ese_extra_parte1_completo

March 20, 2024

1 Esercizi extra - parte 1

In questo notebook troviamo esercizi relativi alla prima parte del programma di numerica, di vario grado di difficoltà (indicato nel testo).

2 Zeri di funzioni

2.0.1 Esercizio 1

Difficoltà: facile. Secondo l'equazione dei gas perfetti il volume specifico v dell'aria è dato da

$$v = \frac{TR^\star}{p}$$

dove $R^\star = 287.78\, J\, kg^{-1}\, K^{-1}$. Per $T = 288\, K$, $p = 101325\, Pa$ si ottiene $v = 0.8180$ ossia densitá $\rho = 1.225\, kgm^{-3}$. Consideriamo ora la legge dei gas reali

$$f(v) = \left(p + \frac{a}{v^2}\right)(vb - c) - dR^\star T = 0.$$

con $a = 1.2425e03$, $b = 2.8890e-05$, $c = 3.8700e-05$, $d = 1.0e-5$ (dati non realistici). Risolvere l'equazione dei gas reali con il metodo di bisezione partendo da un intervallo iniziale adeguato. Che densità si ottiene? Calcolare il risultato con un errore massimo di $\epsilon = 1.0e-5$ stabilendo a priori il numero di iterazioni necessarie dato l'intervallo scelto.

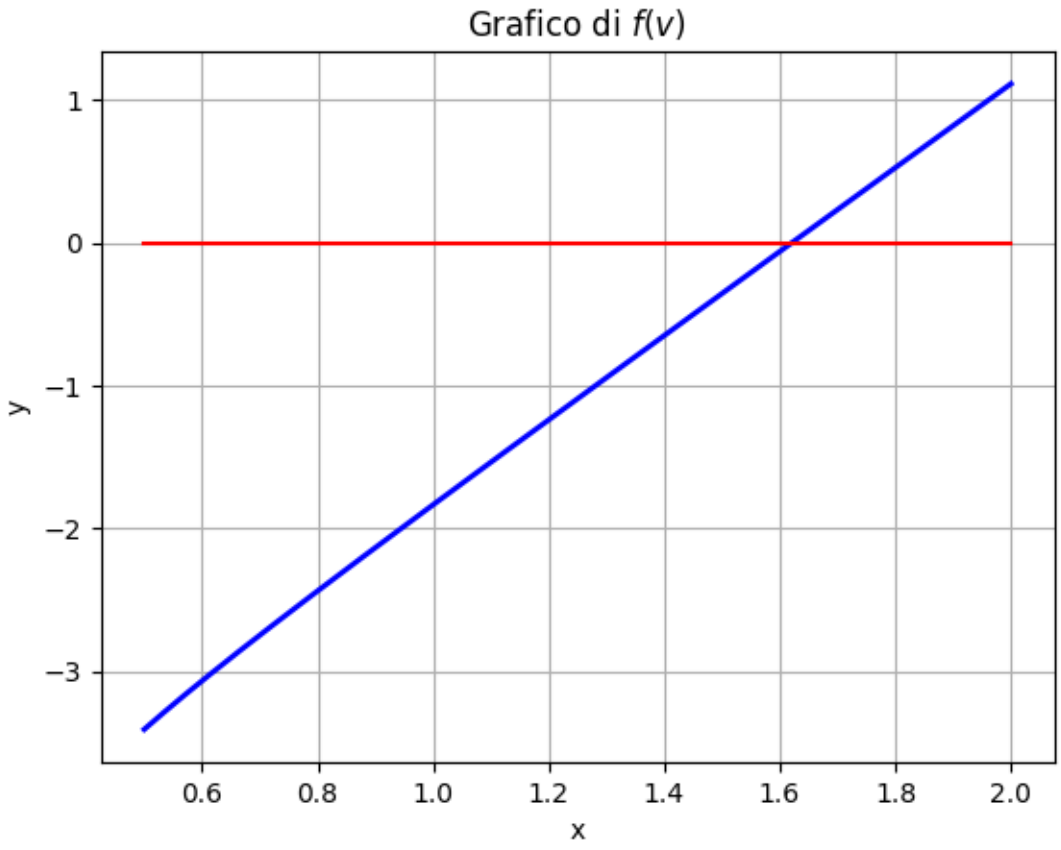
[37]:
```
# step 0 -> ricordarsi di importare i pacchetti numpy and matplotlib

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

Definiamo la funzione e plottiamo il suo andamento.

[38]:
```
a = 1.2425e+03
b = 2.8890e-05
c = 3.8700e-05
d = 1.0e-5
R = 287.78
T = 288
p = 101325
f=lambda v: ( p + a / ( v *v ) ) * ( v * b - c) - d * R * T
```

