**SISTEMI LINEARI**

Nel file sistemi\_lineari.py vengono trattati i seguenti argomenti:

* Fattorizzazione LU senza il pivoting;
* Fattorizzazione LU con la tecnica del pivoting parziale;
* Fattorizzazione LU con la tecnica del pivoting totale;
* Risoluzione di un sistema lineare usando le tre fattorizzazioni sopracitate.

Si passa alla spiegazione teorica degli argomenti.

La fattorizzazione LU è un metodo molto efficace per la risoluzione dei sistemi lineari, poiché trasforma una matrice generica A in un prodotto tra una matrice triangolare inferiore L e una matrice triangolare superiore U, quindi **Ax = b** => **LUx = b** => **Ly = b**. Infatti, sfruttando l’algoritmo di sostituzione in avanti (per risolvere un sistema lineare in cui la matrice è una matrice triangolare inferiore) e l’algoritmo di sostituzione all’indietro (per risolvere un sistema lineari in cui la matrice è una matrice triangolare superiore), la risoluzione di un sistema lineare Ax = b si riduce a:

1. Trovare la fattorizzazione LU
2. Risolvere Ly = b con l’algoritmo di sostituzione in avanti
3. Risolvere Ux = y con l’algoritmo di sostituzione all’indietro

Questo algoritmo può essere migliorato aumentando il numero di matrici che possono essere fattorizzate. Durante la fattorizzazione LU, ci sono delle divisioni in cui il denominatore può essere molto piccolo, con successivi errori di overflow. Per risolvere questo problema, esistono due tecniche di pivoting:

* Parziale: si cerca il massimo in valore assoluto della colonna e si scambia la sua la riga con quella dell’elemento aii. Quindi, la risoluzione del sistema lineare diventa:
  1. Trovare la fattorizzazione PA = LU
  2. Risolvere il sistema Ly = Pb
  3. Risolvere il sistema Ux = y
* Totale: si cerca il massimo in valore assoluto della matrice e si scambiano riga e colonna con quella dell’elemento aii. Quindi, la risoluzione del sistema lineare diventa:
  1. Trovare la fattorizzazione PAQ = LU
  2. Risolvere il sistema Ly = Pb
  3. Risolvere il sistema Uz = y
  4. Calcolare x = Qz

