CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA ELETTRICA

# Equazioni non lineari

## Metodi iterativi per l'approssimazione di radici

**Corso di calcolo numerico 2**

**01/11/2010**

Manuela Carta

❖ INDICE

## *Introduzione*

In questo breve testo saranno analizzati alcuni metodi di tipo iterativo utilizzati per la risoluzione di equazioni non lineari. Si analizzeranno alcune applicazioni su Matlab in modo da capire e apprendere il funzionamento degli algoritmi trattati. In questo modo sarà possibile valutare quali caratteristiche e differenze li contraddistinguono e cosa porta quindi alla scelta di un metodo piuttosto che un altro.

## ❖ Metodo di bisezione

Il metodo consiste nel generare iterativamente una serie di intervalli [a,b] contenenti la radice della funzione f(x). Utilizzando un punto intermedio c all'intervallo,[a,b] viene suddiviso in due sottocampi [a,c] e [b,c]. Individuando l'intervallo in cui la funzione cambia segno si associa ad esso il nuovo intervallo [a,b] da analizzare. In questo modo l'algoritmo genererà ad ogni iterazione un intervallo [a,b] più stringente dell'iterazione precedente. La convergenza avviene quando l'intervallo [a,b] ha dimensioni tali da poter considerare il punto intermedio c punto soluzione.

*Implementazione Matlab*

```
a, b, f(x);
c=(a+b)/2;

k=0;
tao=1.0e-8; N=100;

x=c;
fa=f(a);
fb=f(b);
fc=f(c);

if fa*fb>=0
    disp('non ha passaggio per lo zero')
    break
end
while abs(b-a)>= tao & fc ~=0 & k<=N;
    k=k+1;
    if fa*fc <=0;
        b=c; fb=fc;
    else a=c; fa = fc;
    end
    c= (a+b)/2;fc= f(c);
    x=c;
end
c, k
```

Una volta definito l'intervallo e la funzione da analizzare l'algoritmo calcola la funzione nel punto a, b e nel punto intermedio c. Mediante l'istruzione *if* valuta la presenza dello zero all'interno dell'intervallo analizzando il segno del prodotto f(a)* f(b). Se il prodotto è positivo non vi è passaggio per lo zero, quindi segnala mediante l'istruzione *disp* il messaggio di errore 'non ha passaggio per lo zero' e si arresta. Viceversa l'algoritmo prosegue.
Il ciclo while permette di iterare il metodo fin tanto che l'intervallo non si riduca ad un valore tale da poter considerare "c" zero della funzione. Per poter valutare in quale intervallo tra [a,c] e [b,c] vi è il passaggio per lo zero, l'algoritmo valuta mediante l'istruzione *if* il segno del prodotto f(a)*f(c). se quest'ultimo è minore di zero definisce il nuovo estremo dell'intervallo 'b' pari al vecchio punto intermedio 'c' e definisce il nuovo intervallo [a,b] pari al vecchio intervallo [a,c]. Viceversa definisce il nuovo estremo dell'intervallo 'a' pari al vecchio punto 'c' e definisce il nuovo intervallo [a,b] pari al vecchio intervallo [c,b].

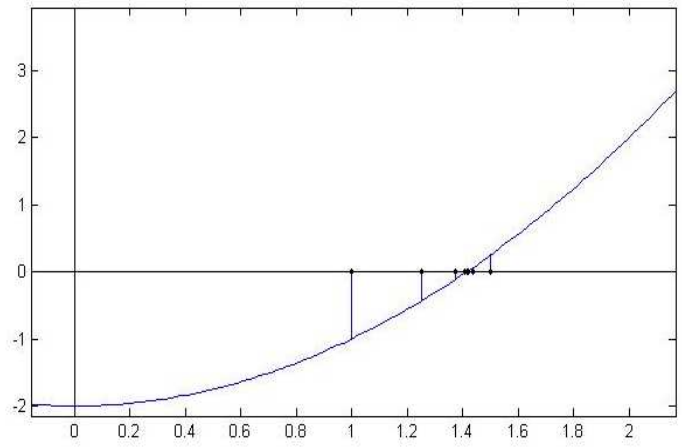
*Applicazione*

Si applica il metodo a tre funzioni diverse sullo stesso intervallo di partenza [a,b].
Prima funzione:

$$F(x) = x^2 - 2 \ , \ \ [a, b] = [0; 2]$$

L'algoritmo esegue 28 iterazioni. Si riportano in una tabella le prime 12 iterazioni in modo da capire come si comporta il metodo.

| K | a | b | X |
|---|---|---|---|
| 1 | 0 | 2.0000 | 1.0000 |
| 2 | 1.0000 | 2.0000 | 1.5000 |
| 3 | 1.0000 | 1.5000 | 1.2500 |
| 4 | 1.2500 | 1.5000 | 1.3750 |
| 5 | 1.3750 | 1.5000 | 1.4375 |
| 6 | 1.3750 | 1.4375 | 1.4063 |
| 7 | 1.4063 | 1.4375 | 1.4219 |
| 8 | 1.4063 | 1.4219 | 1.4141 |
| 9 | 1.4141 | 1.4219 | 1.4180 |
| 10 | 1.4141 | 1.4180 | 1.4160 |
| 11 | 1.4141 | 1.4160 | 1.4150 |
| 12 | 1.4141 | 1.4150 | 1.4146 |

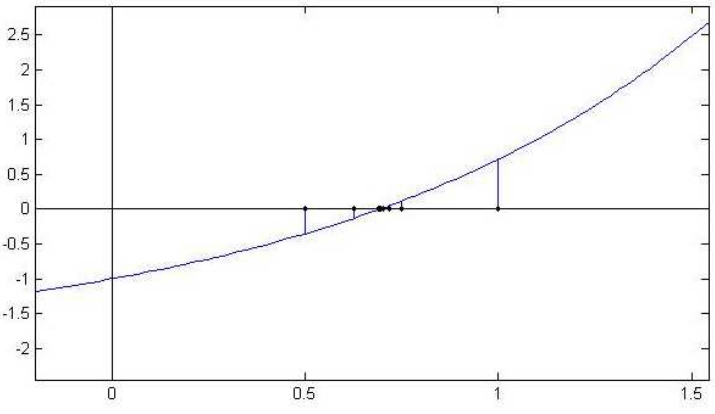



2°funzione

$$F(x) = e^x - 2 \ , \ \ [a, b] = [0; 2]$$

Come con la funzione precedente l'algoritmo esegue 28 iterazioni.

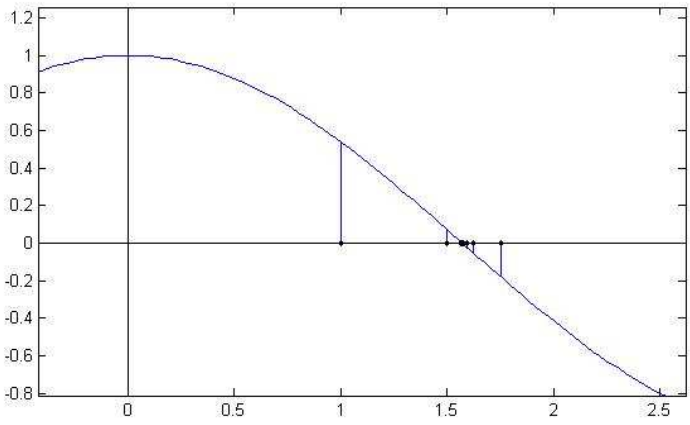
| K | a | B | X |
|---|---|---|---|
| 1 | 0 | 2.0000 | 1.0000 |
| 2 | 0 | 1.0000 | 0.5000 |
| 3 | 0.5000 | 1.0000 | 0.7500 |
| 4 | 0.5000 | 0.7500 | 0.6250 |
| 5 | 0.6250 | 0.7500 | 0.6875 |
| 6 | 0.6875 | 0.7500 | 0.7188 |
| 7 | 0.6875 | 0.7188 | 0.7031 |
| 8 | 0.6875 | 0.7031 | 0.6953 |
| 9 | 0.6875 | 0.6953 | 0.6914 |
| 10 | 0.6914 | 0.6953 | 0.6934 |
| 11 | 0.6914 | 0.6934 | 0.6924 |
| 12 | 0.6924 | 0.6934 | 0.6929 |



3°funzione

$F(x) = \cos(x)$ , $[a, b] = [0; 2]$

| K | a | B | X |
|---|---|---|---|
| 1 | 0 | 2.0000 | 1.0000 |
| 2 | 1.0000 | 2.0000 | 1.5000 |
| 3 | 1.5000 | 2.0000 | 1.7500 |
| 4 | 1.5000 | 1.7500 | 1.6250 |
| 5 | 1.5000 | 1.6250 | 1.5625 |
| 6 | 1.5625 | 1.6250 | 1.5938 |
| 7 | 1.5625 | 1.5938 | 1.5781 |
| 8 | 1.5625 | 1.5781 | 1.5703 |
| 9 | 1.5703 | 1.5781 | 1.5742 |
| 10 | 1.5703 | 1.5742 | 1.5723 |
| 11 | 1.5703 | 1.5723 | 1.5713 |
| 12 | 1.5703 | 1.5713 | 1.5708 |




Anche in quest'ultima funzione l'algoritmo esegue 28 iterazioni. Il numero di passi non dipende infatti dal tipo di funzione ma dall'intervallo scelto.