```
# rappresentazione grafica
# discretizzazione dell'intervallo [-2 2]
xgrid=np.linspace(0.5,2,1000)
# plot di f
plt.plot(xgrid, f(xgrid),'b-',linewidth=2.0)
# plot asse y=0
plt.plot(xgrid,np.zeros(np.shape(xgrid)),'r-') # o semplicemente plt.
 ↪plot([-2,2],[0,0],'r-')
# specifiche grafiche titolo, legenda, grid,
plt.title("Grafico di $f(v)$")
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.show()
```



Implementiamo il metodo di bisezione:

[39]:
```
# definizione del metodo di bisezione
def bisez(f, a, b, toll = 1e-6):
```

```
    # controllo se gli estremi sono una bracket (opzionale)
    if (f(a)*f(b)>=0):
     raise RuntimeError('ERRORE: l''intervallo [a,b] non è una bracket')

    # inizializzazione
    xvect = []

    while (abs(b-a)>toll):
      x = 0.5*(a+b)

      # primo controllo se x è uno zero (opzionale)
      if (f(x)==0):
        xvect.append(x)
        print("x è esattamente uno zero della funzione")
        break

      # metodo di bisezione
      if (f(a)*f(x)>0):
          a = x
      else:
          b = x

      xvect.append(x)

    return  np.array(xvect)
```

Dal grafico ottenuto osserviamo che possiamo scegliere come estremi iniziali $a_0 = 1$, $b_0 = 2$ ottenendo un intervallo iniziale di lunghezza $|I_0| = |b_0 - a_0| = 1$. Ricaviamo il numero di iterazioni necessarie come:

$$|e_k| \leq \frac{|I_0|}{2^{k+1}} = \epsilon \tag{1}$$

$$2^{k+1} = \epsilon^{-1}|I_0| \tag{2}$$

$$k = log_2(\epsilon^{-1}|I_0|) - 1 \tag{3}$$

[40]:
```
N_it= np.log2(1.0e5)-1
print("Numero minimo di iterazioni", np.ceil(N_it))
```

```
Numero minimo di iterazioni 16.0
```

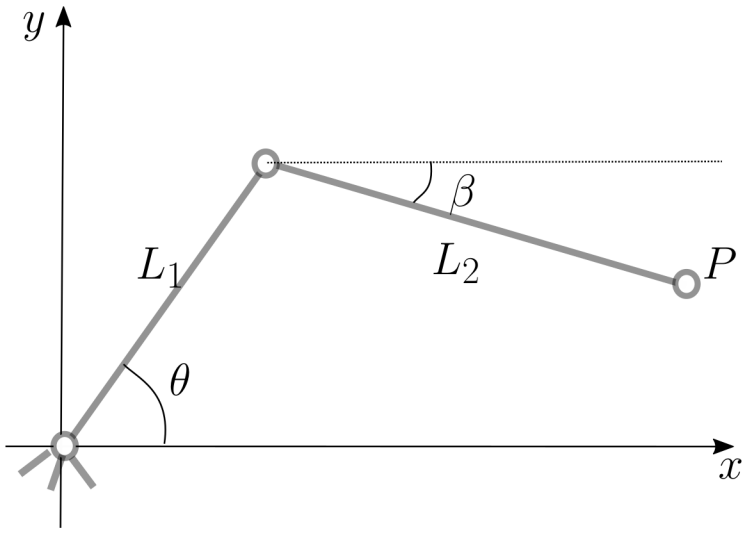
[41]:
```
xvect = bisez(f, 1, 2, 1e-5)
print("Numero iterazioni: %d." % len(xvect))
print("Ultimo valore di x: %f" % xvect[-1])
```

