Calcolo Numerico

A.A. 2019-2020

Ingegneria chimica

Equazioni non lineari

D. Vitulano

Esercizio

```
% sep_radici_tab
% script per la separazione delle radici dell'equazione non lineare
% f(x)=0 contenute nell'intervallo I = [a,b]

f = input('introduci la funzione f = ')
a = input('introduci l''estremo inferiore dell''intervallo a = ')
b = input('introduci l''estremo superiore dell''intervallo b = ')
np = input('numero di punti in cui suddividere l''intervallo np = ')
if isempty(np)
   np=10;
end
x=linspace(a,b,np);
y=f(x);
pos = find(abs(diff(sign(y))));
intervallo(1,:) = x(pos);
intervallo(2,:) = x(pos+1);
fprintf('l''estremo inferiore e'' %6.5f\nl''estremo  superiore e''
%6.5f\n', intervallo)
```

Esercizio

Dal Command Window

```
>> sep_radici_tab
introduci la funzione f = @(x)[x.^3-10*x.^2+5];
f =
    @(x)[x.^3-10*x.^2+5]
introduci l'estremo inferiore dell'intervallo a = 0.6
a =
   0.6000
introduci l'estremo superiore dell'intervallo b = 0.8
b =
   0.8000
numero di punti in cui suddividere l'intervallo np =
np =
     []
l'estremo inferiore e' 0.73333
l'estremo  superiore e' 0.75556
```

Esercizio

Dal Command Window

```
>> sep_radici_tab
introduci la funzione f = @(x)[x.^3-10*x.^2+5];
f =
    @(x)[x.^3-10*x.^2+5]
introduci l'estremo inferiore dell'intervallo a = 0.6
a =
   0.6000
introduci l'estremo superiore dell'intervallo b = 0.8
b =
   0.8000
numero di punti in cui suddividere l'intervallo np =
np =
    11
l'estremo inferiore e' 0.72000
l'estremo  superiore e' 0.74000
```

Bisezione

```
function [xn,err1,err2,iter] = bisezione_fun(f,a,b,eps)
% [xn,err1,err2,n_iter] = bisezione_fun(f,a,b,eps)
% cerca la radice della funzione f nell'intervallo [a,b] con precisione eps
utilizzando il doppio criterio di arresto
%
% INPUT
% f = espressione della funzione della quale si vuole cercare la radice
% a = estremo inferiore dell'intervallo in cui è stata isolata la radice
% b = estremo superiore dell'intervallo in cui è stata isolata la radice
% eps = limite superiore dell'errore da usare come criterio di arresto
%
% OUTPUT
% xn = approssimazione della radice
% err1 = |xn-x(n-1)| differenza in valore assoluto tra due approssimazioni
successive
% err2 = f(xn) valore della funzione nell'approssimazione xn
% iter = numero di iterazioni eseguite
```

Bisezione

```
format long;
%inizializzazione dei parametri
iter = 0; err1 = b-a; err2 = eps+1; x0=a;

% controllo esistenza
if f(a)*f(b)>0
    error('Attenzione! Non sono verificate le condizioni di
applicabilita'' del metodo di bisezione!!!')
elseif f(a)*f(b)==0
    if f(a)==0
        xn=a;
    else
        xn=b;
    end
    return
end
```

Bisezione

```
% calcolo successione
while (err1>eps) && (err2>eps)
    xn = (a+b)/2;
    if ( f(a)*f(xn) < 0 )
        b = xn;
    elseif ( f(xn)*f(b) < 0 )
        a = xn;
    end
    iter = iter+1;
    err1 = abs(xn-x0);
    err2 = abs(f(xn));
    x0 = xn;
end
```

Criterio di arresto a priori

1. Come si può modificare il programma se si vuole introdurre il criterio di arresto a priori?

   a)la variabile max_it va definita all'interno dello script nel modo seguente max_it = ceil(log2(b-a)-log2(eps))

   b)eps è la tolleranza richiesta e deve essere data come input

   Quindi:

max_it = ceil(log2(b-a)-log2(eps))