|  |  |
| --- | --- |
| Scomposizione LU senza pivoting |  |
| Teoria: | Codice: |
| Calcolo L21(n)  Azzero A21(n)  Calcolo Â22(n)  Aggiorno L | Metodo fattorizzazione\_LU\_no\_piv:  Obiettivo: scomporre la matrice A in un prodotto tra la matrice L e la matrice U  Input: MATRICE (matrice da scomporre)  Output: L, U  U = np.copy(MATRICE)  U = np.asmatrix(U)  L = np.identity(len(MATRICE))  L = np.asmatrix(L)  for j in range(0, len(MATRICE)):  ln = np.copy(U[j+1:, j]/U[j,j])  U[j+1:, j] = 0  U[j+1:, j+1:] = U[j+1:, j+1:] - ln.dot(U[j, j+1:])  L[j+1:,j] = np.copy(ln)  return L,U |
| Scomposizione LU utilizzando il metodo del pivoting parziale |  |
| Teoria: | Codice: |
| Trovo le coordinate pivot considerando un U[j:,j]  Posiziono il pivot nella posizione corretta, quindi aggiorno L, U e P  Calcolo L21(n)  Azzero A21(n)  Calcolo Â22(n)  Aggiorno L | Metodo fattorizzazione\_LU\_piv\_parziale:  Obiettivo: scomporre la matrice A in un prodotto tra la matrice L e la matrice U usano la tecnica del pivoting parziale  Input: MATRICE (matrice da scomporre)  Output: P, L, U  U = np.copy(MATRICE)  U = np.asmatrix(U)  L = np.eye(len(MATRICE))  L = np.asmatrix(L)  P = np.eye(len(MATRICE))  for j in range (0, len(U)):      iMax = np.abs(U[j:,j]).argmax()  iMax += j    if iMax != j:  L[[j, iMax], :j] = L[[iMax,j], :j]  U[[j, iMax], :] = U[[iMax, j], :]  P[[j, iMax], :] = P[[iMax, j], :]    ln = np.copy(U[j+1:, j]/U[j,j])  U[j+1:, j] = 0  U[j+1:, j+1:] = U[j+1:, j+1:] - ln.dot(U[j, j+1:])  L[j+1:,j] = np.copy(ln)    return P,L,U |
| Scomposizione LU utilizzando il metodo del pivoting totale |  |
| Teoria: | Codice: |
| Trovo le coordinate del pivot considerando una sottomatrice di U  Posiziono il pivot nella posizione corretta, quindi aggiorno L, U, P e Q  Calcolo L21(n)  Azzero A21(n)  Calcolo Â22(n)  Aggiorno L | Metodo fattorizzazione\_LU\_piv\_totale:  Obiettivo: scomporre la matrice A in un prodotto tra la matrice L e la matrice U usano la tecnica del pivoting totale  Input: A (matrice da scomporre)  Output: P, L, U, Q  U = np.copy(MATRICE)  U = np.asmatrix(U)  L = np.eye(len(MATRICE  L = np.asmatrix(L)  P = np.eye(len(MATRICE))  Q = np.eye(len(MATRICE))  for j in range (0, len(U)):  iMax,jMax = np.unravel\_index(np.abs(U[j:,j:]).argmax(), U[j:,j:].shape)  jMax += j  iMax += j    if iMax != j or jMax != j:  L[[j, iMax], :j] = L[[iMax,j], :j]  U[[j, iMax], :] = U[[iMax, j], :]  U[:, [j, jMax]] = U[:, [jMax, j]]  P[[j, iMax], :] = P[[iMax, j], :]  Q[:, [j, jMax]] = Q[:, [jMax, j]]    ln = np.copy(U[j+1:, j]/U[j,j])  U[j+1:, j] = 0  U[j+1:, j+1:] = U[j+1:, j+1:] - ln.dot(U[j, j+1:])  L[j+1:,j] = np.copy(ln)  return P,Q,L,U |
| Algoritmo di sostituzione in avanti |  |
| Teoria: | Codice: |
| Calcolo la prima incognita  Calcolo le altre incognite | Metodo algoritmo\_sostituzione\_in\_avanti:  Obiettivo: trovare le soluzioni di un sistema utilizzando una matrice triangolare inferiore  Input: L (matrice triangolare inferiore) e b (vettore contenente i termini noti)  Output: y (vettore contenente le soluzioni del sistema)  y = np.empty(len(b))    y[0] = b[0]/L[0,0]  for i in range (1, len(L)):  y[i] = (b[i] - np.dot(L[i, 0:i], y[0:i]))/ L[i,i]      return y |
| Algoritmo di sostituzione in avanti |  |
| Teoria: | Codice: |
| Calcolo l’ultima incognita  Calcolo le altre incognite | Metodo algoritmo\_sostituzione\_indietro:  Obiettivo: trovare le soluzioni di un sistema utilizzando una matrice triangolare superiore  Input: U (matrice triangolare superiore) e y (vettore contenente i termini noti)  Output: x (vettore contenente le soluzioni del sistema)      x = np.zeros(len(y))  dim = len(x)  x[dim-1] = y[dim-1] / U[dim-1, dim-1]  for i in range (dim-2, -1, -1):  x[i] = (y[i] - np.dot(U[i, i:dim], x[i:dim])) / U[i,i]  return x |
| Risoluzione di un sistema utilizzando la fattorizzazione LU senza il pivoting |  |
| Teoria: | Codice: |
| Calcolo la fattorizzazione LU  Calcolo la soluzione del sistema Ly = b  Calcolo la soluzione del sistema Ux = y | Metodo ris\_sistema\_LU\_no\_piv:  Obiettivo: trovare le soluzioni di un sistema scomponendo la matrice in LU senza effettuare il pivoting  Input: A (matrice con i coefficienti delle incognite) e b (vettore contenente i termini noti)  Output: x (vettore contenente le soluzioni del sistema)  L, U = fattorizzazione\_LU\_no\_piv(A)    y = algoritmo\_sostituzione\_in\_avanti(L, b)    x = algoritmo\_sostituzione\_indietro(U, y)  return x |
| I metodi per la risoluzione del sistema utilizzando la tecnica del pivoting parziale e totale (ris\_sistema\_LU\_piv\_parz e ris\_sistema\_LU\_piv\_tot) sono molto simili al metodo precedente |  |

**TEST:**

Consideriamo la matrice

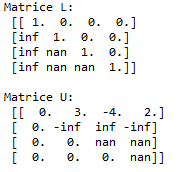
| 0 3 -4 2|  
A= | 2 5 3 1|  
 | 9 -3 -2 3|   
 |4 -1 -3 7|