```
Numero iterazioni: 17.
Ultimo valore di x: 1.621376
```

Nota: otteniamo 17 e non 16 perché subito dopo il calcolo del punto medio il contatore it nella function viene incrementato. Al valore $v = 1.622368$ corrisponde una densità $\rho = 0.6164$.

2.0.2 Esercizio 2

Difficoltà: difficile. Si consideri il cinematismo in figura e si descriva la posizione del punto finale P utilizzando i due angoli e il sistema di riferimento indicati.



Imponendo $P = (2.5, 0)$, $L_1 = 2$, $L_2 = 1$, che angoli θ e β si ottengono? Risolvere il sistema nonlineare con il metodo di Newton per sistemi per diversi valori della guess iniziale, con tolleranza $\epsilon = 10^{-6}$. La soluzione è unica? Abbiamo sempre convergenza? Perché?

La posizione del punto P é descritta dalla seguente legge:

$$\begin{cases} x = L_1 \cos\theta + L_2 \cos\beta \\ y = L_1 \sin\theta - L_2 \sin\beta \end{cases} \tag{4}$$

Se imponiamo valori dati alle coordinate di P otteniamo un sistema nonlineare di due equazioni per le incognite θ e β, i due gradi di libertà del sistema:

$$\begin{cases} f_1(\theta, \beta) = L_1 \cos\theta + L_2 \cos\beta - x_P \\ f_2(\theta, \beta) = L_1 \sin\theta - L_2 \sin\beta - y_P \end{cases} \tag{5}$$

Calcoliamo lo Jacobiano:

$$J = \begin{pmatrix} -L_1 \sin\theta & -L_2 \sin\beta \\ L_1 \cos\theta & -L_2 \cos\beta \end{pmatrix} \tag{6}$$

Definiamo i dati, raggruppando le incognite nel vettore $\mathbf{x}$:

[42]:
```
L1 = 2
L2 = 1
xp = 2.5
yp = 0
f=lambda x: np.array([L1*np.cos(x[0])+ L2*np.cos(x[1]) - xp , L1*np.
 ↪sin(x[0])-L2*np.sin(x[1]) - yp])

J=lambda x: np.array([[-L1*np.sin(x[0]), -L2*np.sin(x[1])],
                      [L1*np.cos(x[0]), -L2*np.cos(x[1]) ] ]  )
```

Adattiamo il metodo di Newton al caso di sistemi di equazioni nonlineari:

[43]:
```
# Definizione del metodo di Newton (questa è già la versione modificata)
def newton (f,J,x0,nmax=100,toll=1e-6) :

  xold = x0

  for nit in range(nmax) :
    dx=scipy.linalg.solve(J(xold),f(xold))
    # calcolo il nuovo punto
    xnew=xold-dx
    # criterio di arresto e aggiorno
    if (np.linalg.norm(dx) < toll) :
        break
    else :
        xold=xnew

  return xold,nit
```

[44]:
```
sol,it=newton (f,J,np.array([0.1,0.1]),100,1.0e-6)
print("convergo a",sol, "in ",it, " iterazioni")
```

```
convergo a [0.38976075 0.86321194] in  6  iterazioni
```

Notiamo che con questa scelta dei dati iniziali convergiamo in 6 iterazioni e gli angoli corrispondenti, convertiti in gradi, sono $\theta = 22.2°$, $\beta = 49.5°$, mentre, ad esempio, partendo da $\theta = 0$, $\beta = 0$ il metodo si arresta perch'e lo jacobiano diventa singolare. Infine, proviamo a partire da una guess iniziale opposta:

[45]:
```
sol,it=newton (f,J,np.array([-0.1,-0.1]),100,1.0e-6)
print("convergo a",sol, "in ",it, " iterazioni")
```

```
convergo a [-0.38976075 -0.86321194] in  6  iterazioni
```

Notiamo che la soluzione non 'e unica, infatti in questo caso otteniamo angoli opposti.

2.1 Sistemi lineari

2.1.1 Esercizio 3

Difficoltà: media.

Si costruisca la matrice di Vandermonde di dimensione 10×10 con il comando np.vander. L'elemento vij di tale matrice è costruito come $v_{ij} = x_i^{n-j-1}$ il numero di righe e xi sono gli elementi del vettore dato in input. La matrice di Vandermonde può essere usata nella costruzione dei polinomi interpolanti su nodi x_i ed è notoriamente malcondizionata.

[46]:
```
V=np.vander(np.linspace(1,10,10))
```

Si costruisca un termine noto $\mathbf{b}$ tale che la soluzione esatta sia $x_e = (1, 1, 1, 1, \dots, 1)^T$.

[47]:
```
xe = np.ones(10).T
b=V@xe
```

Si costruisca ora un termine noto perturbato $\mathbf{b}+\delta\mathbf{b}$ dove la perturbazione $\delta\mathbf{b}$ è un vettore di numeri random compresi fra 0 e 0.1. Suggerimento: usare il comando numpy.random.rand(nrows,ncols) e riscalare il vettore ottenuto.

[48]:
```
db=0.1*np.random.rand(10)
print(db)
```