Variando l'intervallo [a,b] si ottiene che la velocità di convergenza cambia come indicato in tabella

| Funzione | [a,b] | Soluzione | # iterazioni |
|---|---|---|---|
| $f(x) = x^2 - 2;$ | [1,22] | X=1.4142 | 32 |
| $f(x) = e^x - 2;$ | [1,22] | X=0.6931 | 32 |
| $f(x) = \cos(x);$ | [1,22] | X=9*(pi/2) | 32 |

Il metodo di bisezione è un metodo molto robusto, nel senso che converge sempre. Ad ogni iterazione esegue una sola valutazione di funzione, quindi presenta un costo computazionale basso.

## ❖ Metodo di Newton

Il metodo di Newton (o delle tangenti) genera una successione di punti a partire da un punto iniziale $x0$ che dopo un certo numero di iterazioni converge ad un'approssimazione della radice della funzione. Data una funzione $f(x)$ a partire da $x0$ viene calcolata l'approssimazione successiva $x1$ come intersezione della retta tangente a $f(x)$ con l'asse delle ascisse.

Considerando l'equazione del fascio di rette:

$y - f(x_0) = m(x - x_0)$ , posto $y = 0 \quad e \quad m = f'(x_0), x_1$ sarà pari a:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Generalizzando all'iterazione k:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Il metodo è quindi applicabile solo a funzioni derivabili con derivata prima diversa da zero nei punti della successione.

*Implementazione del metodo*

```
F, df, x0

N=100;
k=0;
tao=10^(-8);
error= 1;

x= x0;
while (k<=N & error>=tao);
    fx= f(x);
    dfx=df(x);

    if abs(dfx)==0;
        break
    end
    x1= x-(fx/dfx);
    error= abs(x1-x);
    x=x1;
    k=k+1;
end
x,k
```

E' necessario fornire all'algoritmo la funzione *F,* la sua derivata *df* e il punto iniziale *xo*. Mediante il ciclo *while* è possibile iterare il metodo. L'algoritmo termina quando l'errore sulla soluzione risulta poco significativo o se il numero di iterazioni risulta superiore al limite prestabilito. Ad ogni iterazione l'algoritmo calcola funzione e derivata nel punto *x* corrente. Mediante l'istruzione *if* valuta se la derivata nel punto si annulla ed in tal caso blocca l'algoritmo. Viceversa calcola il nuovo punto *x* ed aggiorna il contatore di iterazioni *k*.

*Test sull'algoritmo*

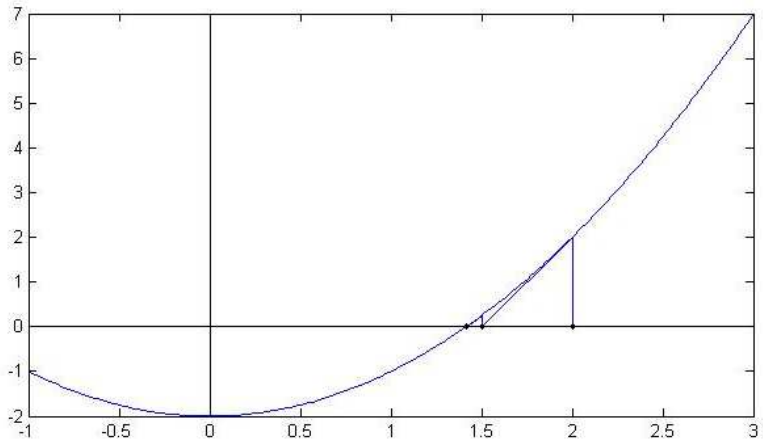
Si analizzano due funzioni distinte. Ogni funzione sarà testata su punti iniziali differenti per valutare la velocità di convergenza al variare di essi.

1° funzione

$f(x) = x^2 - 2$ ;

Punto iniziale $x_0 = 2$

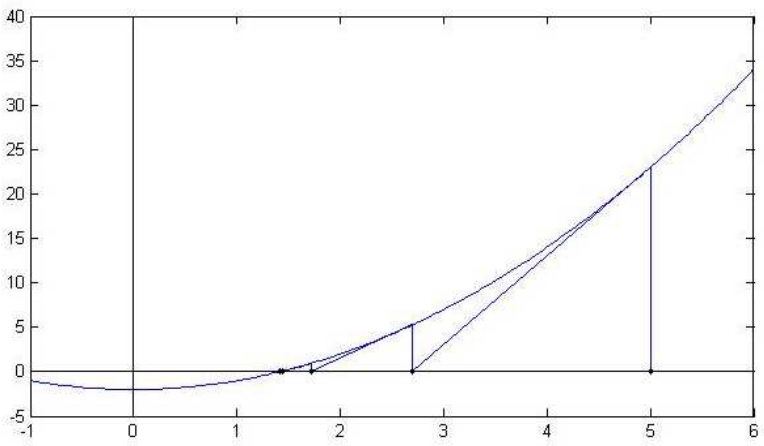
| k | X |
|---|---|
| 1 | 2.0000 |
| 2 | 1.5000 |
| 3 | 1.4167 |
| 4 | 1.4142 |
| 5 | 1.4142 |
| 6 | 1.4142 |




Si può subito notare la velocità di convergenza del metodo rispetto al metodo precedente. Esegue sole 6 iterazioni contro le 28 del metodo di bisezione.

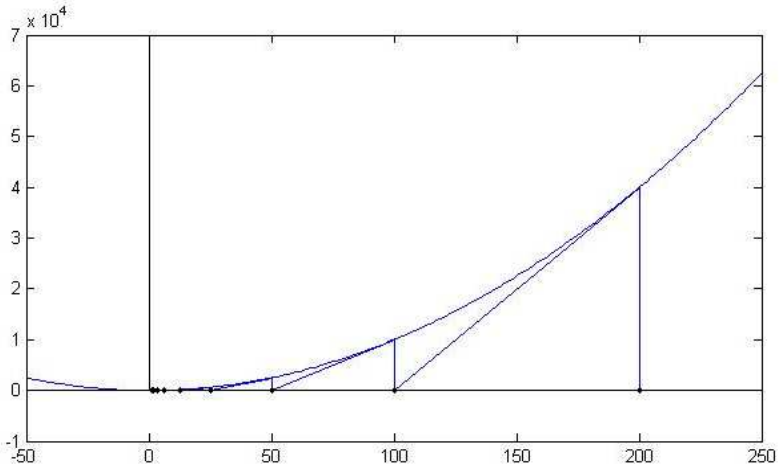
Punto iniziale $x_0 = 5$

| k | x |
|---|---|
| 1 | 5.0000 |
| 2 | 2.7000 |
| 3 | 1.7204 |
| 4 | 1.4415 |
| 5 | 1.4145 |
| 6 | 1.4142 |
| 7 | 1.4142 |
| 8 | 1.4142 |



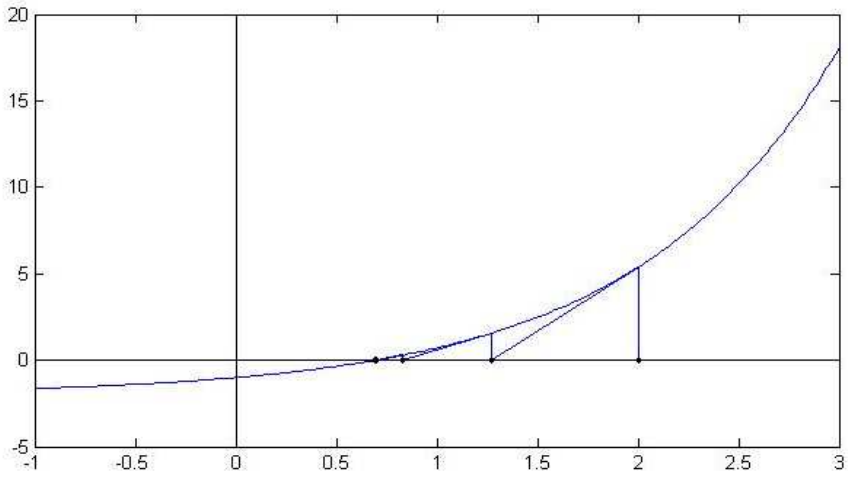

Punto iniziale $x_0 = 200$

| k | x |
|---|---|
| 1 | 200.0000 |
| 2 | 100.0050 |
| 3 | 50.0125 |
| 4 | 25.0262 |
| 5 | 12.5531 |
| 6 | 6.3562 |
| 7 | 3.3354 |
| 8 | 1.9675 |
| 9 | 1.4920 |
| 10 | 1.4162 |
| 11 | 1.4142 |
| 12 | 1.4142 |
| 13 | 1.4142 |