I risulati attesi sono stati arrotondati alla terza cifra decimale

**Fattorizzazione LU senza pivoting**

**Risultato atteso**: errore

**Risultato ottenuto**:



**Commenti**: l’ errore è causato dall’elemento 0 in posizione 11. Questo perchè si effettua una divisione avente 0 come denominatore durante il calcolo di ln

L’errore si può risolvere usando la tecnica del pivoting parziale o pivoting totale. Infatti:

**Fattorizzazione LU con pivoting parziale:**

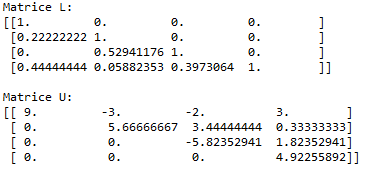
**Risultato** **atteso**:

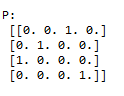
|1 0 0 0|  
L =|0.222 1 0 0|  
 |0 0.529 1 0|  
 |0.444 0.059 0.397 1|

|9 -3 -2 3 |  
U = |0 5.667 3.444 0.333|  
 |0 0 -5.824 1.824|  
 |0 0 0 4.923|

|0 0 1 0|  
P =|0 1 0 0|  
 |1 0 0 0|  
 |0 0 0 1

**Risultato** **ottenuto**:





**Commento**: come previsto, il problema riscontrato nella fattorizzazione LU senza pivoting non è stato riscontrato usando il metodo del pivoting parziale. Infatti, come si evince dalla matrice di permutazione P, la riga dell’elemento 0 (la prima) viene scambiata con la terza riga, evitando la divisione avente 0 come denominatore.

**Fattorizzazione LU con pivoting totale:**  
**Risultato** **atteso**:

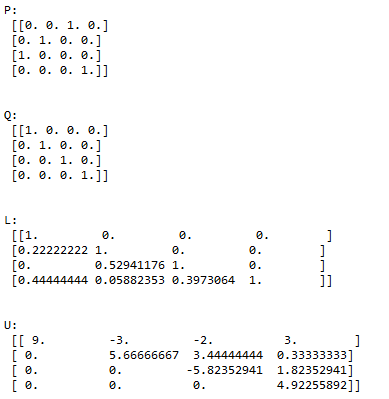
|1 0 0 0|  
L =|0.222 1 0 0|  
 |0 0.529 1 0|  
 |0.444 0.059 0.397 1|

|9 -3 -2 3 |  
U = |0 5.667 3.444 0.333|  
 |0 0 -5.824 1.824|  
 |0 0 0 4.923|

|0 0 1 0|  
P =|0 1 0 0|  
 |1 0 0 0|  
 |0 0 0 1

|1 0 0 0|  
Q =|0 1 0 0|  
 |0 0 1 0|  
 |0 0 0 1|

**Risultato** **ottenuto**:



**Commento**: come per la tecnica del pivoting parziale, anche qui, il problema riscontrato nella fattorizzazione LU senza pivoting non è stato riscontrato. Infatti, come si evince dalla matrice di permutazione P, la riga dell’elemento 0 (la prima) viene scambiata con la terza riga, evitando la divisione avente 0 come denominatore. Inoltre, la matrice Q ci indica che non sono state fatte permutazioni per Colonna.

**Risoluzione Sistema lineare**  
Consideriamo la matrice   
 | 1 3 -4 2|  
A= | 2 5 3 1|  
 | 9 -3 -2 3|   
 |4 -1 -3 7|

e il vettore:

b = |1, 3, -4, 3|

Per tutti e tre i metodi, i risultato ottenuti sono gli stessi

**Risultato atteso**:  
  
x = |-0.549 0.393 0.390 0.965]

**Risultato ottenuto**:



**ZERI DI FUNZIONE**

Nel file zeri\_di\_funzione.py vengono trattati i seguenti argomenti:

* Metodo di bisezione;
* Metodo di Newton (metodo delle tangenti);
* Metodo della direzione costante;
* Metodo della falsa posizione;
* Metodo delle secanti;
* Metodo di Brent.

Come criterio di stop è stato scelto quello in cui si confronta f(xn) con una tolleranza.

Si passa alla spiegazione teorica dei metodi.