```
[0.02660417 0.02588424 0.08960441 0.0665684  0.04616058 0.0869962
 0.06197872 0.05752185 0.07912997 0.01825655]
```

Si risolva il sistema perturbato con la fattorizzazione LU, verificando se è stato eseguito il pivoting. Si calcoli la soluzione $\mathbf{x} + \delta\mathbf{x}$ utilizzando i metodi di sostituzione opportuni.

[49]:
```
P, L, U = scipy.linalg.lu(V)
# verifica che la matrice è la matrice identità
if (P == np.eye(10)).all():
    print("P=I, no pivoting")
else:
    print('Pivoting effettuato')
```

```
Pivoting effettuato
```

Copiamo le function per la sostituzione avanti/indietro:

[50]:
```
def fwsub(A,b):
    n = b.shape[0]
    # inizializzo il vettore
    x = np.zeros(n)
    # costruzione forward substitution
    x[0] = b[0]/A[0,0]
    for i in range(1,n):
        x[i] = (b[i] - A[i,0:i] @ x[0:i]) / A[i,i]
    return x
```

```
def bksub(A,b):
    n = b.shape[0]
    # inizializzo il vettore
    x = np.zeros(n)
    # costruzione forward substitution
    x[-1] = b[-1]/A[-1,-1]
    for i in range(n-2,-1,-1):
        x[i] = (b[i] - A[i,i+1:n] @ x[i+1:n]) / A[i,i]
    return x
```

[51]:
```
y = fwsub(L, P.T@(b+db))
x_pert = bksub(U, y)
print(x_pert)
```

```
[  1.00001072   0.99947602   1.01091395   0.87364681   1.88991274
  -2.92231512  11.71852851 -16.3387054   15.93544827  -4.14031232]
```

Si verifichi se è soddisfatta la stima teorica relativa alla stabilità del metodo di eliminazione di Gauss che prevede che

$$\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq K(V) \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|}$$

dove K(V) è il condizionamento della matrice e $\delta \mathbf{x}$ si ottiene per differenza con la soluzione esatta (nota).

[52]:
```
rel_err=np.linalg.norm(xe-x_pert)/np.linalg.norm(xe)
KV=np.linalg.cond(V)
if (rel_err<KV*np.linalg.norm(db)/np.linalg.norm(b)):
    print("la stima è verificata")
    print("errore relativo=",rel_err)
    print("KV= %.4e" %KV)
    print("perturbazione termine noto (normalizzata)=",np.linalg.norm(db)/np.
 ↪linalg.norm(b))
```

```
la stima è verificata
errore relativo= 8.253487560826173
KV= 2.1063e+12
perturbazione termine noto (normalizzata)= 1.6032544926721194e-10
```

2.1.2 Esercizio 4

Difficoltà: dfficile.

Si consideri la matrice A allegata, da importare con il comando

[53]:
```
matrix = np.loadtxt('matrice.txt')
```

```
A=scipy.sparse.coo_matrix((matrix[:,2], (matrix[:,0].T.astype(int),matrix[:,1].
↪T.astype(int)))).toarray()
```

Si costruisca il termine noto b del sistema lineare $A\mathbf{x} = \mathbf{b}$ tale che la soluzione esatta sia $\mathbf{x}_e = [1, 1, ...]^T$. (Nota: determinare le dimensioni della matrice A con il comando shape.

[54]:
```
n,m=np.shape(A)
print(n)
xe=np.ones(n).T
b=A@xe
```

```
725
```

Calcolare il numero di condizionamento della matrice.

[55]:
```
K=np.linalg.cond(A)
print(K)
```

```
12176.579615307353
```

Vogliamo risolvere il sistema lineare con il metodo di Richardson statico. Calcolare il parametro α_{max} massimo per avere convergenza, e il parametro α_{opt} che garantisce il raggio spettrale minimo per la matrice di convergenza considerando come precondizionatore la matrice identità.

[56]:
```
from scipy.linalg import eigvals
lambda_max = np.max(np.abs(eigvals(A)))
lambda_min = np.min(np.abs(eigvals(A)))
alpha_max = 2/(lambda_max)
alpha_opt = 2/(lambda_max+lambda_min)
print(alpha_max, alpha_opt)
```

```
0.32778809021714816 0.32776117287391804
```

Implementare una function per risolvere il sistema lineare con il metodo di Richardson. La funzione riceve in input la matrice, il termine noto, la guess iniziale x_0 , il precondizionatore P, il parametro α, la tolleranza tol e il numero massimo di iterazioni nmax, e restituisce in output il vettore soluzione e il numero di iterazioni effettuate.

[57]:
```
def richardson_solve(A, b, x0, P, alpha, nmax, rtoll):
    r = A @ x0 - b
    bnorm = np.linalg.norm(b)
    k = 0
    xiter = [x0]
    while( (np.linalg.norm(r)/bnorm) > rtoll and k<nmax):
        xold = xiter[-1]
        dx = np.linalg.solve(P,alpha*r)
        xnew = xold +dx
        xiter.append(xnew)
        r = b-A @ xnew
        k = k+1
```

