Progetto di Esperienze di Programmazione
Implementazione di metodi per la ricerca degli zeri per funzioni non lineari

Massagli Lorenzo

2020-08-25

Indice

1 Descrizione del Problema

1.1 Introduzione

In matematica si presentano spesso problemi che richiedono di calcolare uno zero di una funzione di variabile reale $f(x)$ o, secondo un'espressione equivalente, trovare una radice reale di un'equazione della forma $f(x)$=0.
La risoluzione del problema dipende strettamente dalla forma della funzione f: ad esempio, se essa é un polinomio o una funzione razionale esistono, per i gradi più bassi, formule che permettono di determinare in modo preciso tutti gli zeri, senza approssimazioni. In tutti gli altri casi, a parte alcuni casi elementari risolvibili attraverso le definizioni, non esistono metodi algebrici per ricavare con esattezza i valori degli zeri.
Per questo tipo di problema si preferisce parlare di algoritmi per la soluzione di equazioni, sottintendendo che questi metodi possono applicarsi sia ad equazioni lineari che ad equazioni non lineari. Taluni algoritmi per il calcolo di uno zero di una funzione reale possono essere direttamente generalizzati per risolvere equazioni non lineari.
Andremo a vedere alcuni di questi algoritmi conosciuti, tra cui:

- Metodo della Bisezione
- Metodo di Punto Fisso
- Metodo di Newton (Anche chiamato come Metodo delle tangenti)
- Metodo delle Corde
- Metodo delle Secanti

Andando a confrontare i vari algoritmi tra loro e andando ad analizzare i risultati ottenuti su alcuni esempi.

1.2 Il Problema del Calcolo degli Zeri di una Funzione.

Data una funzione $f\colon [a,b] \to \mathbb{R}$ si cercano (se esistono) valori della variabile per cui la funzione si annulla. Se $\xi \in [a,b]$ é un tale valore, i.e., $f(\xi) = 0$, allora ξ é detto zero della funzione o, equivalentemente, radice dell'equazione $f(x) = 0$.
L'esistenza di almeno uno zero é assicurata dal seguente *teorema di esistenza degli zeri.*

Teorema 1 *Sia* $f\colon [a,b] \to \mathbb{R}, f \in \mathbb{C}^0([a,b])$ *con* $f(a)f(b) < 0$ *allora* $\exists\ \xi \in (a,b)$ *tale che* $f(\xi) = 0$.

Per l'approssimazione numerica degli zeri di una funzione si introducono tecniche iterative che a partire da un dato iniziale x_0 generano una successione x_k di approssimazioni che sotto opportune ipotesi convergono ad uno zero della funzione, i.e.,

$$\lim_{k\to\infty} x_k = \xi, \qquad f(\xi) = 0$$

Il caso non lineare presenta molteplici criticitá rispetto al caso lineare.
In particolare la convergenza dipende fortemente dalla scelta del punto iniziale e dalle proprietá della funzione ed eventualmente delle sue derivate. Qualunque metodo si intenda applicare, sará quindi generalmente necessario effettuare uno studio preliminare della funzione in modo da localizzare le eventuali radici, i.e. determinare un intervallo che contenga una e una sola radice.

2 Metodi Numerici per l'Approssimazione degli Zeri di una Funzione

2.1 Il metodo di Bisezione

Il metodo della bisezione é il metodo più primitivo per la ricerca di radici reali di una funzione $f(x) = 0$ dove f é una funzione continua. Sia $f : [a, b] \in \mathbb{R}$ con $f \in \mathbb{C}^0([a, b])$ e $f(a)f(b) < 0$, dal teorema di esistenza degli zeri segue che $\exists\ \xi \in [a, b]$ tale che $f(\xi) = 0$. Scelti a e b come estremi dell'intervallo e dopo essersi assicurati che dell'esistenza dello zero della funzione nell'intervallo, si procede con l'andare ad approssimare la radice andando a ripetere in modo iterativo il seguente algoritmo.

Algoritmo del metodo di bisezione per la ricerca delle radici
Input:

- $f(x)$ é la funzione data;
- a, b sono i due numeri tali che $f(a)f(b) < 0$.

Output: Una approssimazione della radice di $f(x) = 0$ nell'intervallo $[a, b]$, per $k = 0, 1, 2, 3...$ fin quando non é soddisfatto l'algoritmo.

Iterazione:

- Calcolare $c_k = \frac{a_k+b_k}{2}$;
- Testare se c_k é la radice desiderata. Se si, stop;
- Se c_k non é la radice desiderata, testare se $f(c_k)f(a_k) < 0$. Se si, $a_{k+1} = a_k$ e $b_{k+1} = c_k$. Altrimenti $a_{k+1} = c_k$ e $b_{k+1} = b_k$;
- Poni $k = k + 1$.

Teorema 2 *Sia $f : [a, b] \in \mathbb{R}$ con $f \in \mathbb{C}^0([a, b])$ e $f(a)f(b) < 0$. Per le successioni generate come sopra si ha*

$$\lim_{k\to\infty} a_k = \lim_{k\to\infty} b_k = \lim_{k\to\infty} c_k = \xi \in [a, b]$$

con

$$f(\xi) = 0$$

2.1.1 Critero d'arresto

Come criterio di arresto conviene fissare una tolleranza $\epsilon > 0$ e arrestare l'iterazione quando

$$b_{k+1} - a_{k+1} \leq \epsilon,$$

che garantisce

$$0 \leq \xi - a_{k+1} \leq \epsilon \qquad 0 \leq b_{k+1} - \xi \leq \epsilon \qquad 0 \leq |\xi - c_{k+1}| \leq \frac{\epsilon}{2}$$

Poiché $0 \leq b_{k+1} - a_{k+1} \leq (b - a)/2^k$ si ha che la condizione risulta soddisfatta dopo

$$k \geq \lceil log_2(\frac{b-a}{\epsilon})\rceil$$

iterazioni. Questo numero può essere significativamente elevato richiedendo molte valutazioni della funzione f.

2.2 Il metodo di Punto Fisso

Il metodo di Punto Fisso rappresenta la generalizzazione dei metodi iterativi. Dato un problema senza far riferimento ad un metodo in particolare é possibile generare infiniti metodi iterativi che consentono di ottenere la radice del problema come approssimazione del punto fisso ξ. Siano $f, g : [a, b] \to \mathbb{R}$, le equazioni $f(x) = 0$ e $g(x) - x = 0$ si dicono *equivalenti* se $f(\xi) = 0 \iff g(\xi) = \xi$.
Questo vuol dire che la radice ξ dell'equazione $f(x) = 0$ é detta *punto fisso* della funzione $g(x)$. La soluzione si approssima(scelto un punto x_0 iniziale) con la successione:

$$\begin{cases} x_0 \in [a, b] \\ x_{k+1} = g(x_k), \quad k \geq 0 \end{cases} \tag{1}$$

Teorema del punto fisso:
Sia $g : [a, b] \to \mathbb{R}, g \in \mathbb{C}^1([a, b]), \xi$ punto fisso.
Se $\exists \rho > 0 : |g'(x)| < 1 \; \forall x \in I_\epsilon = [\xi - \rho, \xi + \rho] \subset [a, b] \Rightarrow$ la successione é localmente convergente, cioé:

1. $x_k \in I_\epsilon$ per ogni $k \geq 0$;
2. $\lim_{k\to\infty} x_k = \xi$.

Questo teorema ci dice che se tutti i punti in un intorno di ξ sono tali che $-1 < g'(x) < 1$, allora c'é convergenza locale. Non é peró sempre cosí facile determinare un intervallo siffatto. Se però, si conosce bene il comportamento di g nei pressi del punto fisso, si può sfruttare il seguente teorema:

Teorema 3 *Sia $g : [a, b] \to \mathbb{R}, g \in \mathbb{C}^1([a, b])$, con ξ punto fisso.*
Se $|g'(\xi)| < 1$ allora il metodo di punto fisso é localmente convergente in ξ.

Algoritmo del metodo di Punto Fisso per la ricerca delle radici
Input:

- $g(x)$ é la funzione la quale $g(x) - x = 0$ é equivalente a $f(x) = 0$
- $x_0 \in [a, b]$ punto iniziale

Output: Una approssimazione della radice di $f(x) = 0$ nell'intervallo $[a, b]$, per $k = 0, 1, 2, 3...$ fin quando non é soddisfatto l'algoritmo.

Iterazione:

- Calcolare $x_{k+1} = g(x_k)$;
- Controllare i criteri d'arresto;
- Se x_{k+1} é la radice desiderata, stop. Sennò continuare con l'iterazione.

2.2.1 Critero d'arresto

Il metodo di punto fisso richiede la selezione di un opportuno criterio di arresto del tipo:

$$|x_{k+1} - x_k| \leq tol, \qquad \tfrac{|x_{k+1} - x_k|}{|x_{k+1}|} \leq tol$$

Se $g \in \mathbb{C}^1([a, b])$ allora

$$|x_{k+1} - x_k| = |x_{k+1} - \xi + \xi - x_k| = |g'(\xi_k) - 1||x_k - \xi|,$$

da cui si conclude che

$$|x_k - \xi| \leq \frac{tol}{|g'(\xi_k)-1|}$$

Ne segue che l'approssimazione restituita può essere scadente se $g'(\xi)$ é prossimo ad 1.

2.3 Il metodo di Newton

Il metodo di Newton é il più noto metodo di iterazione funzionale. Esso genera una successione di punti a partire da un punto iniziale x_0 che dopo un certo numero di iterazioni converge ad un'approssimazione della radice della funzione.
Sia $f : [a,b] \to \mathbb{R}, f \in \mathbb{C}^1([a,b]), \xi$ punto fisso, il metodo é definito nel seguente modo:

$$\begin{cases} x_0 \in [a,b] \\ x_{k+1} = g(x_k) = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k \geq 0 \end{cases} \tag{2}$$

Teorema 4 *Sia* $f : [a,b] \to \mathbb{R}, f \in \mathbb{C}^2([a,b]), f(\xi) = 0, f'(\xi) \neq 0, \xi \in (a,b)$.
Allora il metodo é locamente convergente in ξ*, cioé* $\exists \rho > 0$ *tale che* $\forall\, x_0 \in I_\xi = [\xi - \rho, \xi + \rho] \subset [a,b]$ *la successione generata dal metodo soddisfa:*

1. $x_k \in I_\xi$ *per ogni* $k \geq 0$
2. $\lim_{k\to\infty} x_k = \xi$.

Se inoltre tale successione verifica $x_k \neq \xi, k \geq 0$*, allora la convergenza é almeno quadratica, cioé:*

$$\lim_{k\to\infty} \frac{|x_{k+1}-\xi|}{|x_k-\xi|^2} = l \in \mathbb{R}$$

Teorema 5 *Sia* $f : [a,b] \to \mathbb{R}, f \in \mathbb{C}^2([a,b]), f(\xi) = 0, \xi \in (a,b)$.
Se $\exists \delta > 0$ *tale che* $\forall\, x \in I_\delta = (\xi, \xi + \delta) \subset [a,b]$ *si ha*

1. $f'(x) \neq 0$*;*
2. $f(x)f''(x) > 0$*;*

Allora il metodo di Newton $x_0 \in I_\delta$ *genera successioni convergenti a* ξ*.*

Algoritmo del metodo di Newton per la ricerca delle radici

Input:

- $f(x)$ é la funzione data;
- $x_0 \in [a,b]$ punto iniziale.

Output: Una approssimazione della radice di $f(x) = 0$ nell'intervallo $[a,b]$, per $k = 0,1,2,3...$ fin quando non é soddisfatto l'algoritmo.

Iterazione:

- Calcolare $f(x_k)$;
- Calcolare $f'(x_k)$;
- Calcolare $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$;
- Controllare i criteri d'arresto;
- Se x_{k+1} é la radice desiderata, stop. Sennò continuare con l'iterazione.

2.3.1 Critero d'arresto

Il metodo di Newton richiede la selezione di un opportuno criterio di arresto del tipo

$$|x_{k+1} - x_k| \leq tol, \qquad \frac{|x_{k+1}-x_k|}{|x_{k+1}|} \leq tol$$

2.4 Metodi quasi-Newton

I metodi quasi-Newton rappresentano una variante al metodo di Newton. La differenza sta nell'iterazione, infatti:

$$\begin{cases} x_0 \in [a, b] \\ x_{k+1} = x_k - \frac{f(x_k)}{m_k}, \quad k \geq 0 \end{cases} \tag{3}$$

Dove come coefficiente angolare m_k é utilizzata un'approssimazione del valore della derivata della funzione.

2.4.1 Metodo delle Corde

Nel metodo delle corde il termine m_k é un valore che resta costante ad ogni iterazione. Per esempio, si può considerare m_k pari al valore della derivata sul punto iniziale $f'(x_0)$. In questo caso avremmo che il metodo delle corde é praticamente identico a quello di Newton, tranne per il fatto che la derivata sará costante nel calcolo, e si dovrá solo calcolare $f(x)$ ad ogni iterazione.

Algoritmo del metodo delle Corde per la ricerca delle radici
Input:

- $f(x)$ é la funzione data;
- m é la derivata della funzione $f(x)$ nel punto iniziale;
- $x_0 \in [a, b]$ punto iniziale;

Output: Una approssimazione della radice di $f(x) = 0$ nell'intervallo $[a, b]$, per $k = 0, 1, 2, 3...$ fin quando non é soddisfatto l'algoritmo.

Iterazione:

- Calcolare $f(x_k)$;
- Calcolare $x_{k+1} = x_k - \frac{f(x_k)}{m}$
- Controllare i criteri d'arresto;
- Se x_{k+1} é la radice desiderata, stop. Sennò continuare con l'iterazione.

Il metodo delle corde converge linearmente se, detta ξ la soluzione corretta, vale: $0 < \frac{f'(\xi)}{m} < 2$.
Negli altri casi i metodo potrebbe non convergere affatto.

2.4.2 Metodo delle Secanti

Nel metodo delle Secanti il termine m_k é il coefficente angolare della retta secante la curva $f(x)$ in due punti d'ascissa x_k e x_{k-1}. Il metodo quindi richiede due punti iniziali x_0 e x_1. Quindi avremo che m_k sará: $m_k = \frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$.

Algoritmo del metodo delle Secanti per la ricerca delle radici
Input:

- $f(x)$ é la funzione data;
- $x_0 \in [a, b]$ punto iniziale;

Output: Una approssimazione della radice di $f(x) = 0$ nell'intervallo $[a, b]$, per $k = 0, 1, 2, 3...$ fin quando non é soddisfatto l'algoritmo.

Iterazione:

- Calcolare $f(x_k)$;
- Calcolare $f(x_{k-1})$;
- Calcolare $m = \frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$
- Assegnamento x_{k-1}=x_k;
- Calcolare $x_{k+1} = x_k - \frac{f(x_k)}{m}$
- Controllare i criteri d'arresto;
- Se x_{k+1} é la radice desiderata, stop. Sennò continuare con l'iterazione.

A partire dai punti iniziali x_0, x_1 il metodo calcola la retta secante passante per i due punti e calcola il nuovo punto come intersezione della retta con l'asse delle ascisse. Rinomina i nuovi x_0, x_1 e itera il procedimento fino ad ottenere un intervallo talmente ristretto da considerare x_1 soluzione del problema.