Soluzione

2) Come si può modificare lo script precedente se si vuole introdurre almeno un criterio di arresto a posteriori?

```
x0 = a;
for k = 1:max_it
    c = (a+b)/2;
    if f(c) == 0
        fprintf('la radice e'' %6.5f \n',c)
        break
    elseif abs(x0-c)<eps
        break
    end
    x0=c;
    if f(c)*f(a) > 0
        a = c;
    else
        b = c;
    end
end
```

E' necessario definire una variabile in cui si conserva il valore dell'elemento della successione calcolato all'iterazione precedente in modo da poter valutare la differenza tra due approssimazioni successive

Newton-Raphson

```
function [xn,n_iter,err] = newton_fun(f,df,x0,eps)
% [xn,n_iter,err] = newton_fun(f,df,x0,eps)
% approssima lo zero della funzione f usando il metodo di Newton con
% precisione eps e scegliendo x0 come punto iniziale.
% Il procedimento iterativo si interrompe quando la differenza tra due
% approssimazioni successive risulta in modulo <= alla precisione richiesta
% Nota:  [xn,n_iter,err] = newton_fun(f,df,x0,eps,maxiter)
%
% INPUT
% f = funzione di cui trovare lo zero
% df = derivata prima di f
% x0 = approssimazione iniziale
% eps = precisione richiesta alla approssimazione
%
% OUTPUT
% xn = approssimazione prodotta dal metodo di Newton
% n_iter = numero di iterazioni eseguite
% err = |xn-x(n-1)| valore assoluto della differenza tra due approssimazioni
successive
```

Newton-Raphson

```
format long;
%inizializzazione dei parametri
n_iter = 0; err = eps+1;

% iterazioni (si usa un solo criterio di arresto)
while (err>eps) % && (n_iter<maxiter) aggiungere se nella lista degli input si
aggiunge il numero massimo di iterazioni consentite
n_iter = n_iter+1;
xn = x0-f(x0)/df(x0);
err = abs(xn-x0);
x0 = xn;
end
```

Secanti

```
function [xn,n_iter,err] = secanti_fun(f,x0,x1,eps)
% approssima lo zero della funzione f usando il metodo delle secanti con
% precisione eps e scegliendo x0 e x1 come punti iniziali.
% Il procedimento iterativo si interrompe quando la differenza tra due
% approssimazioni successive risulta in modulo <= alla precisione richiesta
%
% INPUT
% f = funzione di cui trovare lo zero
% x0, x1 = approssimazioni iniziali
% eps = precisione richiesta alla approssimazione
%
% OUTPUT
% xn = approssimazione prodotta dal metodo delle secanti
% n_iter = numero di iterazioni eseguite
% err = |xn-x(n-1)| valore assoluto della differenza tra due approssimazioni
successive
```

Secanti

```
%inizializzazione dei parametri
n_iter = 0; err = eps+1;

% iterazioni
while (err>eps)
    xn = x1-f(x1) *(x1-x0)/(f(x1)-f(x0));
    n_iter = n_iter + 1;
    err = abs(xn-x1);
    x0 = x1;
    x1 = xn;
end
```

Esercizio

Problema: flusso di liquidi e gas attraverso i sistemi di raffreddamento

La resistenza al flusso in tali condotti è parametrizzata da un numero adimensionale, detto coefficiente di attrito. L'equazione di Colebrook definisce il coefficiente di attrito di Darcy f per flussi turbolenti attraverso la seguente equazione

$$\frac{1}{\sqrt{f}} + 2log_{10}\left(\frac{\varepsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}}\right) = 0,$$

