

1 Problema minimi quadrati

Data una matrice A di dimensioni $m \times n$ con $m \geq n$ ed un vettore y di m componenti, il problema dei minimi quadrati è il seguente:

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}} \|A\alpha - y\|_2^2$$

Questo problema di minimo può essere risolto in due modi:

- **Metodo delle equazioni normali.** se A ha rango massimo il problema di minimo può essere riscritto in maniera equivalente come segue:

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

Questo sistema si può risolvere utilizzando la fattorizzazione LU o di Cholesky (dato che la matrice $A^T A$ è simmetrica).

- **SVD.** se A non ha rango massimo il problema è sottodeterminato quindi avere più di una soluzione. In questo caso si considera la decomposizione SVD della matrice $A = USV^T$ dove $U \in \mathbb{R}^{m \times m}$ e $V^T \in \mathbb{R}^{n \times n}$ matrici ortogonali e $S \in \mathbb{R}^{m \times n}$ diagonale. e da questa decomposizione si può calcolare esplicitamente la soluzione di minima norma del problema di minimi quadrati come segue:

$$\alpha = \sum_{i=0}^r \frac{(u_i^T y) v_i}{s_i},$$

Exercise 1.1. Assegnata una matrice A di numeri casuali di dimensione $m \times n$ con $m > n$, generata utilizzando la funzione `np.random.rand`, scegliere un vettore α (per esempio con elementi costanti) come soluzione per creare un problema test e calcolare il termine noto $y = A\alpha$.

Definito quindi il problema di minimi quadrati con la matrice A ed il termine noto y calcolato:

Genero la matrice casuale $m \times n$ di numeri casuali e calcolo il termine noto $y = A@\alpha$

```
m = 100
n = 10
A = np.random.rand(m, n)      # matrice casuale m x n
alpha = np.ones(n)            # alpha con elementi costanti (ad esempio tutti uno)
y = A@alpha
print('alpha test', alpha)
```

Output:

```
alpha test: [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Calcolare la soluzione del problema risolvendo le equazioni normali mediante la fattorizzazione LU e Cholesky:

```
ATA = A.T@A
ATy = A.T@y

lu, piv = LUdec(ATA)
alpha_LU = scipy.linalg.lu_solve((lu,piv), ATy)
print("Soluzione con fattorizzazione LU:", alpha_LU)

L = scipy.linalg.cholesky(ATA)
x = scipy.linalg.solve_triangular(np.transpose(L), ATy, lower=True)
alpha_chol = scipy.linalg.solve_triangular(L, x, lower=False)
print("Soluzione con Cholesky:", alpha_chol)
```

Output:

```
Soluzione con fattorizzazione LU: [1. 1. 1. 1. 1. 1. 1. 1. 1.]
Soluzione con Cholesky: [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Calcolare la soluzione del problema usando la SVD della matrice A:

```
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)

alpha_svd = np.zeros(s.shape)
for i in range(n):
    ui = U[:, i] #accesso alla colonna i
    vi = Vh[i, :] #accesso alla riga i
    alpha_svd = alpha_svd + np.dot(ui, y) * vi / s[i]

print("Soluzione con SVD:", alpha_svd)
```

Output:

```
Shape of U: (100, 100)
Shape of s: (10,)
Shape of V: (10, 10)
Soluzione con SVD: [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Calcolare l'errore relativo delle soluzioni trovate, rispetto al vettore α , soluzione esatta, utilizzata per generare il problema test:

```
relative_error_svd = np.linalg.norm(alpha_svd - alpha) / np.linalg.norm(alpha)
print("Errore relativo:", relative_error_svd)
```

Output:

```
Errore relativo: 1.7543631680830378e-15
```

2 Approssimazione di un set di dati tramite Minimi Quadrati

Sia $\{(x_i, y_i)\}_{i=0}^m$ un set di dati, che devono essere approssimati da un polinomio di grado $n \in \mathbb{N}$ fissato.

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

Per calcolare i coefficienti del polinomio si deve risolvere un problema di minimi quadrati in cui la matrice A è definita come segue:

$$A = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{bmatrix}$$

Mentre il termine noto è:

$$y = \begin{bmatrix} y_0 \\ \vdots \\ y_m \end{bmatrix} \quad (1)$$

Exercise 2.1. Date le seguenti funzioni:

- $f(x) = \exp(x/2) \quad x \in [-1, 1]$
- $f(x) = \frac{1}{1+25*x^2} \quad x \in [-1, 1]$
- $f(x) = \sin(x) + \cos(x) \quad x \in [0, 2\pi]$

Si eseguano le seguenti richieste per ciascuna delle funzioni date:

Calcolare $m = 10$ coppie di punti $(x_i, f(x_i))$:

```
case = 1
m = 10
m_plot = 1000
grado_pol_approssimante = [1, 2, 3, 5, 7]
if case==0:
    x = np.linspace(-1,1,m)
    y = np.exp(x/2)
    x1 = np.linspace(-1,1,m_plot)
    y1 = np.exp(x1/2)
elif case==1:
    x = np.linspace(-1,1,m)
    y = 1/(1+25*(x**2))
    x1 = np.linspace(-1,1,m_plot)
    y1 = 1/(1+25*(x1**2))
elif case==2:
    x = np.linspace(0,2*np.pi,m)
    y = np.sin(x)+np.cos(x)
    x1 = np.linspace(0,2*np.pi,m_plot)
    y1 = np.sin(x1)+np.cos(x1)
```

In base a quale problema test si vuole andare a valutare si setta la variabile ‘case’ con valori fra 0 e 2.

Per n fissato calcolare una soluzione del problema di minimi quadrati, descritto sopra, utilizzando un metodo a scelta tra quelli utilizzati nell'esercizio precedente.

```
A = np.zeros((m, n+1))
for i in range(n+1):
    A[:, i] = x**i
U, s, Vh = scipy.linalg.svd(A)

alpha_svd = np.zeros(n+1)
for i in range(n+1):
    ui = U[:, i]
    vi = Vh[i, :]
    alpha_svd += ui@y * vi / s[i] #alpha_svd = alpha_svd + np.dot(ui, y) * vi / s[i]
print(alpha_svd)
```

Output con n fissato pari a 5 e case 0:

```
[1.00000060e+00 5.00000139e-01 1.24988092e-01 2.08320882e-02 2.63709877e-03 2.63076547e-04]
```

Output con n fissato pari a 5 e case 1:

```
[ 6.42191979e-01 -1.21704962e-15 -1.99646993e+00 4.23316618e-15 1.42497265e+00 -2.68306842e-15]
```

Output con n fissato pari a 5 e case 2:

```
[ 0.99315181 1.18619381 -0.77082962 -0.07787167 0.06176099 -0.00551072]
```

Per ciascun valore di $n \in \{1, 2, 3, 5, 7\}$, creare una figura con il grafico della funzione esatta $f(x)$ insieme a quello del polinomio di approssimazione $p(x)$, evidenziando gli m punti noti, e per ciascun valore di $n \in \{1, 2, 3, 5, 7\}$, calcolare e stampare il valore del residuo in norma 2 commesso nei punti xi.

```
x_plot = np.linspace(x[0], x[-1], m_plot)
A_plot = np.zeros((m_plot, n+1))

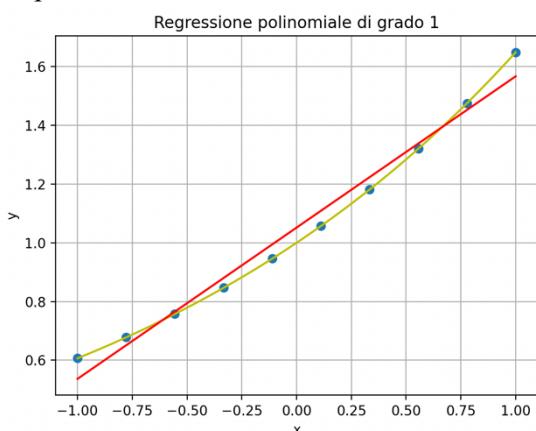
for i in range(n+1):
    A_plot[:, i] = x_plot**i