Rispetto al metodo delle corde, richiedendo un punto iniziale in più, ha una velocitá di convergenza superlineare. Si dimostra infatti che, detta ξ la soluzione corretta, se $f \in \mathtt{C}^2([a, b])$, $f''(\xi) \neq 0$ e x_0, x_1 sono abbastanza vicini ad ξ, allora il metodo converge con ordine: $p = \frac{1+\sqrt{5}}{2} \simeq 1.618$

3 Scelte Implementative

L'implementazione dei vari metodi per la ricerca della radice di una funzione é stata effettuata in MATLAB, ovvero un ambiente desktop ottimizzato per l'analisi iterativa e i processi di progettazione, con un linguaggio di programmazione che esprime le operazioni matematiche con matrici e array in modo diretto. E' usato principalmente per l'analisi di dati e la simulazione completa di sistemi wireless. Per tutte l'implementazioni dei vari metodi é stato scelto l'approccio simbolico di rappresentazione delle funzioni (syms) al posto dell'approccio "function handle". Questa scelta é stata effettuata per poter sviluppare programmi non ad-hoc per il problema, ma soluzione generale di un insieme di essi. Inoltre nelle varie implementazioni non é stata settata una tolleranza e numero massimo di iterazioni fisse, ma vengono passate come parametri della funzione. Solitamente la tolleranza viene passata alla funzione con valori pari a 10^{-8} o 10^{-12} a seconda della precisione che si vuole ottenere del risultato. Ogni implementazione ha la visualizzazione grafica e tabulare dei vari risultati ottenuti.

3.1 Metodo di bisezione

Il metodo di bisezione per essere implementato ha bisogno principlamente di 3 parametri i quali sono:

- f: Funzione data per la ricerca della radice
- a : Estremo sinistro dell'intervallo iniziale
- b : Estremo destro dell'intervallo iniziale
- itr e tol : Numero massimo di iterazioni e tolleranza sull'errore.

Nell'iterazione é stato utilizzato come criterio di arresto che la tolleranza sia maggiore dell'errore prodotto dall'approssimazione della radice.
Per la rappresentazione grafica del metodo é stata utilizzata una combinazione tra plot e fplot, per poter rappresentare la funzione e i vari intervalli nel tempo.
Per lo store dei risultati ottenuti durante l'esecuzione é stata utilizzata una matrice la quale conterrá per ogni iterazione i seguenti valori:

- a_k : Valore dell'estremo sinistro all'iterazione k-esima;
- b_k : Valore dell'estremo destro all'iterazione k-esima;
- x_k : Valore approssimato della possibile radice all'iterazione k-esima;
- err : Valore dell'errore prodotto nell'approssimazione della radice all'iterazione k-esima.

3.2 Metodo di Punto Fisso

Il metodo di punto fisso per essere implementato ha bisogni principalmente di 2 parametri:

- g : Funzione la quale $g(x) - x = 0$ é equivalente a $f(x) = 0$ (Se $g(\xi) = \xi$ e $f(\xi) = 0$);
- x_0 : Punto iniziale x_0 del metodo di Punto Fisso.

Nell'iterazione é stato utilizzato come criterio di arresto che la tolleranza sia maggiore dell'errore prodotto dall'approssimazione della radice.
Per la rappresentazione grafica del metodo é stata utilizzata una combinazione tra plot e fplot, per poter rappresentare la funzione e i vari passaggi che il metodo fa per trovare il nuovo x_k.
Per lo store dei risultati ottenuti durante l'esecuzione é stata utilizzata una matrice la quale conterrá per ogni iterazione i seguenti valori:

- x : Valore approssimato della possibile radice all'iterazione k-esima;;
- $f(x)$: Valore della funzione in x;
- err : Valore dell'errore prodotto nell'approssimazione della radice all'iterazione k-esima.

3.3 Metodo di Newton

Il metodo di Newton per essere implementato ha bisogni principalmente di 2 parametri:

- f : Funzione data per la ricerca della radice
- x_0 : Punto iniziale x_0 del metodo di Newton.

Nell'iterazione é stato utilizzato come criterio di arresto che la tolleranza sia maggiore dell'errore prodotto dall'approssimazione della radice.
Per la rappresentazione grafica del metodo é stata utilizzata una combinazione tra plot e fplot, per poter rappresentare la funzione e i vari passaggi che il metodo fa per trovare il nuovo x_k tramite le tangenti.
Per lo store dei risultati ottenuti durante l'esecuzione é stata utilizzata una matrice la quale conterrá per ogni iterazione i seguenti valori:

- x : Valore approssimato della possibile radice all'iterazione k-esima;;
- $f(x)$: Valore della funzione in x;
- err : Valore dell'errore prodotto nell'approssimazione della radice all'iterazione k-esima.

3.4 Metodi Quasi-Newton

3.4.1 Metodo delle corde

Per il metodo delle Corde l'implementazione é pressoché identica a quella del metodo di Newton. L'unica differenza presente tra i due é che il metodo delle corde utilizza la derivata solo all'inizio su x_0 invece che andare a calcolare la derivata in ogni sua iterazione.

3.4.2 Metodo delle secanti

A differenza del metodo delle corde, l'implementazione del metodo delle Secanti é leggermente differente da quello del metodo di Newton. Infatti esso utilizza 2 punti iniziali x_0 e x_1. Per questo motivo, per essere implementato ha bisogno principalmente di 3 parametri:

- f : Funzione data per la ricerca della radice
- x_0 : Primo punto iniziale
- x_1 : Secondo punto iniziale

Nell'iterazione é stato utilizzato come criterio di arresto che la tolleranza sia maggiore dell'errore prodotto dall'approssimazione della radice.
Per la rappresentazione grafica del metodo é stata utilizzata una combinazione tra plot e fplot, per poter rappresentare la funzione e i vari passaggi che il metodo esegue per trovare il nuovo x_1.
Per lo store dei risultati ottenuti durante l'esecuzione é stata utilizzata una matrice la quale conterrá per ogni iterazione i seguenti valori:

- x : Valore approssimato della possibile radice all'iterazione k-esima, esso corrisponde a x_1;
- $f(x)$: Valore della funzione in x_1;
- err : Valore dell'errore prodotto nell'approssimazione della radice all'iterazione k-esima.

4 Testing

Si procede con il testing, andando a confrontare i risultati ottenuti con i vari metodi descritti.

4.1 Primo caso di applicazione: $f(x) = x^2 - 2$

Come primo caso di applicazione si inizia con un polinomio di secondo grado.

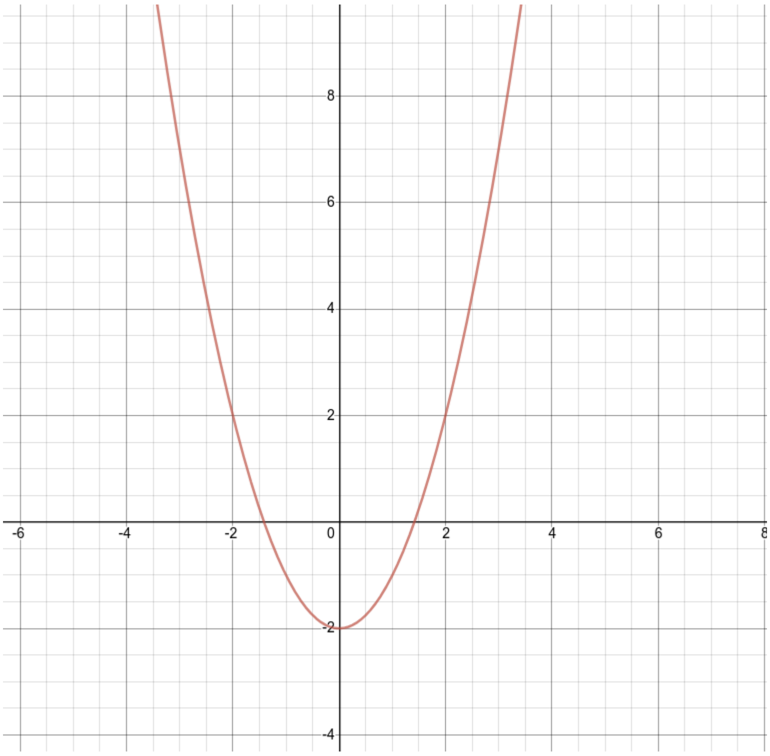

Figure 1: Grafico funzione $f(x) = x^2 - 2$

Come si può vedere dal grafico di questa funzione, abbiamo che le due radici sono negli intervalli $[-2, 0]$ e $[0, 2]$. Questo risultato si può ottenere teoricamente andando ad applicare a questi estremi il teorema dell'esistenza degli zeri. Dopo aver determinato in quali intervalli sono le due radici, possiamo andare ad utilizzare i metodi per la ricerca degli zeri di una funzione. Nei vari metodi, andremo alla ricerca della radice nell'intervallo $[0; 2]$ usando come tolleranza 10^{-8} e numero massimo di iterazioni pari a 100.

4.1.1 Metodo di bisezione

Con il metodo di bisezione, andando a dare come parametri $f(x) = x^2 - 2$ e $[a, b] = [0; 2]$ otteniamo:

```
>> bisection(x^2-2,10^-8,100,0,2,[-1,3],[-3,3])

T =

  29×5 table
```

k	a	b	x	err
0	0	2	1	Inf
1	1	2	1.5	1
2	1	1.5	1.25	0.5
3	1.25	1.5	1.375	0.25
4	1.375	1.5	1.4375	0.125
5	1.375	1.4375	1.40625	0.0625
6	1.40625	1.4375	1.421875	0.03125
7	1.40625	1.421875	1.4140625	0.015625
8	1.4140625	1.421875	1.41796875	0.0078125
9	1.4140625	1.41796875	1.416015625	0.00390625
10	1.4140625	1.416015625	1.4150390625	0.001953125
11	1.4140625	1.4150390625	1.41455078125	0.0009765625
12	1.4140625	1.41455078125	1.414306640625	0.00048828125
13	1.4140625	1.414306640625	1.4141845703125	0.000244140625
14	1.4141845703125	1.414306640625	1.41424560546875	0.0001220703125
15	1.4141845703125	1.41424560546875	1.41421508789062	6.103515625e-05
16	1.4141845703125	1.41421508789062	1.41419982910156	3.0517578125e-05
17	1.41419982910156	1.41421508789062	1.41420745849609	1.52587890625e-05
18	1.41420745849609	1.41421508789062	1.41421127319336	7.62939453125e-06
19	1.41421127319336	1.41421508789062	1.41421318054199	3.814697265625e-06
20	1.41421318054199	1.41421508789062	1.41421413421631	1.9073486328125e-06
21	1.41421318054199	1.41421413421631	1.41421365737915	9.5367431640625e-07
22	1.41421318054199	1.41421365737915	1.41421341896057	4.76837158203125e-07
23	1.41421341896057	1.41421365737915	1.41421353816986	2.38418579101562e-07
24	1.41421353816986	1.41421365737915	1.41421359777451	1.19209289550781e-07
25	1.41421353816986	1.41421359777451	1.41421356797218	5.96046447753906e-08
26	1.41421353816986	1.41421356797218	1.41421355307102	2.98023223876953e-08
27	1.41421355307102	1.41421356797218	1.4142135605216	1.49011611938477e-08
28	1.4142135605216	1.41421356797218	1.41421356424689	7.45058059692383e-09

Figure 2: Tabella Metodo di Bisezione su $f(x) = x^2 - 2$

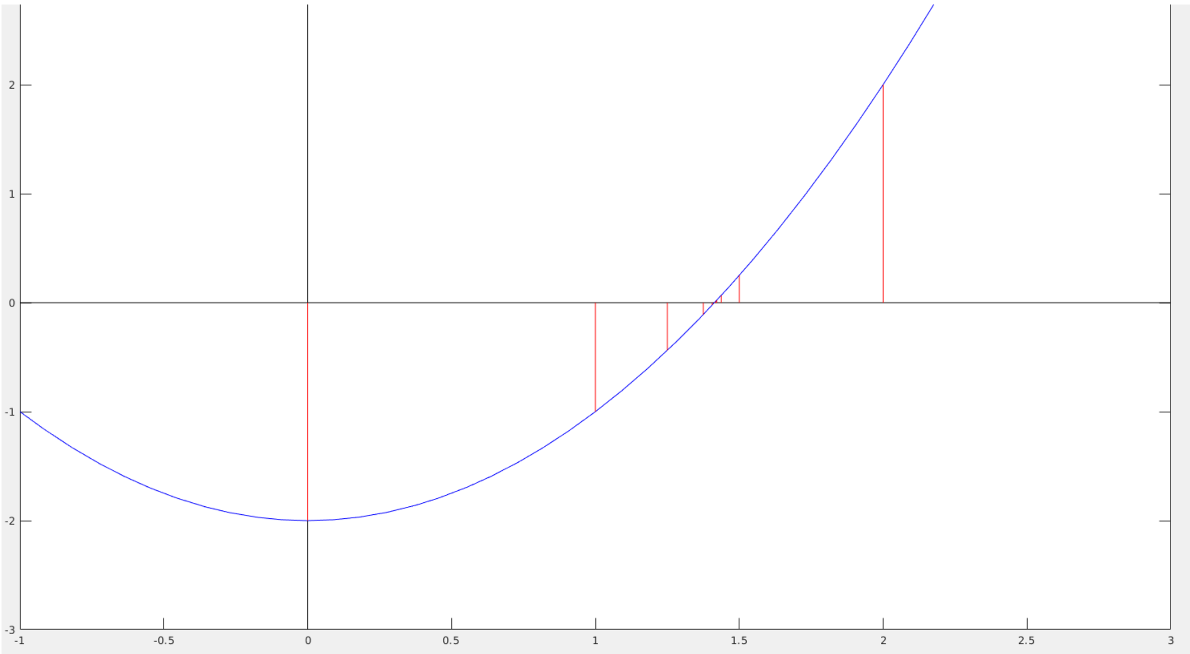

Figure 3: Grafico Metodo di Bisezione su $f(x) = x^2 - 2$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB.

Come si può vedere, l'algoritmo esegue 28 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Nelle varie iterazioni, il metodo di Bisezione si avvicina sempre di più alla radice, restrigendo l'intervallo e spostando gli estremi (Nel grafico segnati con la linea rossa).
Infine si ottiene che la radice della funzione é approssimativamente $\sqrt{2}$.