Metodo di bisezione

Sia [a,b] un intervallo in cui f(a)\*f(b) < 0. Questo metodo, dimezza l’intervallo in cui c’è lo zero, fino a quando l’errore misto non si avvicina ad una certa tolleranza. Quindi:

1. ci = (ai + bi) / 2
2. Calcolo l’errore misto
3. Se l’errore misto è minore di rtol, il procedimento iterativo termina
4. Altrimenti,
   1. se f(ai) \* f(ci) < 0 => ai+1 = ai, bi+1 = ci
   2. se f(ci) \* f(bi) < 0 => ai+1 = ci, bi+1 = bi
5. Si ritorna al punto 1

Metodo di Newton

Sia x0 un punto iniziale, ricavo la tangente di f in x0 e calcolo lo zero. Quest’ultimo sarà x1 e così via fino a quando l’errore misto non si avvicina ad una certa tolleranza. Quindi:

1. Scelgo x0
2. Calcolo l’errore misto
3. Se l’errore misto è minore di rtol, il procedimento iterativo termina
4. Altrimenti mi calcolo xn = xn-1 – f(xn-1) / f’(xn-1)
5. Si ritorna al punto 2

Ad ogni passo quindi, viene valutata la derivata di f in un punto. Dato che può non essere possible, si usano approssimazioni con l’uso dei metodi quasi newtoniani.

Metodo della direzione costante

Questo metodo è simile a quello di newton e la derivate di f in un punto viene approssimata con il differenziale. Quindi:

1. Scelgo x0
2. Calcolo l’errore misto
3. Se l’errore misto è minore di rtol, il procedimento iterativo termina
4. Altrimenti mi calcolo xn = xn-1 – f(xn-1) / df
5. Si ritorna al punto 2

Metodo della falsa posizione

Partendo da due punti x0 e x1, si calcola la retta che passa da (x0,f(x0)) e (x1,f(x1)), si calcola lo zero e si ripete il procedimento, lasciando fisso x0 . Quindi:

1. Scelgo x0 e x1
2. Calcolo la retta passante per (x0,f(x0)) e (x1,f(x1))
3. Calcolo lo zero della retta (xn)
4. Calcolo l’errore misto
5. Se l’errore misto è minore di rtol, il procedimento iterativo termina
6. Altrimenti sostituisco xn a x1
7. Si ritorna al punto 2

Metodo delle secanti

Questo metodo è molto simile a quello della falsa posizione. L’unica differenza è nel cambiare ad ogni passo entrambi i punti, anzichè lasciarne uno fisso

1. Scelgo x0 e x1
2. Calcolo la retta passante per (x0,f(x0)) e (x1,f(x1))
3. Calcolo lo zero della retta (xn)
4. Calcolo l’errore misto
5. Se l’errore misto è minore di rtol, il procedimento iterativo termina
6. Altrimenti sostituisco xn a x1 e xn-1 a x0
7. Si ritorna al punto 2

Metodo di Brent

Questo metodo unisce i vantaggi del metodo di bisezione (converge globalmente) e del metodo delle secant (converge superlinearmente). Dato un intervallo [a,b] tale che f(a) \* f(b) < 0, si inizia con il metodo delle secanti per il calcolo di c, se c non si trova nell’intervallo, si procede con il metodo delle bisezioni. Quindi:

1. Inizializzo a1 = a, b1 = b, x0 = a1, x1 = b
2. Calcolo c con il metodo delle secanti
3. Se c si trova al di fuori dell’intervallo [ak, bk],
   1. Calcolo c con il metodo di bisezione
   2. Se f(ak) \* f(c) < 0 => a k+1 = ak, b k+1 = c, x k+1 = c, xk = ak
   3. Se f(c) \* f(bk) < 0 => a k+1 = c, b k+1 = bk, x k+1 = c, xk = bk
4. Altrimenti, se c si trova all’interno dell’intervallo [ak, bk]
   1. x k+1 = c
   2. Se f(ak) \* f(c) < 0 => ak+1 = ak, b k+1 = c
   3. Se f(c) \* f(bk) < 0 => a k+1 = c, b k+1 = bk
5. Incremento k
6. Torno al punto 2

