

Equazioni non lineari

Gabriella Puppo

Equazioni non lineari

- Passare una function come argomento
- Metodo di bisezione
- Metodo di Newton
- Metodo delle secanti
- Funzione fzero

Passare una function come argomento

Molte funzioni “lavorano” su altre funzioni, predefinite dall’utente. Supponiamo di voler scrivere una function che stampa il grafico di una funzione sull’intervallo $[a,b]$.

La function richiesta deve avere in input il nome della function di cui vogliamo il grafico e l’intervallo $[a,b]$.

La sua intestazione sarà quindi:

```
function plot_funz(funz,a,b)
```

La difficoltà nasce dal fatto che per Matlab (prima della versione 6) il parametro “funz” deve essere una stringa di caratteri, contenente il nome della function.

Supponiamo quindi di aver creato un M-file f.m che contiene la function di cui vogliamo il grafico:

```
function f=f(x)
% Calcola la funzione exp(x)*cos(4*x)
f=exp(x).*cos(4*x);
```

Poiché il nome della function deve essere passato come stringa di caratteri, la chiamata a `plot_funz` per disegnare il grafico di $f(x)$ sull'intervallo $[0,5]$ sarà:

```
>> plot_funz('f',0,5)
```

Function feval

All'interno della function `plot_funz`, è necessario valutare la funzione passata come argomento sui valori `x` delle ascisse. Per far questo devo usare la function `feval`:

```
f=feval(funz,x);
```

Questa istruzione trasforma la stringa `funz`, associando a `funz` il file `funz.m`. Poi, esegue il comando `funz(x)`. Possiamo ora dare il listato della function `plot_funz`

Listato di plot_funz

```
function plot_funz(funz,a,b)
% PLOT_FUNZ(FUNZ,A,B): disegna il grafico della funzione
%     FUNZ sull'intervallo [a,b], usando 201 punti.
%     FUNZ deve essere una stringa di caratteri
h=(b-a)/200;
x=a:h:b;
f=feval(funz,x); %Valuta funz nei valori x
plot(x,f)
```

Inline functions

Per costruire funzioni semplici, posso usare l'istruzione `inline`, che genera funzioni di una riga. Consideriamo il file `f.m`:

```
function f=f(x)
% Calcola la funzione exp(x)*cos(4*x)
f=exp(x).*cos(4*x);
```

Un modo più semplice di generare la stessa funzione è:

```
>> f=inline('exp(x).*cos(4*x)')
f =
    Inline function:
    f(x) = exp(x).*cos(4*x)
```

Attenzione!

L'oggetto f creato dall'istruzione `inline` è una stringa di caratteri. Quindi se voglio disegnare il grafico di f usando la function `plot_funz`, ho due possibilità:

- 1) -creo un file `f.m`, che contiene la funzione desiderata.
- chiamo `plot_funz` con il comando:

```
>> plot_funz('f',0,5)
```

- 2) -creo f come inline function e poi chiamo `plot_funz`:

```
>> f=inline('exp(x).*cos(4*x)');  
>> plot_funz(f,0,5)
```

Metodo di bisezione

Per implementare il metodo di bisezione al problema $f(x)=0$, devo scrivere una function che:

- *Accetti in input il nome della funzione f e l'intervallo $[a,b]$ in cui f cambia segno.*
- *Dia in output la soluzione x ed il residuo $f(x)$*
- *Contenga una iterazione in cui ad ogni passo si dimezza l'intervallo sul quale si cerca la soluzione*
- *Contenga un opportuno test di arresto (preferibilmente, lasciando la possibilità di impostare in input la tolleranza relativa)*

Function bisez.m

Riporto il listato della function bisez.m, aggiungendo alcuni commenti

```
function [x,res]=bisez(funz,a0,b0,toll)
%BISEZ(FUNZ,A0,B0,TOLL) Calcola la radice X di FUNZ(X)=0
%   nell'intervallo [A0,B0] con il metodo di bisezione.
%   FUNZ deve essere fornita come stringa di caratteri
%   oppure puo' essere il nome di una inline function
%   Come parametro di convergenza usa TOLL
% Sintassi: [X,RES]=BISEZ(FUNZ,A0,B0,TOLL)
%   RES=FUNZ(X) è il residuo.
%   (TOLL=EPS e' il valore di default)
%   Ritorna con un messaggio di errore se FUNZ(A0)*FUNZ(B0) >0
```

continua...