4.1.2 Metodo di punto fisso

Con il metodo di punto fisso, andando a dare come parametri $g(x) = \frac{2}{x}$ e $x_0 = 1.5$ otteniamo:

```
>> iterative_method(2/x,1.5, 10^-8, 100, [-1,2],[-1,3])

T =

  101×4 table
```

k	x	fx	err
0	1.5	1.33333333333333	Inf
1	1.33333333333333	1.5	0.166666666666667
2	1.5	1.33333333333333	0.166666666666667
3	1.33333333333333	1.5	0.166666666666667
4	1.5	1.33333333333333	0.166666666666667
5	1.33333333333333	1.5	0.166666666666667
6	1.5	1.33333333333333	0.166666666666667
7	1.33333333333333	1.5	0.166666666666667
8	1.5	1.33333333333333	0.166666666666667
9	1.33333333333333	1.5	0.166666666666667
10	1.5	1.33333333333333	0.166666666666667
11	1.33333333333333	1.5	0.166666666666667
12	1.5	1.33333333333333	0.166666666666667
13	1.33333333333333	1.5	0.166666666666667
14	1.5	1.33333333333333	0.166666666666667
15	1.33333333333333	1.5	0.166666666666667
16	1.5	1.33333333333333	0.166666666666667
17	1.33333333333333	1.5	0.166666666666667

Figure 4: Tabella Metodo di Punto Fisso su $g(x) = \frac{2}{x}$

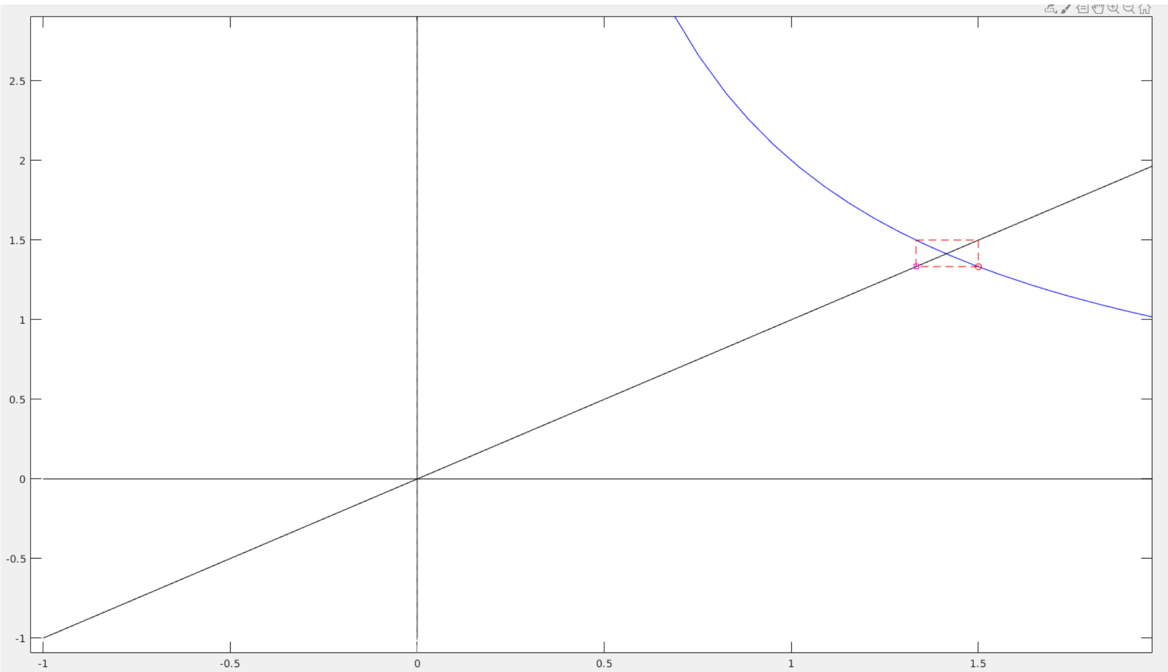

Figure 5: Grafico Metodo di Punto Fisso su $g(x) = \frac{2}{x}$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB. Come si può vedere,il programma esegue sempre le solite operazioni in loop, infatti esso fa più delle 100 iterazioni massime. Questo é giustificato dal fatto che se andiamo a studiare teoricamente la funzione $g(x)$ otteniamo che la sua derivata é: $g'(x) = -\frac{2}{x^2}$. Per il teorema 2 precedentemente esposto nella descrizione del punto fisso, abbiamo che la successione converge localmente se in un intorno di ξ vale $-1 < g'(x) < 1$, ma non è questo il caso perchè la $g'(x)$ è monotona decrescente a sinistra di ξ dove $g'(\sqrt{2}) = 1$, quindi non ho convergenza in ξ perchè non ho intorni che soddisfano la definizione di convergenza locale.

4.1.3 Metodo di Newton

Con il metodo di Newton, andando a dare come parametri $f(x) = x^2 - 2$ e $x_0 = 2$ otteniamo:

```
>> tangentmethod(x^2 - 2, 2, 10^-8, 100, [-1,3],[0,7])

T =

  6×4 table

    k            x                      fx                      err
    _    ________________    ____________________    ____________________

    0                   2                       2                     Inf
    1                 1.5                    0.25                     0.5
    2    1.41666666666667     0.00694444444444444      0.0833333333333333
    3    1.41421568627451    6.00730488273741e-06     0.00245098039215685
    4    1.41421356237469     4.5108350476556e-12    2.12389982001682e-06
    5     1.4142135623731                       0    1.59472435257157e-12


ans =

   1.414213562373095
```

Figure 6: Tabella Metodo di Newton su $f(x) = x^2 - 2$

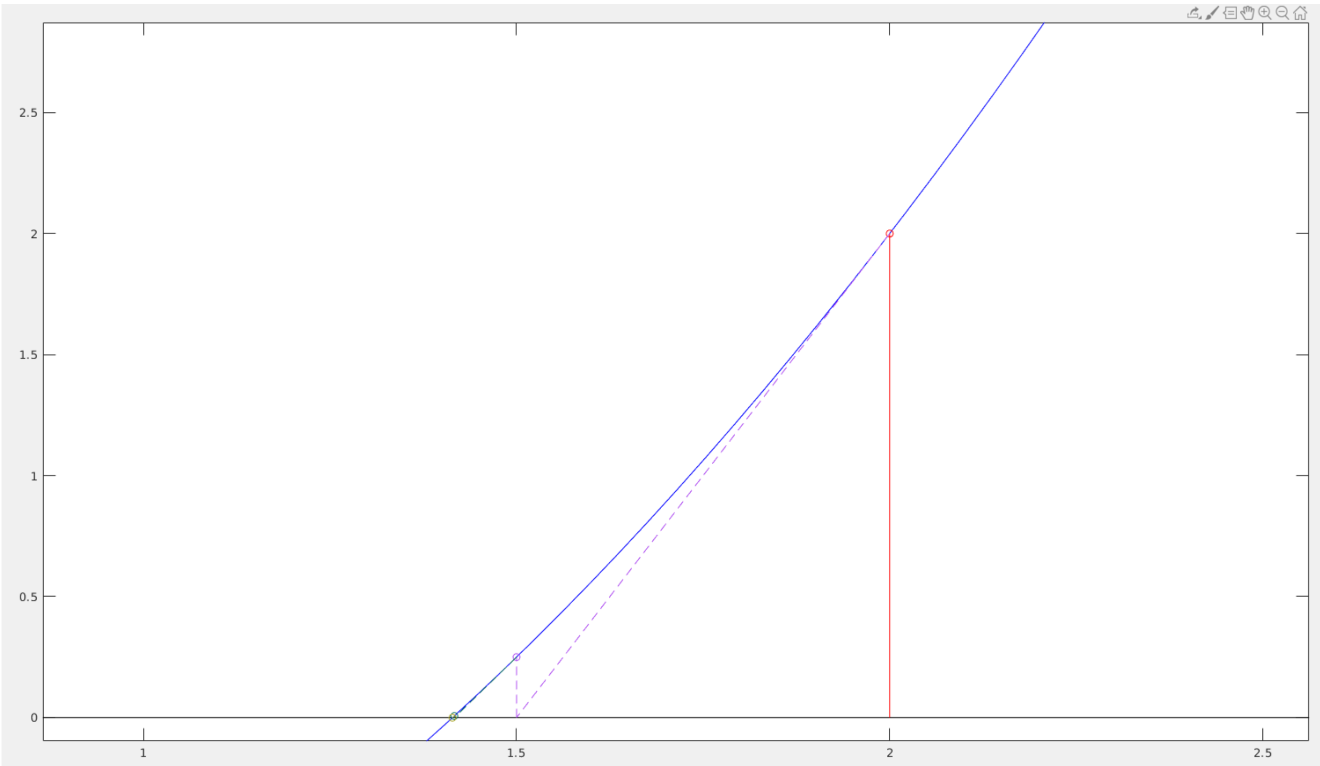

Figure 7: Grafico Metodo di Newton su $f(x) = x^2 - 2$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB.
Come si può vedere,l'algoritmo esegue 5 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Questo é giustificato teoricamente dato che per il teorema 4, precedentemente esposto, che se $f(\xi) = 0$, $\xi \in (a, b)$ e $f \in \mathtt{C}^2[(a, b)]$, se $f'(\xi) \neq 0 \implies$ Il metodo di Newton é localmente convergente ad ξ. Infatti la derivata di $f(x) = x^2 - 2$ é $f'(x) = 2x$, quindi $f'(\xi) = 2 * \sqrt{2} \neq 0$ quindi c'é convergenza locale in ξ.

4.1.4 Metodo delle Corde

Con il metodo delle Corde, andando a dare come parametri $f(x) = x^2 - 2$ e $x_0 = 2$ otteniamo:

```
>> Corda(x^2 - 2, 2, 10^-8, 100, [-1,3],[-2,7])

T =

  16×4 table
```

k	x	fx	err
0	2	2	Inf
1	1.5	0.25	0.5
2	1.4375	0.06640625	0.0625
3	1.4208984375	0.0189523696899414	0.0166015625
4	1.41616034507751	0.00551012297006537	0.00473809242248535
5	1.414782814335	0.00161041173765829	0.00137753074251634
6	1.41438021140058	0.000471382401559769	0.000402602934414631
7	1.41426236580019	0.000138039318761238	0.000117845600389854
8	1.4142278559705	4.04286029271611e-05	3.45098296903323e-05
9	1.41421774881977	1.18410768631701e-05	1.01071507316863e-05
10	1.41421478855056	3.46815509394698e-06	2.96026921575709e-06
11	1.41421392151178	1.01579773439959e-06	8.67038773444406e-07
12	1.41421366756235	2.97520149881621e-07	2.53949433703937e-07
13	1.41421359318231	8.71416242674254e-08	7.43800374625181e-08
14	1.41421357139691	2.55231897334671e-08	2.17854061457246e-08
15	1.41421356501611	7.47556908532888e-09	6.38079744597064e-09

```
ans =

   1.414213565016108
```

Figure 8: Tabella Metodo della Corda su $f(x) = x^2 - 2$

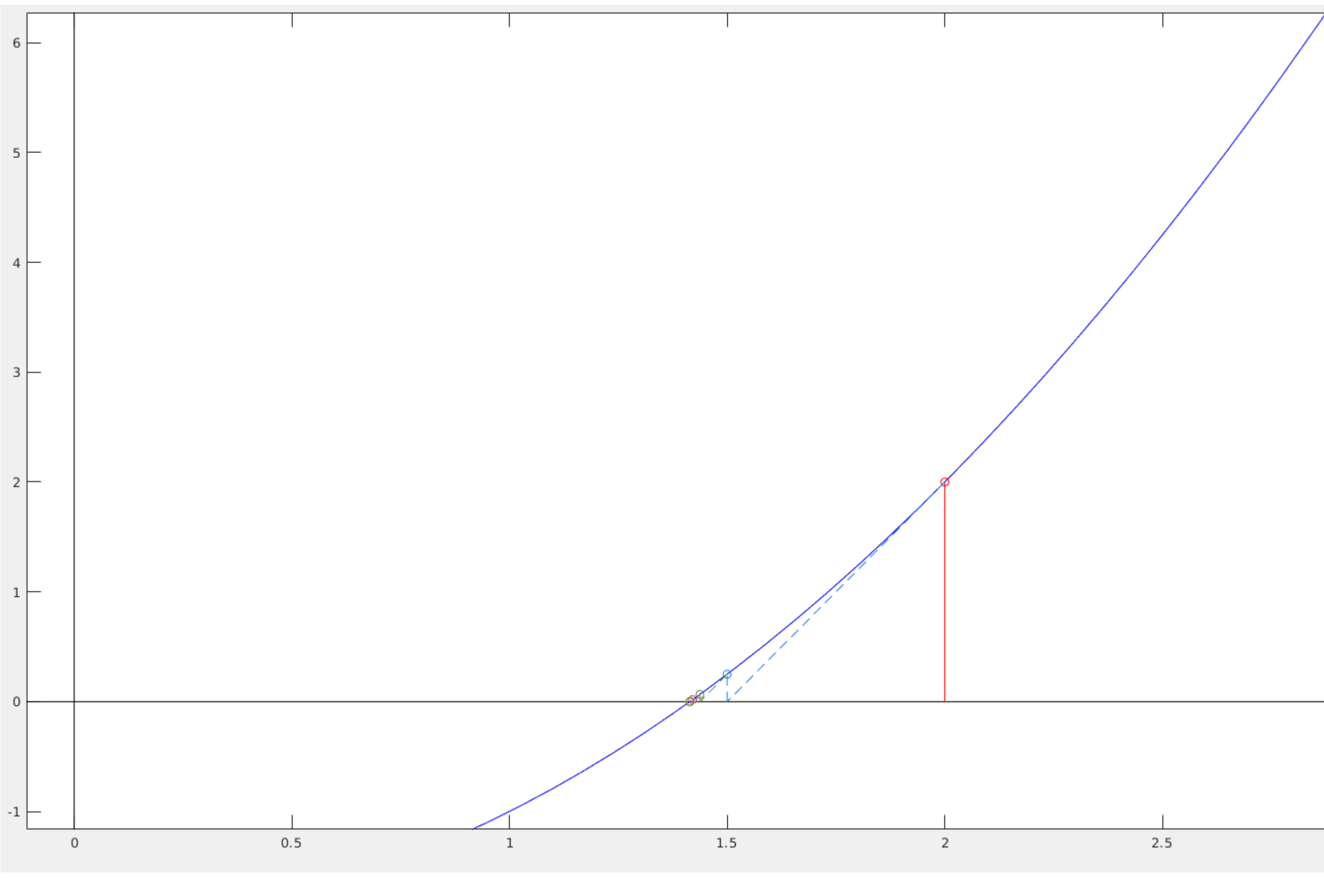

Figure 9: Grafico Metodo della Corda su $f(x) = x^2 - 2$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB. Come si può vedere, l'algoritmo esegue 15 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Il risultato é dovuto al fatto che non va ad aggiornare $f'(x)$ ogni volta, ma lo calcola solo inizialmente. Quindi avendo la sempre la solita derivata, ci mette più iterazioni rispetto al metodo di Newton.

4.1.5 Metodo delle Secanti

Con il metodo delle Secanti, andando a dare come parametri $f(x) = x^2 - 2$, $x_0 = 2$ e x_1=3 otteniamo:

T =

8×4 table

k	x	fx	err
0	3	7	Inf
1	1.6	0.56	1.4
2	1.47826086956522	0.185255198487713	0.121739130434783
3	1.4180790960452	0.0109483226403652	0.0601817735200196
4	1.41429904164193	0.000241779189282457	0.00378005440326734
5	1.4142136790323	3.29962083599627e-07	8.53626096266602e-05
6	1.41421356237662	9.97161661896765e-12	1.16655683113365e-07
7	1.41421356237309	0	3.52584628160457e-12

Figure 10: Tabella Metodo delle Secanti su $f(x) = x^2 - 2$

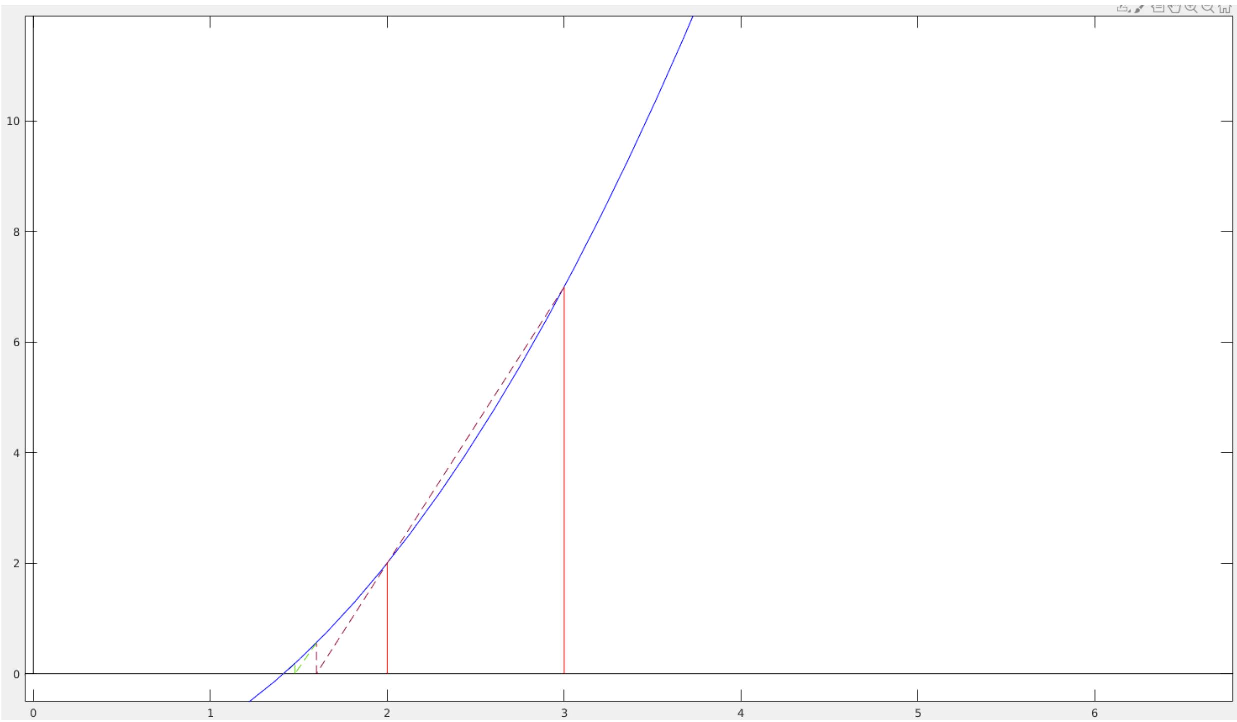

Figure 11: Grafico Metodo delle Secanti su $f(x) = x^2 - 2$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB. Come si può vedere,l'algoritmo esegue 7 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Usando due punti iniziali, riesce ad approssimare, come si é visto precedentemente nell'introduzione dei vari metodi, con velocitá di convergenza superlineare, quindi più veloce rispetto al metodo delle corde.