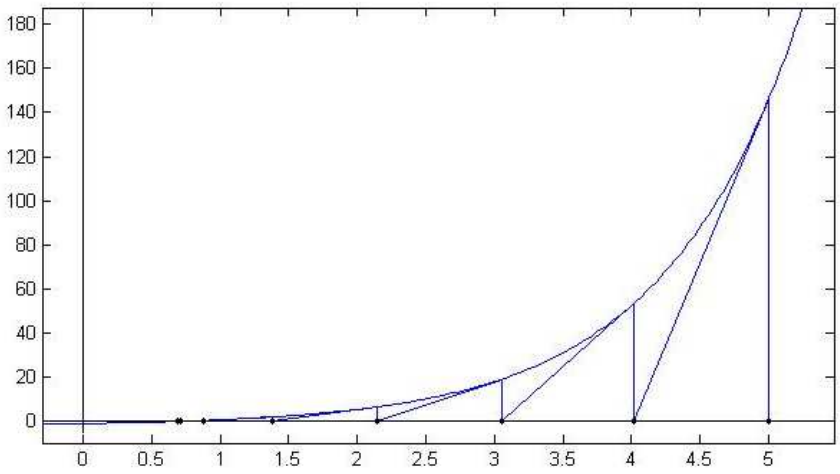

2° funzione $f(x) = e^x - 2$ ; Punto iniziale $x_0 = 2$

| k | x |
|---|---|
| 1 | 2.0000 |
| 2 | 1.2707 |
| 3 | 0.8320 |
| 4 | 0.7024 |
| 5 | 0.6932 |
| 6 | 0.6931 |



Punto iniziale $x_0 = 5$

| k | x |
|---|---|
| 1 | 5.0000 |
| 2 | 4.0135 |
| 3 | 3.0496 |
| 4 | 2.1444 |
| 5 | 1.3787 |
| 6 | 0.8825 |
| 7 | 0.7100 |
| 8 | 0.6933 |
| 9 | 0.6931 |




Punto iniziale $x_0 = 200$ k> 100 .

Per $x_0 = 200$ l'algoritmo realizza un numero di iterazioni maggiore di 100, quindi l'algoritmo si blocca. Il numero di iterazioni infatti dipende principalmente dalla posizione del punto iniziale rispetto allo zero. In base alla funzione analizzata e al valore di x0 assunto come prima iterazione il metodo può convergere o meno. Il metodo risulta meno solido rispetto al metodo di bisezione.

Come costo computazionale, ad ogni iterazione esegue due valutazioni di funzione. È quindi più pesante rispetto al precedente metodo ma esegue un numero di iterazioni significativamente inferiore.

## ❖ Metodi quasi-Newton

I metodi quasi-Newton rappresentano una variante al metodo di Newton. infatti, in quest'ultimo metodo è richiesto l'aggiornamento della funzione $f\ \ (x_k)$ e della sua derivata a $f'(x_k)$ ad ogni passo, mentre nei metodi quasi newton l'iterazione è calcolata nel seguente modo:

$$x_{k+1} = x_k - \frac{f(x_k)}{m_k}$$

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

*Corde*

Nel metodo delle corde il termine $m_k$ è un valore che resta costante ad ogni iterazione. Per esempio si potrebbe pensare di considerare $m_k$ pari al valore della derivata sul punto iniziale $f'(x_0)$ . in questo modo il metodo è analogo al metodo di newton classico con l'eccezione di non aggiornare la derivata ad ogni passo, che resta quindi costante, ma aggiornare la sola funzione $f\ \ (x_k)$.

*Sperimentazione su Matlab*

```
f ,df,x0;
m=df(x0);
N=200;
k=0;
tao=10^(-8);
error= 1;
x= x0;
if abs(m)==0;
    break
end

while (k<=N & error>=tao);
    fx= f(x);
    x1= x-(fx/m);
    error= abs(x1-x);
    x=x1;
    k=k+1;
end
```

Come si può notare, a parte qualche accorgimento, l'implementazione è uguale al metodo di Newton. è necessario fornire in ingresso la funzione *F,* la sua derivata *df* e il punto iniziale *xo*. Si è scelto di applicare il

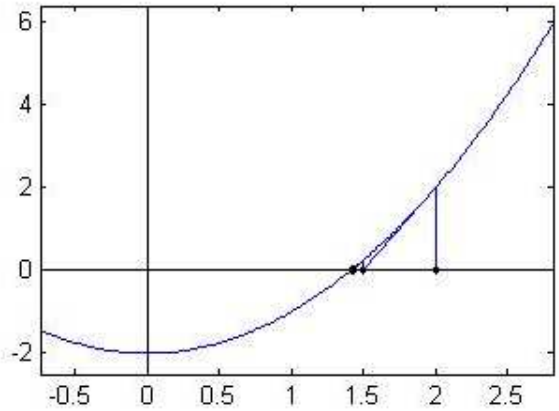
metodo a due funzioni diverse. Per entrambe le funzioni si utilizza come valore di $m_k$ la derivata sul punto iniziale $f'(x_0)$.

*Test algoritmo*

1° funzione $f(x) = x^2 - 2$

Punto iniziale $x_0 = 2$ ; soluzione $x = \sqrt{2}$; iterazioni totali 15; per visualizzare il comportamento dell'algoritmo si riportano le prime 13 iterazioni

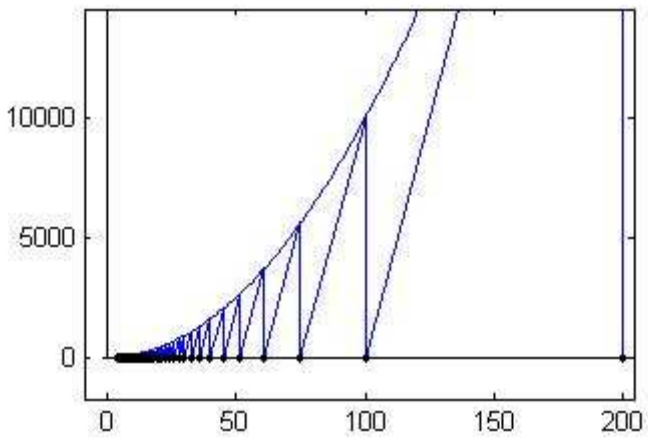
| k | x |
|---|---|
| 1 | 2.0000 |
| 2 | 1.5000 |
| 3 | 1.4375 |
| 4 | 1.4209 |
| 5 | 1.4162 |
| 6 | 1.4148 |
| 7 | 1.4144 |
| 8 | 1.4143 |
| 9 | 1.4142 |
| 10 | 1.4142 |
| 11 | 1.4142 |
| 12 | 1.4142 |
| 13 | 1.4142 |




Punto iniziale $x_0 = 200$

In questo caso l'algoritmo realizza più di 100 iterazioni, numero massimo prefissato e si blocca.

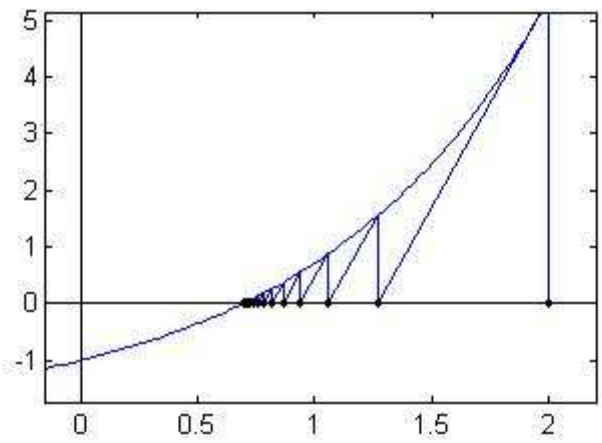
| k | x |
|---|---|
| 1 | 200.0000 |
| 2 | 100.0050 |
| 3 | 75.0075 |
| 4 | 60.9472 |
| 5 | 51.6658 |
| 6 | 44.9974 |
| 7 | 39.9405 |
| 8 | 35.9574 |
| 9 | 32.7300 |
| 10 | 30.0569 |
| 11 | 27.8034 |
| 12 | 25.8758 |
| 13 | 24.2069 |



2° funzione $f(x) = e^x - 2$; Punto iniziale $x_0 = 2$; soluzione $x = log2$; iterazioni totali 54;

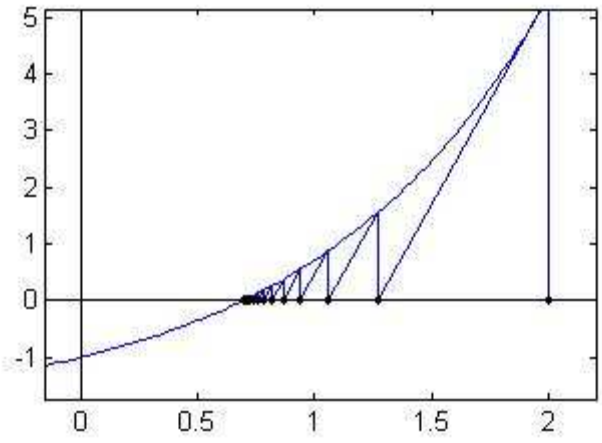
si riportano le prime 13 soluzioni.