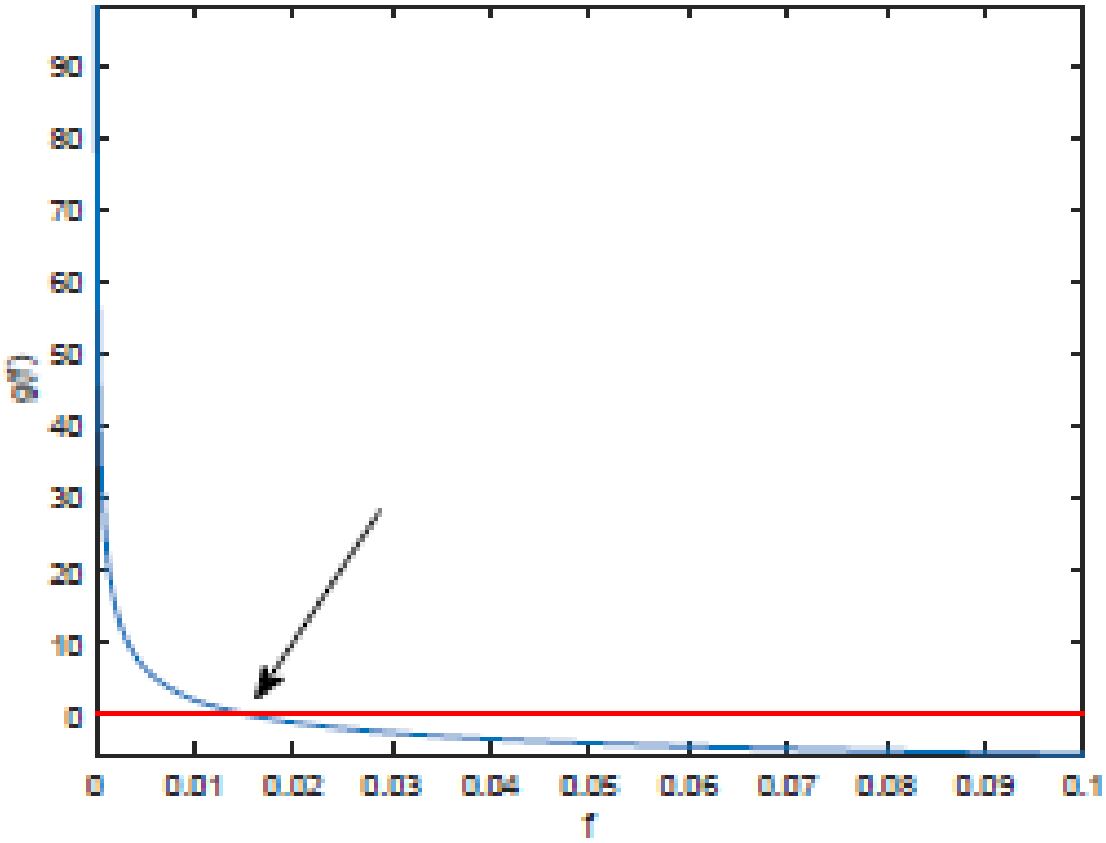
dove Re rappresenta il numero di Reynolds ($4000 < Re < 10^8$), D il diametro della condotta e ε la rugosità della superficie

Oss: f rappresenta un parametro adimensionale

Si vuole stimare f con precisione 10^{-8} quando $Re = 300000$ e $\frac{\varepsilon}{D} = 0.0001$

