$
$$VSU^T USV^T \alpha$$
$$= VSU^T y \iff$$
$$VS^2 V^T \alpha$$
$$= VSU^T y \iff$$
$$SV^T \alpha = U^T y$$
$$\iff$$
$$\alpha = S^{-1} VU^T y$$

E quindi

$$\alpha_i = \sum_{j=1}^{N} \frac{(u_i^T y) v_i}{s_i}$$

In [ ]:

```python
help(scipy.linalg.svd)
```

```
Help on function svd in module scipy.linalg.decomp_svd:

svd(a, full_matrices=True, compute_uv=True, overwrite_a=False, check_finite=True, lapack_
driver='gesdd')
    Singular Value Decomposition.

    Factorizes the matrix `a` into two unitary matrices ``U`` and ``Vh``, and
    a 1-D array ``s`` of singular values (real, non-negative) such that
    ``a == U @ S @ Vh``, where ``S`` is a suitably shaped matrix of zeros with
    main diagonal ``s``.

    Parameters
    ----------
    a : (M, N) array_like
        Matrix to decompose.
    full_matrices : bool, optional
        If True (default), `U` and `Vh` are of shape ``(M, M)``, ``(N, N)``
```

```
    If True (default), `u` and `vh` are of shape ``(M, M)``, ``(N, N)``.
    If False, the shapes are ``(M, K)`` and ``(K, N)``, where
    ``K = min(M, N)``.
compute_uv : bool, optional
    Whether to compute also ``U`` and ``Vh`` in addition to ``s``.
    Default is True.
overwrite_a : bool, optional
    Whether to overwrite `a`; may improve performance.
    Default is False.
check_finite : bool, optional
    Whether to check that the input matrix contains only finite numbers.
    Disabling may give a performance gain, but may result in problems
    (crashes, non-termination) if the inputs do contain infinities or NaNs.
lapack_driver : {'gesdd', 'gesvd'}, optional
    Whether to use the more efficient divide-and-conquer approach
    (``'gesdd'``) or general rectangular approach (``'gesvd'``)
    to compute the SVD. MATLAB and Octave use the ``'gesvd'`` approach.
    Default is ``'gesdd'``.

    .. versionadded:: 0.18

Returns
-------
U : ndarray
    Unitary matrix having left singular vectors as columns.
    Of shape ``(M, M)`` or ``(M, K)``, depending on `full_matrices`.
s : ndarray
    The singular values, sorted in non-increasing order.
    Of shape (K,), with ``K = min(M, N)``.
Vh : ndarray
    Unitary matrix having right singular vectors as rows.
    Of shape ``(N, N)`` or ``(K, N)`` depending on `full_matrices`.

For ``compute_uv=False``, only ``s`` is returned.

Raises
------
LinAlgError
    If SVD computation does not converge.

See also
--------
svdvals : Compute singular values of a matrix.
diagsvd : Construct the Sigma matrix, given the vector s.

Examples
--------
>>> from scipy import linalg
>>> m, n = 9, 6
>>> a = np.random.randn(m, n) + 1.j*np.random.randn(m, n)
>>> U, s, Vh = linalg.svd(a)
>>> U.shape,  s.shape, Vh.shape
((9, 9), (6,), (6, 6))

Reconstruct the original matrix from the decomposition:

>>> sigma = np.zeros((m, n))
>>> for i in range(min(m, n)):
...     sigma[i, i] = s[i]
>>> a1 = np.dot(U, np.dot(sigma, Vh))
>>> np.allclose(a, a1)
True

Alternatively, use ``full_matrices=False`` (notice that the shape of
``U`` is then ``(m, n)`` instead of ``(m, m)``):

>>> U, s, Vh = linalg.svd(a, full_matrices=False)
>>> U.shape, s.shape, Vh.shape
((9, 6), (6,), (6, 6))
>>> S = np.diag(s)
>>> np.allclose(a, np.dot(U, np.dot(S, Vh)))
True
```

```
>>> s2 = linalg.svd(a, compute_uv=False)
>>> np.allclose(s, s2)
True
```