| k | x |
|---|---|
| 1 | 2.0000 |
| 2 | 1.2707 |
| 3 | 1.0591 |
| 4 | 0.9395 |
| 5 | 0.8639 |
| 6 | 0.8135 |
| 7 | 0.7789 |
| 8 | 0.7546 |
| 9 | 0.7375 |
| 10 | 0.7252 |
| 11 | 0.7164 |
| 12 | 0.7100 |
| 13 | 0.7054 |




In questo caso già per valori di x0 vicini alla soluzione si ha un numero di iterazioni abbastanza elevato. Inserendo come punto iniziale $x_0 = 200$ l'algoritmo effettua un numero di iterazioni ben più superiore al centinaio. Come si può notare dalle prime 13 soluzioni il metodo converge alla soluzione molto più lentamente rispetto a quanto valutato con la prima funzione.

| k | x |
|---|---|
| 1 | 200.0000 |
| 2 | 199.0000 |
| 3 | 198.6321 |
| 4 | 198.3775 |
| 5 | 198.1801 |
| 6 | 198.0180 |
| 7 | 197.8802 |
| 8 | 197.7602 |
| 9 | 197.6537 |
| 10 | 197.5580 |
| 11 | 197.4710 |
| 12 | 197.3913 |
| 13 | 197.3176 |




*Secanti*

In questo metodo come termine $m_k$ viene utilizzato il coefficiente 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 $x_1$.

Il termine $m_k$ sarà calcolato come:

$m_k = \frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$;

*Implementazione matlab*

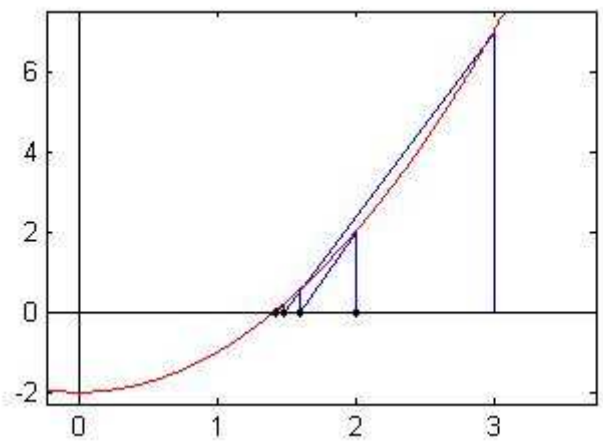
```
f, x0, x1;
N=100;
k=0;
tao=10^(-8);
error= 1;
error=abs(x1-x0);

X=x0;
while (k<=N  & error >=tao);
    fx1= f(x1);
    fx0=f(x0);
    m = (fx1-fx0)/(x1-x0);
    if fx1==0;
        break
    end
    x2= x1-(fx1/m);
    x0=x1;
    x1=x2;
    error= abs(x1-x0);

    k=k+1;
end
```

A partire dai punti iniziali xo,x1 il metodo calcola la retta secante i due punti e calcola il nuovo punto come intersezione della retta con asse delle ascisse. Rinomina il nuovi x0,x1 e itera il procedimento fino ad ottenere un intervallo talmente ristretto da considerare x1 soluzione del problema.

*Test*

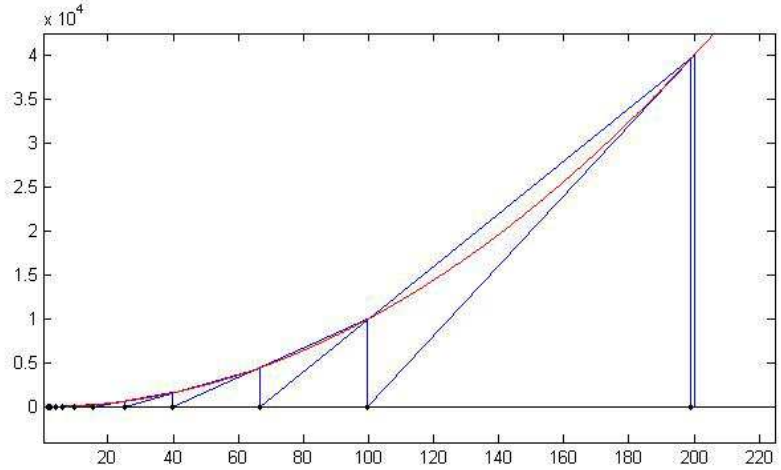
1° funzione $f(x) = x^2 - 2$
Punto iniziale $x_0 = 2$ ; $x_1 = 3$ ; soluzione $x = \sqrt{2}$; iterazioni totali 7

| k | x |
|---|---|
| 1 | 3.0000 |
| 2 | 1.6000 |
| 3 | 1.4783 |
| 4 | 1.4181 |
| 5 | 1.4143 |
| 6 | 1.4142 |
| 7 | 1.4142 |
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |



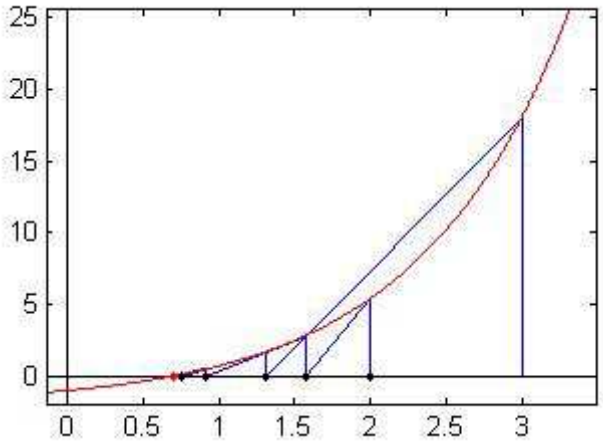
Punto iniziale $x_0 = 199$ ; $x_1 = 200$ ; soluzione $x = \sqrt{2}$; iterazioni totali 16

| k | x |
|---|---|
| 1 | 200.0000 |
| 2 | 99.7544 |
| 3 | 66.5641 |
| 4 | 39.9358 |
| 5 | 24.9793 |
| 6 | 15.3981 |
| 7 | 9.5755 |
| 8 | 5.9841 |
| 9 | 3.8112 |
| 10 | 2.5325 |
| 11 | 1.8368 |
| 12 | 1.5224 |
| 13 | 1.4278 |



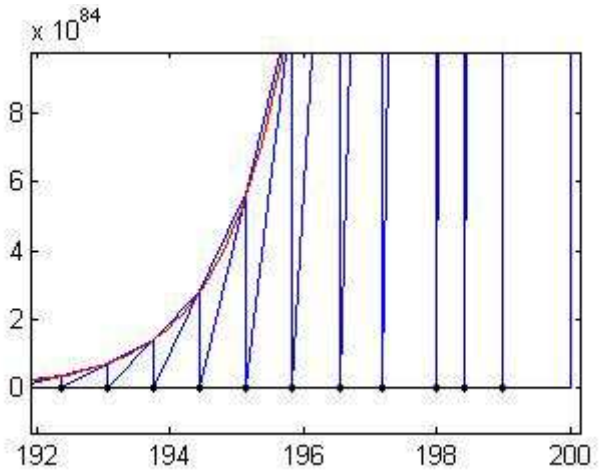

2° funzione $f(x) = e^x - 2$; Punto iniziale $x_0 = 2$; $x_1 = 3$; soluzione $x = log2$; iterazioni totali 9;

| k | x |
|---|---|
| 1 | 3.0000 |
| 2 | 1.5755 |
| 3 | 1.3109 |
| 4 | 0.9084 |
| 5 | 0.7511 |
| 6 | 0.6991 |
| 7 | 0.6933 |
| 8 | 0.6931 |
| 9 | 0.6931 |
| 10 | 0.6931 |
| 11 | |
| 12 | |
| 13 | |



Punto iniziale $x_0 = 199$; $x_1 = 200$; soluzione $x = log2$; iterazioni totali >100;

| k | x |
|---|---|
| 1 | 200.0000 |
| 2 | 198.4180 |
| 3 | 198.0087 |
| 4 | 197.1994 |
| 5 | 196.5501 |
| 6 | 195.8398 |
| 7 | 195.1533 |
| 8 | 194.4576 |
| 9 | 193.7654 |
| 10 | 193.0719 |
| 11 | 192.3789 |
| 12 | 191.6857 |
| 13 | 190.9926 |




La differenza col metodo delle corte è evidente già per punti iniziali piccoli, man mano che l'intervallo si restringe i punti trovati tendono ai punti trovati col metodo di Newton, perché la secante tenderà a coincidere con la derivata. Rispetto al metodo di bisezione esegue un numero di iterazioni minore, quindi è una via di mezzo tra newton e bisezione. Il metodo più lento è quindi quello delle corde che risulta scarsamente soddisfacente anche paragonato al metodo di bisezione.