y_interpolation = A_plot@alpha_svd

plt.plot(x, y, 'o' )           #mostro gli m punti
plt.plot(x1, y1, '-y')         #per plot funzione
plt.plot(x_plot, y_interpolation, 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.title(f'Regressione polinomiale di grado {n}')
plt.grid()
plt.show()
```

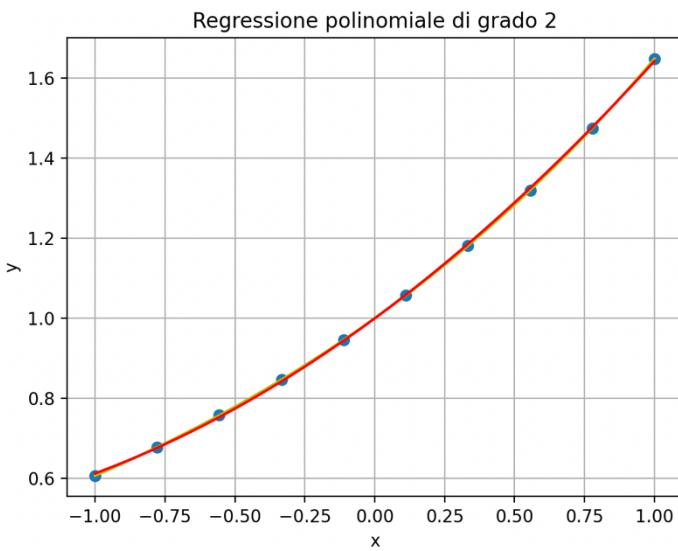
case 0:

Output per n=1:



Alpha svd: [1.05169894 0.51523297]
Residual: 0.14543338288469418

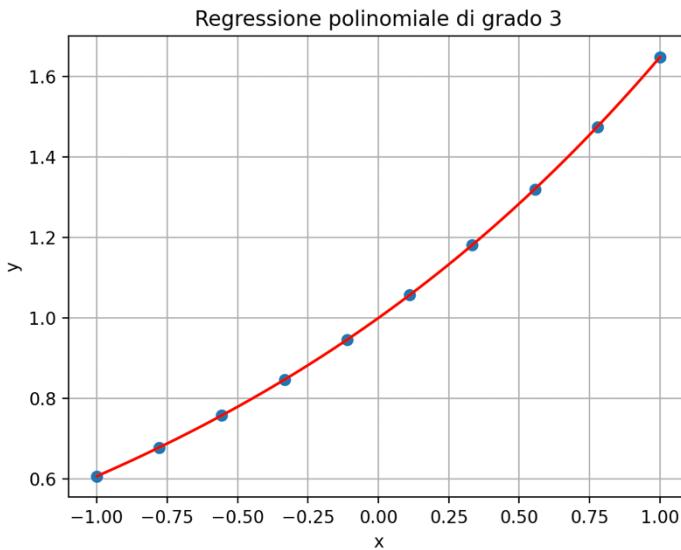
Output per n=2