4.2 Secondo caso di applicazione: $f(x) = e^{-x} - 2x^2$

Come secondo caso di applicazione utilizziamo una funzione più complessa rispetto alla precedente.

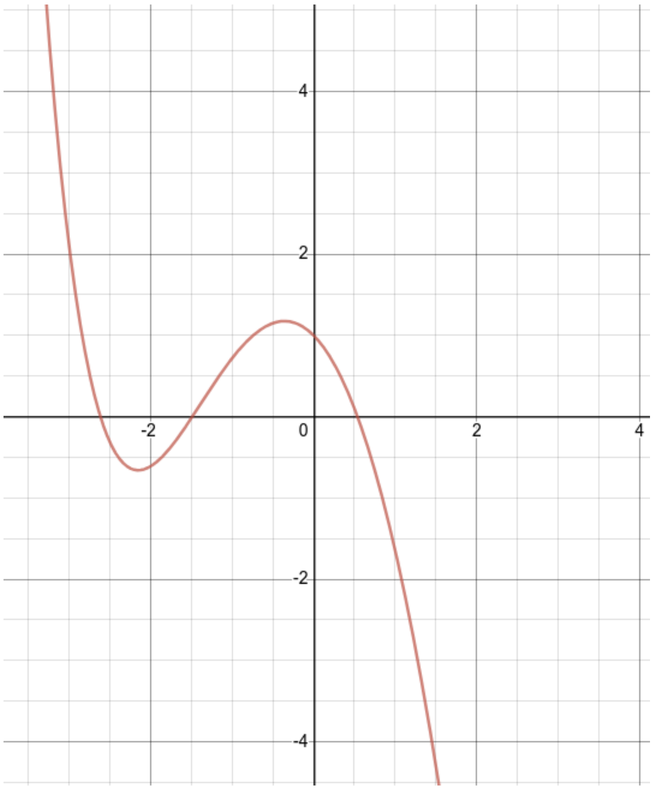

Figure 12: Grafico funzione $f(x) = e^{-x} - 2x^2$

Come si può vedere dal grafico di questa funzione, abbiamo 3 radici negli intervalli $[-3,-2]$,$[-2,-1]$,$[0,1]$. Questo risultato si può ottenere teoricamente andando ad applicare a questi estremi il teorema dell'esistenza degli zeri. Dopo aver determinato in quali intervalli sono le tre radici, possiamo andare ad utilizzare i metodi per la ricerca degli zeri di una funzione. Andremo a studiare come testing la radice nell'intervallo $[-2,-1]$ usando come tolleranza 10^{-8} e numero massimo di iterazioni pari a 100.

4.2.1 Metodo di bisezione

Con il metodo di bisezione, andando a dare come parametri $f(x) = e^{-x} - 2x^2$ e $[a, b] = [-2, 0]$ otteniamo:

```
>> bisection(exp(-x)-2*x^2,10^-8,100,-2,0,[-5,2],[-2,6])

T =

  29×5 table
```

k	a	b	x	err
0	-2	0	-1	Inf
1	-2	-1	-1.5	1
2	-1.5	-1	-1.25	0.5
3	-1.5	-1.25	-1.375	0.25
4	-1.5	-1.375	-1.4375	0.125
5	-1.5	-1.4375	-1.46875	0.0625
6	-1.5	-1.46875	-1.484375	0.03125
7	-1.5	-1.484375	-1.4921875	0.015625
8	-1.4921875	-1.484375	-1.48828125	0.0078125
9	-1.48828125	-1.484375	-1.486328125	0.00390625
10	-1.48828125	-1.486328125	-1.4873046875	0.001953125
11	-1.48828125	-1.4873046875	-1.48779296875	0.0009765625
12	-1.48828125	-1.48779296875	-1.488037109375	0.00048828125
13	-1.488037109375	-1.48779296875	-1.4879150390625	0.000244140625
14	-1.488037109375	-1.4879150390625	-1.48797607421875	0.0001220703125
15	-1.48797607421875	-1.4879150390625	-1.48794555664062	6.103515625e-05
16	-1.48797607421875	-1.48794555664062	-1.48796081542969	3.0517578125e-05
17	-1.48797607421875	-1.48796081542969	-1.48796844482422	1.52587890625e-05
18	-1.48796844482422	-1.48796081542969	-1.48796463012695	7.62939453125e-06
19	-1.48796463012695	-1.48796081542969	-1.48796272277832	3.814697265625e-06
20	-1.48796272277832	-1.48796081542969	-1.487961769104	1.9073486328125e-06
21	-1.48796272277832	-1.487961769104	-1.48796224594116	9.5367431640625e-07
22	-1.48796224594116	-1.487961769104	-1.48796200752258	4.76837158203125e-07
23	-1.48796224594116	-1.48796200752258	-1.48796212673187	2.38418579101562e-07
24	-1.48796212673187	-1.48796200752258	-1.48796206712723	1.19209289550781e-07
25	-1.48796206712723	-1.48796200752258	-1.48796203732491	5.96046447753906e-08
26	-1.48796206712723	-1.48796203732491	-1.48796205222607	2.98023223876953e-08
27	-1.48796206712723	-1.48796205222607	-1.48796205967665	1.49011611938477e-08
28	-1.48796206712723	-1.48796205967665	-1.48796206340194	7.45058059692383e-09

Figure 13: Tabella Metodo di bisezione su $f(x) = e^{-x} - 2x^2$

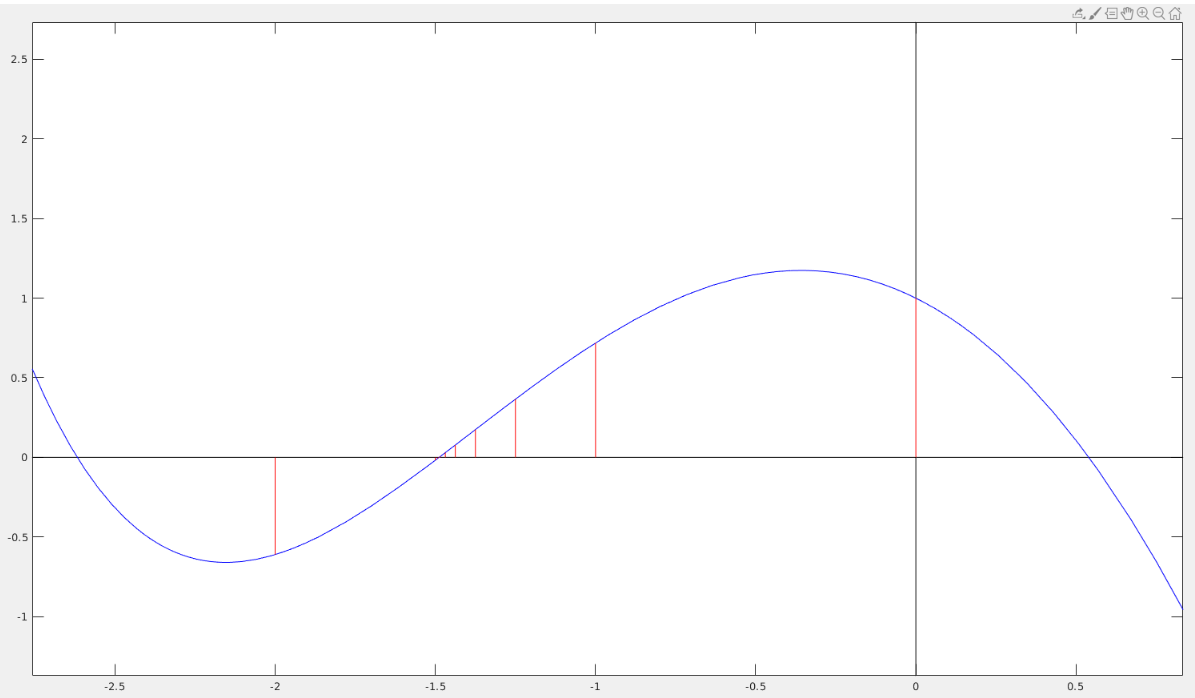

Figure 14: Grafico Metodo di bisezione su $f(x) = e^{-x} - 2x^2$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB.
Come si può vedere, l'algoritmo esegue 28 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Nelle varie iterazioni, il metodo di Bisezione si avvicina sempre di più alla radice, restrigendo l'intervallo e spostando gli estremi (Nel grafico riportati con la linea rossa).
Infine si ottiene che la radice della funzione é approssimativamente -1.487.

4.2.2 Metodo di punto fisso

Con il metodo di punto fisso, andando a dare come parametri $g(x) = \frac{e^{-x}}{2x}$ e $x_0 = -0.3$ otteniamo:

```
>> iterative_method(exp(-x)/(2*x),-0.3, 10^-8, 100, [-5,2],[-2,6])

T =

  29×4 table
```

k	x	fx	err
0	-0.3	-2.24976467929334	Inf
1	-2.24976467929334	-2.10811013383662	0.141654545456723
2	-2.10811013383662	-1.9526180821357	0.155492051700916
3	-1.9526180821357	-1.80452937418923	0.14808870794647
4	-1.80452937418923	-1.68384921220112	0.120680161988104
5	-1.68384921220112	-1.59938576918226	0.084463443018864
6	-1.59938576918226	-1.54746626760258	0.0519195015796801
7	-1.54746626760258	-1.51846530904143	0.0290009585611457
8	-1.51846530904143	-1.50323276961462	0.0152325394268196
9	-1.50323276961462	-1.49551050021756	0.00772226939705578
10	-1.49551050021756	-1.49166910752314	0.00384139269442074
11	-1.49166910752314	-1.48977667709156	0.00189243043157905
12	-1.48977667709156	-1.48884889687811	0.0009277802134533
13	-1.48884889687811	-1.48839513275194	0.000453764126165268
14	-1.48839513275194	-1.48817346391464	0.000221668837305122
15	-1.48817346391464	-1.4880652384984	0.00010822541623412
16	-1.4880652384984	-1.48801241443706	5.28240613459907e-05
17	-1.48801241443706	-1.48798663492506	2.57795120008897e-05
18	-1.48798663492506	-1.48797405469761	1.25802274448805e-05
19	-1.48797405469761	-1.4879679158325	6.13886511180439e-06
20	-1.4879679158325	-1.48796492025364	2.99557886229707e-06
21	-1.48796492025364	-1.48796345851394	1.46173969817198e-06
22	-1.48796345851394	-1.48796274523783	7.13276105512861e-07
23	-1.48796274523783	-1.48796239718554	3.48052296761381e-07
24	-1.48796239718554	-1.48796222734907	1.69836464225526e-07
25	-1.48796222734907	-1.48796214447528	8.28737867220042e-08
26	-1.48796214447528	-1.48796210403601	4.04392723751812e-08
27	-1.48796210403601	-1.48796208430318	1.97328329232249e-08
28	-1.48796208430318	-1.4879620746743	9.62887436450899e-09

Figure 15: Tabella Metodo di Punto Fisso su $g(x) = \frac{e^{-x}}{2x}$

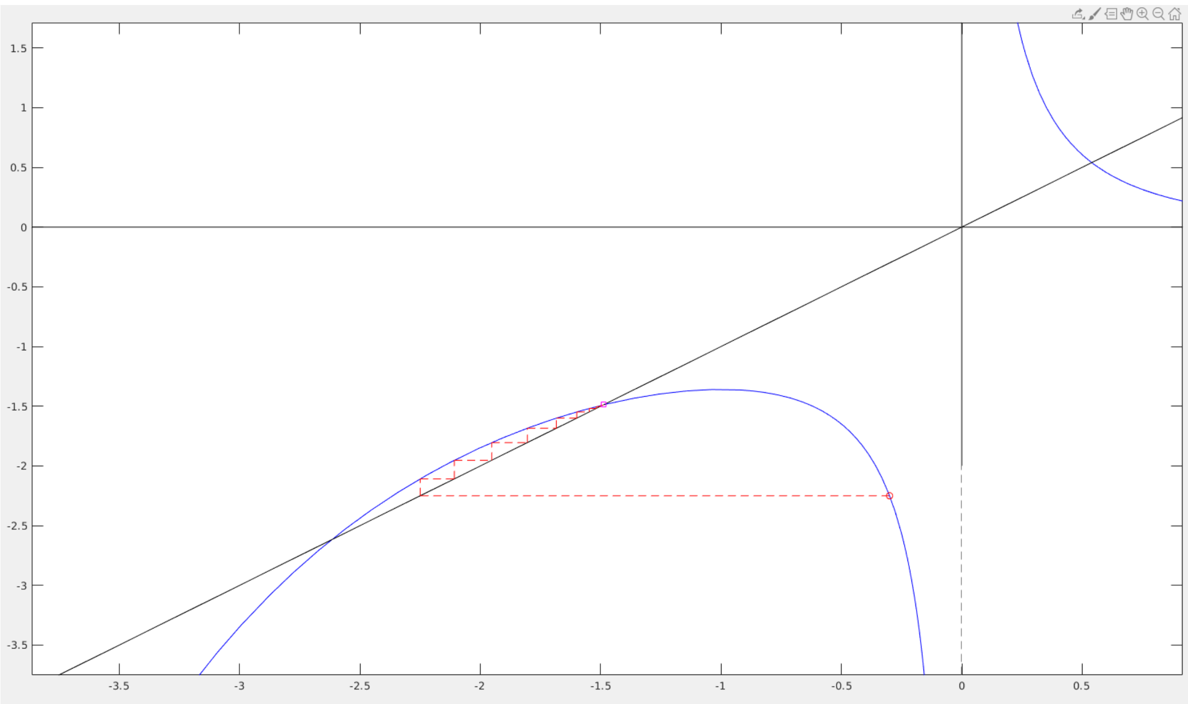

Figure 16: Grafico Metodo di Punto Fisso su $g(x) = \frac{e^{-x}}{2x}$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB. Come si può vedere, l'algoritmo esegue 28 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Questo é giustificato dal fatto che se andiamo a studiare teoricamente la funzione $g(x)$ otteniamo che la sua derivata é: $g'(x) = -\frac{2}{x^2}$. Studiando il grafico della derivata, scopriremo che é monotona decrescente nell'intervallo $[-2, -1]$, andando a valutare la derivata negli estremi dell'intervallo, otterremo che $f'(-2) = 0.924$ e $f'(-1) = 0$. Per il teorema del Punto Fisso esposto nella descrizione del omonimo metodo, otteniamo quindi la convergenza locale perché $-1 < g'(x) < 1$ in tutti i punti compresi nell'intervallo $[-2, -1]$.