Si riportano di seguito un quadro riassuntivo dei risultati ottenuti testando i metodi.

| $f(x) = x^2 - 2$, soluzione x $= \sqrt{2}$; | | | |
|---|---|---|---|
| *Metodo* | *Punto iniziale x0* | *# iterazioni* | *Costo/iterazione* |
| *Bisezione* | $[a, b] = [0; 2]$ | 28 | 1 valutazione di funzione |
| *Newton* | $x_0 = 2$ | 6 | 2 valutazioni di funzione |
| *Corde* | $x_0 = 2$ | 15 | 1 valutazione di funzione |
| *secanti* | $[x_0, x_1] = [2; 3]$ | 7 | 1 valutazione di funzione |

| $f(x) = e^x - 2$; soluzione $x = log2$; | | | |
|---|---|---|---|
| *Metodo* | *Punto iniziale x0* | *# iterazioni* | *Costo/iterazione* |
| *Bisezione* | $[a, b] = [0; 2]$ | 28 | 1 valutazione di funzione |
| *Newton* | $x_0 = 2$ | 6 | 2 valutazioni di funzione |
| *Corde* | $x_0 = 2$ | 54 | 1 valutazione di funzione |
| *secanti* | $[x_0, x_1] = [2; 3]$ | 9 | 1 valutazione di funzione |

## ❖ Iterazioni di punto fisso

Rappresenta la generalizzazione dei metodi iterativi. Dato un problema, senza far riferimento ad un metodo in particolare è possibile generare infiniti metodi iterativi che consentono di ottenere la radice del problema come approssimazione del punto fisso α.

La funzione di iterazione è definita come:

$x_{k+1} = g(x_k)$,

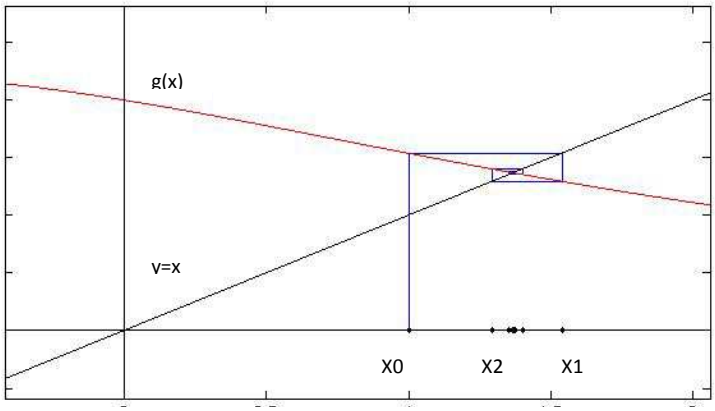
Iterativamente si produce una successione di $x_k$. Se $x_k$converge e se la $g(x_k)$ è continua allora $x_k$ converge ad un punto fisso α.

α è definito punto fisso per $g(x)$se

α = $g(\alpha)$.

*convergenza della successione*

A partire da un punto iniziale x0, la convergenza viene illustrata graficamente nella seguente figura.




Alla prima iterazione il metodo calcola:

$x_1 = g(x_0)$ per poter calcolare poi $x_2$ è necessario valutare $g(x_1)$ ; graficamente si proietta il punto $x_1$ sulla bisettrice, si riporta su $g(x_1)$ per poi proiettarlo nuovamente sulla bisettrice trovando cosi $x_2$ .

Il punto fisso risulta quindi l'intersezione di $g(x)$ con la bisettrice $y = x$.

Non esiste un unico modo per scegliere la funzione di iterazione, a seconda dell'equazione è possibile ricavare diverse $g(x)$ che non è detto convergano al punto fisso e soprattutto con lo stesso numero di iterazioni.

La convergenza del metodo dipende, infatti ,dalla pendenza della curva direttamente collegata alla definizione di contrattività .

La $g(x)$si definisce contrattiva se esiste c<1 e se

$|g(x) - g(y)| \leq c|x - y|$ , ∀ x,y

Quindi significa che considerando due punti $x, y$ la distanza tra $g(x)$ e $g(y)$ è minore stretto della distanza tra $x$ e $y$. Ossia la $g(x)$ ha la caratteristica di avvicinare due punti, prendendo $x, y$ la contrattività fa si che $g(x)$ e $g(y)$ si avvicinino.

La convergenza è inoltre legata alla definizione di derivabilità,infatti se la funzione è definita tale in [x,y] si può scrivere

$|g(x) - g(y)| = g'(\xi)\,|x - y|$

Maggiorando

$|g(x) - g(y)| = g'(\xi)\,|x - y| \leq c|x - y|$

C= $\max_{x \in D} g'(x)$ .

per la contrattività c deve essere minore di uno. Si può dire quindi che se le derivate hanno modulo sempre minore di uno allora il metodo converge.

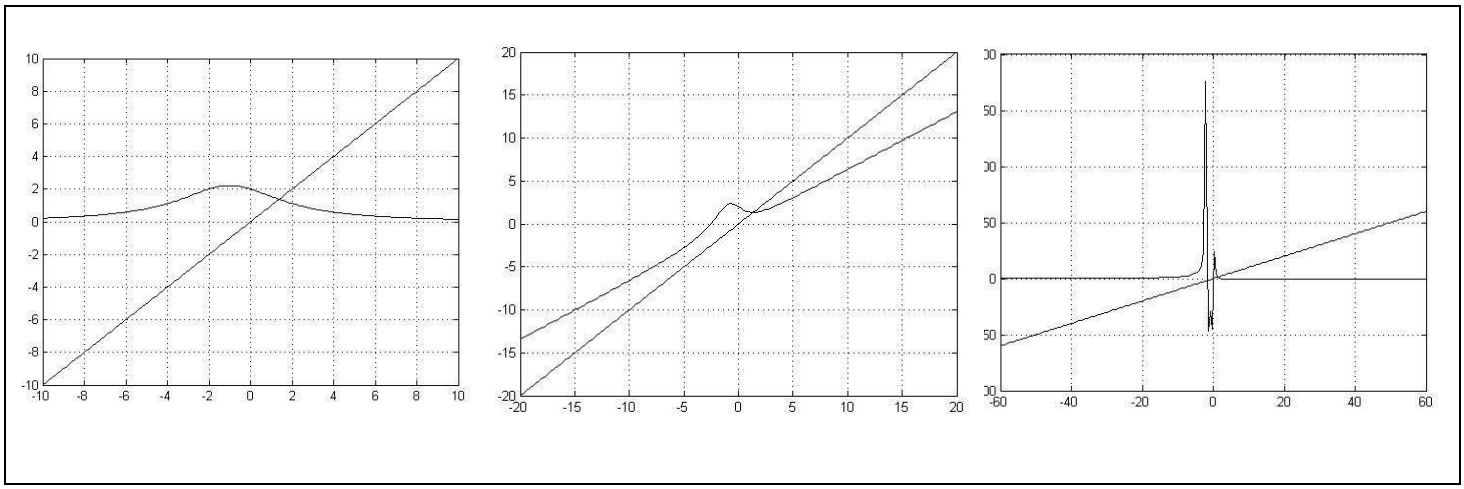
*Applicazione*

Calcolo dell'approssimazione della radice reale dell'equazione di Fibonacci.

$x^3+2x^2+10\,x$ -20=0

Da questa relazione è possibile ottenere diverse $g(x)$ tra cui:

1) $g(x)= \frac{20}{(x^2+2\,x\,+10)}$ 2) $g(x)= x - \frac{x^3+2x^2+10X-20}{(3x^2+4X+10)}$ 3) $g(x)= \frac{20-10\,x}{(x^2+2x\ )}$



*Implementazione Matlab*

```
g,den,x0;  % def funzione di iterazione g, denominatore di g, punto iniziale

N=100;
k=0;
tao=1.0e-10;
error= 1;
x= x0;
while (k<=N & error>= tao);
```

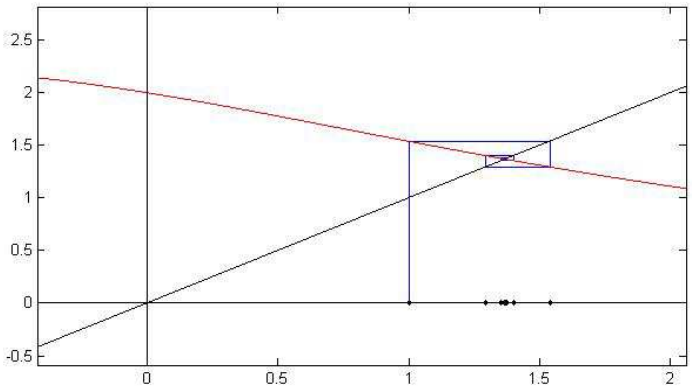
```
        if den(x)==0;
            break
        end
    gx= g(x);
    error= abs(gx-x);
    x=gx;
    k=k+1;
end
```

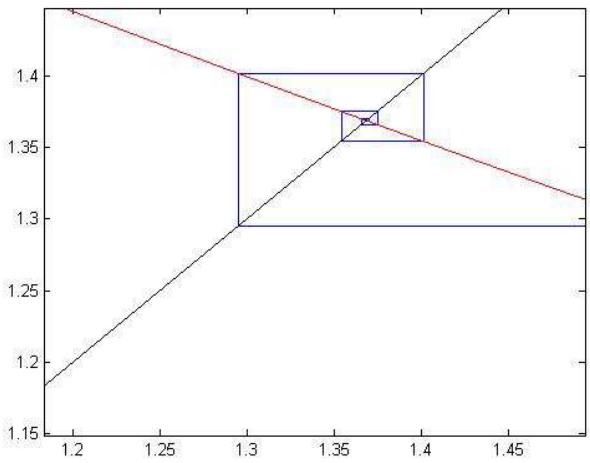
Non è possibile avere per tutte le funzioni la stessa implementazione è necessario adattare il programma al problema.