Alpha svd: [0.99969023 0.51523297 0.12765775]

Residual: 0.01293656315783785

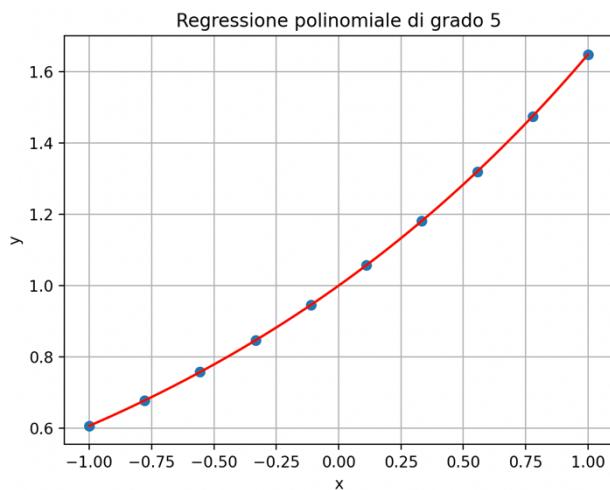
Output per n=3



Alpha svd: [0.99969023 0.49991905 0.12765775 0.0211677]

Residual: 0.0008263727821352719

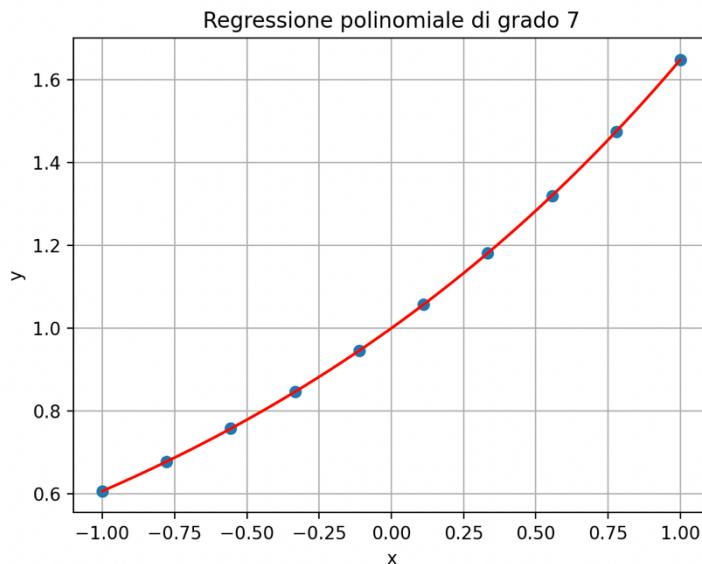
Output per n=5



Alpha svd: [1.00000060e+00 5.00000139e-01 1.24988092e-01 2.08320882e-02 2.63709877e-03 2.63076547e-04]

Residual: 1.4777299184732303e-06

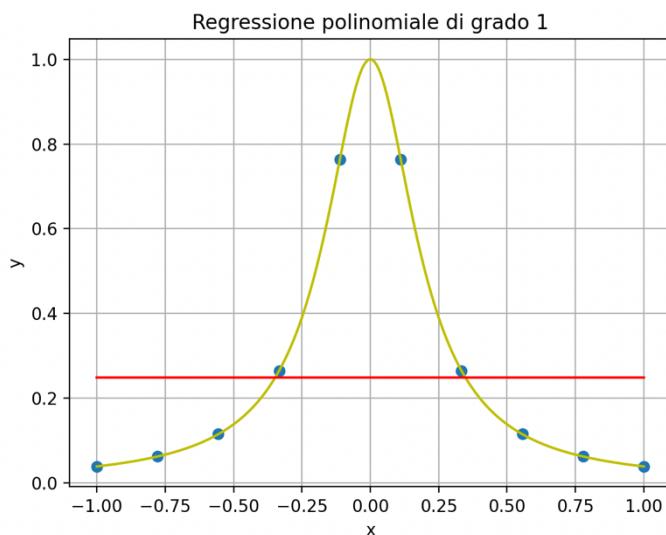
Output per n=7:



Alpha svd: [1.00000000e+00 5.00000000e-01 1.25000017e-01 2.08333348e-02 2.60406567e-03 2.60410030e-04
2.18828985e-05 1.56077950e-06]
Residual: 8.737823524609127e-10

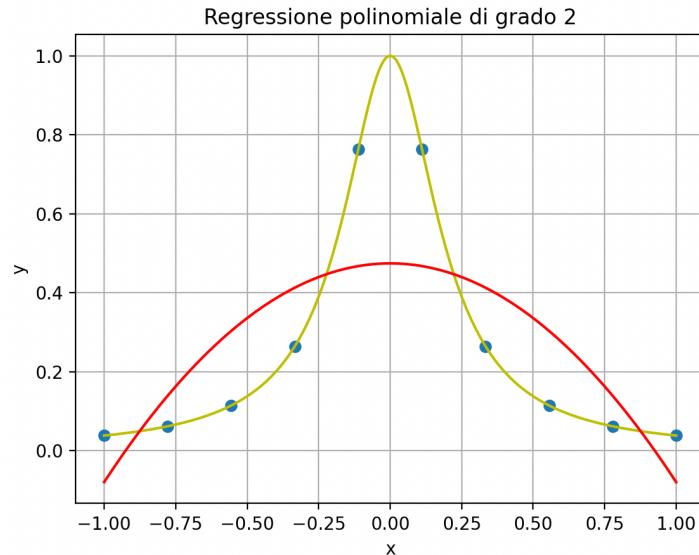
Case 1:

Output per n=1:



Alpha svd: [2.48814136e-01 1.18862397e-16]
Residual: 0.8519889898800778

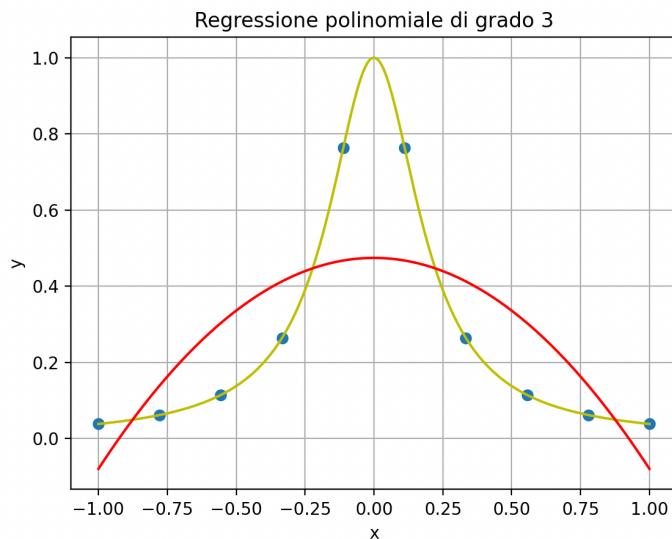
Output per n=2:



Alpha svd: [4.74479148e-01 1.33300079e-16 -5.53905029e-01]

Residual: 0.01293656315783785

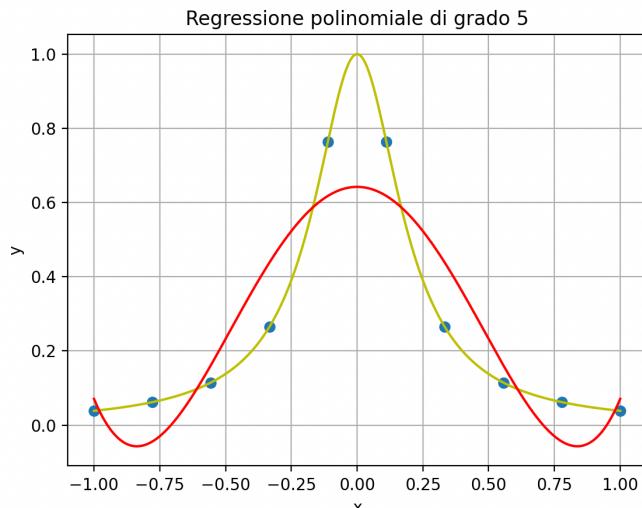
Output per n=3:



Alpha svd: [4.74479148e-01 -1.49111060e-16 -5.53905029e-01 3.57225108e-16]

Residual: 0.5751808755886042

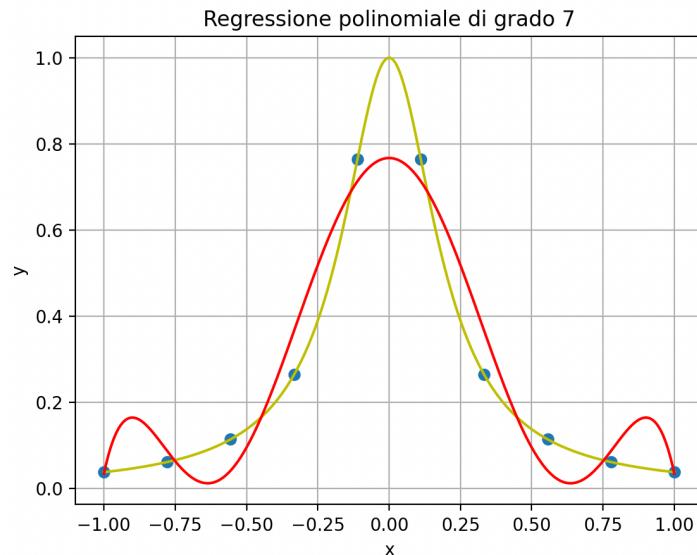
Output per n=5:



Alpha svd: [6.42191979e-01 -1.21704962e-15 -1.99646993e+00 4.23316618e-15 1.42497265e+00 -2.68306842e-15]

Residual: 0.3631838354754461

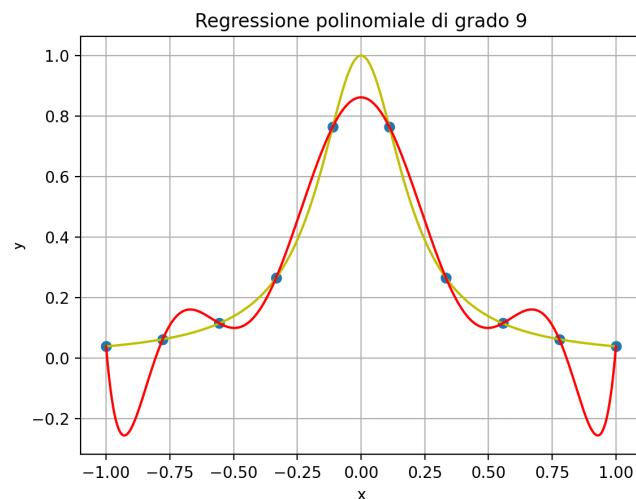
Output per n=7



Alpha svd: [7.67384095e-01 2.10900315e-15 -4.47552598e+00 -9.58983588e-15 8.29218923e+00 2.03601170e-14
-4.54921187e+00 -1.15571217e-14]

Residual: 0.1939183846601576

Output per n=9

