

Equazioni nonlineari in Matlab

Alvise Sommariva

Università degli Studi di Padova

21 aprile 2022

Si supponga di dover risolvere l'equazione $f(x) = 0$ con $f : [a, b] \subset \mathbb{R} \rightarrow \mathbb{R}$ continua.

- Il **metodo di bisezione** (cf. [5]) genera una successione di intervalli (a_k, b_k) con

- $f(a_k) \cdot f(b_k) < 0$,
- $[a_k, b_k] \subset [a_{k-1}, b_{k-1}]$,
- $|b_k - a_k| = \frac{1}{2} |b_{k-1} - a_{k-1}|$.

- Quale criterio di arresto utilizziamo il **residuo pesato**.

Siano $a < b$ e $c = (a + b)/2$. Diciamo **residuo pesato** $|f(c) \cdot w|$ con

$$w := \left(\frac{f(b) - f(a)}{b - a} \right)^{-1}.$$

Fissata una tolleranza `tol1`, concluderemo le iterazioni se $f(c) = 0$ oppure $|w \cdot f(c)| \leq \text{tol1}$.

Si può vedere sperimentalmente che per questo metodo è più adatto del classico test sul residuo, ovvero $|f(c)| \leq \epsilon$, qualora le funzioni siano molto **piatte** o molto **ripide**.

Implementiamo tale metodo mediante la routine Matlab `bisezione.m`.

```
function [aa,bb,wres,flag]=bisezione(f,a,b,toll,maxit)

% Algoritmo di bisezione, con criterio di arresto sul residuo pesato e
% ampiezza dell'intervallo.
%
% Dati di ingresso:
% f: funzione (inline function)
% a: estremo sinistro
% b: estremo destro
% toll: tolleranza richiesta per il test del residuo pesato
% maxit: massimo indice dell'iterata permesso
%
% Dati di uscita:
% aa: sequenza degli estremi di sinistra degli intervalli [a_k,b_k],
%     immagazzinata in vettore colonna;
% bb: sequenza degli estremi di destra degli intervalli [a_k,b_k],
%     immagazzinata in vettore colonna;
% wres: sequenza dei residui pesati, immagazzinata in vettore colonna;
% flag: 0 processo terminato correttamente,
%       1 processo non terminato correttamente.
```

```

if b < a, s=b; b=a; a=s; end % Aggiusta errori utente.
flag=0; fa=feval(f,a); fb=feval(f,b); aa=[a]; bb=[b]; wres=[];

% hp. bisezione non soddisfatte (uscita con errore).
if fa*fb > 0, flag=1; return; end
% zero all'estremo iniziale "a" (uscita con successo).
if fa == 0, aa=[a]; bb=[a]; return; end % a sol.
% zero all'estremo iniziale "b" (uscita con successo).
if fb == 0, aa=[b]; bb=[b]; return; end % b sol.

for k=1:maxit % iterazioni bisezione
    c=(a+b)/2; fc=feval(f,c); % punto medio di [a_k,b_k]
    w=(b-a)/(fb-fa); % peso "w".
    wres(k)=abs(fc*w); % residuo pesato.
    % uscita per raggiungimento risultato.
    if (wres(k)<toll) |(fc==0)
        aa(k+1)=c; bb(k+1)=c; return;% OK exit.
    end
    % determinazione intervallo [a_{k+1},b_{k+1}]
    if fc*fa > 0 % "c" sostituisce "a"
        aa(k+1)=c; bb(k+1)=b; a=c; fa=fc; % aggiorna aa,bb,a,fa
    else % "c" sostituisce "b"
        aa(k+1)=a; bb(k+1)=c; b=c; fb=fc; % aggiorna aa,bb,a,fa
    end
end

flag=1; % raggiunto questo punto, si sono fatte troppe iterazioni

```

Nella routine, abbiamo salvato nei vettori `aa`, `bb`, tutti gli intervalli utili per bisezione. Se la loro lunghezza è m , allora l'ultimo intervallo analizzato è $[aa_m, bb_m]$.