Fissa i valori iniziali e controlla se è presente uno zero in $[a,b]$:

```
if nargin < 4, toll=eps; end
fa0 = feval(funz,a0); fb0 = feval(funz,b0);
fa0*fb0
if fa0*fb0 >= 0
    fprintf('f non cambia segno in [a,b] \n')
    return
end
test = (abs( fa0)+abs(fb0 ) ) * toll;
nmax = 100; % numero massimo di iterazioni
a=a0; b=b0;
fprintf(' k      a_k      b_k      |f(x_k)| \n')
```

Iterazione principale:

```
for iter=1:nmax
    c=(a+b)/2.d0;
    fa = feval(funz,a);
    fc = feval(funz,c);
    if fa*fc < 0
        b=c;
    else
        a=c;
    end
    fprintf('%2.0f: %20.16e %20.16e %9.2e \n',iter,a,b,abs(fc))
    if abs( fc ) <= test
        x = c;
        res = fc;
        return
    end
end
display('BISEZ richiede più di 100 iterazioni')
x=c; res=fc;
```

Function fprintf

La function fprintf serve a stampare output formattato (per esempio, per una tabella).

Il formato è il seguente:

```
fprintf('istruzioni di formato ',variabili da stampare)
```

Esempio:

```
fprintf('%10.3f %20.16e \n',a,b)
```

Stampa a con formato floating di 10 caratteri, di cui 3 dopo la virgola
Stampa b con formato esponenziale di 20 caratteri, di cui 16 dopo la virgola. Poi va a capo (\n).

Esempio

Applico il metodo di bisezione alla ricerca del valore di $\sqrt{2}$.

Devo trovare il punto in cui $f(x)=x^2 - 2 = 0$.

Il valore cercato si trova nell'intervallo $[0,2]$.

Quindi devo dare i comandi:

```
>> f=inline('x.^2-2');
```

```
>> [x,res]=bisez(f,0,2)
```

In output, ottengo:

k	a_k	b_k	f(x_k)
1:	1.0000000000000000e+000	2.0000000000000000e+000	1.00e+000
2:	1.0000000000000000e+000	1.5000000000000000e+000	2.50e-001
3:	1.2500000000000000e+000	1.5000000000000000e+000	4.38e-001
4:	1.3750000000000000e+000	1.5000000000000000e+000	1.09e-001
5:	1.3750000000000000e+000	1.4375000000000000e+000	6.64e-002
6:	1.4062500000000000e+000	1.4375000000000000e+000	2.25e-002
7:	1.4062500000000000e+000	1.4218750000000000e+000	2.17e-002

.....

50:	1.4142135623730940e+000	1.4142135623730958e+000	2.89e-015
51:	1.4142135623730949e+000	1.4142135623730958e+000	4.44e-016

x =

1.4142

res =

-4.4409e-016

Metodo di Newton

Per implementare il metodo di Newton al problema $f(x)=0$, devo scrivere una function che:

- *Accetti in input il nome della funzione f , il nome della funzione f' e l'intervallo $[a,b]$ in cui f cambia segno.*
- *Dia in output la soluzione x ed il residuo $f(x)$*
- *Contenga una iterazione in cui ad ogni passo si applica il metodo di Newton*
- *Contenga un opportuno test di arresto (preferibilmente, lasciando la possibilità di impostare in input la tolleranza relativa)*

Function newton.m

Riporto il listato della function newton.m, aggiungendo alcuni commenti

```
function [x,res]=newton(funz,fder,a0,b0,toll)
%NEWTON(FUNZ,FDER,A0,B0,TOLL) Calcola la radice X di FUNZ(X)=0
% nell'intervallo [A0,B0] con il metodo di Newton.
% FUNZ e FDER devono essere fornite come stringa di caratteri
% oppure possono essere il nome di una inline function
% FDER e' la derivata di FUNZ.
% Sintassi: [X,RES]=NEWTON(FUNZ,FDER,A0,B0,TOLL)
% RES=FUNZ(X) è il residuo.
% (TOLL=EPS e' il valore di default)
% Ritorna con un messaggio di errore se FUNZ(A0)*FUNZ(B0) >0
% Come parametro di convergenza usa TOLL
```

continua...