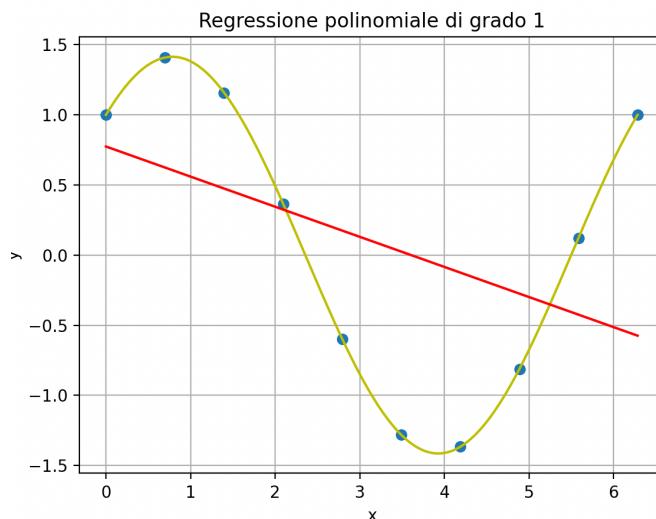


Alpha svd: [8.61538152e-01 9.21616982e-14 -8.26092333e+00 -7.73081304e-13 3.07285300e+01 2.74019473e-12
-4.49154581e+01 -4.33806207e-12 2.16247748e+01 2.28408092e-12]

Residual: 4.05204265350537e-14

Case 2:

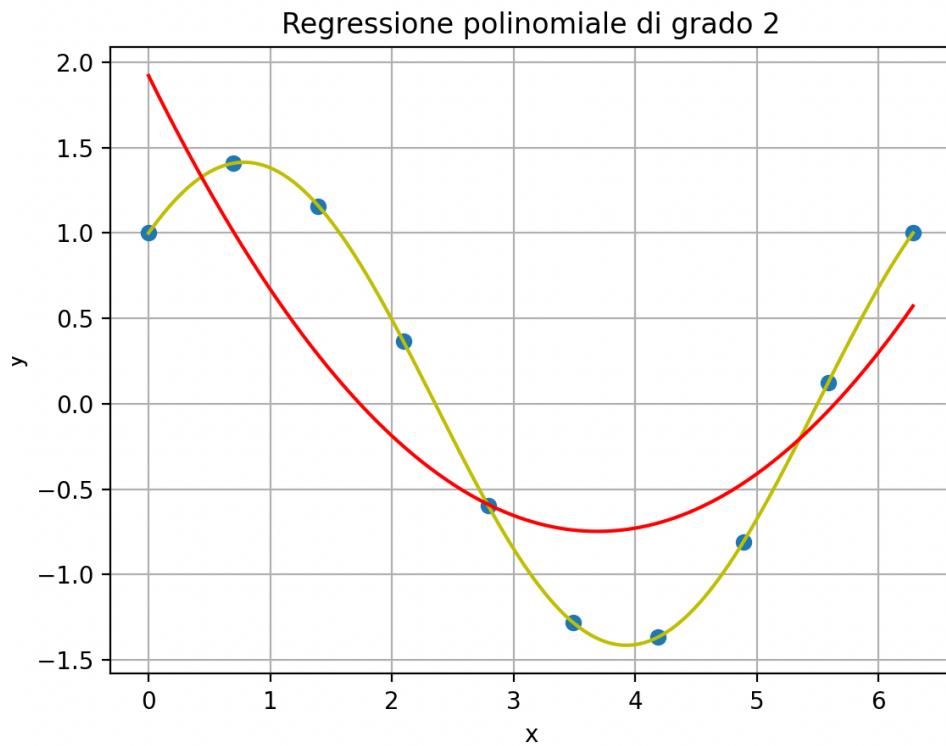
Output per n=1:



Alpha svd: [0.77438082 -0.21466208]

Residual: 2.836750592104937

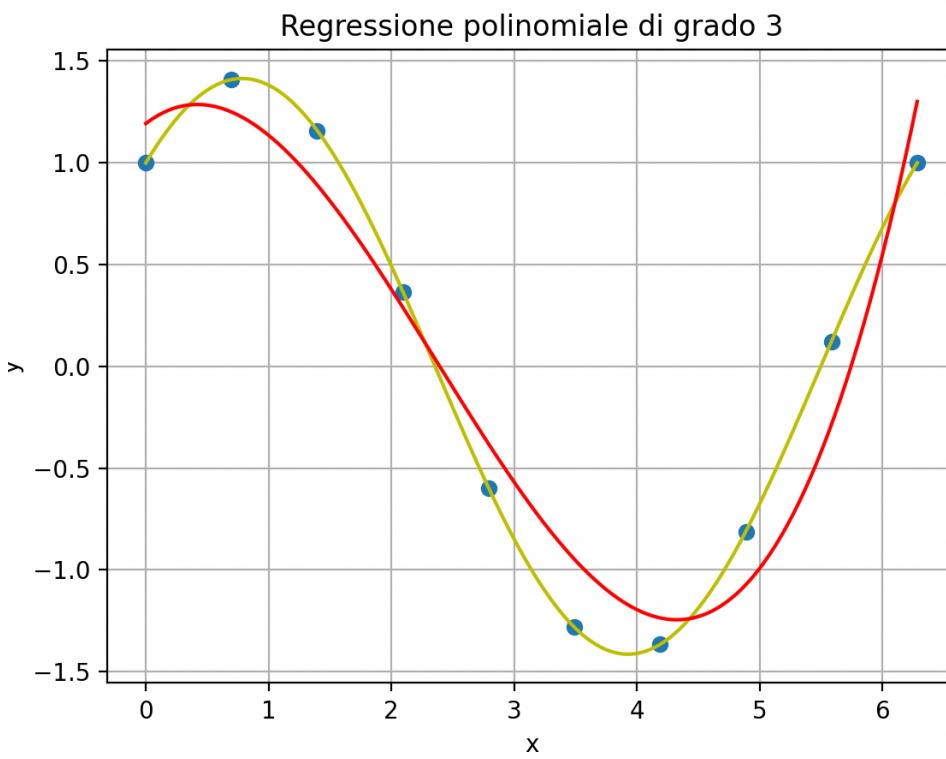
Output per n=2:



Alpha svd: [1.92140002 -1.44690007 0.19611677]

Residual: 1.7952974022498318