Soluzione: separazione grafica

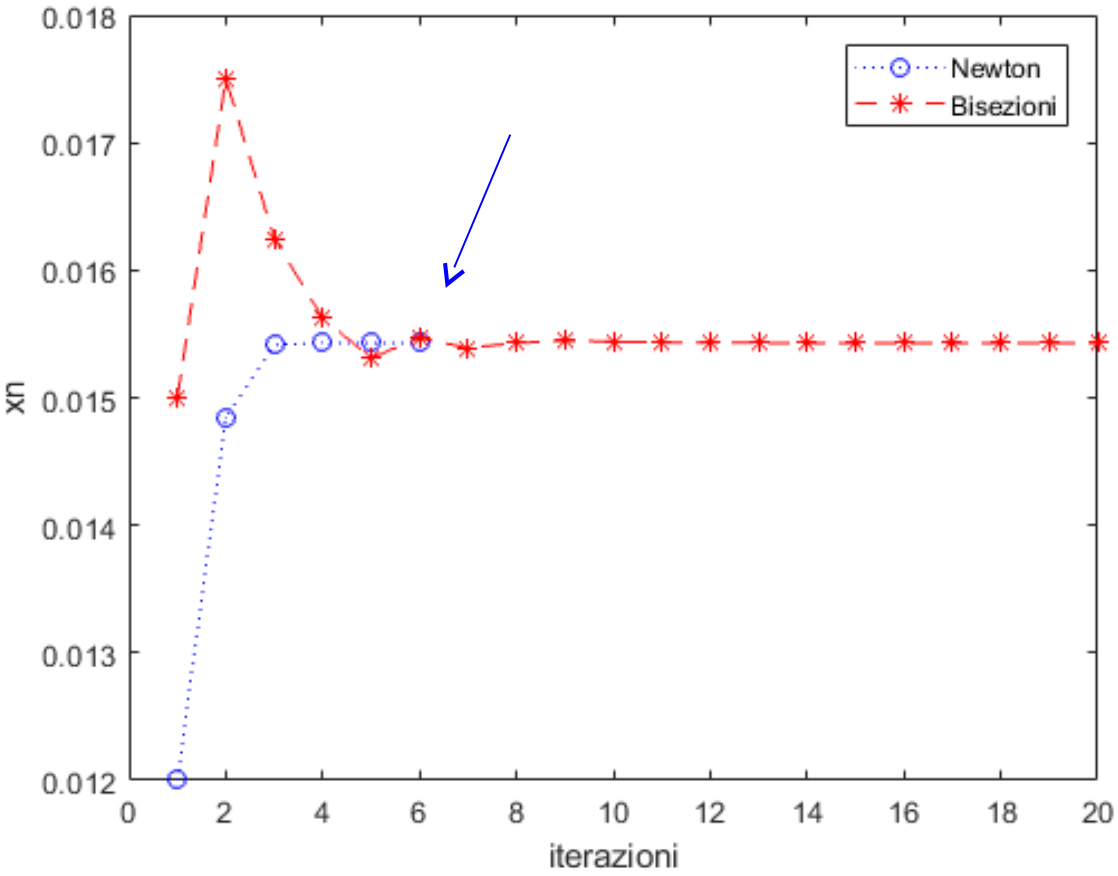
```
f = @(x)[1./sqrt(x) + 2*log10(0.0001/3.7 + 2.51./(300000*sqrt(x)))]
figure, fplot(f,[.0001,.1])
hold on, fplot(@(x)[0.*x],[.0001,.1])
```



La radice è contenuta nell'intervallo I =[0.01, 0.02]

Soluzione

numero di Reynolds Re = 300000
rapporto rugosità della superficie/diametro della condotta eD = 0.0001
estremo inferiore dell'intervallo a = 0.01