4.2.3 Metodo di Newton

Con il metodo di Newton, andando a dare come parametri $f(x) = e^{-x} - 2x^2$ e $x_0 = -1$ otteniamo:

```
>> tangentmethod(exp(-x)-2*x^2,-1, 10^-8, 100, [-5,2],[-2,6])

T =

  6×4 table
```

k	x	fx	err
0	-1	0.718281828459045	Inf
1	-1.5604054342114	-0.108979212983355	0.560405434211396
2	-1.48681412598308	0.00174949514057712	0.0735913082283199
3	-1.48796188192829	2.79721232423241e-07	0.00114775594521865
4	-1.48796206549817	7.17417379359241e-15	1.83569877743039e-07
5	-1.48796206549818	6.88559149374006e-17	4.66293670342566e-15

Figure 17: Tabella Metodo di Newton su $f(x) = e^{-x} - 2x^2$

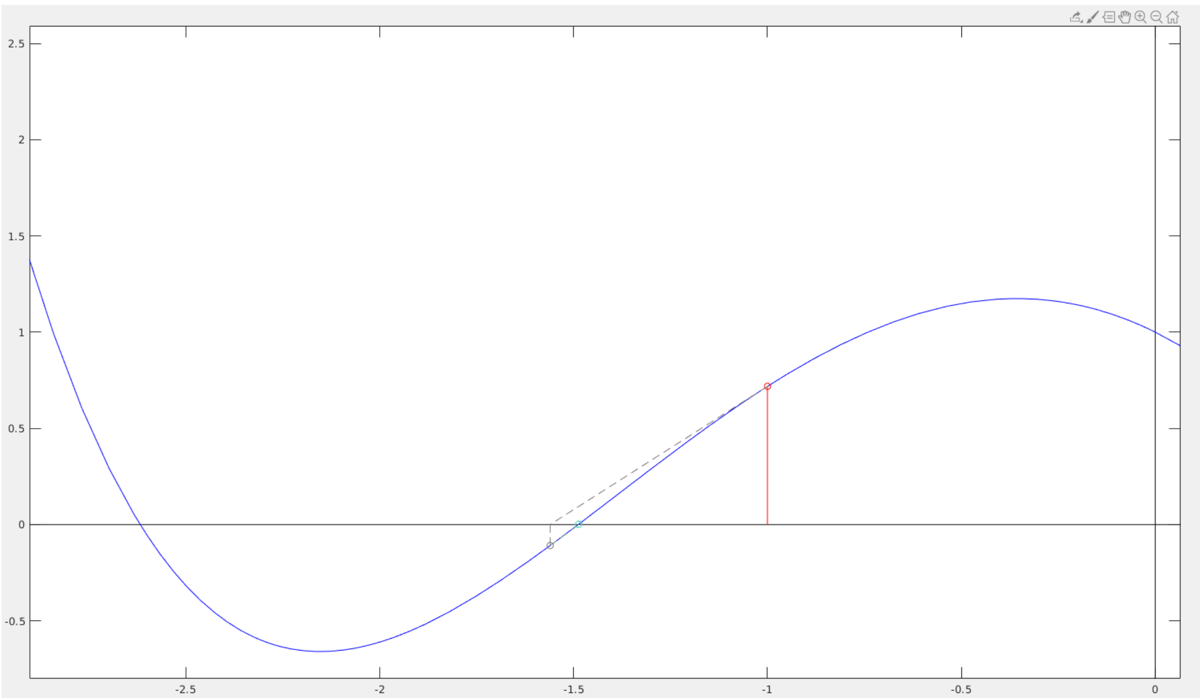

Figure 18: Grafico Metodo di Newton su $f(x) = e^{-x} - 2x^2$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB. Come si può vedere,l'algoritmo esegue 5 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Questo é giustificato teoricamente perché sappiamo che, per il teorema 4, se $f(\xi) = 0$, $\xi \in (a, b)$ e $f \in \mathtt{C}^2[(a, b)]$, se $f'(\xi) \neq 0 \implies$ Il metodo di Newton é localmente convergente a ξ. Infatti la derivata di $f(x) = e^{-x} - 2x^2$ é $f'(x) = -e^{-x} - 4x$, andando a studiare il grafico della derivata, scopriremo che $f'(x) > 0$, $\forall\, x \in [-2, -1]$ quindi anche $f'(\xi) \neq 0$.

4.2.4 Metodo delle Corde

Con il metodo delle Corde, andando a dare come parametri $f(x) = e^{-x} - 2x^2$ e $x_0 = -1$ otteniamo:

```
>> Corda(exp(-x)-2*x^2,-1, 10^-8, 100, [-5,2],[-2,6])

T =

  13x4 table
```

k	x	fx	err
0	-1	0.718281828459045	Inf
1	-1.5604054342114	-0.108979212983355	0.560405434211396
2	-1.47537955613374	0.019205471945177	0.085025878077654
3	-1.49036371752867	-0.00365835910319286	0.0149841613949244
4	-1.48750945581621	0.000689724094374658	0.00285426171245939
5	-1.48804758042869	-0.000130304892150944	0.000538124612484481
6	-1.48794591619778	2.46081343975927e-05	0.000101664230908405
7	-1.4879651155311	-4.64759561875535e-06	1.91993333198415e-05
8	-1.48796148946437	8.77752343210171e-07	3.62606672976717e-06
9	-1.48796217428914	-1.65774142772052e-07	6.84824763119707e-07
10	-1.4879620449517	3.13084357134924e-08	1.29337436538535e-07
11	-1.48796206937863	-5.91297441335914e-09	2.44269267035691e-08
12	-1.48796206476531	1.11673611298967e-09	4.61331861600911e-09

```
ans =

  -1.487962064765308
```

Figure 19: Tabella Metodo delle Corde su $f(x) = e^{-x} - 2x^2$

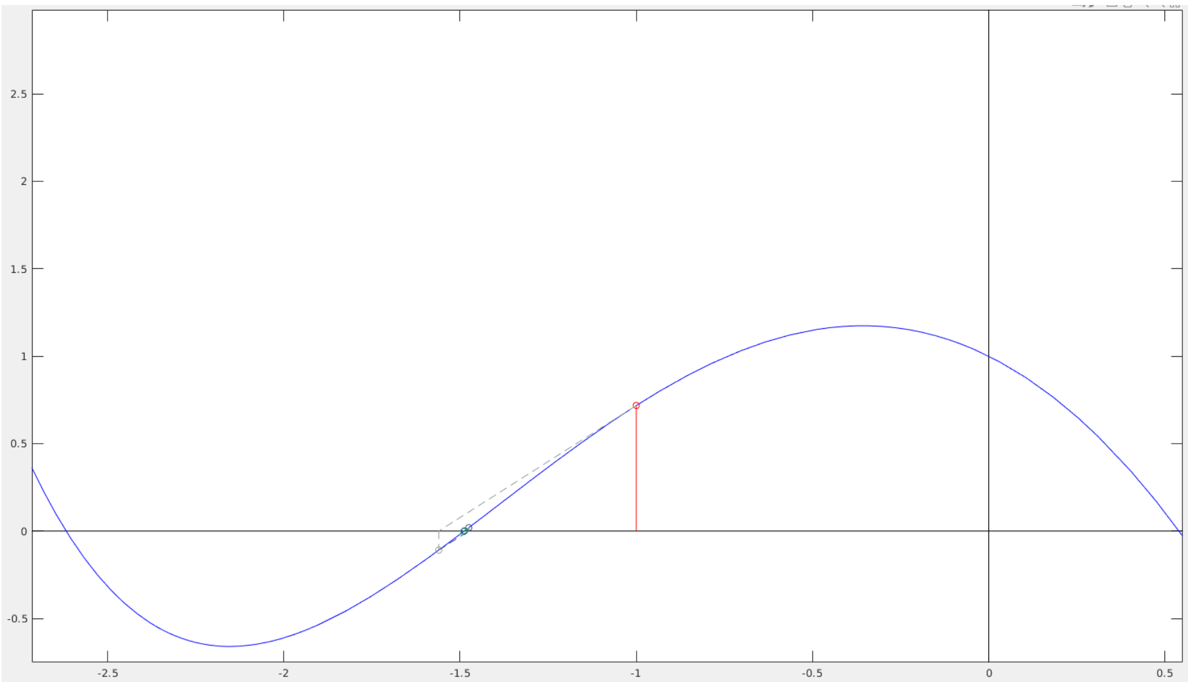

Figure 20: Grafico Metodo delle Corde su $f(x) = e^{-x} - 2x^2$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB. Come si può vedere,l'algoritmo esegue 12 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Il risultato é dovuto al fatto che non va ad aggiornare $f'(x)$ ad ogni iterazione, ma lo calcola solo inizialmente. Quindi avendo la medesima derivata, ci impiega più iterazioni rispetto al metodo di Newton.

4.2.5 Metodo delle Secanti

Con il metodo delle Secanti, andando a dare come parametri $f(x) = e^{-x} - 2x^2$ e $x_0 = -1$, $x_1 = -0.9$ otteniamo:

```
>> SecantMethod(exp(-x)-2*x^2,-1,-0.9, 10^-8, 100, [-5,2],[-2,6])

T =

  7×4 table
```

k	x	fx	err
0	-0.9	0.83960311115695	Inf
1	-1.592049319366	-0.155433490417011	0.692049319366001
2	-1.48394511387283	0.00612438062814386	0.108104205493167
3	-1.48804315810714	-0.000123566378046606	0.00409804423430526
4	-1.48796211060744	-6.87368639395951e-08	8.10474996999488e-05
5	-1.48796206549766	7.82668867978281e-13	4.51097759146535e-08
6	-1.48796206549818	6.88559149374006e-17	5.13589171191597e-13

Figure 21: Tabella Metodo delle Secanti su $f(x) = e^{-x} - 2x^2$

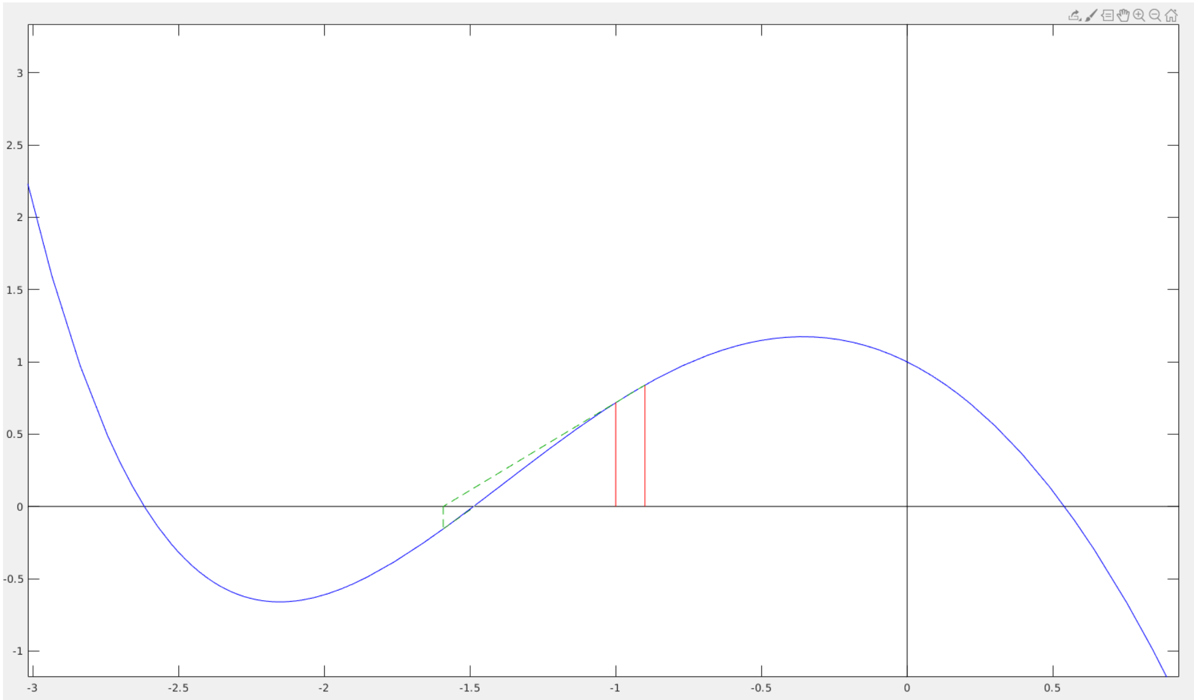

Figure 22: Grafico Metodo delle Secanti su $f(x) = e^{-x} - 2x^2$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB. Come si può vedere,l'algoritmo esegue 6 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Usando due punti iniziali, riesce ad approssimare, come prima riportato, con velocitá di convergenza superlineare, quindi più veloce rispetto al metodo delle corde.

4.3 Terzo caso di applicazione: $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$

Come terzo caso di applicazione utilizziamo una funzione la quale ha una radice tripla in 1.

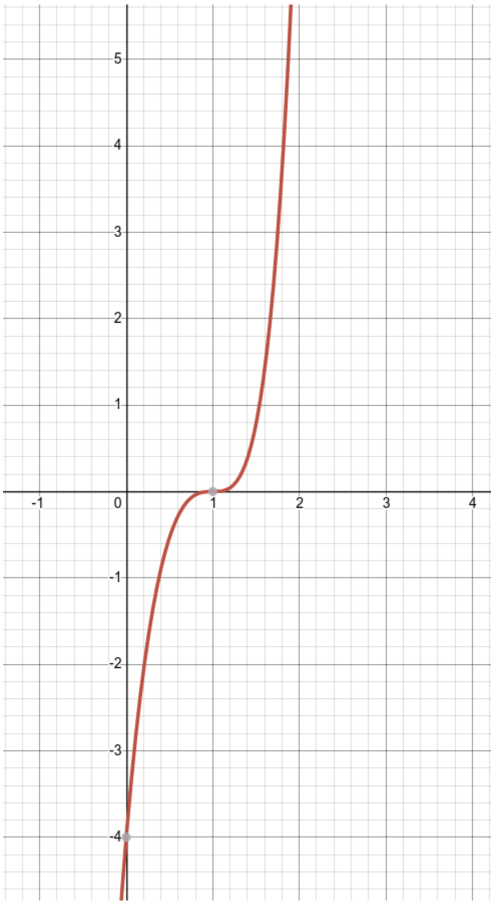

Figure 23: Grafico funzione $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$

Andremo a studiare come testing la radice tripla in 1 usando come tolleranza 10^{-8} e numero massimo di iterazioni pari a 100.