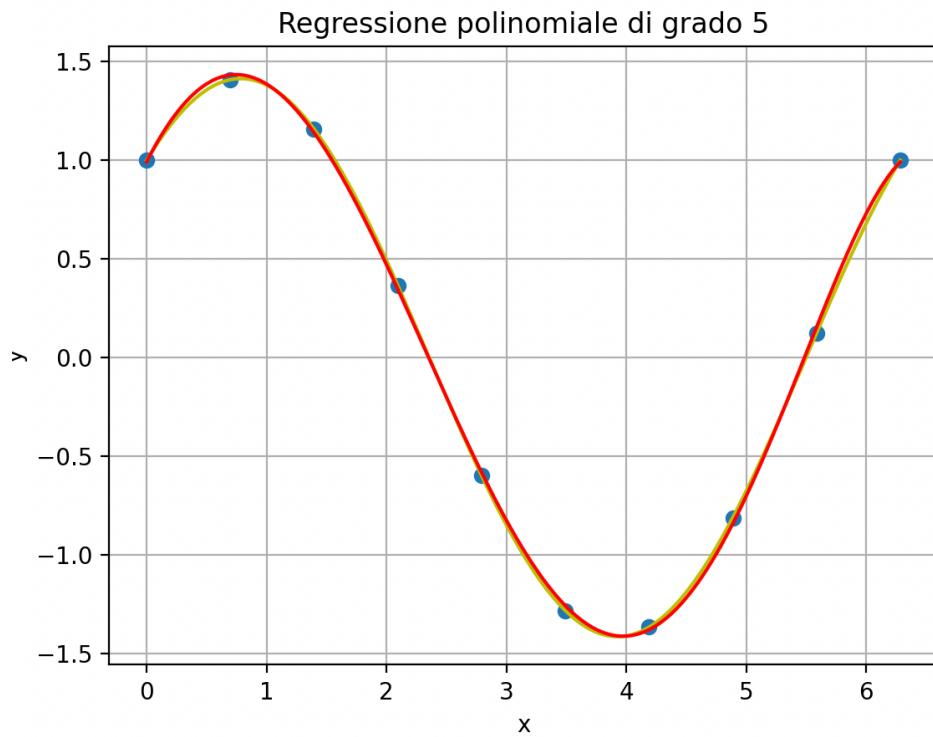
Output per n=3:



Alpha svd: [1.19282899 0.46222741 -0.60469494 0.08496876]

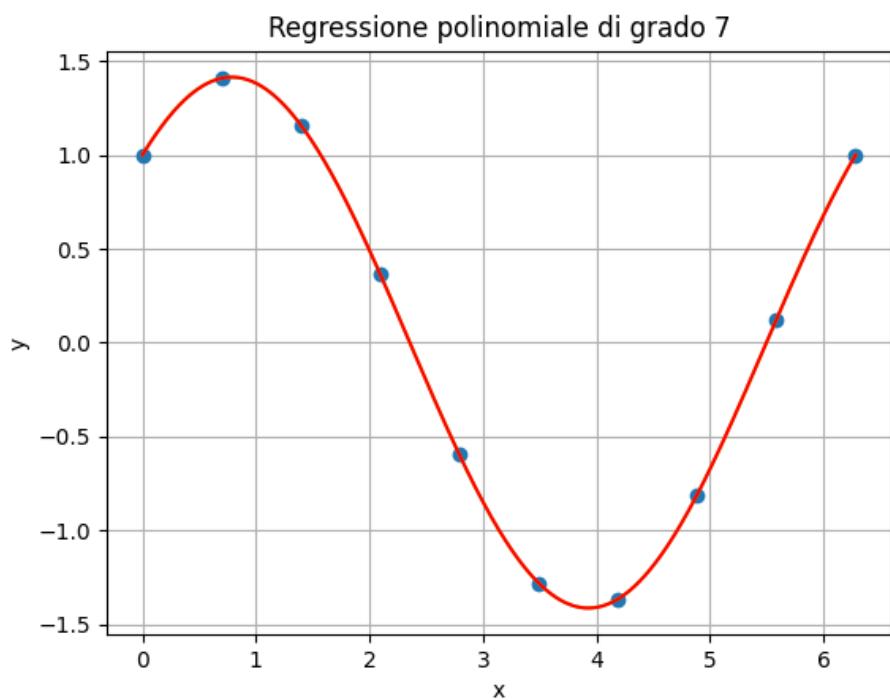
Residual: 0.8007707717731798

Output per n=5:



Alpha svd: [0.99315181 1.18619381 -0.77082962 -0.07787167 0.06176099 -0.00551072]
Residual: 0.0655943161754875

Output per n=7



Alpha svd: [1.00003083e+00 9.77531789e-01 -4.34564755e-01 -2.32885792e-01 7.02987169e-02 4.47279780e-03 -2.32165017e-03 1.50714072e-04]
Residual: 0.0016911098620951087