*Primo caso*

$g(x)= \frac{20}{(x^2+2x+10)}$ con punto iniziale x0=1; k=29

| k | X | g(x) |
|---|---|---|
| 0 | 1 | 1.53846153846 |
| 1 | 1.53846153846 | 1.29501915708 |
| 2 | 1.29501915708 | 1.40182530944 |
| 3 | 1.40182530944 | 1.35420939040 |
| 4 | 1.35420939040 | 1.37529809248 |
| 5 | 1.37529809248 | 1.36592978817 |
| 6 | 1.36592978817 | 1.37008600340 |
| 7 | 1.37008600340 | 1.36824102361 |
| 8 | 1.36824102361 | 1.36905981200 |
| 9 | 1.36905981200 | 1.36869639755 |
| 10 | 1.36869639755 | 1.36885768862 |
| 11 | 1.36885768862 | 1.36878610257 |
| 12 | 1.36878610257 | 1.36881787439 |
| 13 | 1.36881787439 | 1.36880377314 |
| 14 | 1.36880377314 | 1.36881003167 |
| 15 | 1.36881003167 | 1.36880725396 |
| 16 | 1.36880725396 | 1.36880848678 |
| 17 | 1.36880848678 | 1.36880793962 |
| 18 | 1.36880793962 | 1.36880818247 |
| 19 | 1.36880818247 | 1.36880807468 |
| 20 | 1.36880807468 | 1.36880812252 |
| 21 | 1.36880812252 | 1.36880810129 |
| 22 | 1.36880810129 | 1.36880811071 |
| 23 | 1.36880811071 | 1.36880810653 |
| 24 | 1.36880810653 | 1.36880810839 |
| 25 | 1.36880810839 | 1.36880810756 |
| 26 | 1.36880810756 | 1.36880810793 |
| 27 | 1.36880810793 | 1.36880810777 |
| 28 | 1.36880810777 | 1.36880810784 |
| 29 | 1.36880810784 | 1.36880810784 |



Il metodo converge, infatti la funzione è contrattiva con derivata prima sempre minore di 1 in modulo . La convergenza avviene in 29 iterazioni per poter valutare quanto il metodo è veloce si valuta la costante asintotica C. Più la costante si avvicina all'unità più il metodo è lento. Infatti per un metodo di ordine 1

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|} = C$$

Si può dimostrare che al limite

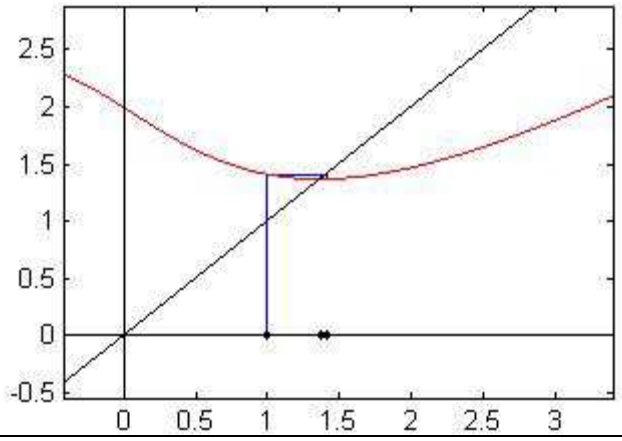
C = g'(α)

In questo caso C= 0,44;

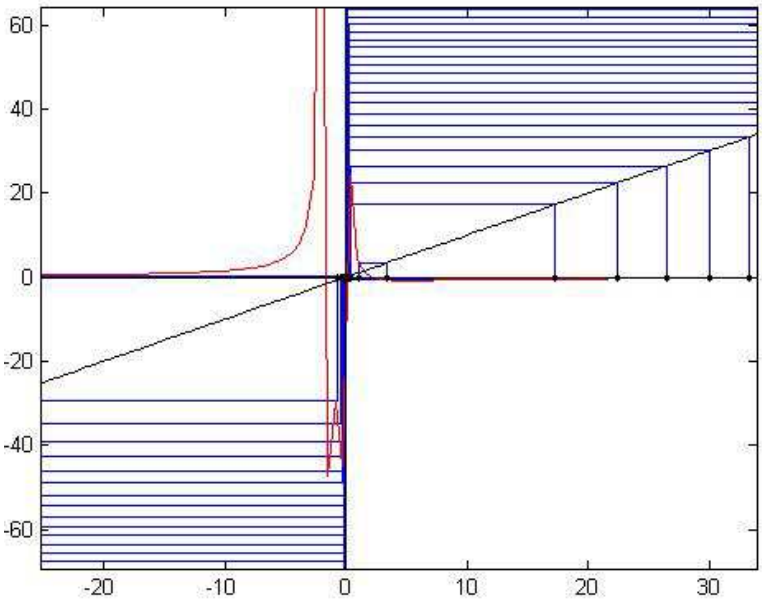
*Secondo caso*

$g(x)= x - \frac{x^3+2x^2+10X-20}{(3x^2+4X+10)}$ punto iniziale x0=1;

| k | X | g(x) |
|---|---|---|
| 0 | 1 | 1.41176470588 |
| 1 | 1.41176470588 | 1.36933647058 |
| 2 | 1.36933647058 | 1.36880818861 |
| 3 | 1.36880818861 | 1.36880810782 |
| 4 | 1.36880810782 | 1.36880810782 |
| 5 | 1.36880810782 | 1.36880810782 |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |




In questo caso la convergenza avviene in sole 6 iterazioni. Calcolando g'(α) si nota che il metodo è del secondo ordine p=2 , per questo motivo si ha maggiore velocità di convergenza.

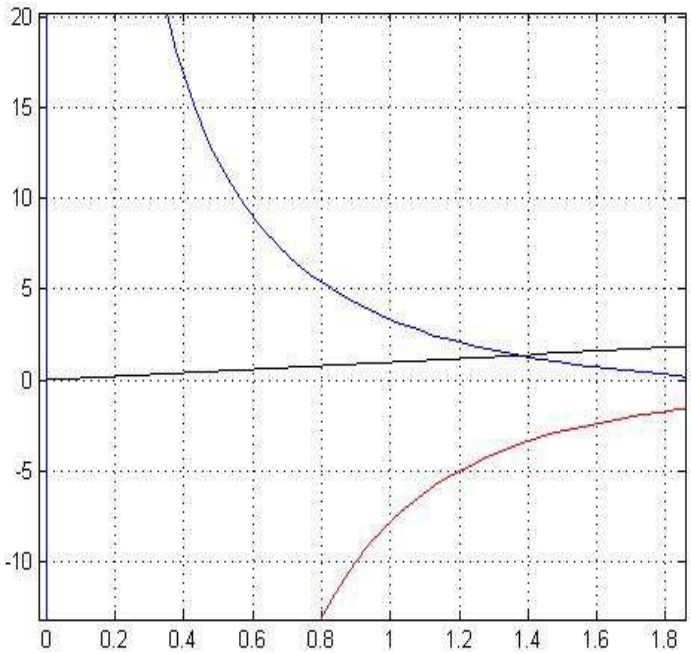



Terzo caso

$g(x)= \frac{20-10\,x}{(x^2+2x\ )}$ *x0=1;*

in quest'ultimo caso il metodo non converge.

La successione xk tende ad allontanarsi dal punto soluzione. La funzione non è contrattiva nell'intervallo in cui si trova la radice. Valutando la derivata nel punto soluzione si ottiene, infatti ,un valore superiore all'unità( tratto rosso nella figura seguente).