estremo superiore dell'intervallo b = 0.02
punto iniziale **x0 = 0.012**
precisione con cui si vuole produrre la soluzione, eps = 10^-8;



Soluzione metodo di Newton
xn = 0.015430606110171
N_iter = 6

Soluzione metodo di bisezione
xn = 0.015430612564087
N_iter = 20

Soluzione

$I = [a,b] = [0.01, 0.02]$

$f(a)f''(a) > 0$

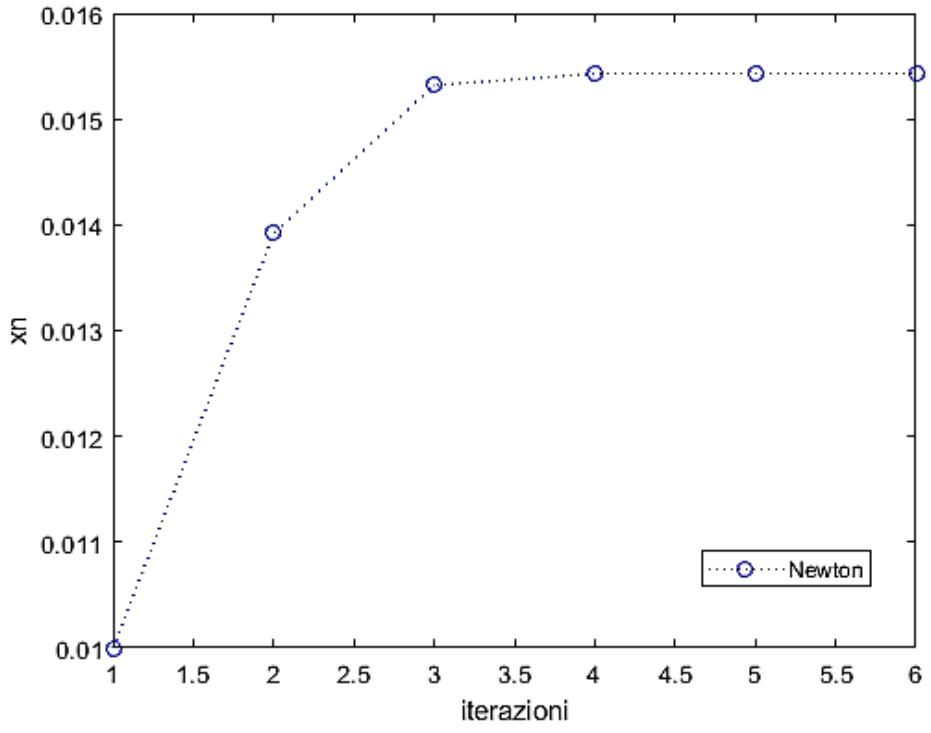
$f(b)f''(b) < 0$ ⇒ a=0.01 è estremo di Fourier