In [ ]:

```
U, s, Vh = # TODO

print('Shape of U:', U.shape)
print('Shape of s:', s.shape)
print('Shape of V:', Vh.T.shape)

alpha_svd = np.zeros(s.shape)

for i in range(n+1):
  ui = # TODO
  vi = # TODO

  alpha_svd = alpha_svd + # TODO
```

```
Shape of U: (11, 11)
Shape of s: (6,)
Shape of V: (6, 6)
```

Verifica e confronto delle soluzioni

In [ ]:

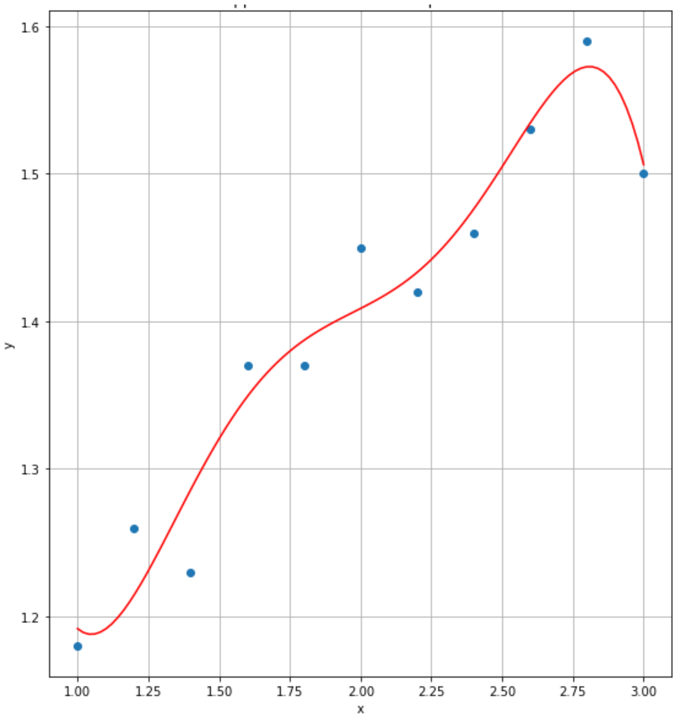
```
def p(alpha, x):
  N = len(x)
  n = len(alpha)

  A = np.zeros((N,n))

  for i in range(n):
    # TODO

  return # TODO
```

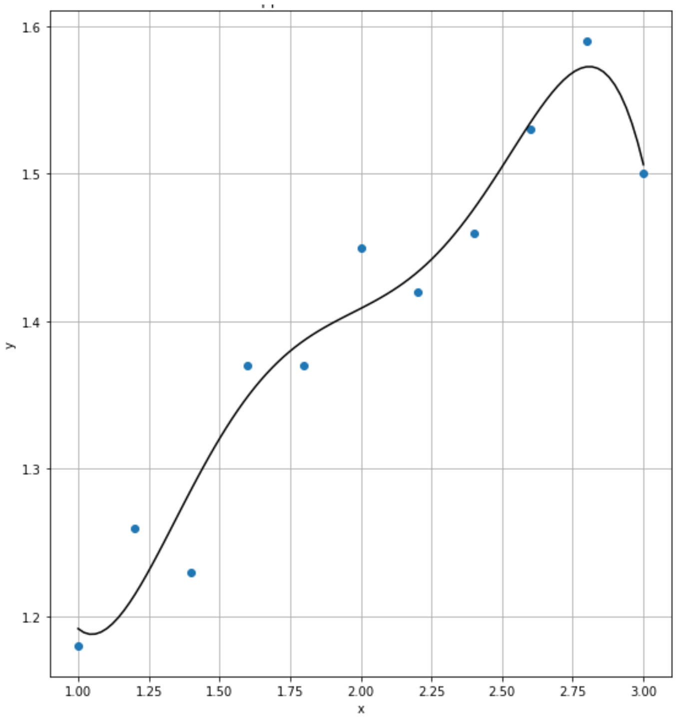
In [ ]:

```
# VETTORE PUNTI PER IL GRAFICO
x_plot = np.linspace(1, 3, 100)

#VALUTAZIONE POLINOMIO NEI PUNTI X_PLOT
y_normali = p(alpha_normali, x_plot)
y_svd = p(alpha_svd, x_plot)

plt.figure(figsize=(20, 10))

plt.subplot(1, 2, 1)
plt.plot(x, y, 'o')
plt.plot(x_plot, y_normali, 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Approssimazione tramite Eq. Normali')
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(x, y, 'o')
plt.plot(x_plot, y_svd, 'k')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Approssimazione tramite SVD')
plt.grid()

plt.show()
```