Nel codice,

- abbiamo commentato adeguatamente le variabili di input e output, utile per chi utilizza il software senza costringerlo a leggere e capire il codice;
- abbiamo *verificato* che fosse $a < b$, e in caso contrario, invertito i valori delle variabili a e b ;
- se $a < b$, abbiamo *verificato* che effettivamente $f(a) \cdot f(b) < 0$, e se così non è stato, posto la variabile `flag` uguale a 1, in quanto il codice non è terminato correttamente e siamo usciti dalla routine mediante `return`;
- se $f(a) = 0$ allora a è uno zero x^* e quindi l'intervallo finale $[a, a]$ contiene un tale x^* ; di seguito si esce per `return`;
- se $f(b) = 0$ allora b è uno zero x^* e quindi l'intervallo finale $[b, b]$ contiene un tale x^* ; di seguito si esce per `return`;
- se così non è stato, abbiamo che certamente $f(a) \cdot f(b) < 0$ ed entriamo nel ciclo `for`, da eseguire al più `maxit` volte;

- alla k -sima iterazione abbiamo **calcolato il punto medio** c_k di $[a_k, b_k]$, lo abbiamo salvato in c e determinato la variabile w e valutato il residuo pesato $wres(k)$;
- di seguito abbiamo distinto due casi:
 - caso 1 se un tale c ha **residuo pesato sotto la tolleranza** $tol1$ allora l'intervallo $[c, c]$ contiene il valore desiderato, e quindi dopo aver modificato le variabili aa , bb siamo usciti con **return**;
 - caso 2 **altrimenti** abbiamo aggiornato gli intervalli cosicchè l'ultima componente a_{k+1} di a , b_{k+1} di b siano tali che $[a_{k+1}, b_{k+1}]$ contiene uno zero x^* di f , in quanto $f(a_k)f(b_k) < 0$ con $f \in C([a_k, b_k])$.
- osserviamo che se il codice esce per **return** all'interno del ciclo **for**, allora $flag$ è pari a 0, mentre se fa un numero di iterazioni maggiori di $maxit$, allora dopo esser uscito dal ciclo **for**, pone $flag$ uguale a 1;
- se bisezione termina correttamente, l'approssimazione è immagazzinata nell'ultima componente del vettore aa cioè in $aa(end)$ come pure come nell'ultima componente del vettore bb cioè in $bb(end)$;
- si noti che qualsiasi sia il motivo per cui si esce dal codice, le variabili di output sono comunque fornite.

Introduciamo una demo di bisezione in [demo_bisezione.m](#).

```
function demo_bisezione

% default
toll=10^(-6); maxit=1000; esempio=1;
% esempi.
switch esempio
    case 1 % funzione piatta
        f=inline('exp(x)-2+x');
        a=0; b=1; sol=0.4428544010023885;
    case 2
        f=inline('sin(x)-x');
        a=-2; b=3; sol=0;
end
% bisezione
[aa,bb,wr,ko]=bisezione(f,a,b,toll,maxit);
% statistiche
fprintf('\n \t soluzione : %1.15e',aa(end));
fprintf('\n \t tolleranza: %1.15e',toll);
fprintf('\n \t numero it.: %4d',length(aa));

if ko == 0
    fprintf('\n \t La procedura e'' terminata correttamente');
else
    fprintf('\n \t La procedura non e'' terminata correttamente');
end
fprintf('\n \n');
```

```
% ----- plot -----
indici=(1:length(wr))';
% grafico risultati
clf;
semilogy(indici,wr);
hold on;
title('Residuo pesato bisezione'); % titolo
xlabel('Indice'); % etichetta asse x
ylabel('Residuo pesato'); % etichetta asse y
% nome del file da salvare che vari con l'esempio,
% ottenuto concatenando 3 stringhe
% salva figura come pdf.
print('bisezione_esempio.pdf','-dpdf');
hold off;
% ----- salvataggio risultati su file -----
% creazione del file con facolta' di scrittura.
fid=fopen('bisezione_esempio.txt','rw');
% dati immagazzinati nella matrice A (si immagazzinino come vettori riga,
% ma bisogna ricordare che "aa", "bb" sono colonna.
A=[1:length(aa); aa'; bb'];
% scrittura dei dati su file.
fprintf(fid,'\n %3.0f  %1.15e %1.15e',A);
% chiusura file
fclose(fid);
```


In particolare,

- 1. se **esempio** vale 1 si cerca di approssimare lo zero $x^* \approx 0.4428544010023885$ della funzione

$$f(x) = \exp(x) - 2 + x,$$

avendo quale a il valore 0 e quale b il valore 1 (si noti che $f(0) = -1$, $f(1) = \exp(1) - 2 + 1 \approx 2.7 - 2 + 1 = 1.7 > 0$),

- 2. se **esempio** vale 2 si cerca di approssimare lo zero $x^* = 0$ della funzione

$$f(x) = \sin(x) - x,$$

avendo quale a il valore -2 e quale b il valore 3 (si noti che $f(-2) = \sin(-2) - (-2) > 0$, $f(3) = \sin(3) - 3 < 0$);

- se bisezione termina correttamente, l'approssimazione è immagazzinata nell'ultima componente del vettore aa cioè in aa(end);
- di seguito dopo aver scritto su monitor alcune statistiche, il codice elimina possibili plot precedenti con il comando `clf`, ed in scala semilogaritmica disegna le coppie (k, wr_k) dove wr_k è il residuo pesato alla k -sima iterazione.