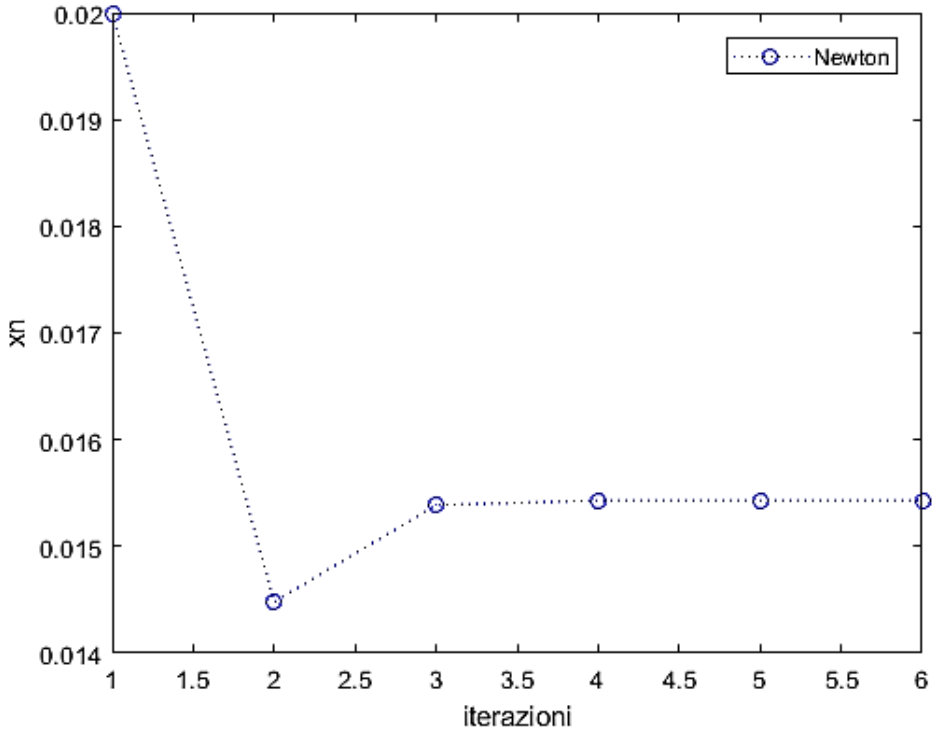
punto iniziale x0 = 0.01

Soluzione metodo di Newton
xn = 0.015430606110171
N_iter = 6

punto iniziale x0 =0.02

Soluzione metodo di Newton
xn = 0.015430606110171
N_iter = 6



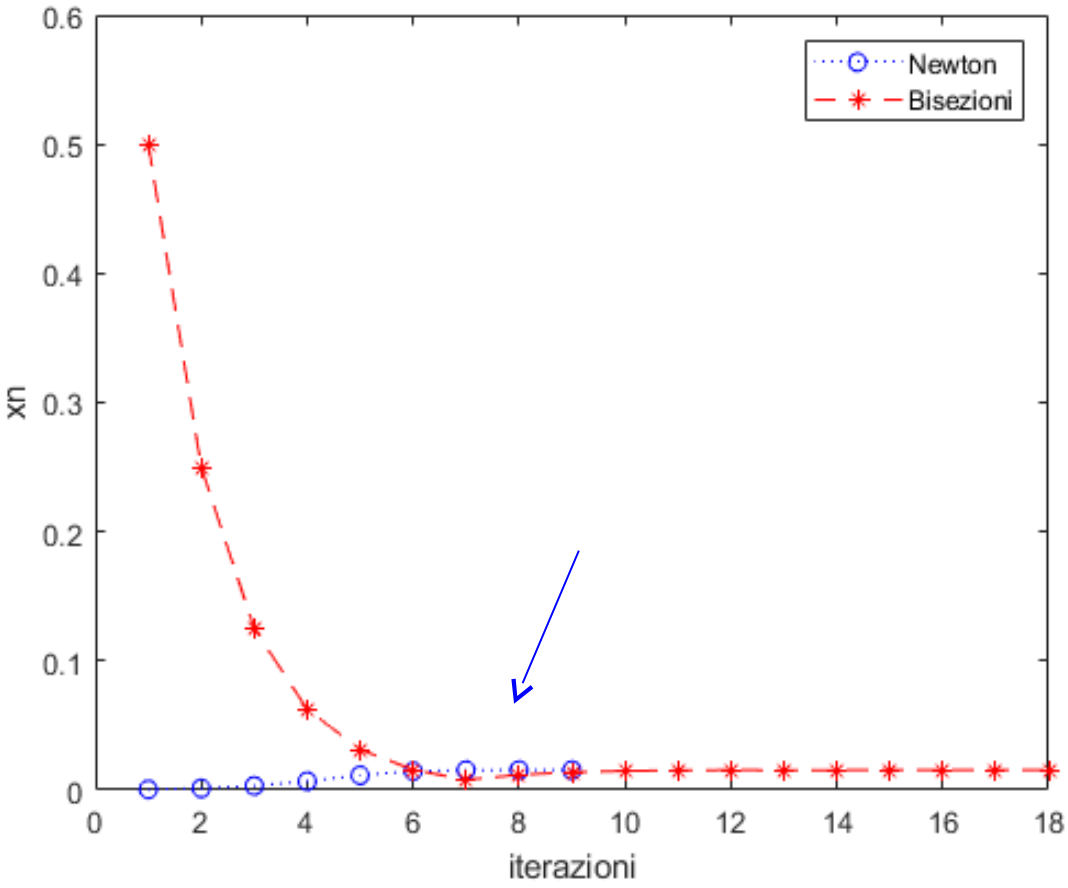


Soluzione

numero di Reynolds Re = 300000
rapporto rugosità della superficie/diametro della condotta eD = 0.0001
estremo inferiore dell'intervallo **a = 0.0001**
estremo superiore dell'intervallo **b = 1**
punto iniziale **x0 = 0.0005**
precisione con cui si vuole produrre la soluzione, **eps = 0.5*10^-5**;