|  |  |
| --- | --- |
| Metodo di bisezione |  |
| Teoria: | Codice: |
| Calcolo c  Calcolo l’errore misto  Fino a quando l’errore misto è maggiore di rtol  Lo zero si trova nel primo intervallo  Lo zero si trova nel secondo invervallo  Calcolo c  Non esiste uno zero in [a,b] | Metodo met\_bisezione:  Obiettivo: ricerca dello zero della funzione usando il metodo di bisezione  Input:  a: inizio dell’intervallo  b: fine dell’intervallo  atol  rtol  f: funzione  x: insieme di punti  Output:  c zero della funzione  if(f(a) == 0):  c = a  else:  if(f(b) == 0):  c = b  else:  if(f(a) \* f(b) < 0):  passo = 1  c = (a + b)/2  errore = errore\_misto(c, atol, rtol, f)      while errore > rtol:  passo += 1  if (f(a)\*f(c)) < 0:  #a quella precedente  b = c  else:  #b quella precedente  if (f(c)\*f(b)) < 0:  a = c  c = (a + b)/2  errore = errore\_misto(c, atol, rtol,f)  else:  c = None  return c |
| Metodo di Newton |  |
| Teoria: | Codice: |
| Se la derivata di f(x0) è uguale a 0 allora non è possibile calcolare lo zero  Calcolo l’errore misto  Fino a quando l’errore misto è maggiore di rtol  Calcolo xn+1  Calcolo l’errore | Metodo met\_newton:  Obiettivo: ricerca dello zero della funzione usando il metodo di newton  Input:  x0: punto iniziale  f: funzione  fder: derivata della funzione  x: insieme di punti  atol  rtol  Output:  xsucc: zero della funzione  if(fder(x0) == 0):  xsucc = None  else:    errore = errore\_misto(x0, atol, rtol, f)  xsucc = x0  while errore > rtol:  xprec = xsucc  xsucc = xprec - f(xprec)/fder(xprec)  errore = errore\_misto(xsucc, atol, rtol,f)  return xsucc |
| Metodo della direzione costante |  |
| Teoria: | Codice: |
| Se il differenziale è uguale a 0 allora non è possibile calcolare lo zero  Calcolo l’errore misto  Fino a quando l’errore misto è maggiore di rtol  Calcolo xn+1  Calcolo l’errore | Metodo met\_dir\_costante:  Obiettivo: ricerca dello zero della funzione usando il metodo della direzione costante  Input:  x0: punto iniziale  f: funzione  fd: differenziale  x: insieme di punti  atol  rtol  Output:  xsucc: zero della funzione  if(df == 0):  xsucc = None  else:  errore = errore\_misto(x0, atol, rtol, f)  xsucc = x0  while errore > rtol:  xprec = xsucc  xsucc = xprec - f(xprec)/df  errore = errore\_misto(xsucc, atol, rtol,f)    return xsucc |
| Metodo della falsa posizione |  |
| Teoria: | Codice: |
| Se la differenza tra x1 e x0 è uguale a 0 allora non è possibile calcolare lo zero  Calcolo l’errore misto  Fino a quando l’errore misto è maggiore di rtol  Calcolo xn+1  Calcolo l’errore | Metodo met\_falsa\_posizione  Obiettivo: ricerca dello zero della funzione usando il metodo della falsa posizione  Input:  x0: punto iniziale  x1: secondo punto  f: funzione  x: insieme di punti  atol  rtol  Output:  xsucc: zero della funzione  if((x1-x0) == 0):  xsucc = None  else:  errore = errore\_misto(x0, atol, rtol, f)  xsucc = x1    while errore > rtol:  xprec = xsucc  xsucc = xprec - f(xprec)/((f(xprec)-f(x0))/(xprec-x0))  errore = errore\_misto(xsucc, atol, rtol,f)    return xsucc |
| Metodo delle secanti |  |
| Teoria: | Codice: |
| Se la differenza tra x1 e x0 è uguale a 0 allora non è possibile calcolare lo zero  Calcolo l’errore misto  Fino a quando l’errore misto è maggiore di rtol  Calcolo xn+1  Calcolo l’errore | Metodo met\_secanti:  Obiettivo: ricerca dello zero della funzione usando il metodo della falsa posizione  Input:  x0: punto iniziale  x1: secondo punto  f: funzione  punti: insieme di punti  atol  rtol  Output:  xsucc: zero della funzione  if((x1-x0) == 0):  xsucc = None  else:  errore = errore\_misto(x0, atol, rtol, f)  xsucc = x1  x = x0  while errore > rtol:  passo += 1  xprec = x #Xn-1  x = xsucc #Xn  xsucc = x - f(x)/((f(x)-f(xprec))/(x-xprec))  errore = errore\_misto(xsucc, atol, rtol,f)        return xsucc |
| Metodo di Brent |  |
| Teoria: | Codice: |
| Se la differenza tra b e a è uguale a 0 allora non è possibile calcolare lo zero  Calcolo l’errore misto  Fino a quando l’errore misto è maggiore di rtol  Calcolo c con il metodo delle secanti  Se c si trova fuori dall’intervallo calcolo c con il metodo di bisezione | Metodo met\_Brent:  Obiettivo: ricerca dello zero della funzione usando il metodo di Brent  Input:  a: inizio dell’intervallo  b: fine dell’intervallo f: funzione  punti: insieme di punti  atol  rtol  Output:  xsucc: zero della funzione  if((b-a) == 0):  c = None  else:  xprec = a  x = b  k = 1  errore = errore\_misto(x, atol, rtol, f)  while errore > rtol:  c = x - f(x)/((f(x) - f(xprec))/(x - xprec))    if c < a or c > b:  c = (a+b) / 2  if(f(a) \* f(c) < 0):  b = c  x = c  xprec = a    else :  if f(c) \* f(b) < 0:  a = c  x = c  xprec = b  else:  x = c  if f(a) \* f(c) < 0:  b = c  else:  if f(c) \* f(b) < 0:  a = c  errore = errore\_misto(c, atol, rtol, f)  k+=1    return c |