```
    return xiter,k
```

Risolvere il sistema con la function scritta e i seguenti dati: il vettore nullo come guess iniziale, precondizionatore pari alla matrice identità, $\alpha = \alpha_{opt}$, nmax=1000 e tol=1.0e-6. Cosa si osserva?

[58]:
```
xiter,k = richardson_solve(A,b,np.zeros(n), np.eye(n), alpha_opt, 1000, 1.0e-6)
print(k)
```

```
1000
```

Osserviamo che il metodo non converge nel numero di iterazioni prescritto. Infatti l'errore relativo è ancora molto alto:

[59]:
```
print(np.linalg.norm(xiter[-1]-xe)/np.linalg.norm(xe))
```

```
0.8445671774800815
```

Eseguire la fattorizzazione LU della matrice A e ripetere il punto precedente usando come precondizionatore P = L. Cosa si osserva? Come si spiega questo risultato? (Calcolare il numero di condizionamento di $L^{-1}A$).

[60]:
```
PIV, L, U = scipy.linalg.lu(A)
P=L
P_A=scipy.linalg.solve(P,A)   #calcolo P^-1A sfruttando il fatto che è␣
 ↪triangolare
lambda_max = np.max(np.abs(eigvals(P_A)))  #ricalcolo gli autovalori e gli alpha
lambda_min = np.min(np.abs(eigvals(P_A)))
alpha_opt = 2/(lambda_max+lambda_min)
xiter,k = richardson_solve(A,b,np.zeros(n), P, alpha_opt, 1000, 1.0e-6)
print(k)
print(np.linalg.cond(P_A))
```

```
123
372.8575696506422
```

Osserviamo anche che la matrice P non è simmetrica, infatti il condizionamento trovato NON corrisponde al rapporto fra il massimo e minimo autovalore che risulterebbe

[61]:
```
print(lambda_max/lambda_min)
```

```
14.127263857180967
```

2.1.3 Esercizio 5

Difficoltà: media.

Si consideri la matrice A allegata, da importare con il comando

[62]:
```
matrix = np.loadtxt('matrice.txt')
A=scipy.sparse.coo_matrix((matrix[:,2], (matrix[:,0].T.astype(int),matrix[:,1].
 ↪T.astype(int)))).toarray()
```

Si costruisca il termine noto $\mathbf{b}$ del sistema lineare $A\mathbf{x} = \mathbf{b}$ tale che la soluzione esatta sia $\mathbf{x}_e = [1, 1, ...]^T$. (Nota: determinare le dimensioni della matrice A con il comando shape).

[63]:
```
n,m=np.shape(A)
print(n)
xe=np.ones(n).T
b=A@xe
```

```
725
```

Calcolare il numero di condizionamento della matrice.

[64]:
```
K=np.linalg.cond(A)
print(K)
```

```
12176.579615307353
```

Verificare che la matrice è simmetrica e definita positiva.

[65]:
```
print("simmetrica?",np.min(np.abs(A-A.T))<1e-15)
lambda_min=np.min(eigvals(A))
print("definita positiva?",lambda_min>0)
```

```
simmetrica? True
definita positiva? True
```

Risolvere il sistema con il metodo del gradiente (gdescent) e i seguenti dati: il vettore nullo come guess iniziale, nmax=1000 e tol=1.0e-6. Cosa si osserva?

[66]:
```
from utils import gdescent
xg = gdescent(A, b, np.zeros(n), nmax = 1000, rtoll = 1e-6)
print("Soluzione approssimata (Gradient descent):")
print(xg[-1])
```