Approssimazione tramite Eq. Normali

Approssimazione tramite SVD





2) Import dataset da Kaggle

**Come secondo esercizio andremo ad eseguire approssimazione polinomiale su un dataset caricato dall'esterno. Nello specifico, utilizzeremo un data set di Kaggle (www.kaggle.com) contenente dati riguardanti gli anni di esperienza e lo stipendio di alcuni individui (nello specifico 30). Il data set è scaricabile al seguente indirizzo: https://www.kaggle.com/karthickveerakumar/salary-data-simple-linear-regression/ . **

Una volta scaricato, è necessario caricarlo su Colab. Per leggere il file utilizzeremo una libreria chiamata `pandas` molto utilizzata quando si lavora coi dati. La funzione per caricarlo è `pandas.read_csv` che darà come output il dataset, che dovrà successivamente essere convertito in numpy array.

In [ ]:

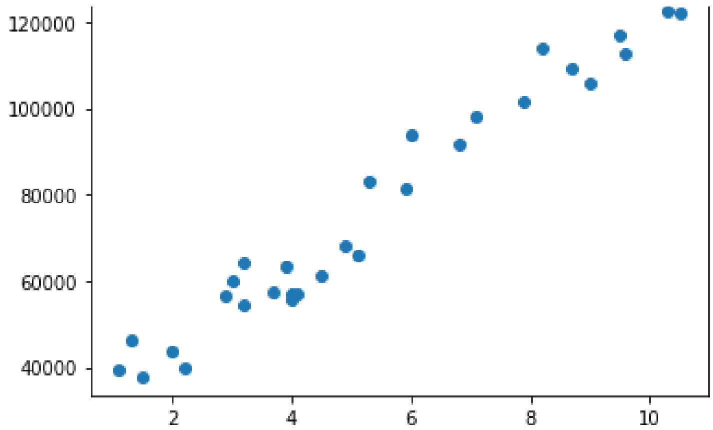
```
import pandas as pd
data = # TODO
data = np.array(data)

print(data.shape)

plt.plot(x,y, 'o')
plt.show()
```

```
(30, 2)
```

In [ ]:

```
x = data[:, 0]
y = data[:, 1]

print(x.shape)
print(y.shape)

n = 5
N = x.size

A = np.zeros((N, n+1))

for i in range(n+1):
  A[:, i] = # TODO
```

```
(30,)
(30,)
```



Equazioni Normali

In [ ]:

```
ATA = # TODO
ATy = # TODO

lu, piv = LUdec.lu_factor(ATA)

print('LU = \n', lu)
print('piv = ', piv)

alpha_normali = LUdec.lu_solve((lu, piv), ATy)
```

```
LU =
 [[ 6.45033578e+05  5.96456220e+06  5.63460131e+07  5.40475744e+08
   5.24431260e+09  5.13504684e+10]
 [ 1.11649414e-01 -2.09062959e+04 -3.26437140e+05 -3.99778703e+06
  -4.50486847e+07 -4.88937106e+08]
 [ 1.31121329e-02  2.96107509e-01  2.87765528e+03  6.15471755e+04
   9.21143105e+05  1.19395264e+07]
 [ 1.67510659e-03  7.33516580e-02  5.47928190e-01 -8.00030098e+02
  -2.05449828e+04 -3.49150920e+05]
 [ 4.65092067e-05  5.64457025e-03  1.05116484e-01  7.28928847e-01
   5.37774269e+02  1.60690157e+04]
 [ 2.47118918e-04  1.88199841e-02  2.35310330e-01  9.85660222e-01
   6.99433595e-01  8.96366625e+01]]
piv =  [5 4 3 3 5 5]
```

SVD

In [ ]:

```
U, s, Vh = # TODO

print('Shape of U:', U.shape)
print('Shape of s:', s.shape)
print('Shape of V:', Vh.T.shape)

alpha_svd = 0

for i in range(n+1):
  ui = # TODO
  vi = # TODO

  alpha_svd = alpha_svd + # TODO
```