Soluzione metodo di Newton
xn = 0.015430606110063
N_iter = 9

Soluzione metodo di bisezone
xn = 0.015429735183716
N_iter = 18

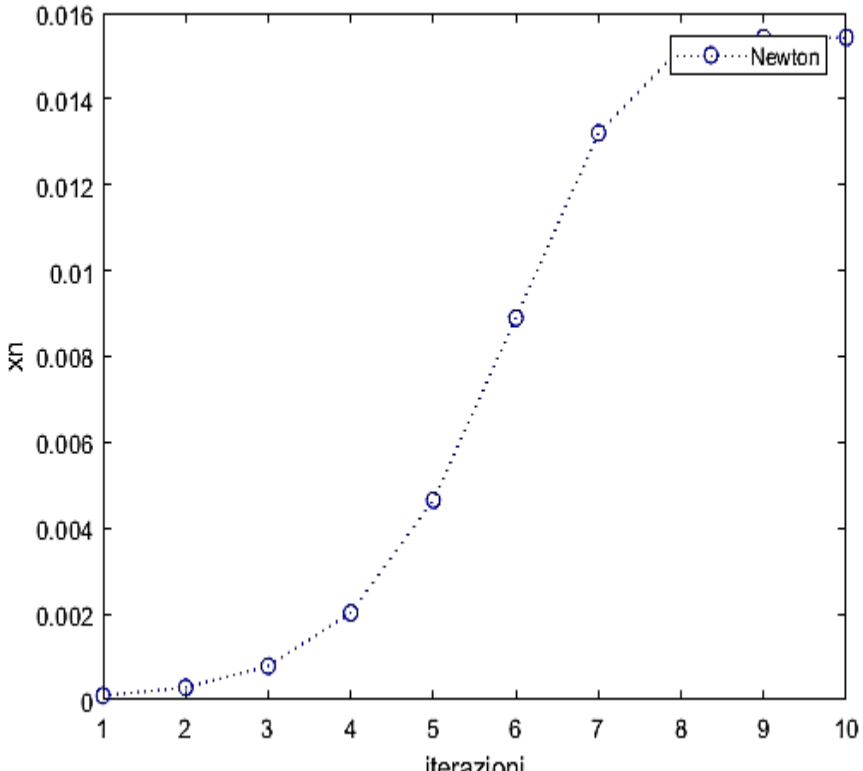


Soluzione

I = [a,b] = [0.0001,1]
f(a)f''(a) > 0
f(b)f''(b) < 0 ⇒ **a=0.0001 è estremo di Fourier**

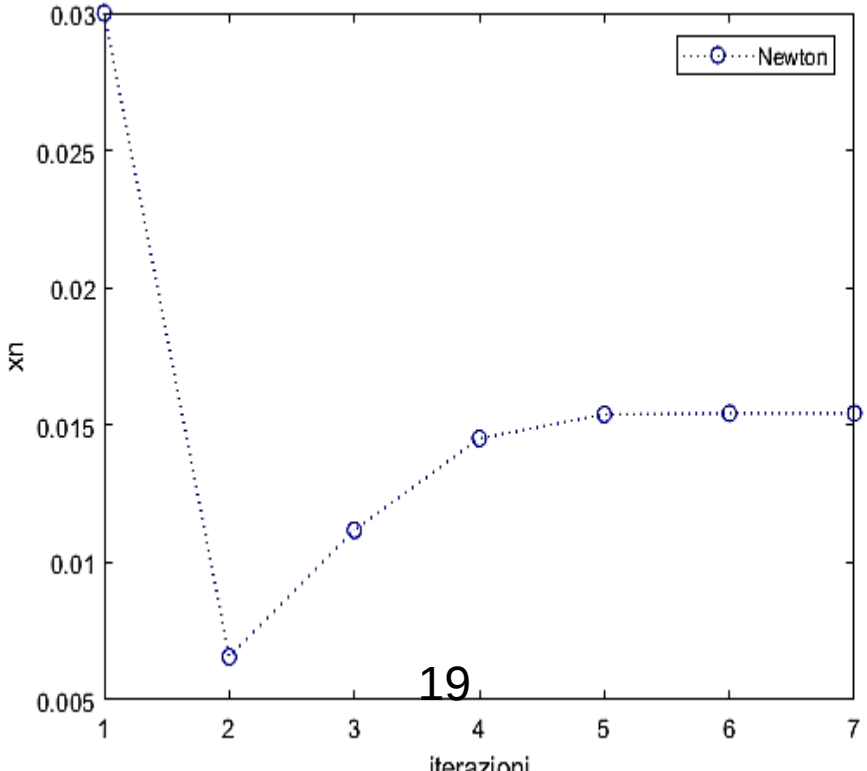
punto iniziale x0 = 0.0001

Soluzione metodo di Newton
xn = 0.015430605727842
N_iter = 10



punto iniziale x0 =0.03

Soluzione metodo di Newton
xn = 0.015430606109881
N_iter = 7



Cenni sul Calcolo Simbolico

Matlab esegue non solo operazioni numeriche ma anche manipolazioni e soluzioni di espressioni matematiche (in forma analitica) simboliche .

Usando il Symbolic Math Toolbox è possibile risolvere/calcolare:

equazioni algebriche e trascendenti, integrali, derivate, limiti, serie, ODE etc.