Fissa i valori iniziali e controlla se è presente uno zero in $[a,b]$:

```
if nargin < 5, toll=eps; end
fa0 = feval(funz,a0); fb0 = feval(funz,b0);
fa0*fb0
if fa0*fb0 >= 0
    fprintf('f non cambia segno in [a,b] \n')
    return
end
test = ( abs(fa0)+abs(fb0) ) * toll;
nmax = 100;
```

Iterazione principale:

```
xk = (a0+b0)/2;
fprintf(' k      x_k      |f(x_k)| \n')
for iter=1:nmax
    fxk = feval(funz,xk);
    fprintf('%2.0f:  %20.16e %9.2e \n',iter,xk,abs(fxk))
    if abs( fxk ) <= test
        x = xk;
        res = fxk;
    return
end
fpxk = feval(fder,xk);
xk = xk - fxk/fpxk;
end
display('NEWTON richiede più di 100 iterazioni')
x=xk; res=fxk;
```

Esempio

Applico il metodo di Newton alla ricerca del valore di $\sqrt{2}$.
Devo trovare il punto in cui $f(x)=x^2 - 2 = 0$.
Il valore cercato si trova nell'intervallo $[0,2]$.
Quindi devo dare i comandi:

```
>> f=inline('x.^2-2');  
>> fp=inline('2*x');  
>> [x,res]=newton(f,fp,0,2)
```

In output ottengo:

k	x_k	f(x_k)
1:	1.00000000000000000000e+000	1.00e+000
2:	1.50000000000000000000e+000	2.50e-001
3:	1.4166666666666666667e+000	6.94e-003
4:	1.4142156862745099e+000	6.01e-006
5:	1.4142135623746899e+000	4.51e-012
6:	1.4142135623730951e+000	4.44e-016

x =

1.4142

res =

4.4409e-016

N.B. Il numero di iterazioni è diminuito moltissimo!

Metodo delle secanti

Per implementare il metodo delle secanti al problema $f(x)=0$, devo scrivere una function che:

- *Accetti in input il nome della funzione f e l'intervallo $[a,b]$ in cui f cambia segno.*
- *Dia in output la soluzione x ed il residuo $f(x)$*
- *Contenga una iterazione in cui ad ogni passo si applica il metodo delle secanti*
- *Contenga un opportuno test di arresto (preferibilmente, lasciando la possibilità di impostare in input la tolleranza relativa)*

Function secanti.m

Riporto il listato della function secanti.m, aggiungendo alcuni commenti

```
function [x,res]=secanti(funz,a0,b0,toll)
%SECANTI Calcola la radice X di FUNZ(X)=0 con il metodo delle secanti
%   SECANTI(FUNZ,A0,B0,TOLL): A0 e B0 definiscono l'intervallo iniziale
%   FUNZ deve essere fornita come stringa di caratteri
%   oppure puo' essere il nome di una inline function
% Sintassi: [X,RES]=SECANTI(FUNZ,A0,B0,TOLL)
%   RES=FUNZ(X) è il residuo.
%   Ritorna con un messaggio di errore se FUNZ(A0)*FUNZ(B0) >0
%   Come parametro di convergenza usa TOLL (TOLL=EPS e' il valore
%   di default)
```

continua...

Fissa i valori iniziali e controlla se è presente uno zero in $[a,b]$:

```
if nargin < 4, toll=eps; end
fa0 = feval(funz,a0); fb0 = feval(funz,b0);
if fa0*fb0 >= 0
    fprintf('f non cambia segno in [a,b] \n')
    return
end
test = ( abs(fa0)+abs(fb0) ) * toll;
nmax = 100;
a=a0; b=b0;
fprintf(' k          x_k          |f(x_k)| \n')
```

Iterazione principale:

```
for iter=1:nmax
    fa = feval(funz,a);
    fb = feval(funz,b);
    if abs(fa) <= test
        x = a; res=abs(fa);
        return
    end
    den = fa-fb;
    if abs(den) > 10e-16    %evita di dividere per 0
        xnew=(fa*b-fb*a)/den;
    else
        xnew = (a+b)/2;
    end
    fprintf('%02.0f:  %20.16e  %09.2e \n',iter,xnew,abs(feval(funz,xnew)))
    b=a;
    a = xnew;
end
display('SECANTI richiede più di 100 iterazioni')
x=xnew; res=abs(feval(funz,xnew));
```

Esempio

Applico il metodo delle secanti alla ricerca del valore di $\sqrt{2}$.