L'ultima parte del codice tratta la descrizione grafica dei risultati e il salvataggio su file dei vari intervalli analizzati.

Per quanto riguarda il plot in scala semilogaritmica,

- Abbiamo salvato un vettore di indici pari alla lunghezza del vettore `wr`.
- Disegnato il grafico in scala semilogaritmica del valore del residuo pesato al variare degli indici.
- Descritto il significato del grafico, delle ascisse (con il comando `xlabel`), delle ordinate (con il comando `ylabel`).
- Abbiamo salvato la figura in `'bisezione_esempio.pdf'`.

Vediamo di seguito i risultati.

1. Se **esempio** vale 1 abbiamo

```
>> demo_bisezione

soluzione : 4.428539276123047e-01
tolleranza: 1.000000000000000e-06
numero it.: 20
La procedura e' terminata correttamente

>>
```

Sul file **bisezione_esempio.txt** viene registrato:

```
1  0.000000000000000e+00 1.000000000000000e+00
2  0.000000000000000e+00 5.000000000000000e-01
3  2.500000000000000e-01 5.000000000000000e-01
4  3.750000000000000e-01 5.000000000000000e-01
5  4.375000000000000e-01 5.000000000000000e-01
6  4.375000000000000e-01 4.687500000000000e-01
7  4.375000000000000e-01 4.531250000000000e-01
8  4.375000000000000e-01 4.453125000000000e-01
... ..
18 4.428482055664062e-01 4.428558349609375e-01
19 4.428520202636719e-01 4.428558349609375e-01
20 4.428539276123047e-01 4.428539276123047e-01
```

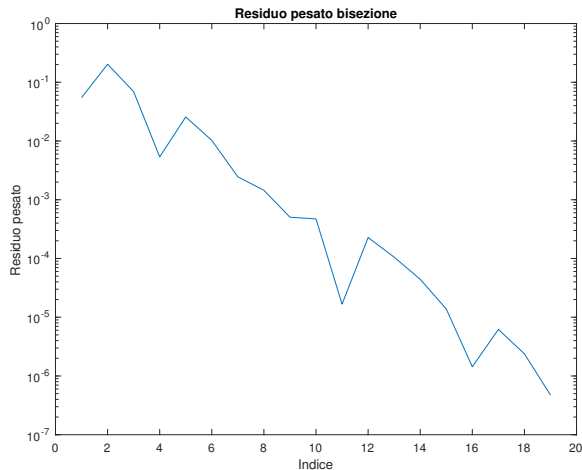


Figura: Residuo pesato delle iterazioni fornite dal metodo di bisezione, relativamente al metodo di bisezione per lo studio dello zero di $f(x) = \exp(x) - 2 + x$, con $a = 0$, $b = 1$.

2. Se **esempio** vale 2 (basta cambiare manualmente il valore della variabile nella function) abbiamo

```
>> demo_bisezione

soluzione : 7.629394531250000e-06
tolleranza: 1.000000000000000e-06
numero it.: 18
La procedura e' terminata correttamente

>>
```

Sul file **bisezione_esempio.txt** viene registrato:

```
1 -2.000000000000000e+00 3.000000000000000e+00
2 -2.000000000000000e+00 5.000000000000000e-01
3 -7.500000000000000e-01 5.000000000000000e-01
4 -1.250000000000000e-01 5.000000000000000e-01
5 -1.250000000000000e-01 1.875000000000000e-01
6 -1.250000000000000e-01 3.125000000000000e-02
7 -4.687500000000000e-02 3.125000000000000e-02
8 -7.812500000000000e-03 3.125000000000000e-02

... ..
16 -3.051757812500000e-05 1.220703125000000e-04
17 -3.051757812500000e-05 4.577636718750000e-05
18 7.629394531250000e-06 7.629394531250000e-06
```