```
Soluzione approssimata (Gradient descent):
[1.         0.15643625 0.12177743 0.15636829 0.63682208 0.51241335
 0.43707034 0.38426187 0.34397654 0.31163152 0.28448519 0.2619033
 0.24271719 0.22599495 0.21177653 0.19944253 0.18898463 0.18028577
 0.17303055 0.16712251 0.16243244 0.15934452 0.15731083 0.15653224
 0.15603581 0.15506617 0.15372212 0.15200754 0.14991954 0.14761738
 0.14514975 0.1425329  0.13975374 0.13702938 0.13438213 0.13188849
 0.12957742 0.12746655 0.12560624 0.12402744 0.12302237 0.12226988
 0.12229336 0.1230817  0.12419419 0.12569924 0.12751804 0.12955976
 0.13183001 0.13434964 0.13700441 0.13961763 0.14226461 0.14483446
 0.14731826 0.1496911  0.1517864  0.15350046 0.15483963 0.15589037
 0.15644752 0.15728009 0.159338   0.16259271 0.16719239 0.17311709
 0.18043622 0.189161   0.19964427 0.21194477 0.22604233 0.24266421
 0.26189228 0.28454157 0.3114605  0.34392756 0.38444554 0.43621047
 0.51287168 0.63664891 0.17382334 0.40088708 0.16002488 0.12538674
 0.15948416 0.15688847 0.12230871 0.15694666 0.65652284 0.24686745
```

0.17679672 0.13977295 0.16198154 0.16438117 0.12554731 0.15577314
0.39460085 0.15825522 0.12347509 0.1577985 0.49178675 0.15045861
0.19825709 0.14284109 0.22830109 0.30662398 0.13078753 0.15589822
0.17332318 0.16244417 0.12467821 0.1559671 0.42077223 0.44420897
0.16028957 0.12427944 0.1573028 0.16936685 0.12673517 0.15363558
0.36391705 0.53726389 0.15689786 0.12299136 0.15826341 0.15680189
0.12280531 0.15798693 0.55679123 0.17202763 0.15897422 0.29227231
0.13255899 0.19313607 0.15861872 0.15686824 0.19887173 0.24120171
0.13885065 0.20251315 0.14685477 0.20411036 0.13746021 0.14565828
0.24646009 0.1271182 0.15439595 0.16892879 0.3550105 0.47908756
0.12356004 0.15646858 0.15977935 0.43179149 0.15559339 0.12445651
0.16251033 0.15754799 0.12551485 0.16229247 0.39647883 0.15463301
0.16510314 0.12520553 0.40220978 0.15789792 0.12434845 0.15976799
0.44011116 0.17384594 0.17046711 0.21106311 0.14928668 0.19892509
0.14691218 0.1524581 0.21647592 0.18233983 0.15132849 0.1317202
0.29275444 0.34618654 0.16356105 0.16018542 0.12792388 0.27860074
0.1489182 0.13303061 0.18872284 0.22682179 0.14031392 0.14106971
0.22218045 0.28173904 0.13396484 0.16923987 0.16288701 0.37209911
0.12638478 0.16391902 0.15743163 0.20588099 0.13624214 0.14423071
0.25025657 0.12572546 0.16626345 0.15440846 0.38537845 0.12367504
0.1605014 0.15603388 0.47364381 0.45234909 0.12386396 0.16145344
0.1554486 0.24332936 0.14024219 0.17614127 0.16270273 0.19153225
0.17049285 0.17795549 0.16065003 0.18792733 0.13678047 0.25741424
0.15165152 0.26659637 0.15442655 0.13568942 0.18293648 0.2104129
0.15367096 0.14782042 0.16343415 0.14945596 0.21695554 0.18145251
0.19745321 0.14401596 0.21701898 0.14351896 0.21945593 0.20380238
0.24901844 0.13663175 0.14480283 0.32456855 0.12930008 0.1555397
0.17147783 0.15200417 0.17734553 0.12978897 0.31317155 0.15908919
0.16584179 0.12860908 0.3347961 0.19008162 0.14793601 0.13251906
0.28008067 0.13163184 0.16363138 0.16543617 0.30211059 0.15091315
0.30996786 0.1299587 0.17967468 0.13877693 0.14135518 0.21913871
0.23282995 0.37121111 0.16262554 0.15889591 0.12654137 0.12708071
0.15286162 0.17088117 0.35375034 0.12549504 0.1576711 0.16208298
0.39579187 0.37951472 0.12601661 0.15597939 0.1649287 0.12465688
0.41608818 0.15470378 0.16383107 0.16875223 0.16205465 0.18095442
0.20083165 0.1527342 0.14150482 0.23516105 0.19305388 0.18443609
0.17058312 0.1626034 0.16950073 0.14283661 0.1532538 0.2309119
0.20402992 0.22965416 0.18464098 0.14368388 0.15922466 0.16412617
0.17071412 0.13661337 0.26239635 0.19978293 0.2365591 0.14856343
0.1403142 0.27262144 0.14652504 0.19331118 0.13333977 0.19856031
0.15183844 0.21613239 0.20940885 0.15439459 0.20559328 0.1456558
0.14718526 0.16814677 0.15529598 0.12724451 0.35583795 0.19423854
0.15455706 0.13533119 0.14882814 0.13186408 0.26181201 0.17723936
0.29567931 0.26605262 0.15939706 0.13611432 0.17618695 0.30917267
0.15993801 0.16833571 0.13082598 0.25689821 0.13600064 0.14749519
0.28810529 0.19644207 0.13262225 0.15380233 0.17934209 0.21205272
0.24020733 0.14264098 0.13771493 0.184787   0.13097143 0.14907517
0.29535716 0.32120752 0.12904186 0.17746845 0.15097207 0.3213377

0.1297443 0.15927822 0.16750421 0.22856141 0.14063993 0.21624828
0.14312772 0.14048409 0.2138962 0.2299867 0.14365539 0.12747135
0.16661037 0.15675273 0.35101714 0.17675481 0.15295549 0.1301514
0.31118281 0.12697942 0.36215488 0.15981278 0.16238586 0.19609289
0.26792871 0.14625186 0.13378695 0.12833719 0.33225169 0.17504398
0.15182755 0.18696233 0.15789216 0.17923392 0.25784765 0.13716688
0.15985998 0.14390886 0.22961877 0.18353695 0.17759504 0.15226534
0.16846173 0.2532958 0.1383514 0.16758259 0.16876312 0.20789813
0.14724995 0.14371065 0.22208956 0.14634133 0.16269931 0.22179214
0.18292438 0.29434386 0.14862646 0.18520621 0.13115693 0.13322315
0.28618812 0.15760643 0.17475184 0.1748203 0.1766184 0.16541449
0.13817349 0.25328675 0.15688651 0.18222241 0.13727952 0.25331931
0.19273236 0.15024416 0.18956253 0.18749081 0.20166266 0.16952727
0.16283574 0.15428609 0.17679391 0.16205494 0.19258857 0.13935708
0.15130597 0.24329612 0.23591932 0.13872273 0.14284128 0.21426018
0.20927796 0.16101346 0.18735951 0.15216569 0.20855763 0.25466545
0.14447788 0.13552556 0.20253245 0.15449802 0.17207492 0.32683797
0.12892009 0.14712926 0.20610084 0.22695336 0.14237151 0.20282677
0.15343419 0.15734879 0.19377219 0.26374157 0.14691395 0.13463806
0.19641496 0.14936123 0.20773871 0.14925031 0.21695922 0.20745548
0.14638776 0.15086637 0.19927961 0.20547582 0.14512441 0.13798948
0.24271602 0.33282736 0.12849887 0.15321663 0.17331107 0.16374772
0.13423245 0.16854403 0.27925278 0.27205501 0.15171578 0.18653875
0.13437915 0.14656226 0.26227132 0.19557083 0.13478995 0.34837413
0.12774424 0.15756906 0.16621087 0.25016206 0.13754881 0.1485395
0.19577906 0.16464424 0.12640426 0.15688542 0.37316495 0.32884338
0.16241241 0.16317141 0.1291586 0.13077152 0.15697644 0.17198752
0.30823657 0.14606969 0.20636537 0.14017652 0.23449469 0.12827171
0.15280733 0.17325873 0.33396604 0.15339441 0.13175762 0.17671104
0.29263354 0.14074625 0.13910673 0.23085663 0.22089799 0.18507736
0.14958102 0.29092476 0.13156247 0.16917803 0.15708605 0.12920557
0.32540623 0.18124619 0.15864536 0.13991018 0.24416714 0.18382132
0.23213753 0.17673397 0.14337929 0.16467626 0.14769161 0.17493071
0.16859463 0.1660886 0.23059268 0.14397837 0.17532881 0.1685484
0.16769442 0.27071534 0.13506653 0.15553818 0.18024609 0.2198628
0.13427691 0.18692047 0.27265629 0.15090067 0.20446426 0.15749548
0.14764957 0.21733708 0.19021526 0.19374999 0.17162753 0.17626966
0.15784662 0.16357684 0.17158877 0.26283259 0.13669023 0.16508242
0.1840783 0.15727099 0.19446486 0.18830835 0.15912246 0.14861863
0.21460898 0.18056578 0.23761129 0.13851566 0.1433428 0.21139819
0.19896216 0.19941531 0.14943658 0.13891853 0.14063674 0.15319453
0.15075072 0.22551357 0.24724132 0.14364566 0.23608369 0.18923989
0.16455081 0.17220039 0.13863746 0.25096976 0.18318116 0.1306663
0.14985629 0.30517892 0.26938121 0.13546844 0.17432904 0.16008452
0.13084005 0.29963869 0.16517666 0.18065914 0.16498216 0.16082652
0.14968349 0.197064   0.21223909 0.1568159 0.19662487 0.15735286
0.18236379 0.20326041 0.23990101 0.18603619 0.15638439 0.14093619
0.22264144 0.19231292 0.15445272 0.14526627 0.20843036 0.14600725