4.3.1 Metodo di bisezione

Con il metodo di bisezione, andando a dare come parametri $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$ e $[a, b] = [-2.5, 2.5]$ otteniamo:

```
>> bisection(x^5-3*x^4+7*x^3-13*x^2+12*x-4,10^-8,100,-2.5,2.5,[-3,3],[-3,3])

T =

  30×5 table
```

k	a	b	x	err
0	-2.5	2.5	0	Inf
1	0	2.5	1.25	2.5
2	0	1.25	0.625	1.25
3	0.625	1.25	0.9375	0.625
4	0.9375	1.25	1.09375	0.3125
5	0.9375	1.09375	1.015625	0.15625
6	0.9375	1.015625	0.9765625	0.078125
7	0.9765625	1.015625	0.99609375	0.0390625
8	0.99609375	1.015625	1.005859375	0.01953125
9	0.99609375	1.005859375	1.0009765625	0.009765625
10	0.99609375	1.0009765625	0.99853515625	0.0048828125
11	0.99853515625	1.0009765625	0.999755859375	0.00244140625
12	0.999755859375	1.0009765625	1.0003662109375	0.001220703125
13	0.999755859375	1.0003662109375	1.00006103515625	0.0006103515625
14	0.999755859375	1.00006103515625	0.999908447265625	0.00030517578125
15	0.999908447265625	1.00006103515625	0.999984741210938	0.000152587890625
16	0.999984741210938	1.00006103515625	1.00002288818359	7.62939453125e-05
17	0.999984741210938	1.00002288818359	1.00000381469727	3.814697265625e-05
18	0.999984741210938	1.00000381469727	0.999994277954102	1.9073486328125e-05
19	0.999994277954102	1.00000381469727	0.999999046325684	9.5367431640625e-06
20	0.999999046325684	1.00000381469727	1.00000143051147	4.76837158203125e-06
21	0.999999046325684	1.00000143051147	1.00000023841858	2.38418579101562e-06
22	0.999999046325684	1.00000023841858	0.999999642372131	1.19209289550781e-06
23	0.999999642372131	1.00000023841858	0.999999940395355	5.96046447753906e-07
24	0.999999940395355	1.00000023841858	1.00000008940697	2.98023223876953e-07
25	0.999999940395355	1.00000008940697	1.00000001490116	1.49011611938477e-07
26	0.999999940395355	1.00000001490116	0.999999977648258	7.45058059692383e-08
27	0.999999977648258	1.00000001490116	0.99999999627471	3.72529029846191e-08
28	0.99999999627471	1.00000001490116	1.00000000558794	1.86264514923096e-08
29	0.99999999627471	1.00000000558794	1.00000000093132	9.31322574615479e-09

Figure 24: Tabella Metodo di bisezione su $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$

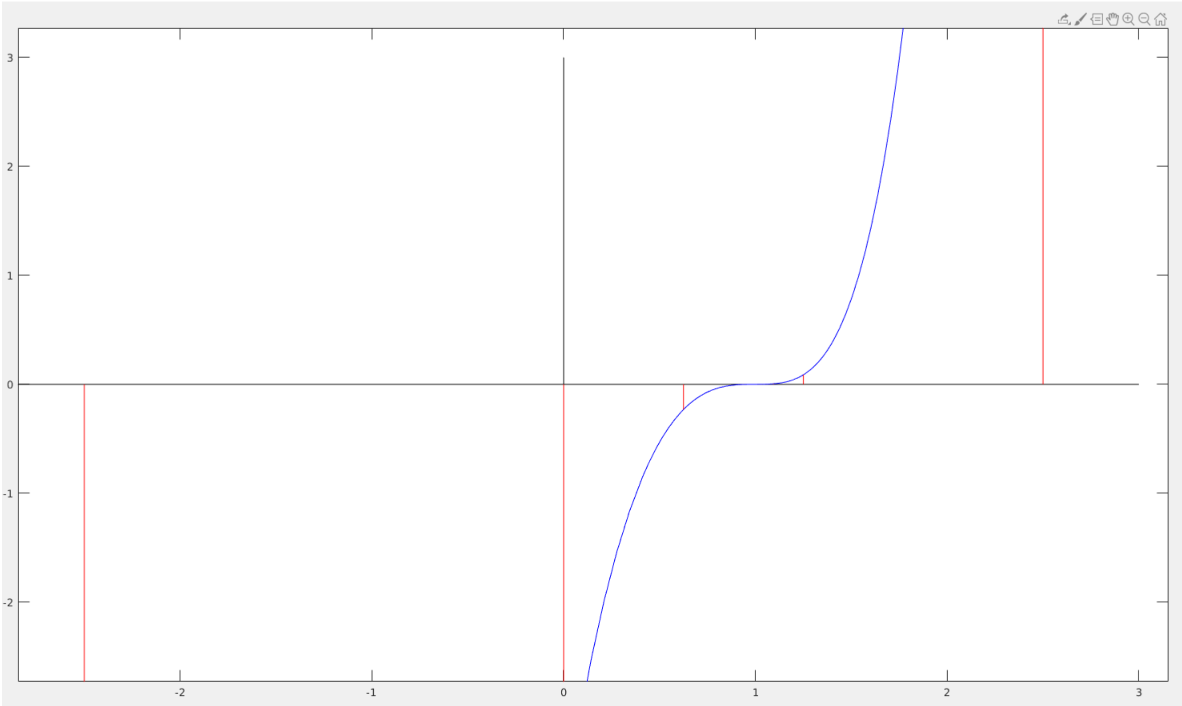

Figure 25: Grafico Metodo di bisezione su $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB.

Come si può vedere, l'algoritmo esegue 29 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Nelle varie iterazioni, il metodo di Bisezione si avvicina sempre di più alla radice, restrigendo l'intervallo e spostando gli estremi (Nel grafico evidenziati con la linea rossa).
Infine si ottiene che la radice della funzione é approssimativamente 1.

4.3.2 Metodo di punto fisso

Con il metodo di punto fisso, andando a dare come parametri $g(x) = \frac{3x^4-7x^3+13x^2-12x+4}{x^4}$ e $x_0 = -1.5$ otteniamo:

```
>> iterative_method((3*x^4-7*x^3+13*x^2-12*x+4)/x^4,1.5, 10^-8, 100, [0.5,2],[0.5,2])

T =

  101×4 table
```

k	x	fx	err
0	1.5	1.34567901234568	Inf
1	1.34567901234568	1.27248207066115	0.0731969416845288
2	1.27248207066115	1.2291227552868	0.0433593153743483
3	1.2291227552868	1.20008027044996	0.0290424848368458
4	1.20008027044996	1.17907221314482	0.0210080573051394
5	1.17907221314482	1.16305714563641	0.0160150675084092
6	1.16305714563641	1.15037512805846	0.0126820175779447
7	1.15037512805846	1.14003904064645	0.0103360874120084
8	1.14003904064645	1.13142277915609	0.0086162614903682
9	1.13142277915609	1.12410876401328	0.00731401514280572
10	1.12410876401328	1.11780705148908	0.00630171252419665
11	1.11780705148908	1.11230956959375	0.00549748189533106
12	1.11230956959375	1.10746283258437	0.00484673700938143
13	1.10746283258437	1.1031509492649	0.0043118833194733
14	1.1031509492649	1.09928464592444	0.00386630334045446
15	1.09928464592444	1.09579394866527	0.00349069725917772
16	1.09579394866527	1.09262317083128	0.00317077783398823
17	1.09262317083128	1.08972739690189	0.00289577392939044
18	1.08972739690189	1.0870699639113	0.00265743299058818
81	1.03908525675781	1.03882507785112	0.0002601789066905
82	1.03882507785112	1.03856983182544	0.000255246025681055
83	1.03856983182544	1.03831936698719	0.000250464838251974
84	1.03831936698719	1.03807353804253	0.000245828944658522
85	1.03807353804253	1.03783220575614	0.000241332286388118
86	1.03783220575614	1.03759523663188	0.000236969124264297
87	1.03759523663188	1.03736250261369	0.000232734018185354
88	1.03736250261369	1.03713388080533	0.000228621808362917
89	1.03713388080533	1.0369092532074	0.000224627597930738
90	1.0369092532074	1.03668850647059	0.000220746736811162
91	1.03668850647059	1.03647153166385	0.000216974806736658
92	1.03647153166385	1.03625822405652	0.000213307607329405
93	1.03625822405652	1.03604848291337	0.000209741143156972
94	1.03604848291337	1.03584221130168	0.000206271611683961
95	1.03584221130168	1.03563931590963	0.000202895392049873
96	1.03563931590963	1.03543970687502	0.000199609034609249
97	1.03543970687502	1.03524329762385	0.000196409251172147
98	1.03524329762385	1.03505000471795	0.000193292905896758
99	1.03505000471795	1.03485974771117	0.000190257006778438
100	1.03485974771117	1.03467244901348	0.00018729869769496

Figure 26: Tabella Metodo di Punto Fisso, iterazioni: 0-10, 81-100, su $g(x) = \frac{3x^4-7x^3+13x^2-12x+4}{x^4}$

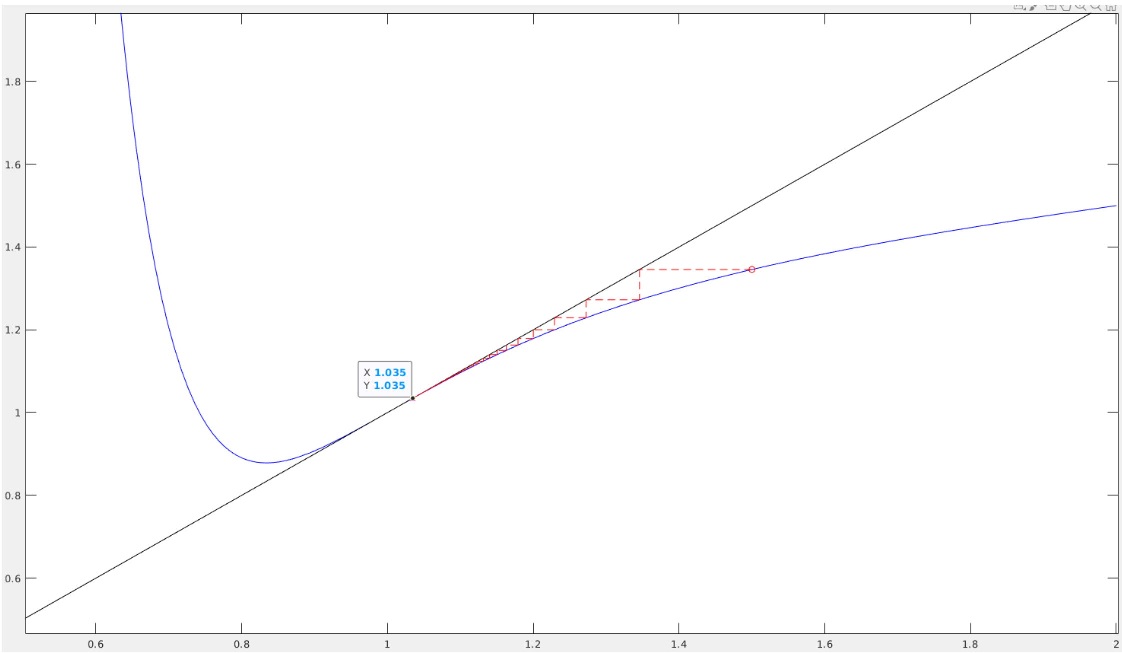

Figure 27: Grafico Metodo di Punto Fisso su $g(x) = \frac{3x^4-7x^3+13x^2-12x+4}{x^4}$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB. Come si può vedere, l'algoritmo esegue più di 100 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Studiando teoricamente la convergenza locale ottengo che $g'(x) = \frac{7x^3-26x^2+36x-16}{x^5}$ la funzione ha un intorno di ξ dove $-1 < g'(x) < 1$ però essendo che $g'(\xi) = 1$ non posso dire nulla sulla sua convergenza per il teorema 3.

4.3.3 Metodo di Newton

Con il metodo di Newton, andando a dare come parametri $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$ e $x_0 = 1.5$ otteniamo:

```
>> tangentmethod(x^5-3*x^4+7*x^3-13*x^2+12*x-4,1.5, 10^-8, 100, [-1,3],[0,7

T =

  44×4 table
```

k	x	fx	err
0	1.5	0.78125	Inf
1	1.34567901234568	0.240026311853401	0.154320987654321
2	1.23629055173695	0.0729356156199997	0.109388460608729
3	1.16020721839813	0.0219827366732772	0.0760833333388251
4	1.10801457488386	0.00658805826674029	0.0521926435142683
5	1.07255098038385	0.00196683494319815	0.0354635945000039
6	1.04860847813095	0.000585693574267146	0.0239425022529074
7	1.03251290428685	0.000174116363683725	0.0160955738440987
8	1.02172293543511	5.17039271238671e-05	0.0107899688517352
9	1.0145031365443	1.53421473304415e-05	0.00721979889081492
10	1.00967816831512	4.55025405147976e-06	0.00482496822917722
11	1.00645629376083	1.34909828599641e-06	0.00322187455429357
12	1.00430605398765	3.99905538035276e-07	0.00215023977317785
13	1.00287152839893	1.18524632182364e-07	0.00143452558871826
14	1.00191471922975	3.5125143966769e-08	0.00095680916918206
15	1.00127664257173	1.04087800754823e-08	0.000638076658016296
16	1.00085116752719	3.08434567075915e-09	0.000425475044546042
17	1.00056747723029	9.13932082868532e-10	0.000283690296900563
18	1.00037833246977	2.70804939360106e-10	0.000189144760511573
19	1.00025222800921	8.02405247988502e-11	0.000126104460559384
20	1.00016815483399	2.37753701521333e-11	8.40731752289603e-05
21	1.00011210447947	7.0446330988223e-12	5.60503545166124e-05
22	1.00007473687823	2.08731429669208e-12	3.73676012419555e-05
23	1.00004982483374	6.18466576922072e-13	2.49120444846263e-05
24	1.00003321666616	1.83249964835019e-13	1.66081675774254e-05
25	1.00002214449315	5.4296406115563e-14	1.10721730164354e-05
26	1.00001476301723	1.60878477851135e-14	7.38147592116434e-06
27	1.00000984202117	4.76677440555445e-15	4.92099605575724e-06
28	1.00000656135175	1.41237852836191e-15	3.28066941857053e-06
29	1.00000437423642	4.18482709965289e-16	2.18711533750415e-06
30	1.00000291615846	1.23994913189736e-16	1.45807795459696e-06
31	1.00000194410602	3.67392406740114e-17	9.7205244231624e-07
32	1.00000129607085	1.08857023540268e-17	6.48035171302297e-07
33	1.00000086404731	3.2253935690312e-18	4.32023540941628e-07
34	1.00000057603157	9.55672223539183e-19	2.88015735483427e-07
35	1.00000038402106	2.83162150977809e-19	1.92010508826002e-07
36	1.00000025601405	8.38998988391876e-20	1.28007013877607e-07
37	1.00000017067603	2.48592296660736e-20	8.5338013100511e-08
38	1.00000011378402	7.365697769946e-21	5.68920102139714e-08
39	1.00000007585602	2.18242898048143e-21	3.79280076234778e-08
40	1.00000005057068	6.46645625823919e-22	2.52853387117113e-08
41	1.00000003371379	1.91598705179726e-22	1.68568925484891e-08
42	1.00000002247586	5.67699875864179e-23	1.12379283656594e-08
43	1.0000000149839	1.68207370122343e-23	7.49195239180267e-09

Figure 28: Tabella Metodo di Newton su $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB. Come si può vedere,l'algoritmo esegue 43 iterazioni andando a passare come parametro

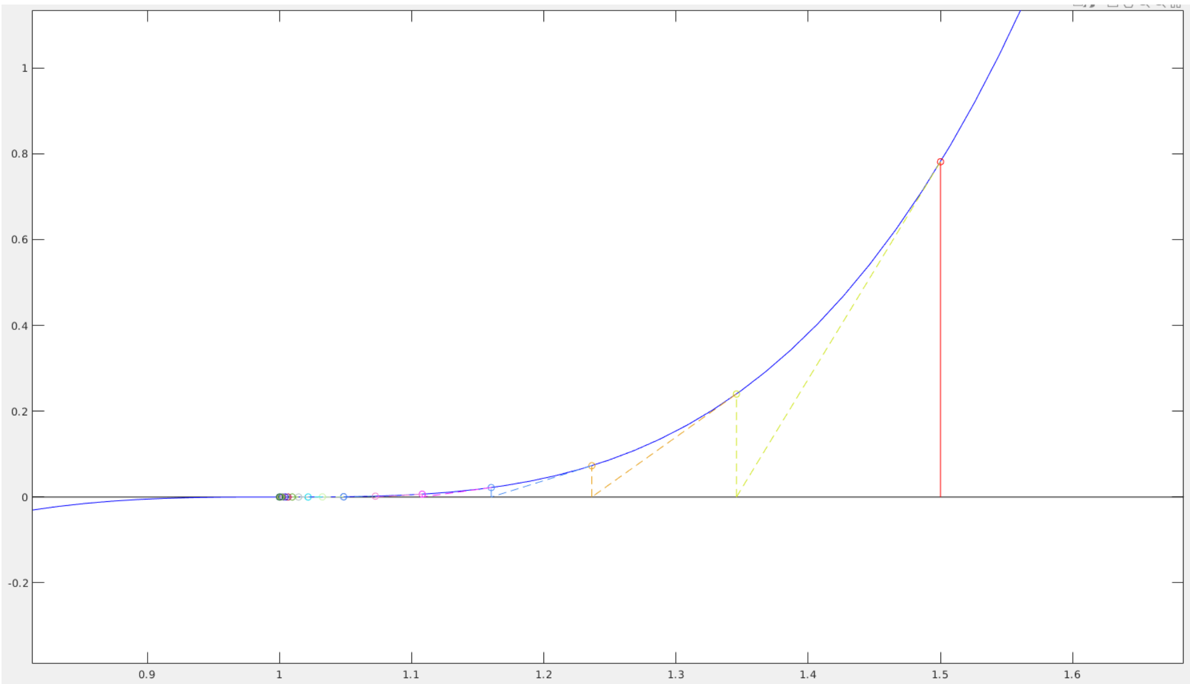

Figure 29: Grafico Metodo di Newton su $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$

una tolleranza pari a 10^{-8}. Questo é giustificato teoricamente perché sappiamo che secondo il teorema 5, sia $f : [a, b] \to \mathbb{R}$, $f(\xi) = 0$, $\xi \in (a, b)$ e $f \in C^2[(a, b)]$, se $\exists \delta > 0 : \forall x \in (\xi, \xi + \delta) = S \subset [a, b]$ si ha che: se $f'(x) \neq 0$ e $f(x) * f''(x) > 0 \, \forall \, x \in S \implies$ Il metodo di Newton é convergente a ξ scegliendo $x_0 \in S$. Andando a studiare la funzione per un intervallo $(\xi, \xi + \delta) = S$ o $(\xi - \delta, \xi) = S$ con $\delta \neq 0$ sappiamo che $\forall x \in S$, $f'(x) \neq 0$ e $f(x) * f''(x) > 0$, quindi c'è convergenza. Di seguito il grafico che mostra le varie funzioni:

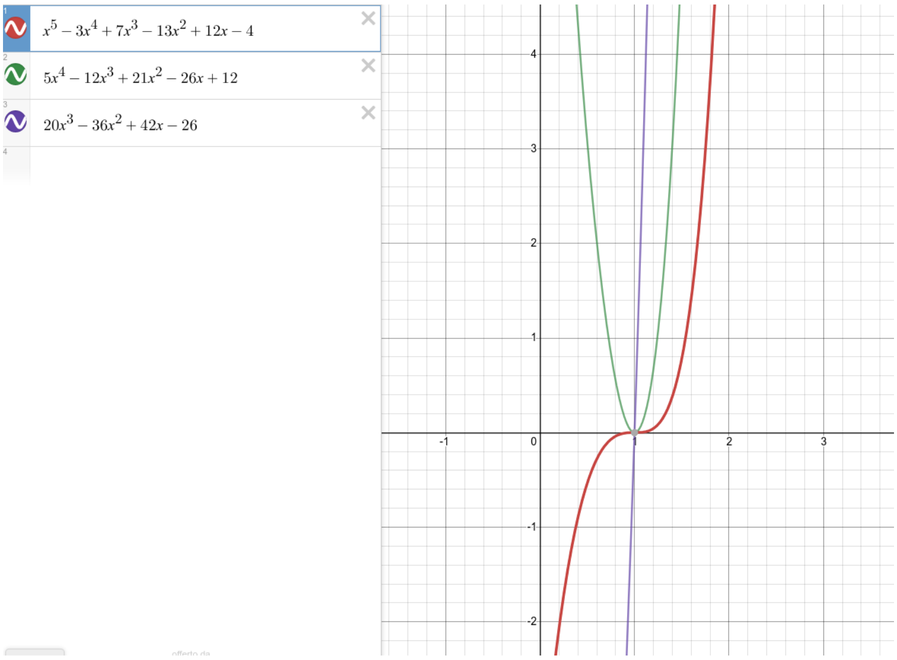

Figure 30: Grafico delle funzione $f(x), f'(x), f''(x)$

4.3.4 Metodo delle Corde

Con il metodo delle Corde, andando a dare come parametri $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$ e $x_0 = 1.5$ otteniamo:

```
>> Corda(x^5-3*x^4+7*x^3-13*x^2+12*x-4,1.5, 10^-8, 100, [-1,3],[0,7])

T =

  101×4 table
```

k	x	fx	err
0	1.5	0.78125	Inf
1	1.34567901234568	0.240026311853401	0.154320987654321
2	1.29826640753513	0.150862521749035	0.0474126048105483
3	1.26846640323902	0.108531502774453	0.0298000042961057
4	1.24702808170333	0.0837392795254774	0.0214383215356944
5	1.23048698945138	0.0675170944679828	0.0165410922519462
6	1.21715027943302	0.0561276701571474	0.013336710018367
7	1.20606333224148	0.0477270035480003	0.0110869471915354
8	1.19663577598509	0.0412992674032236	0.00942755625639502
9	1.1884778960042	0.0362391127696621	0.00815787998088369
10	1.18131955274106	0.0321637659297538	0.00715834326314324
11	1.17496621626111	0.0288196642756301	0.00635333647995129
12	1.16927344307086	0.0260324051024314	0.00569277319024786
13	1.16413123959384	0.0236782482020234	0.00514220347702343
14	1.15945405476381	0.0216670741852773	0.00467718483002932
15	1.15517413887536	0.0199317683907382	0.00427991588844989
16	1.15123699944015	0.0184213747841022	0.00393713943520746
17	1.14759820935934	0.0170965479100606	0.00363879008081036
18	1.14422111347587	0.0159264524951548	0.00337709588346868
19	1.1410751475509	0.0148866014910184	0.00314596592496885
20	1.13813458429342	0.0139573179776555	0.00294056325748504
21	1.13537758321141	0.0131226211647874	0.00275700108200594
22	1.13278546051219	0.0123694065023777	0.00259212269921738
23	1.13034212095617	0.011686833444574	0.00244333955602527
24	1.12803361064613	0.0110658622222828	0.00230851031003931
25	1.12584776131827	0.0104988991291567	0.00218584932785837
26	1.12377390470017	0.00997952190126745	0.00207385661810511
27	1.12180264111473	0.00950226494757904	0.00197126358543565
28	1.1199256505078	0.00906244981510062	0.00187699060692914
29	1.11813553696408	0.00865605020226803	0.00179011354372349
30	1.11642569988709	0.00827958361622944	0.00170983707699124
31	1.11479022658018	0.0079300237644101	0.0016354733069095
32	1.11322380213289	0.00760472921771478	0.00156642444729083
33	1.11172163339852	0.00730138494389029	0.00150216873436348
80	1.07501862410684	0.00217666692376922	0.00043760698926909
81	1.07458866520832	0.00213907168126664	0.000429958898522376
82	1.07416613253053	0.0021025543479269 1	0.00042253267778114
83	1.07375081315317	0.00206707194681415	0.000415319377368384
84	1.07334250264515	0.00203258368653023	0.000408310508012599
85	1.072941004633	0.00199905082654962	0.000401498012154056
86	1.07254613039566	0.0019664365522716	0.00039487423734319
87	1.0721576984841	0.00193470585898828	0.000388431911559906
88	1.0717755343638	0.0019038254440411	0.000382164120293904
89	1.07139947007856	0.00187376360650514	0.000376064285242617
90	1.07102934393407	0.00184449015380033	0.000370126144494831
91	1.07066500019998	0.00181597631468277	0.000364343734084116
92	1.07030628882918	0.00178819465811734	0.000358711370801457
93	1.06995306519301	0.00176111901757714	0.000353223636171407
94	1.06960518983151	0.00173472442035448	0.000347875361496719
95	1.06926252821761	0.00170898702150368	0.000342661613897288
96	1.06892495053435	0.00168388404206876	0.000337577683259926
97	1.06859233146432	0.00165939371127757	0.000332619070038342
98	1.06826454999048	0.0016354952124114	0.0003277814738325
99	1.06794148920779	0.00161216863208219	0.000323060782698503
100	1.06762303614466	0.00158939491267211	0.000318453063127278

Figure 31: Tabella Metodo delle Corde, iterazioni:0-33, 80-100, su $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$

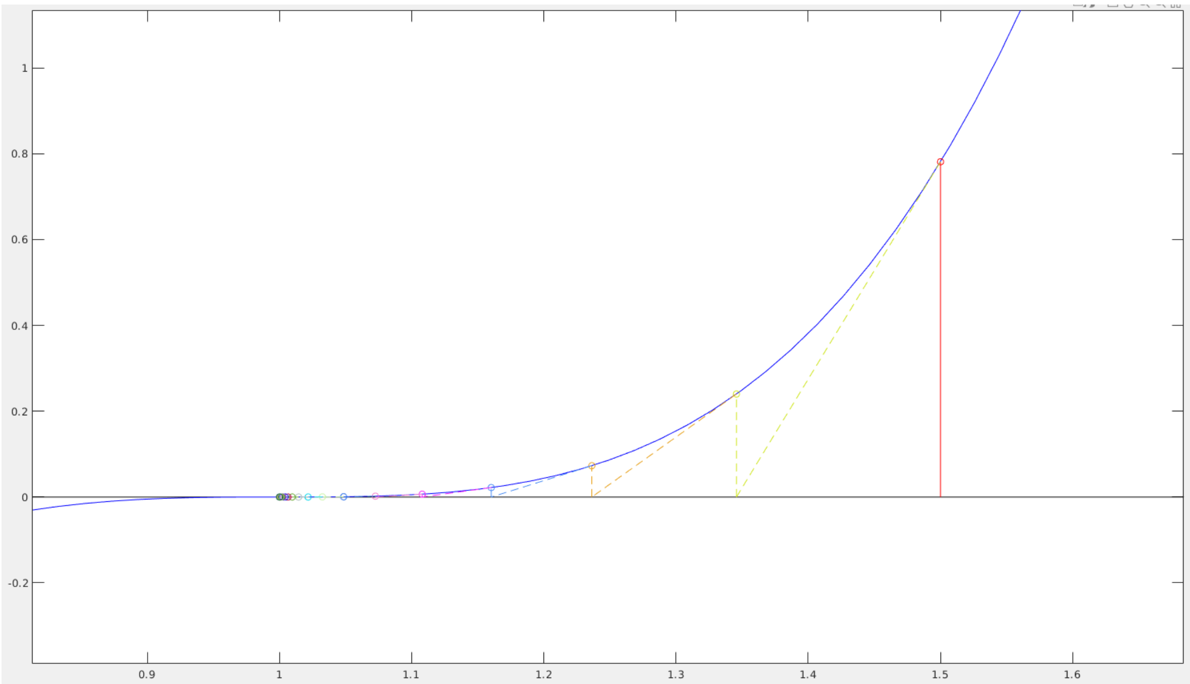

Figure 32: Grafico Metodo delle corde su $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$

Questi sono i risultati ottenuti dall'esecuzione del programma in MATLAB.
Come si può vedere,l'algoritmo esegue più di 100 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Il risultato é dovuto al fatto che non va ad aggiornare $f'(x)$ ogni volta, ma lo calcola solo inizialmente. Quindi avendo la sempre la solita derivata, ci mette più iterazioni rispetto al metodo di Newton.

4.3.5 Metodo delle Secanti

Con il metodo delle Secanti, andando a dare come parametri $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$, $x_0 = 1.5$ e $x_1 = 1.6$ otteniamo: Questi sono i risultati ottenuti

```
>> SecantMethod(x^5-3*x^4+7*x^3-13*x^2+12*x-4,1.5,1.6, 10^-8, 100, [-1,3],[0,7])

T =

  61×4 table
```

k	x	fx	err
0	1.6	1.41696	Inf
1	1.37710591307357	0.316212103082433	0.222894086926429
2	1.3130750771401	0.17565386916768	0.0640308359334734
3	1.23305654130265	0.0698806080213136	0.0800185358374437
4	1.18019115853301	0.031551416940473	0.0528653827696408
5	1.1366739935095	0.0135107596250583	0.0435171650235113
6	1.10408372000022	0.00588485525919821	0.0325902735092853
7	1.07893404158064	0.00253973001594915	0.0251496784195764
8	1.05983957170912	0.0010977711646328	0.0190944698715161
9	1.04530284730138	0.000473501120280757	0.014536724407749
10	1.03427692133463	0.000204169162296588	0.0110259259667442
11	1.02591863335045	8.79717829023738e-05	0.00835828798417859
12	1.01959066365085	3.78913873345744e-05	0.00632796969960148
13	1.01480285101058	1.63150708549206e-05	0.00478781264026629
14	1.01118251560534	7.02324188978748e-06	0.00362033540524354
15	1.00844608003721	3.02277985241376e-06	0.00273643556813186
16	1.00637840829692	1.30081971091596e-06	0.00206767174028921
17	1.00481642780117	5.59735785277461e-07	0.0015619804957463
18	1.00363667426837	2.40832813003519e-07	0.00117975353280175
19	1.00274573443194	1.03615080620284e-07	0.00089093983643207
20	1.00207297296463	4.45770428352721e-08	0.000672761467314764
21	1.00156500017502	1.9177198854537e-08	0.000507972789604016
22	1.00118147438619	8.24989179614807e-09	0.000383525788832495
23	1.00089192031381	3.54897674792757e-09	0.000289554072379739
24	1.00067332017754	1.52669359449658e-09	0.00021860013627184
25	1.00050829114724	6.5674375894916e-10	0.000165029030299912
26	1.00038370719852	2.82511746056659e-10	0.000124583948713175
27	1.00028965744418	1.21527457016435e-10	9.40497543477914e-05
28	1.00021865904087	5.2276958013937e-11	7.09984033042677e-05
29	1.00016506259605	2.24876821359184e-11	5.35964448207249e-05
30	1.00012460307566	9.67337308884722e-12	4.04595203884472e-05
31	1.00009406065357	4.16112077333029e-12	3.05424220905515e-05
32	1.00007100461409	1.78995475537991e-12	2.30560394833823e-05
33	1.00005359998396	7.6996909704135e-13	1.74046301275599e-05
34	1.00004046153713	3.31210553077327e-13	1.31384468340379e-05
36	1.00002305669435	6.12865447393782e-14	7.48687695462102e-06
37	1.0000174050033	2.63630320502911e-14	5.65169105448682e-06
38	1.00001313865948	1.13403238715015e-14	4.2663438164503e-06
39	1.000009918087	4.87815355864004e-15	3.22057248602903e-06
40	1.000007486946	2.09838613551712e-15	2.43114099029995e-06
41	1.0000056517304	9.0264150081444e-16	1.83521560281896e-06
42	1.00000426636624	3.8828011290144e-16	1.38536416383062e-06
43	1.00000322058526	1.67022495310322e-16	1.04578097537278e-06
44	1.00000243114827	7.18463593295876e-17	7.89436991777137e-07
45	1.00000183521975	3.09054127051715e-17	5.95928519198097e-07
46	1.00000138536653	1.32942645354334e-17	4.49853223782881e-07
47	1.00000104578232	5.71865735297817e-18	3.39584205422128e-07
48	1.00000078943776	2.45993610841645e-18	2.5634456313206e-07
49	1.00000059592896	1.05816544120916e-18	1.93508802981412e-07
50	1.00000044985347	4.55180151472969e-19	1.46075483486996e-07
51	1.00000033958435	1.95800165582894e-19	1.10269125830342e-07
52	1.00000025634464	8.4225343530014e-20	8.32397035743782e-08
53	1.00000019350885	3.62303497726026e-20	6.28357947896063e-08
54	1.00000014607551	1.55848368672534e-20	4.74333392563864e-08
55	1.00000011026914	6.70396891164528e-21	3.58063692029731e-08
56	1.00000008323971	2.88377734565865e-21	2.70294284732131e-08
57	1.0000000628358	1.24048482152047e-21	2.04039123374855e-08
58	1.00000004743334	5.33606590130478e-22	1.5402457753666e-08
59	1.00000003580637	2.2953605607408e-22	1.16269713856809e-08
60	1.00000002702943	9.87371620271433e-23	8.7769411738492e-09