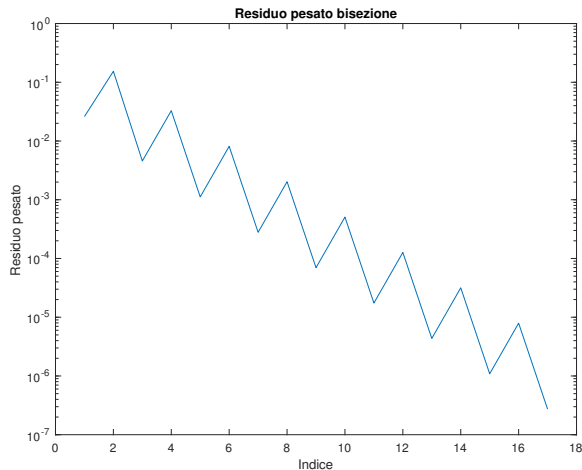


Figura: Residuo pesato delle iterazioni fornite dal metodo di bisezione per lo studio dello zero di $f(x) = \sin(x) - x$, con $a = -2$, $b = 3$.

Esercizio (Facoltativo)

- 1 Utilizzando il programma `bisezione.m`, si scriva la routine `bisezione2.m` cosicchè termini le sue iterazioni qualora l'ampiezza dell'ultimo intervallo analizzato sia inferiore a una tolleranza `tollintv` o il residuo pesato sia inferiore a `toll`.
Osservazione: si aggiunga la variabile `tollintv` agli input della funzione.
- 2 Si modifichi `demo_bisezione` in `demo_bisezione2` che utilizzi `bisezione2`. In particolare si assegni a `tollintv` il valore di 10^{-6} .
- 3 Si effettuino entrambi gli esperimenti, aventi quali valori del parametro esempio i numeri 1 e 2.

Definizione

Il **metodo di Newton** genera la successione (cf. [6])

$$x_{k+1} = x_k + s_k, \quad s_k = -\frac{f(x_k)}{f^{(1)}(x_k)}, \quad k = 0, 1, \dots \quad (1)$$

supposto che sia $f^{(1)}(x_k) \neq 0$ per $k = 0, 1, \dots$

Un **pseudo-codice** del metodo di Newton che si arresta quando la differenza tra due iterate successive è inferiore alla soglia di tolleranza è la seguente

```
[x,n,flag]=newton(f,f1,x0,toll,nmax)

n=1; flag=0; step=toll+1; x=x0; % settings iniziali
while (step >= toll) & (n < nmax) & (flag == 0) do
    if f1(x(n)) == 0 then % problemi: derivata prima nulla!
        flag=1;
    else % calcola iterazione e suo step abs(x(n+1)-x(n))
        s=-f(x(n))/f1(x(n));
        x(n+1)=x(n)+s;
        step=abs(s);
        n=n+1;
    end if
end while
if (n == nmax) & (step >= toll)
    flag = 1; % uscita per troppe iterazioni
end
```


Alcune osservazioni.

- Questo è un pseudo-codice e non un codice Matlab.
- Il codice inizialmente assegna le possibili variabili di output, scrivendo

```
n=1; flag=0; step=toll+1; x=x0;
```

- L'istruzione del ciclo while (cf. [2])

```
while (step >= toll) & (n < nmax) & (flag == 0) do
    ...
end while
```

in qualche senso dice **continua a iterare se lo step è sopra la soglia della tolleranza, se le iterazioni non sono troppe, e se finora tutto è andato bene.**

- all'interno del ciclo while si valuta la derivata e se è nulla il metodo esce perchè non può procedere (si dovrebbe dividere per zero nel valutare lo step successivo), mentre se ciò non succede
 - si calcola il nuovo step step,
 - la nuova iterata $x(n+1)$,
 - il valore assoluto dello step, utile nel test di arresto, e lo si assegna a step,
 - si incrementa il numero n di iterazioni eseguite.
 - si osservi che la variabile x è in generale un vettore e non uno scalare.