## ❖ Sistemi di equazioni non lineari

In questo breve testo si analizza il metodo di Newton multidimensionale. Considerato un sistema di n equazioni in m incognite:

$$\begin{cases} f_1(x_1, \dots, x_n) = 0 \\ f_2(x_1, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, \dots, x_n) = 0 \end{cases} \text{, posto } \quad F(x) = \begin{bmatrix} f_1(x_1, \dots, x_n) \\ f_2(x_1, \dots, x_n) \\ \vdots \\ f_n(x_1, \dots, x_n) \end{bmatrix},$$

il sistema può essere scritto in forma

$F(x) =0$;

Per ottenere l'iterazione si approssima $F(x)$ mediante la serie di Taylor troncata al primo ordine

$$F(x) \approx F(x^k) + F'(x^k)(x - x^k)$$

Per risolvere il sistema è richiesta quindi la valutazione della matrice Jacobiana $F'(x^k)$ che contiene le derivate parziali della funzione F valutata in $x^k$.

$$F'(x^k) = \begin{bmatrix} \frac{df_1}{x_1}\left(x_1^{(k)}, \dots, x_n^{(k)}\right) & \cdots \cdots & \frac{df_1}{x_n}\left(x_1^{(k)}, \dots, x_n^{(k)}\right) \\ & \vdots & \\ & \vdots & \\ \frac{df_n}{x_1}\left(x_1^{(k)}, \dots, x_n^{(k)}\right) & \cdots \cdots & \frac{df_n}{x_n}\left(x_1^{(k)}, \dots, x_n^{(k)}\right) \end{bmatrix};$$

la procedura iterativa può essere generalizzata come segue:

$$x^{k+1} = x^k - \left(F'(x^k)\right)^{-1} F(x^k);$$

A partire da un punto $x_0$ arbitrario, ad ogni passo l'algoritmo approssima $F(x^k)$ mediante la serie di Taylor troncata e calcola come $x^{k+1}$ il valore di x che annulla l'approssimazione di $F(x^k)$

La convergenza del metodo avviene quando il valore di $F(x^k)$ può essere considerato nullo ossia sotto un limite prestabilito. Il problema principale di questo metodo ,oltre alla necessità di esplicitare le derivate parziali della F, è che ad ogni iterazione è necessaria la valutazione della matrice Jacobiana. L'onere computazionale può essere comunque ridotto imponendo all'algoritmo la valutazione della matrice solo ogni tot iterazioni.

*Implementazione Matlab*

É necessario fornire all'algoritmo il vettore F(x), punto iniziale e Jacobiano.

```
x0;
nmax= 100;
k=0;
tao=10.^(-3);
error=1;
while (k<=nmax & error>=tao)
    f1,f2, J0;
    f0=[f1;f2]

    if det(J0)==0;
        disp ('singolare in x0')
        k=k+1;
        x=0;
                break
    end
    x= x0- J0\f0              %iterazione passo k
    error= norm(x-x0);
    k=k+1;
    x0=x;
end
```

*Esempio applicativo*

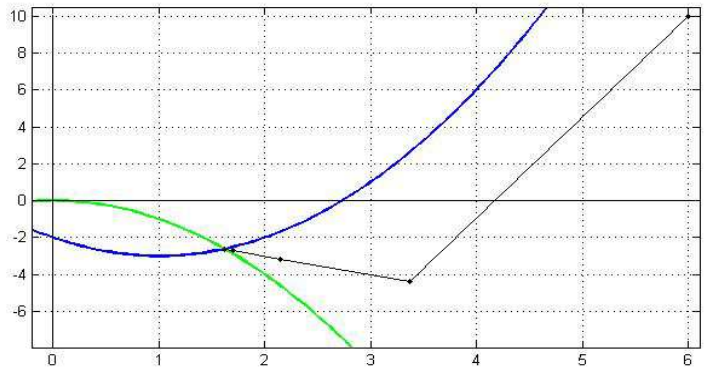
1) $\begin{cases} x^2 - 2x - y - 2 = 0 \\ x^2 - y = 0 \end{cases}$;

Definizione funzione F(x) e Jacobiano J0 su matlab.

```
f1=x0(1).^2-2*x0(1)-x0(2)-2;
f2=x0(1).^2+x0(2);

J0=[2*x0(1)-2, -1;2*x0(1),+1];

x0=[6,10];
```

nel grafico, le approssimazioni successive sono rappresentate dai puntini lungo la direzione di convergenza.

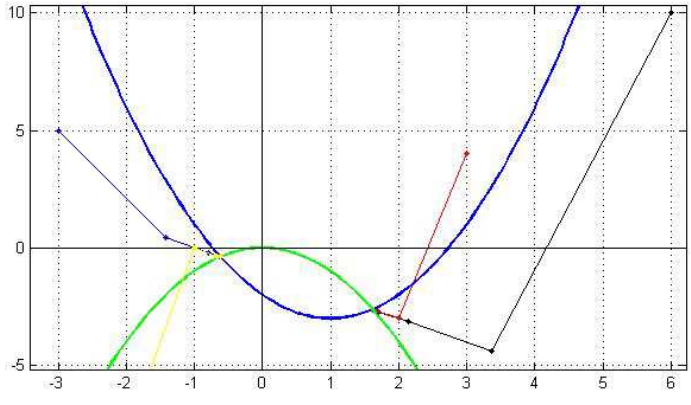
| k | x | F=[f1;f2] |
|---|---|---|
| 0 | [6;10] | [12;46] |
| 1 | [3.3636; -4.3636] | [6.9504;6.9504] |
| 2 | [2.1501; -3.1501] | [1.4727;1.4727] |
| 3 | [1.7038; -2.7038] | [0.1992;0.1992] |
| 4 | [1.6211; -2.6211] | [0.0068;0.0068] |
| 5 | [1.6180; -2.6180] | 1.0e-005 * [0.9312; 0.9312] |
| | | |
| 6 | [1.6180; -2.6180] | |
| Tratto nero | | |




Ad ogni passo F(x) si riduce , in questo caso f1 ed f2 dopo il primo passo tendono a zero seguendo una direzione retilinea. Questo comportamento può essere imputato alla particolare forma dello Jacobiano, nel caso generale il metodo ad ogni iterazione individua una direzione da seguire.

Il sistema presenta due soluzioni, a seconda del punto iniziale scelto il metodo converge ad una o all'altra soluzione. In entrambi i casi il numero di iterazioni è limitato come si può notare nella seguente tabella.

| # k | x0 | x |
|---|---|---|
| 6 | [6;10] | [1.6180; -2.6180] |
| | | Tratto nero |
| | | |
| 5 | [3;4] | [1.6180; -2.6180] |
| | | Tratto rosso |
| | | |
| 5 | [-3;5] | [-0.6180; -0.3820] |
| | | Tratto blu |
| | | |
| 5 | [-2;-8] | [-0.6180; -0.3820] |
| | | Tratto giallo |




Per ridurre il costo computazionale si possono utilizzare delle strategie di approssimazione dello Jacobiano. In questo senso operano i metodi di Newton-Jacobi e Newton-Gauss-Seidel che utilizzano come approssimazione di F'(x) rispettivamente:

F'(x)= D(x) e

F'(x)= D(x)-L(x).

Dove D(x) rappresenta la matrice diagonale e L(x) la sotto triangolare inferiore.

Considerando il seguente sistema si procede nel testare i metodi su Matlab

$$\begin{cases} x^2 - 2y^2 - 4 = 0 \\ xy - xy^2 = 0 \end{cases}$$

1° caso F'(x)= D(x)-L(x)-U(x) Newton
2° caso F'(x)=D(x) Newton-Jacobi
3° caso F'(x)=D(x)-L(x) Newton-Gauss-Seidel

| Caso | Punto iniziale | # iterazioni | soluzione |
|---|---|---|---|
| 1 | X0=[5;5] | 9 | X=[2.4495;1.0000] |
| 2 | X0=[5;5] | 9 | X=[2.4495;1.0000] |
| 3 | X0=[5;5] | 10 | X=[2.4495;1.0000] |

Il metodo di Newton-Jacobi calcola ad ogni iterazione solo due elementi su sei dello jacobiano mentre il metodo di Newton-Gauss-Seidel ne valuta 3 elementi su 6. Nonostante le approssimazioni utilizzate i metodi convergono alla soluzione con un numero di iterazioni limitato.

## ❖ Zeri di polinomi

Il problema del calcolo degli zeri di un polinomio può essere riformulato come un problema di auto valori.

Dato un polinomio $p(x)$

$p(x)= a_n\, x^n + a_{n-1} x^{n-1} + \cdots + a_1\, x \;\; + a_0\,,$

Dopo aver reso monico il polinomio è possibile associare la matrice in forma compagna C

$\tilde{p}(x) = x^n - b_{n-1} x^{n-1} - \cdots - b_1\, x \;\; + b_0\,,$

$$C = \begin{pmatrix} b_{n-1} & b_{n-2} & \cdots & \cdots & b_0 \\ 1 & \emptyset & \cdots & \cdots & \emptyset \\ \emptyset & 1 & \ddots & & \emptyset \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \emptyset & \cdots & \emptyset & 1 & \emptyset \end{pmatrix};$$

Gli auto valori della matrice C rappresentano gli zeri del polinomio $p(x)$. È possibile ottenere un approssimazione degli auto valori della matrice $\lambda(C)$ applicando la fattorizzazione QR, dove Q è una matrice ortogonale e R triangolare superiore. La fattorizzazione consiste nel produrre una successione di matrici $C_k = Q_k R_k$ unitariamente simili che converge ad una matrice $C_k$ triangolare superiore che quindi presenta in diagonale gli auto valori di C.