Figure 33: Tabella Metodo delle Secanti, su $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$

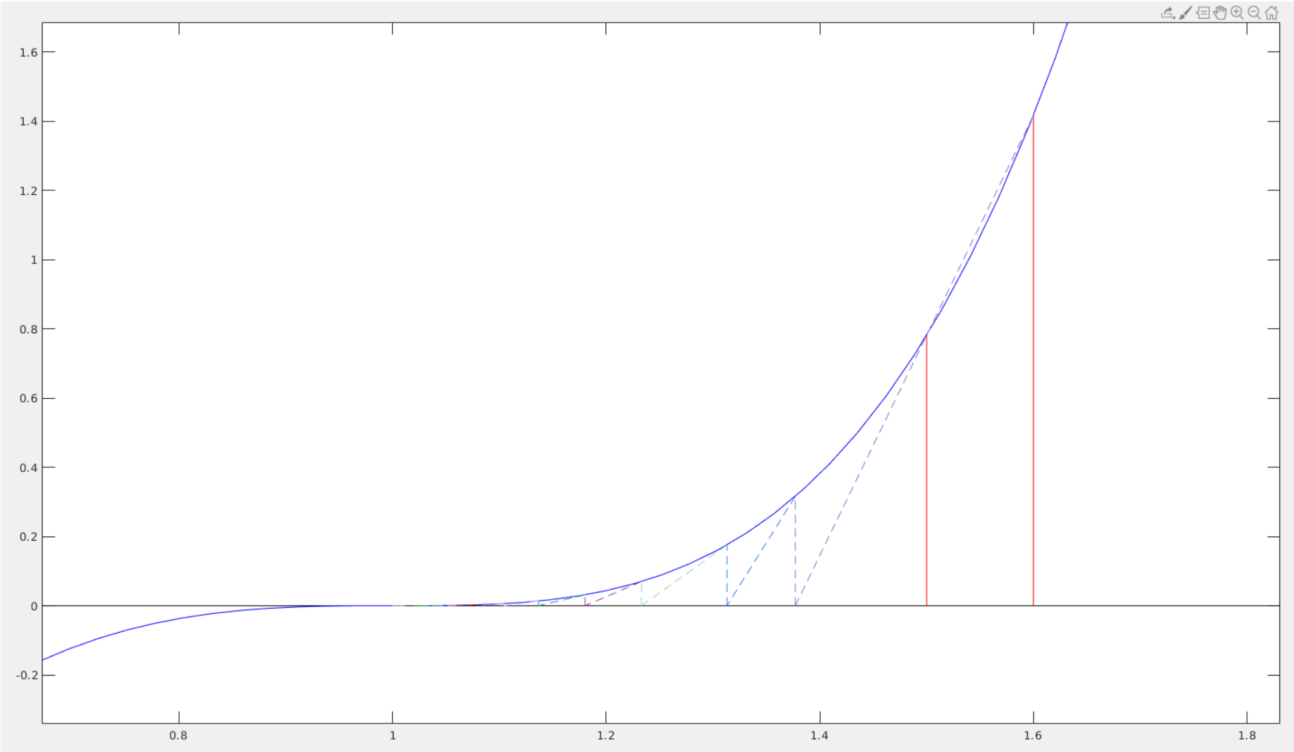

Figure 34: Grafico Metodo delle corde su $f(x) = x^5 - 3x^4 + 7x^3 - 13x^2 + 12x - 4$

dall'esecuzione del programma in MATLAB.
Come si può vedere,l'algoritmo esegue 60 iterazioni andando a passare come parametro una tolleranza pari a 10^{-8}. Usando due punti iniziali, riesce ad approssimare, come si é visto prima, con velocitá di convergenza superlineare, quindi più veloce rispetto al metodo delle corde.

4.4 Conclusioni

Come ci aspettavamo dalla descrizione data dei vari metodi per la ricerca degli zeri, il metodo di Newton é il più veloce tranne nel caso delle radici multiple. Il metodo di punto fisso, può convergere sia linearmente che quadraticamente. Nel caso $e^{-x} - 2x^2$ convergeva linearmente, infatti é stato più lento rispetto ai 2 metodi Quasi-Newton. Infine il metodo di Bisezione é il metodo più lento tra tutti quelli descritti, questo perché compie un numero di iterazioni giá conosciuto pari a: $k \geq \lceil log_2(\frac{b-a}{\epsilon}) \rceil$. Questo numero può essere significativamente elevato richidendo molte valutazioni della funzione f.

5 Codice MATLAB

5.1 Metodo di Bisezione

```
function [xk,k,err]=bisection(f,tol, itr, a, b, x_interval,y_interval)

% Plot the function f and the iterations of bisection method starting from interval [a,b] until
% |x_k - x_(k-1)| < tol or k>=nmax or f(c)~=0

% f = function
% tol = tolerance
% itr = Max number of iterations
% [a,b] = Initial interval of bisection
% x_interval        width of plot
% y_interval        height of plot


syms x
format long

nmax=itr; %Number of iterations
err = inf; %Initial error
k=0; %Iteration counter

c = (a+b)/2; %Interval's middle point
%Calculating function in a,b,c
fa = double(subs(f,x,a));
fb = double(subs(f,x,b));
fc = double(subs(f,x,c));

if (fa*fb>=0) %There aren't zeroes in the interval
    disp("The function doesn't pass from zero")
    return
end

plot(x_interval, [0,0], 'k') %x-axis
hold on
plot([0,0], y_interval, 'k') %y-axis
fplot(f,x_interval,'b') %plot f
plot([a,a], [0,fa],'r') %Plotting red line from 0 to f(a)
plot([b,b], [0,fb],'r') %Plotting red line from 0 to f(b)

%Results container
VarNames = {'k', 'a', 'b', 'x', 'err'};
datasave=[];
datasave=[datasave; k, a, b, c, err];
while ((tol<err) && k<nmax && fc ~=0)
    %Checking if update a or b
    if (fa*fc <= 0)
        b=c;
        fb = fc;
        plot([b,b], [0,fb],'r') %Plotting red line from 0 to f(b)
    else
        a=c;
        fa = fc;
        plot([a,a], [0,fa],'r') %Plotting red line from 0 to f(a)
    end
    %Result approximation
    c = (a+b)/2;
    fc = double(subs(f,x,c));
    err = abs(b-a);
    k=k+1;
    datasave=[datasave; k, a, b, c, err]; %Saving iteration results
end
axis([x_interval y_interval]) % rescaling
hold off
T = table(datasave(:,1),datasave(:,2),datasave(:,3),datasave(:,4),datasave(:,5), 'VariableNames',VarNames)
display(T) %Displaying iteration results
end
```

5.2 Metodo di Punto Fisso

```
function i=iterative_method(g_fun, x_0, tol, itr, x_interval, y_interval)
% % % % % % % % % % % % % % % % % % %
% Plot g_fun in a cartesian plane of x_interval width
% and show the iterative method starting from x_0 until
% |x_k - x_(k-1)| < tol or i==itr
%
% % % % % % % % % % % % % % % % % % %
% Parameters
%
% g_fun              function to plot
% x_0                starting point
% tol                tollerance
% itr                number of max iterations
% x_interval         width of plot
% y_interval         height of plot
% % % % % % % % % % % % % % % % % % %

syms x
format long

    plot(x_interval, [0,0], 'k') %x-axis
    hold on
    plot([0,0], y_interval, 'k') %y-axis
    fplot(g_fun, x_interval, 'b') %plot g_func
    fplot(@(x) x, x_interval, 'k') %plot y=x function
    plot(x_0, double(subs(g_fun,x,x_0)), 'or') %plot starting point x_0

    i = 0; %Iteration counter
    err = inf; %Initial Error

    succ = double(subs(g_fun,x,x_0)); % f(x_0)
    prec = x_0;

    %Saving iteration datas
    VarNames = {'k', 'x', 'fx', 'err'};
    datasave=[];
    datasave=[datasave; i, prec, succ, err];

    %iterative method
    while ( err>=tol  &&  i<itr )

        plot([prec, succ], [succ, succ], '--r')
        prec = succ;
        succ = double(subs(g_fun,x,prec)); % f(prec)
        err = abs(succ - prec);
        plot([prec, prec], [prec, succ], '--r')

        i = i+1;
        datasave=[datasave; i, prec, succ, err]; %Saving iteration datas
    end
    plot([prec, succ], [succ, succ], '--r')
    plot(succ, succ, 'sm') % plot last found point
    axis([x_interval y_interval]) % rescaling
    hold off

    T = table(datasave(:,1),datasave(:,2),datasave(:,3),datasave(:,4), 'VariableNames',VarNames);
    display(T) %Printing results
end
```

5.3 Metodo di Newton

```
function [xk,k,err]=tangentmethod(f,x0,tol, itr, x_interval,y_interval)

% Plot the function f and the iterations of tangent iterate method starting from x0 until
% |x_k - x_(k-1)| < tol or k>=nmax

% f = function (es: 4*log(1/x) - 2*x + 1/x)
% x_0 = starting point
% tol = tolerance
% x_interval        width of plot
% y_interval        height of plot


syms x
format long

nmax=itr; %Number of iterations
err = inf; %Initial error
k=0; %Iteration counter

fderiv = diff(f); %Derivative of f

plot(x_interval, [0,0], 'k') %x-axis
hold on
plot([0,0], y_interval, 'k') %y-axis
fplot(f,x_interval,'b') %plot f
y0 = double(subs(f,x,x0));
plot(x0, y0,'or') %plot starting point (x0,f(x0))
plot([x0,x0],[0,y0],'r')

%Saving datas
VarNames = {'k', 'x', 'fx', 'err'};
datasave=[];
datasave=[datasave; k, x0, y0, err];

while ((tol<err) && k<nmax)
    color=rand(1,3); %Random color for grafic visualization
    fx = subs(f,x,x0); %Calculating f(x)
    fdx = subs(fderiv,x,x0); %Calculating f'(x)
    if (abs(fdx)==0)
        break
    end
    xk=double(x0-(fx/fdx)); %Calculating x(k+1)=x(k)- f(xk)/f'(xk)
    %Plotting the grafic visualization of iteration
    plot([xk,xk], [0, double(subs(f,x,xk))],'--','color',color)
    plot([x0,xk], [double(subs(f,x,x0)),0],'--','color',color)
    err = abs(xk-x0);
    x0=xk;
    y0 = double(subs(f,x,x0));
    plot(x0, y0,'or','color',color) %%plot point (x(k+1),f(x(k+1)))
    %fplot(tangent,x_interval,'color',color);
    k=k+1;
    datasave=[datasave; k, x0, y0, err]; %Saving datas
end
axis([x_interval y_interval]) % rescaling
hold off
T = table(datasave(:,1),datasave(:,2),datasave(:,3),datasave(:,4), 'VariableNames',VarNames);
display(T) %Displaying datas
end
```

5.4 Metodo delle Corde

```
function [xk,k,err]=Corda(f,x0,tol, itr, x_interval,y_interval)

% f = function
% x_0 = starting point
% tol = tolerance
% x_interval        width of plot
% y_interval        height of plot

syms x

nmax=itr; %Number of iterations
err = inf; %Initial error
k=0; %Iteration counter

fderiv = diff(f); %Derivative of f
m = subs(fderiv,x,x0);

if(abs(m)==0)
    return
end

plot(x_interval, [0,0], 'k') %x-axis
hold on
plot([0,0], y_interval, 'k') %y-axis
fplot(f,x_interval,'b') %plot f
y0 = double(subs(f,x,x0)); %Calculating f(x0)
plot(x0, y0,'or') %plot starting point (x0,f(x0))
plot([x0,x0],[0,y0],'r')

%Saving datas
VarNames = {'k', 'x', 'fx', 'err'};
datasave=[];
datasave=[datasave; k, x0, y0, err];

while ((tol<err) && k<nmax)
    color=rand(1,3); %Random color for grafic visualization
    fx = subs(f,x,x0); %Calculating f(x)
    xk=double(x0-(fx/m)); %Calculating x(k+1)=x(k)- f(xk)/f'(xk)
    plot([xk,xk], [0, double(subs(f,x,xk))],'--','color',color)
    plot([x0,xk], [double(subs(f,x,x0)),0],'--','color',color)
    err = abs(xk-x0);
    x0=xk;
    y0 = double(subs(f,x,x0));
    plot(x0, y0,'or','color',color) %%plot point (x(k+1),f(x(k+1)))
    k=k+1;
    datasave=[datasave; k, x0, y0, err]; %Saving datas
end
axis([x_interval y_interval]) % rescaling
hold off
T = table(datasave(:,1),datasave(:,2),datasave(:,3),datasave(:,4), 'VariableNames',VarNames);
display(T) %Displaying datas
end
```

5.5 Metodo delle Secanti

```
function [xk,k,err]=SecantMethod(f,x0,x1,tol, itr, x_interval,y_interval)

% Plot the function f and the iterations of secant method starting from x0,x1 until
% |x_k - x_(k-1)| < tol or k>=nmax

% f = function
% x_0 = First starting point
% x_1 = Second starting point
% tol = tolerance
% x_interval        width of plot
% y_interval        height of plot


syms x

nmax=itr; %Number of iterations
err = inf; %Initial error
k=0; %Iteration counter

plot(x_interval, [0,0], 'k') %x-axis
hold on
plot([0,0], y_interval, 'k') %y-axis
fplot(f,x_interval,'b') %plot f
y0 = double(subs(f,x,x0)); %Calculating f(x0)
y1 = double(subs(f,x,x1)); %Calculating f(x1)
plot([x0,x0],[0,y0],'r')
plot([x1,x1],[0,y1],'r')

%Saving datas
VarNames = {'k', 'x', 'fx', 'err'};
datasave=[];
datasave=[datasave; k, x1, y1, err];

while ((tol<err) && k<nmax)
    color=rand(1,3); %Random color for grafic visualization
    fx0 = subs(f,x,x0); %Calculating f(x0)
    fx1 = subs(f,x,x1); %Calculating f(x1)
    m = (fx1-fx0)/(x1-x0); %Calculating m
    if (fx1==0)
        break
    end
    x0 = x1;
    x1=eval(x1-(fx1/m)); %Calculating the new x1
    err = abs(x1-x0);
    %Grafic visualization of the iteration
    plot([x1,x1], [0, double(subs(f,x,x1))],'--','color',color)
    plot([x0,x1], [double(subs(f,x,x0)),0],'--','color',color)
    y1 = double(subs(f,x,x1));
    k=k+1;
    datasave=[datasave; k, x1, y1, err]; %Saving datas
end

axis([x_interval y_interval]) % rescaling
hold off
T = table(datasave(:,1),datasave(:,2),datasave(:,3),datasave(:,4), 'VariableNames',VarNames);
display(T) %Displaying results
end
```

Bibliografia

[Gem] Luca Gemignani. *Lezioni di Calcolo Numerico.*

[Rob] Ornella Menchi Roberto Bevilacqua. *Appunti di Calcolo Numerico.*

Sitografia

[EA14] J.C. Ehiwario and S.O. Aghamie. *Comparative Study of Bisection, Newton-Raphson and Secant Methods of Root- Finding Problems*. Apr. 2014. URL: `http://www.iosrjen.org/Papers/vol4_issue4%20(part-1)/A04410107.pdf`.

[Ahm15] Abdulaziz G. Ahmad. *Comparative Study of Bisection and Newton-Rhapson Methods of Root-Finding Problems*. Mar. 2015. URL: `https://www.ijmttjournal.org/2015/Volume-19/number-2/IJMTT-V19P516.pdf`.

[Wika] Wikipedia. *Calcolo di uno zero di una funzione*. URL: `https://it.wikipedia.org/wiki/Calcolo_di_uno_zero_di_una_funzione`.

[Wikb] Wikipedia. *Metodo di Bisezione, Metodo di Punto Fisso, Metodo di Newton, Metodo delle corde, Metodo delle Secanti*. URL: `https://it.wikipedia.org/`.