Devo trovare il punto in cui $f(x)=x^2 - 2 = 0$.

Il valore cercato si trova nell'intervallo $[0,2]$.

Quindi devo dare i comandi:

```
>> f=inline('x.^2-2');  
>> [x,res]=secanti(f,0,2)
```

In output ottengo

k	x_k	f(x_k)
1:	1.00000000000000000000e+000	1.00e+000
2:	2.00000000000000000000e+000	2.00e+000
3:	1.33333333333333333333e+000	2.22e-001
4:	1.39999999999999999999e+000	4.00e-002
5:	1.4146341463414633e+000	1.19e-003
6:	1.4142114384748701e+000	6.01e-006
7:	1.4142135620573204e+000	8.93e-010
8:	1.4142135623730951e+000	4.44e-016

x =
1.4142

res =
4.4409e-016

N.B. Il numero di iterazioni è leggermente superiore al caso di Newton

Function fzero

Matlab calcola la soluzione del problema $f(x)=0$ con la function *fzero*.

In *fzero* sono accoppiati un metodo a convergenza veloce (interpolazione quadratica inversa) insieme ad un metodo tipo bisezione, per determinare un intervallo in cui f cambi segno, e per evitare che il metodo diverga.

La chiamata più semplice è:

```
>> x=fzero(funz,x0)
```

Qui `funz` è una stringa di caratteri che contiene il nome della funzione di cui si sta cercando uno zero, e `x0` è un valore di partenza, dal quale cominciare la ricerca.

Per conoscere il residuo, devo:

```
>> [x,res]=fzero(funz,x0)
```

L'esito della ricerca (e la velocità di convergenza) dipendono fortemente dalla scelta di `x0`. Provare:

```
>> [x,res]=fzero('x.^2-2',24)
```

```
>> [x,res]=fzero('x.^2-2',0)
```

```
>> [x,res]=fzero('x.^2-2',1)
```

Si ottiene un controllo maggiore su fzero, se x0 è assegnato come vettore a due componenti (p.e. x0=[a,b]). In questo caso fzero limita la ricerca all'interno di [a,b]:

```
>> [x,res]=fzero('x.^2-2',[0,2])
```

```
x =
```

```
1.4142
```

```
res =
```

```
-4.4409e-016
```

```
>> [x,res]=fzero('x.^2-2',[-2,0])
```

```
x =
```

```
-1.4142
```

```
res =
```

```
4.4409e-016
```

Per avere un controllo maggiore su fzero e sapere che cosa sta facendo, dobbiamo modificare le opzioni disponibili.

Questo si ottiene con la function optimset:

```
[x,res]=fzero('x.^2-2',0,optimset('Display','iter'))
```

Con questa chiamata, Matlab stampa tutti i tentativi che prova per trovare la soluzione x.

Provare anche con:

```
>>[x,res]=fzero('x.^2-2',24,optimset('Display','iter'))
```

```
>> [x,res]=fzero('x.^2-2',[0,2],optimset('Display','iter'))
```

Per imporre una tolleranza relativa uguale (per esempio) a 0.001:

```
>> [x,res]=fzero('x.^2-2',[0,2],optimset('TolX',1e-3))
```

Esercizio

Calcolare gli zeri della funzione:

$$f(x) = \exp(x) * \cos(4x)$$

nell'intervallo $[0,3]$, usando i metodi studiati

- Costruire una function o una inline function che corrisponda a $f(x)$;
- Fare un grafico di $f(x)$, in modo da individuare gli intervalli che contengono un unico zero;
- Applicare le functions di ricerca di zeri ad ognuno degli intervalli individuati

Esercizio

Calcolare i punti di intersezione della funzione:

$$f(x) = 5(x + 3/5) (x + 1/10) (x - 1/2)$$

con la circonferenza unitaria, usando i metodi studiati

- Fare una figura contenente il grafico di f ed il grafico della circonferenza unitaria;
- Trasformare il problema di intersezione in un problema di ricerca di zeri per una funzione $F(X)$ e individuare gli intervalli che contengono una sola soluzione;
- Applicare le funzioni di ricerca di zeri al problema $F(X)=0$, sugli intervalli individuati.