```
0.14143293 0.22918566 0.14924475 0.18904729 0.27699037 0.13322604
0.13585533 0.26769079 0.16654666 0.16739599 0.18444992 0.2405902
0.14102864 0.17134885 0.1674462  0.17204056 0.17007428 0.14556964
0.16678412 0.17161088 0.14063264 0.24172141 0.25449431 0.16116495
0.13788156 0.17553584 0.18378793 0.15478995 0.18497101 0.25139236
0.13813874 0.23647716 0.14141324 0.1546201  0.18891937 0.20594
0.14894668 0.2161512  0.14601905 0.16166633 0.17618273 0.17389594
0.22726801 0.14397269 0.15477049 0.19180496 0.18194861 0.15135058
0.2211452  0.19076214 0.14170272 0.17955408 0.23719599 0.16099859
0.18908478 0.14798511 0.1467782  0.1664769  0.17707586]
```

Notiamo che la soluzione è molto lontana da quella esatta, la convergenza è estremamente lenta.

Verificare la stima teorica per l'abbattimento dell'errore con il metodo del gradiente.

Sappiamo, dalla teoria, che l'errore all'iterazione n-sima soddisfa la seguente stima:

$$\|\mathbf{e}^k\|_A < C^k \|\mathbf{e}^0\|_A$$

con $C = \dfrac{K(A)-1}{K(A)+1}$.

[67]:
```
e0=xg[0]-xe
ek=xg[-1]-xe
norm_e0=(e0.T@A@e0)**0.5
norm_ek=(ek.T@A@ek)**0.5
C=(K-1)/(K+1)
nit=len(xg)
print("la stima è soddisfatta?", norm_ek<C**nit*norm_e0)
```