*Implementazione matlab*

```
P=polinomio
n=length(P);
if P(1)==1, P= -P, else P= -P/P(1);
end

b=P(2:n);
C=[b;eye(n-2),zeros(n-2,1)];
k=0;
x=0;

while k<=100 & norm(tril(C,-1))>= 1.0e-8;
    [Q,R]=qr(C);
    C=R*Q;
    k=k+1;
end
C,k
x=diag(C)
```

L'algoritmo calcola in primo luogo la matrice compagna in maniera analoga alla funzione *compan* di Matlab. Mediante il ciclo while realizza la successione di matrici C che termina quando il numero di iterazioni supera il limite prefissato o quando gli elementi sottodiagonali sono approssimabili a zero. In quest'ultimo caso la

successione di matrici converge alla matrice triangolare superiore da cui è possibile ricavare gli auto valori dalla diagonale principale e quindi gli zeri di $p(x)$.

Questo metodo è molto utilizzato perché oltre a non dipendere da un valore iniziale, il condizionamento della matrice C è pari al condizionamento del sistema triangolare ad esso equivalente. Nonostante ciò non è detto che si ottenga una convergenza dell'algoritmo. infatti, In caso in cui gli autovalori della matrice non siano reali o non siano distinti in modulo si ha che l'elemento della sotto diagonale corrispondente all'autovalore complesso (o di molteplicità maggiore di uno) non si annulli. Se tutti gli autovalori sono reali e distinti gli elementi della diagonale saranno anch'essi reali e distinti

$$C = \begin{pmatrix} \lambda_1 & * & * & \cdots & * \\ \emptyset & \lambda_2 & * & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & * \\ \emptyset & \cdots & \cdots & \emptyset & \lambda_n \end{pmatrix}$$

Se invece, è presente una coppia di autovalori complessi e coniugati in tal caso il blocco corrispondente convergerà ad una sottomatrice 2x2.

$$C = \begin{pmatrix} \lambda_1 & * & * & \cdots & * \\ \emptyset & \lambda_2 & * & & \vdots \\ \vdots & \emptyset & * & * & \vdots \\ \vdots & & * & * & * \\ \emptyset & \cdots & \cdots & \emptyset & \lambda_n \end{pmatrix}$$

Per poter calcolare gli auto valori della sottomatrice è necessario risolvere il polinomio caratteristico ad essa associato.

*Prova su polinomio con radici reali*

Dato il polinomio $p(x)= x^3-10x^2+31x \quad -30$ calcolata la matrice in forma compagna si procede al calcolo degli auto valori mediante fattorizzazione QR.

Prima fattorizzazione

$$C0 = \begin{pmatrix} 10 & -31 & 30 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = Q1 * R1 = \begin{pmatrix} 0{,}9950 & -0{,}0947 & 0{,}0307 \\ 0{,}0995 & 0{,}9463 & -0{,}3069 \\ 0 & 0{,}3084 & 0{,}9513 \end{pmatrix} * \begin{pmatrix} 10{,}0499 & -30{,}8462 & 29{,}8511 \\ 0 & 3{,}2427 & -2{,}8396 \\ 0 & 0 & 0{,}9206 \end{pmatrix};$$

$$C1=R1*Q1=\begin{pmatrix} 6{,}9307 & -20{,}9426 & 38{,}1700 \\ 0{,}3227 & 2{,}1936 & -3{,}6963 \\ 0 & 0{,}2839 & 0{,}8757 \end{pmatrix};$$

Seconda fattorizzazione

$$C1=Q2*R2=\begin{pmatrix} 0.9989 & -0.0463 & 0.0042 \\ 0.0465 & 0.9949 & -0.0892 \\ 0 & 0.0893 & 0.9960 \end{pmatrix}*\begin{pmatrix} 6.9382 & -20.8179 & 37.9568 \\ 0 & 3.1779 & -5.3672 \\ 0 & 0 & 1.3606 \end{pmatrix};$$

$$C2=R2*C2=\begin{pmatrix} 5.9626 & -17.6428 & 39.6916 \\ 0.1478 & 2.6822 & -5.6294 \\ 0 & 0.1216 & 1.3552 \end{pmatrix};$$

terza fattorizzazione

$$C2=\ Q3*R3=\begin{pmatrix} 0.9997 & -0.0248 & 0.0010 \\ 0.0248 & 0.9989 & -0.0389 \\ 0 & 0.0389 & 0.9992 \end{pmatrix}*\begin{pmatrix} 5.9644 & -17.5709 & 39.5399 \\ 0 & 3.1209 & -6.5533 \\ 0 & 0 & 1.6116 \end{pmatrix}$$

$$C3=R3*Q3=\begin{pmatrix} 5.5272 & -16.1599 & 40.1998 \\ 0.0773 & 2.8624 & -6.6698 \\ 0 & 0.0628 & 1.6104 \end{pmatrix};$$

Già alla settima fattorizzazione è possibile notare come gli elementi sotto diagonali della matrice C tendono a zero e glie elementi della diagonale principale tendono a stabilizzarsi.

$$C6=\begin{pmatrix} 5.0373 & -14.3641 & 40.6477 \\ 0.0088 & 3.0032 & -7.9278 \\ 0 & 0.0085 & 1.9344 \end{pmatrix}=Q7*R7=$$

$$=\begin{pmatrix} 1.0000 & -0.0017 & 0.0000 \\ 0.0017 & 1.0000 & -0.0028 \\ 0 & 0.0028 & 1.0000 \end{pmatrix}*\begin{pmatrix} 5.0624 & -14.4697 & 40.6339 \\ 0 & 3.0283 & -7.9927 \\ 0 & 0 & 1.9569 \end{pmatrix};$$

$$C7=R7*Q7=\begin{pmatrix} 5.0373 & -14.3641 & 40.6445 \\ 0.0052 & 3.0058 & -8.0012 \\ 0 & 0.0055 & 1.9569 \end{pmatrix};$$

Alla trentesima iterazione si ottiene la matrice C30 in cui gli elementi sottodiagonali hanno un valore inferiore a 10.e-5 e gli elementi in diagonale sono gli autovalori di C nonché radici del polinomio p(x).

$$C30=\begin{pmatrix} \mathbf{5.0000} & -14.1520 & 40.7296 \\ 0.0000 & \mathbf{3.0000} & -8.1130 \\ 0 & 0.0000 & \mathbf{2.0000} \end{pmatrix};$$

radici = diag(C30)={$\lambda_1$ , $\lambda_2, \lambda_3$}={5.0000, 3.0000, 2.0000} ;

Stesso risultato che si ottiene applicando la funzione *eig* di Matlab alla matrice C0.

*Prova su polinomio con radici complesse*

Dato il polinomio $p(x)= x^4-7x^3+21x^2-37x+30$ calcolata la matrice in forma compagna si procede col calcolo degli autovalori mediante fattorizzazione QR.

$$C0=\begin{pmatrix} 7 & -21 & 37 & -30 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix};$$

dopo 100 iterazioni la matrice C100 risulta:

$$C100=\begin{pmatrix} \mathbf{3.0000} & 5.4825 & -13.0948 & -47.3797 \\ 0.0000 & \mathbf{1.9229} & \mathbf{-4.7140} & -16.5948 \\ 0 & \mathbf{1.0292} & \mathbf{0.0771} & -0.5119 \\ 0 & 0 & 0.0000 & \mathbf{2.0000} \end{pmatrix};$$

dalla matrice C100 è possibile dedurre che vi sono due radici reali , $\lambda_1$ *e* $\lambda_4$ e una coppia di radici complesse e coniugate in corrispondenza del blocco 2x2. Gli elementi della sottoamatrice non si stabilizzeranno ma gli auto valori del blocco saranno sempre approssimabili a $\lambda$.

Risolvendo la sottomatrice

Polinomio caratteristico : $\lambda^2$-2$*$ $\lambda$+5 = 0 soluzione $\lambda = 1\pm$j2

Radici= {3.000, 1$\pm$j2, 2.000};

*Prova radici multiple*

Si considera il polinomio $p(x)= x^3-7x^2+15x-9$ dopo 100 iterazioni la matrice C110 risulta:

$$C100 = \begin{pmatrix} \mathbf{3.0296} & \mathbf{-8.3771} & 15.4499 \\ \mathbf{0.0001} & \mathbf{2.9704} & -5.3965 \\ 0 & 0.0000 & \mathbf{1.0000} \end{pmatrix};$$

La sottomatrice corrispondente avrà dimensione pari alla molteplicità della radice. Per un numero molto elevato di iterazioni si arriva ad una convergenza con un risultato non preciso. Dopo 3277 iterazioni gli elementi della sottodiagonale si possono considerare approssimabili a zero e la matrice C risulta:

$$C=\begin{pmatrix} \mathbf{3.0001} & -8.3772 & 15.4308 \\ 0.0000 & \mathbf{2.9999} & -5.4509 \\ 0 & 0.0000 & \mathbf{1.0000} \end{pmatrix}$$

Dato il numero di iterazioni e la non precisione dei risultati si risolve ,anche in questo caso, la sottomatrice ottenuta all'iterazione 100 ossia risolvendo la sottomatrice corrispondente alle radici multiple.

Polinomio caratteristico : $\lambda^2$-6 $\lambda$+9 = 0 soluzione $\lambda_{1,2}$= 3

Radici= {3.000}; m=2