Cenni sul Calcolo Simbolico

L'uso del Symbolic Math Toolbox permette di usare funzioni simboliche che hanno lo stesso nome delle funzioni numeriche:

```
>> help sym/nomefunzione
```

Cenni sul Calcolo Simbolico

>> help diff

diff Difference and approximate derivative.
diff(X), for a vector X, is [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)].
diff(X), for a matrix X, is the matrix of row differences,
[X(2:n,:) - X(1:n-1,:)].
diff(X), for an N-D array X, is the difference along the first
non-singleton dimension of X.
diff(X,N) is the N-th order difference along the first non-singleton
dimension (denote it by DIM). If N >= size(X,DIM), diff takes
successive differences along the next non-singleton dimension.
diff(X,N,DIM) is the Nth difference function along dimension DIM.
If N >= size(X,DIM), diff returns an empty array.

Examples:

. . .

Cenni sul Calcolo Simbolico

```
>> help sym/diff
```

diff Differentiate.
diff(S) differentiates a symbolic expression S with respect to its free variable as determined by SYMVAR.
diff(S,'v') or diff(S,sym('v')) differentiates S with respect to v.
diff(S,n), for a positive integer n, differentiates S n times.
diff(S,'v',n) and diff(S,n,'v') are also acceptable.
diff(S,'v1','v2',...) or diff(S,sym('v1'),sym('v2'),...) differentiates S with respect to v1, v2, ...

Examples:

. . .

Cenni sul Calcolo Simbolico

Il Symbolic Math Toolbox permette di definire un oggetto simbolico.

(Struttura dati che memorizza una rappresentazione stringa del simbolo).

Per creare oggetti simbolici in MATLAB si utilizza la funzione sym. Per esempio:

```
>> x = sym('x')

x =

x
```

Cenni sul Calcolo Simbolico

```
>> syms x y real
>> f = log(x^2 + y^2)

f =
log(x^2+y^2)

>> g = sqrt(y/x)

g =
(y/x)^(1/2)

>> h=f+g

f =
 log(x^2 + y^2) + (y/x)^(1/2)
```

Cenni sul Calcolo Simbolico

collect(E)	raccoglie i coefficienti con la stessa potenza di x
expand(E)	applica regole algebriche per espandere l'espressione E
factor(E)	esprime E come prodotto di polinomi con coefficienti razionali
poly2sym(p)	converte i coefficienti del vettore p in un polinomio simbolico
sym2poly(E)	converte l'espressione E nel vettore di coefficienti
pretty(E)	visualizza l'espressione E in forma matematica
simple(E)	ricerca la forma dell'espressione E più corta in termini di numero di caratteri,utilizzando differenti semplificazioni algebriche
simplify(E)	semplifica l'espressione E
subs(E,old,new)	sostituisce *new* al posto di *old* nell'espressione E

Cenni sul Calcolo Simbolico

Esempio:

```
>> syms x y

>> expand((x+1)^3)

        x^3+3*x^2+3*x+1
```

.
.
.

Cenni sul Calcolo Simbolico

Esempio: ***funzione solve***

Risolve equazioni algebriche trascendenti:

```
>> solve('x+8=0')
ans =
-8

>> eq='exp(2*x)=54';

>> solve(eq)
ans =

log(54)/2
```

Cenni sul Calcolo Simbolico

diff(E)	Restituisce la derivata dell'espressione E rispetto alla variabile indipendente di default (x)
int(E)	Restituisce l'integrale dell'espressione E
limit(E)	Restituisce il valore del limite di E per x che tende a 0 (default)
symsum(E)	Restituisce la somma dell'espressione E rispetto alla sua variabile k da 0 a k-1
taylor(f,n,a)	Restituisce il polinomio di Maclaurin di f di ordine n-1, valutato nel punto x=a

Cenni sul Calcolo Simbolico

Esempi:

```
syms x
>> diff(sin(x)+exp(x))

ans =

cos(x) + exp(x)

>> diff(sin(x)*exp(x))

ans =

exp(x)*cos(x) + exp(x)*sin(x)
```

Cenni sul Calcolo Simbolico

Esempi:

```
>> syms x y
>> diff(sin(x)*exp(x*y),y)

ans =

x*exp(x*y)*sin(x)

>> diff(sin(x)*exp(x),2)

ans =

2*exp(x)*cos(x)
```

Cenni sul Calcolo Simbolico

Esempi:

```
>> syms x

>> limit(sin(x)/x)

ans =

1

.     .     .
```