**TEST**

Funzione presa in considerazione nel test è x^2 – 5x – 10.

Intervallo: [-4, 2]

atol: 0.003

rtol: 0.002

Risultato atteso per tutti i metodi: -1.532

**Metodo di bisezione:**

Passo: 1

a: -4

b: 2

c: -1.0

atol: 0.003

rtol: 0.002

errore: 1.6

Passo: 2

a: -4

b: -1.0

c: -2.5

rtol: 0.002

errore: 2.1875

Passo: 3

a: -2.5

b: -1.0

c: -1.75

rtol: 0.002

errore: 0.5576923076923077

Passo: 4

a: -1.75

b: -1.0

c: -1.375

rtol: 0.002

errore: 0.42934782608695654

Passo: 5

a: -1.75

b: -1.375

c: -1.5625

rtol: 0.002

errore: 0.08290816326530612

Passo: 6

a: -1.5625

b: -1.375

c: -1.46875

rtol: 0.002

errore: 0.1680921052631579

Passo: 7

a: -1.5625

b: -1.46875

c: -1.515625

rtol: 0.002

errore: 0.04136981865284974

Passo: 8

a: -1.5625

b: -1.515625

c: -1.5390625

rtol: 0.002

errore: 0.021067641388174806

Passo: 9

a: -1.5390625

b: -1.515625

c: -1.52734375

rtol: 0.002

errore: 0.010075604838709678

Passo: 10

a: -1.5390625

b: -1.52734375

c: -1.533203125

rtol: 0.002

errore: 0.005514779861558274

Passo: 11

a: -1.533203125

b: -1.52734375

c: -1.5302734375

rtol: 0.002

errore: 0.0022757084877537866

Passo: 12

a: -1.533203125

b: -1.5302734375

c: -1.53173828125

rtol: 0.002

errore: 0.0016207099824851023

Lo zero è: -1.53173828125

**Metodo di newton:**

x0 = -2

Passo: 0

x 0 : -2

atol: 0.003

rtol: 0.002

errore: 1.1428571428571428

Passo: 1

x 0 : -2

x 1 : -1.5555555555555556

errore: 0.06464646464646494

Passo: 2

x 1 : -1.5555555555555556

x 2 : -1.5312024353120242

errore: 0.00019565650208176136

Lo zero è: -1.5312024353120242

**Metodo della direzione costante**

x0 = -2

differenziale = -9

passo: 0

x 0 : -2

atol: 0.003

rtol: 0.002

errore: 1.1428571428571428

passo: 1

x 0 : -2

x 1 : -1.5555555555555556

rtol: 0.002

errore: 0.06464646464646494

passo: 2

x 1 : -1.5555555555555556

x 2 : -1.5336076817558297

rtol: 0.002

errore: 0.006589820575634739

passo: 3

x 2 : -1.5336076817558297

x 3 : -1.53138646727587

rtol: 0.002

errore: 0.0006851150644890307

lo zero è: -1.53138646727587

**Metodo della falsa posizione**

Passo: 0

x 0 : -4

x 1 : 2

atol: 0.003

rtol: 0.002

errore: 4.7272727272727275

Passo: 1

x 0: -4

x 0 : 2

x 1 : -0.2857142857142856

errore: 4.754285714285715

Passo: 2

x 0: -4

x 1 : -0.2857142857142856

x 2 : -1.1999999999999997

errore: 0.9481481481481491

Passo: 3

x 0: -4

x 2 : -1.1999999999999997

x 3 : -1.4509803921568627

errore: 0.21679369422187483

Passo: 4

x 0: -4

x 3 : -1.4509803921568627

x 4 : -1.5121951219512195

errore: 0.05055791448602735

Passo: 5

x 0: -4

x 4 : -1.5121951219512195

x 5 : -1.5266821345707657

errore: 0.011838371370207578

Passo: 6

x 0: -4

x 5 : -1.5266821345707657

x 6 : -1.5300859598853869

errore: 0.002774560209175826

Passo: 7