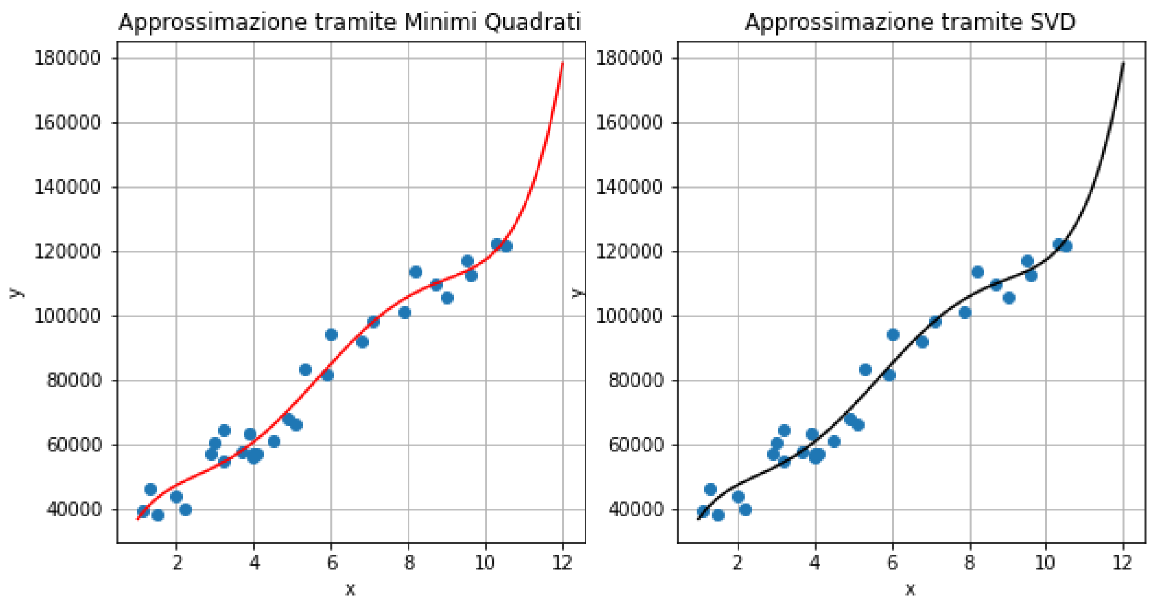
```
Shape of U: (30, 30)
Shape of s: (6,)
Shape of V: (6, 6)
```

Visualizzazione risultati

Visualizzazione risultati

In [ ]:

```
x_new = np.linspace(1, 12, 100)
y_normali = p(alpha_normali, x_new)
y_svd = p(alpha_svd, x_new)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.plot(x, y, 'o')
plt.plot(x_new, y_normali, 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Approssimazione tramite Minimi Quadrati')
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(x, y, 'o')
plt.plot(x_new, y_svd, 'k')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Approssimazione tramite SVD')
plt.grid()

plt.show()
```



3) Compressione di una immagine tramite SVD

Caricare e visualizzare un'immagine A in scala di grigio, di dimensione $m \times n$. Poi:

- **Calcolare la matrice $A_p = \sum_{i=1}^{p} u_i * v_i^T * \sigma_i$, dove $p \leq rango(A)$**
- **Visualizzare le immagini A_p ottenute al variare di p, considerando i valori singolari in ordine prima crescente poi decrescente.**
- **Calcolare l' errore relativo:**

$$\frac{\|A - A_p\|_2}{\|A\|_2}$$

  e plottarlo al variare di p.
- **Calcolare il fattore di compressione**

$$c_p = \frac{1}{p}\min(m, n) - 1$$

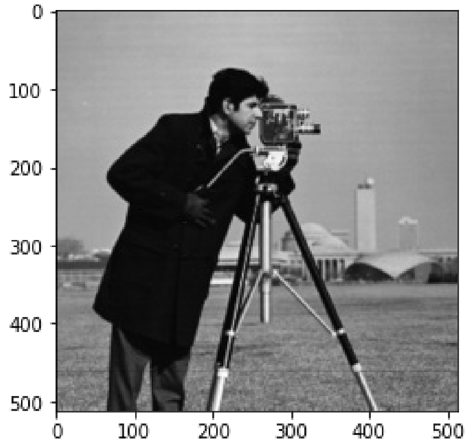
  e plottarlo al variare di p.