```
la stima è soddisfatta? True
```

Ora, risolvere lo stesso problema, con gli stessi dati, utilizzando il metodo del gradiente coniugato allegato di seguito. Il metodo raggiunge la tolleranza desiderata? In quante iterazioni? Verificare che la convergenza soddisfa la stima teorica.

[85]:
```
def cgdescent(A, b, x0, nmax = 1000, rtoll = 1e-6):
  """
  Metodo del gradiente a parametro dinamico per sistemi lineari.

  Input:
   A      Matrice del sistema
   b      Termine noto (vettore)
   x0     Guess iniziale (vettore)
   nmax   Numero massimo di iterazioni
   toll   Tolleranza sul test d'arresto (sul residuo relativo)

  Output:
   xiter  Lista delle iterate
```

```
    """
    norm = np.linalg.norm

    bnorm = norm(b)

    r = b - A@x0

    xiter = [x0]
    iter = 0
    z = r

    while((norm(r) / bnorm)>rtoll  and iter < nmax):
        xold = xiter[-1]

        rho = np.dot(r, z)
        q = A @ z;
        alpha = rho / np.dot(z, q)
        xnew = xold + alpha * z
        r = r - alpha*q
        beta = (r.T @ A @ z)/(z.T @ A @ z)
        z = r - beta*z
        xiter.append(xnew)
        iter = iter + 1

    return xiter
```

[86]:
```
xcg = cgdescent(A, b, np.zeros(n), nmax = 1000, rtoll = 1e-6)
print("numero di iterazioni:")
print(len(xcg))
```

```
Soluzione approssimata (Gradient descent):
121
```

Il metodo converge in 121 iterazioni. Verifichiamo che il metodo soddisfa:

$$\|\mathbf{e}^k\|_A < \frac{2c^k}{1+c^{2k}}\|\mathbf{e}^0\|_A$$

con $c = \dfrac{\sqrt{K(A)}-1}{\sqrt{K(A)}+1}$.

[89]:
```
e0=xcg[0]-xe
ek=xcg[-1]-xe
norm_e0=(e0.T@A@e0)**0.5
norm_ek=(ek.T@A@ek)**0.5
c=(K**0.5-1)/(K**0.5+1)
```

```
nit=len(xcg)
cost=(c**nit)/(1+c**(2*nit))
print("la stima è soddisfatta?", norm_ek<cost*norm_e0)
```

```
la stima è soddisfatta? True
```