x 0: -4

x 6 : -1.5300859598853869

x 7 : -1.5308843537414967

errore: 0.0006504130583414627

Lo zero è: -1.5308843537414967

**Metodo delle secant**

Passo: 0

x 0: -4

x 1: 2

atol: 0.003

rtol: 0.002

errore: 4.7272727272727275

Passo: 1

x 0 : -4

x 1 : 2

x 2 : -0.2857142857142856

errore: 4.754285714285715

Passo: 2

x 1 : 2

x 2 : -0.2857142857142856

x 3 : -2.8695652173913047

errore: 2.8795154661475237

Passo: 3

x 2 : -0.2857142857142856

x 3 : -2.8695652173913047

x 4 : -1.3267326732673266

errore: 0.5681885176258437

Passo: 4

x 3 : -2.8695652173913047

x 4 : -1.3267326732673266

x 5 : -1.5013808921967888

errore: 0.079613672554414

Passo: 5

x 4 : -1.3267326732673266

x 5 : -1.5013808921967888

x 6 : -1.5319056097502781

errore: 0.002065646739008379

Passo: 6

x 5 : -1.5013808921967888

x 6 : -1.5319056097502781

x 7 : -1.531125997827531

errore: 7.650503149549105e-06

Lo zero è: -1.531125997827531

**Metodo di Brent**

Passo: 0

a 1 : -4

b 1 : 2

x 0 : -4

x 1 : 2

atol: 0.003

rtol: 0.002

errore: 4.571428571428571

Passo: 1

c si trova nella prima metà

c: -0.2857142857142856

a 1 : -4

b 1 : -0.2857142857142856

x 0 : -4

x 1 : -0.2857142857142856

atol: 0.003

rtol: 0.002

errore: 4.754285714285715

è stato usato il metodo delle secanti

Passo: 2

c si trova nella prima metà

c: -1.1999999999999997

a 2 : -4

b 2 : -1.1999999999999997

x 1 : -4

x 2 : -1.1999999999999997

atol: 0.003

rtol: 0.002

errore: 0.9481481481481491

è stato usato il metodo delle secanti

Passo: 3

c si trova nella prima metà

c: -1.4509803921568627

a 3 : -4

b 3 : -1.4509803921568627

x 2 : -4

x 3 : -1.4509803921568627

atol: 0.003

rtol: 0.002

errore: 0.21679369422187483

è stato usato il metodo delle secanti

Passo: 4

c si trova nella prima metà

c: -1.5121951219512195

a 4 : -4

b 4 : -1.5121951219512195

x 3 : -4

x 4 : -1.5121951219512195

atol: 0.003

rtol: 0.002

errore: 0.05055791448602735

è stato usato il metodo delle secanti

Passo: 5

c si trova nella prima metà

c: -1.5266821345707657

a 5 : -4

b 5 : -1.5266821345707657

x 4 : -4

x 5 : -1.5266821345707657

atol: 0.003

rtol: 0.002

errore: 0.011838371370207578

è stato usato il metodo delle secanti

Passo: 6

c si trova nella prima metà

c: -1.5300859598853869

a 6 : -4

b 6 : -1.5300859598853869

x 5 : -4

x 6 : -1.5300859598853869

atol: 0.003

rtol: 0.002

errore: 0.002774560209175826

è stato usato il metodo delle secanti

Passo: 7

c si trova nella prima metà

c: -1.5308843537414967

a 7 : -4

b 7 : -1.5308843537414967

x 6 : -4

x 7 : -1.5308843537414967

atol: 0.003

rtol: 0.002

errore: 0.0006504130583414627

è stato usato il metodo delle secanti

Lo zero è: -1.5308843537414967

**INTERPOLAZIONE**

Nel file interpolazione.py vengono trattati i seguenti argomenti:

* Base delle potenze
* Base di Lagrange

Per interpolazione si intende un metodo per individuare i nuovi punti del piano cartesiano a partire da un insieme finito di punti dati.   
Base delle potenze: la base delle potenze è data da {1, x, x2,…,xn}. Quindi, il polinomio è:  
p(x) = a0 + a1x + a2x2 + … + anxn

{ p(x0) = y0

La condizione di interpolazione è | p(x1) = y1  
 | …

{p(xn) = yn

Usando la base delle potenze:

p(x0) = a0 + a1x0 + a2x02 + … + anx0n = y0

…

p(xn) = a0 + a1x1 + a2x12 + … + anx1n = y1

Posso risolvere il problema usando la matrice di Vandermonde e il vettore contenente i coefficienti di x:

| 1 x0 … x0n |

| 1 x1 … x1n |

V = | … … … … |

| 1 xn … xnn |

| a0 |

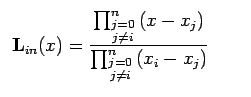
a = | … |

| an |

Risolvendo V\*a = y è possibile ricavare il polinomio che interpola i punti dati

Base di Lagrange:

La base di Lagrange è data da (l0(x), l1(x), … ln(x)) in cui:



Inoltre, li (xi) = 1 e li(xj) = 0 per i != j

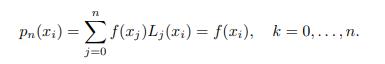
Il polinomio è dato da:

p(x) = a0l0(x) + a1l1(x) + … + anln(x)

p(x0) = a0l0(x0) + a1l1(x0) + … + anln(x0)

…

p(xn) = a0l0(xn) + a1l1(xn) + … + anln(xn)



|  |  |
| --- | --- |
| Costruzione della matrice di Vandermonde |  |
| Teoria: | Codice: |
| Calcolo ogni elemento della matrice | Metodo mat\_Vandermonde:  Obiettivo: Calcolare la matrice di Vandermonde  Input:  x: array di punti  Output:  V: Matrice di Vandermonde  V = np.zeros(shape = (len(x),len(x)))    for i in range (0, len(x)):  for j in range(0, len(x)):  V[i,j] = x[i]\*\*j        return V |
| Calcolo dei valori dei punti usando la base di Lagrande |  |
| Calcolo ogni termine di Lj(xi)  Calcolo p(xi) | Metodo pol\_lagrange:  Obiettivo: Calcolare i valori dei punti dati usando la base di Lagrange  Input:  x: array di punti inziali  y: array dei valori iniziali  punti: array dei punti di cui si deve calcolare i valori  Output:  l: array dei valori dei punti  l = np.zeros(len(punti))    for k in range(0, len(punti)):  somma = 0  for i in range(0, len(x)):  term = y[i]  for j in range(0, len(x)):  if i != j:  term \*= (punti[k] - x[j])/(x[i]-x[j])  somma += term  l[k] = somma  return l |

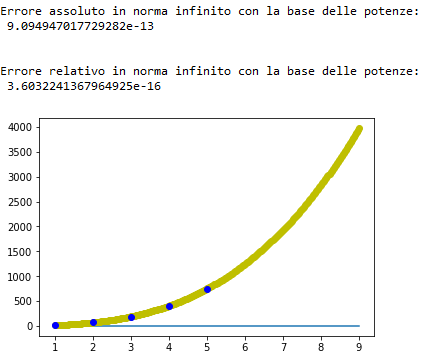
**TEST**

La funzione presa in considerazione è f(x) = 5x3 + 3x2 + 10x -3

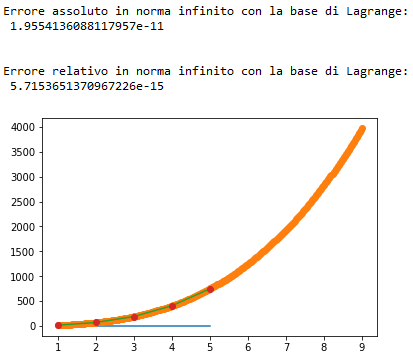
Punti iniziali: 1,2,3,4,5

Valori iniziali: 15,69,189,405,747

**Base delle potenze:**



**Base di Lagrange:**



Commenti: in entrambi i metodi, la curva formata dai punti di cui si vogliono calcolare i valori segue la curva formata dai punti e dai valori iniziali. Inoltre, si può evincere che, con questi dati, la base delle potenze è migliore, poichè gli errori (in norma infinito) sono minori di quelli della base di Lagrange