In [ ]:

```
from skimage import data

#help(data)
```

In [ ]:

```
A = data.camera()

plt.imshow(A, cmap='gray')
plt.show()
```



In [ ]:

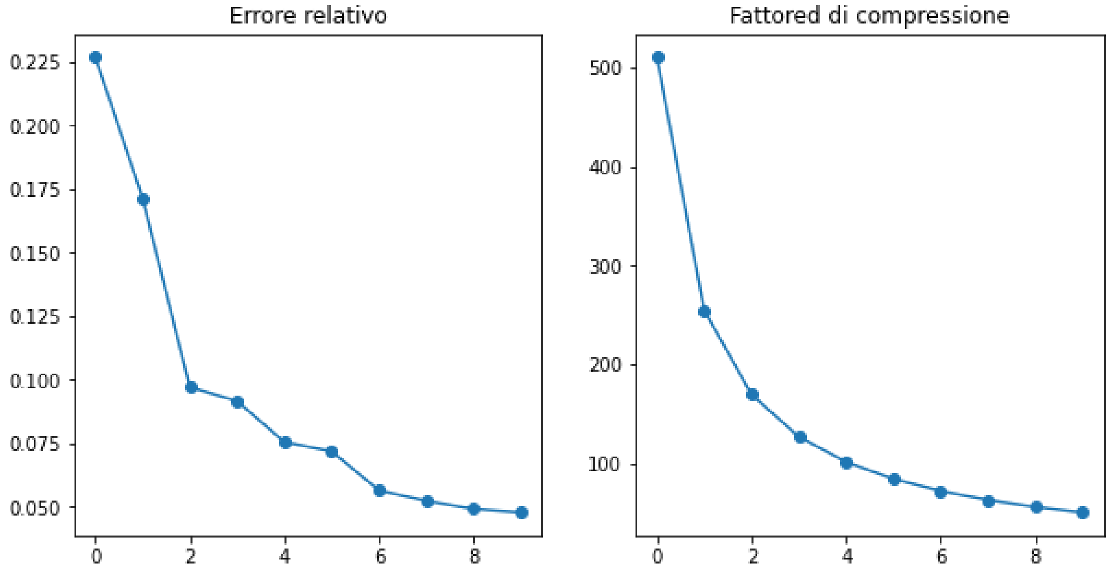
```
U, s, Vh = scipy.linalg.svd(A)

print('Shape of U:', U.shape)
print('Shape of s:', s.shape)
print('Shape of V:', Vh.T.shape)


A_p = np.zeros(A.shape)
p_max = 10

err_rel = np.zeros((p_max))
c = np.zeros((p_max))

for i in range(p_max):
  ui = # TODO
  vi = # TODO

  A_p = A_p + # TODO

  err_rel[i] = # TODO
  c[i] = # TODO

print('\n')
print('L\'errore relativo della ricostruzione di A è', err_rel[-1])
print('Il fattore di compressione è c=', c[-1])
```

```
Shape of U: (512, 512)
Shape of s: (512,)
Shape of V: (512, 512)


L'errore relativo della ricostruzione di A è 0.0477951552426975
Il fattore di compressione è c= 50.2
```

In [ ]:

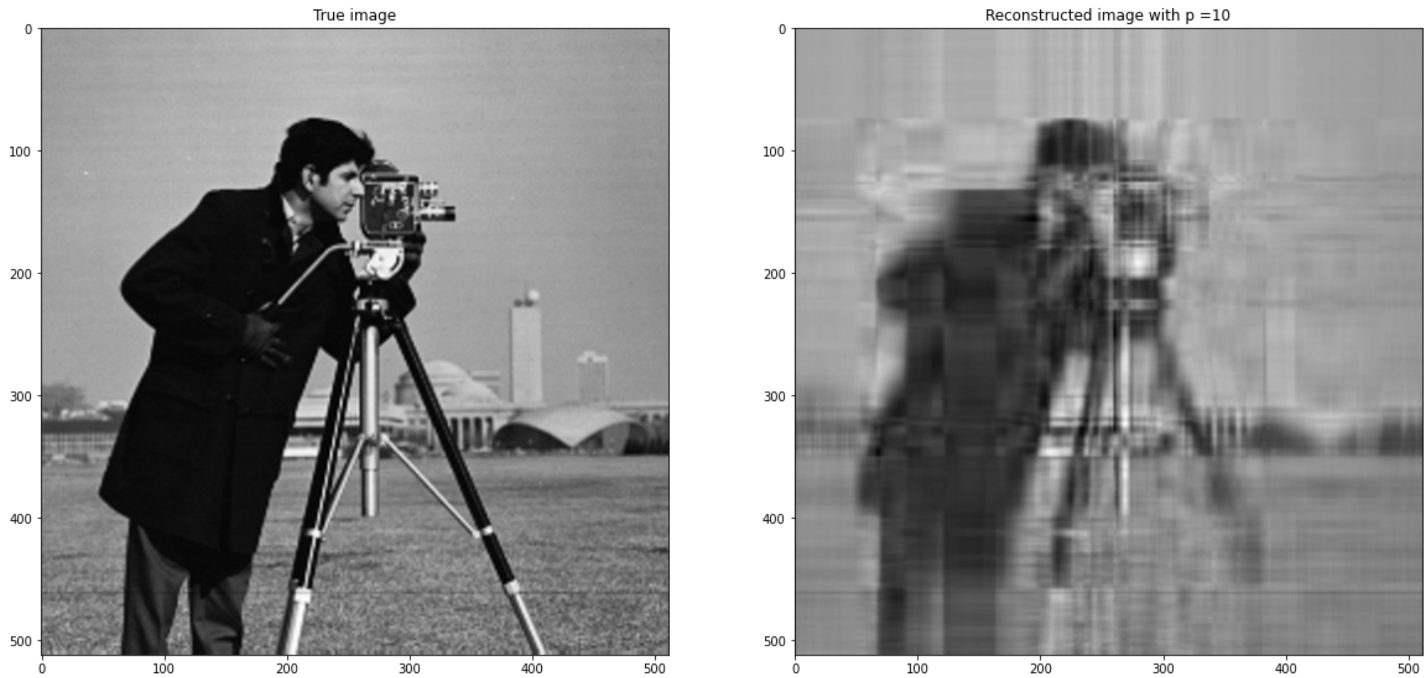
```
plt.figure(figsize=(10, 5))

fig1 = plt.subplot(1, 2, 1)
```

```python
fig1.plot(err_rel, 'o-')
plt.title('Errore relativo')

fig2 = plt.subplot(1, 2, 2)
fig2.plot(c, 'o-')
plt.title('Fattore di compressione')

plt.show()
```



In [ ]:

```python
plt.figure(figsize=(20, 10))

fig1 = plt.subplot(1, 2, 1)
fig1.imshow(A, cmap='gray')
plt.title('True image')

fig2 = plt.subplot(1, 2, 2)
fig2.imshow(A_p, cmap='gray')
plt.title('Reconstructed image with p =' + str(p_max))

plt.show()
```



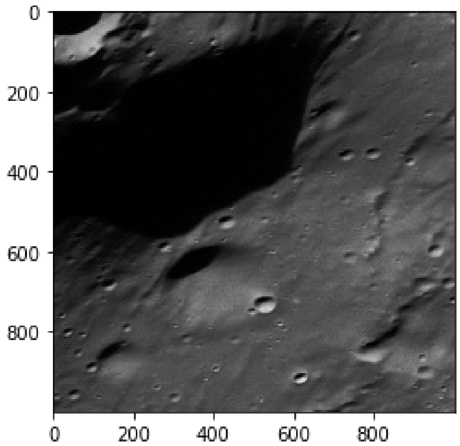
Caricamento immagine da file

Ripetere lo stesso esercizio caricando l'immagine l'immagine
https://upload.wikimedia.org/wikipedia/commons/d/d8/Stereo-1_channel_image_of_Phobos_ESA214117.jpg

In [ ]:

```
A = plt.imread('Stereo-1_channel_image_of_Phobos_ESA214117.jpg')

A = A[3000:4000, 5000:6000]

print(A.shape)

plt.imshow(A, cmap='gray')
plt.show()
```

```
(1000, 1000)
```



In [ ]:

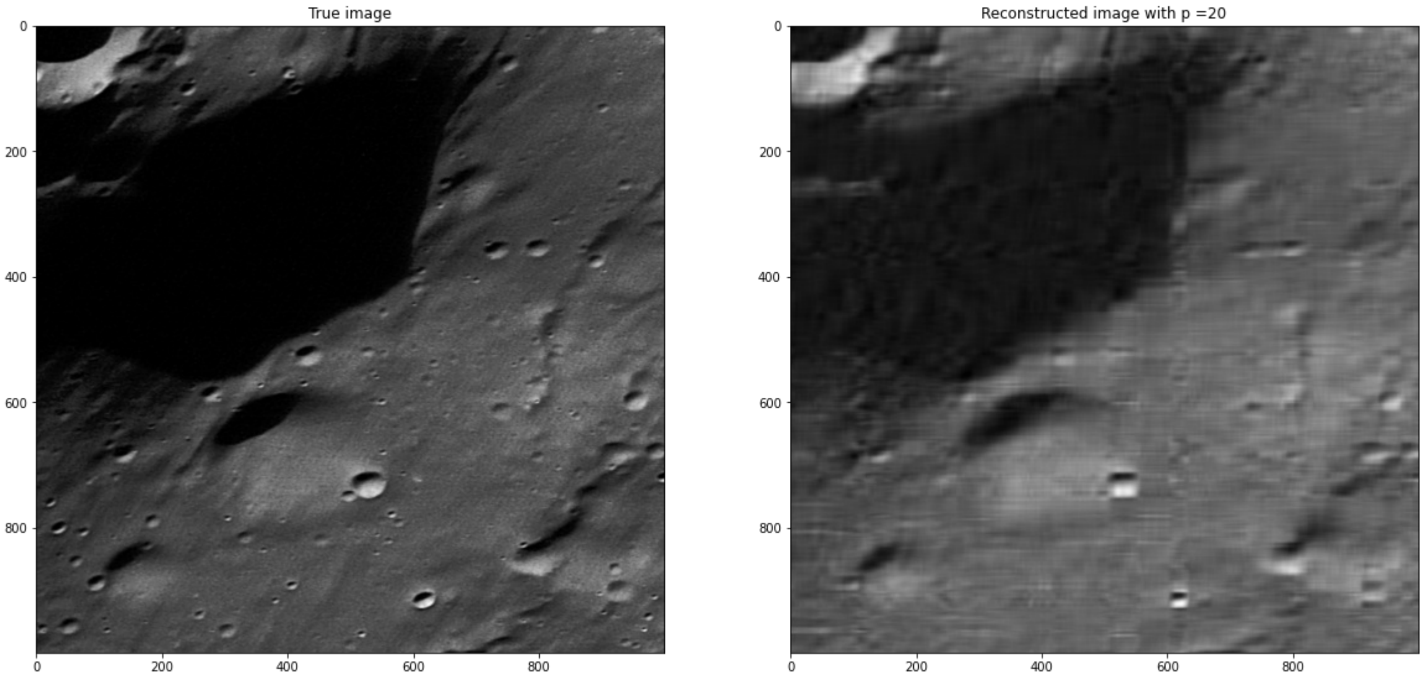
```
U, s, Vh = # TODO

print('Shape of U:', U.shape)
print('Shape of s:', s.shape)
print('Shape of V:', Vh.T.shape)


A_p = np.zeros(A.shape)
p_max = 20

err_rel = np.zeros((p_max))
c = np.zeros((p_max))

for i in range(p_max):
  ui = # TODO
  vi = # TODO

  A_p = A_p + # TODO

  err_rel[i] = # TODO
  c[i] = # TODO

print('\n')
print('L\'errore relativo della ricostruzione di A è', err_rel[-1])
print('Il fattore di compressione è c=', c[-1])
```

```
Shape of U: (1000, 1000)
Shape of s: (1000,)
Shape of V: (1000, 1000)


L'errore relativo della ricostruzione di A è 0.03314248279311608
Il fattore di compressione è c= 49.0
```

In [ ]:

```
plt.figure(figsize=(20, 10))

fig1 = plt.subplot(1, 2, 1)
fig1.imshow(A, cmap='gray')
```

```python
plt.title('True image')

fig2 = plt.subplot(1, 2, 2)
fig2.imshow(A_p, cmap='gray')
plt.title('Reconstructed image with p =' + str(p_max))

plt.show()
```



In [ ]:

```python

```