Esercizio (1)

- Aiutandosi con alcuni dettagli di **bisezione** e con il pseudo-codice fornito in **newton**, si implementi il metodo di Newton in una routine Matlab **newtonfun.m**, utilizzando in particolare l'intestazione:

```
function [xv, fxv, step, flag] = newtonfun (f, f1, x0, toll, maxit)

% Metodo di Newton
% Dati di ingresso:
%   f:      funzione
%   f1:     derivata prima
%   x0:     valore iniziale
%   toll:   tolleranza richiesta per il modulo
%           della differenza di due iterate successive
%   maxit:  massimo numero di iterazioni permesse
% Dati di uscita:
%   xv:     vettore contenente le iterate
%   fxv:    vettore contenente la valutazione di f
%           in xv
%   step:   vettore contenente gli step
%   flag:   0 la derivata prima non si e' annullata.
%           1 la derivata prima si e' annullata,
%           2 eseguite piu' iter. di maxit.
```

(Suggerimenti)

- *Risulta corretta la chiamata della routine nel pseudocodice?*
- *Il vettore in output nell'esercizio richiesto é xv e non x . Cosa fare?*
- *Si richiede in output il vettore delle valutazioni fxv . Cosa fare?*
- *Si noti che il numero massimo di iterazioni nell'input del codice Matlab non é $nmax$ come nel pseudocodice ma $maxit$. Cosa fare?*
- *Si richiede in output il vettore degli step $step$. Cosa fare? Risulta corretto inserire $step(n)=abs(s)$; al posto di $step=abs(s)$; oppure deve essere $step(n+1)=abs(s)$;?*
- *Visto che dopo la modifica necessaria la variabile $step$ diventa un vettore nella terzultima riga é corretto scrivere `if ... (step >= toll)?`*
- *Risulta corretto in Matlab l'uso dell'istruzione `while` del pseudocodice?*
- *Risulta corretto in Matlab l'uso dell'istruzione condizionale del pseudocodice?*
- *Risultano corretti in Matlab gli `end` dell'istruzione condizionale e del ciclo `while` del pseudocodice?*

Per la soluzione si veda [newtonfun.m](#);

Esercizio (2)

- Si implementi una versione di `newtonfun.m`, diciamo `newtonfun_for.m` che utilizzi un ciclo-for invece di un ciclo while (cf. [2]).

A tal proposito si esca per return (cf. [3])

- se la derivata prima si annulla in una iterazione ponendo flag uguale a 1,
- o se il valore assoluto dello step è minore della tolleranza ponendo flag uguale a 0.

Se dopo `maxit` iterazioni il valore assoluto dello step è ancora maggiore o uguale alla tolleranza si ponga flag uguale a 2.

- Utilizzando quale base `demo_newton`, si implementi la routine `demo_newton_for` che testi il metodo di Newton relativamente al calcolo degli zeri di

1 $f(x) = \exp(x) - 2 + x$ partendo dal valore iniziale x_0 uguale a 1,

2 $f(x) = \sin(x) - x$ partendo dal valore iniziale x_0 uguale a 1.

utilizzando `newtonfun_for.m`.

Si pongano in particolare:

- `toll` uguale a 10^{-6} ,
- `maxit` uguale a 1000.

- *La demo deve contenere il codice relativo al plot del valore assoluto dello step e della stampa su file*
 - 1** *degli indici della componente del vettore `xv` in formato decimale con 4 cifre intere,*
 - 2** *il vettore `xv` in formato esponenziale con 1 cifra prima della virgola e 15 dopo,*
 - 3** *il valore assoluto dello step `step`, in formato esponenziale con 1 cifra prima della virgola e 15 dopo.*
- *Si vedano i risultati ottenuti dal metodo di Newton per ogni singolo esempio. Il numero di iterazioni è inferiore a quello di bisezione?*

Nota.

*Le correzioni di questi due esercizi sono reperibili alla homepage del corso.
In particolare si vedano i files Matlab*

- *`newtonfun.m`;*
- *`newtonfun_for.m`;*
- *`demo_newton_for.m`.*

Si supponga di voler calcolare un certo x^* tale che $x = \phi(x)$. Il metodo di **punto fisso** definisce, partendo da un certo $x^{(0)}$ la successione, detta delle **approssimazioni successive**, $x^{(k+1)} = \phi(x^{(k)})$.

Esercizio

*Basandosi sulla routine **newtonfun.m**, definire una routine **puntofisso.m** che*

- *risolva il problema di punto fisso mediante la successione delle approssimazioni successive,*
- *arresti il processo quando il valore assoluto dello step è minore di una tolleranza `tol`,*
- *se vengono eseguite più di `maxit` iterazioni si esca comunque dalla procedura ponendo `flag=1` altrimenti esca con `flag=0`.*

Nota.

*La correzione di questo esercizio é reperibile alla homepage del corso.
In particolare si veda il file Matlab*

- *[puntofisso.m](#).*



Mathworks, Ciclo For,

<https://www.mathworks.com/help/matlab/ref/for.html>



Mathworks, Ciclo While,

<https://www.mathworks.com/help/matlab/ref/while.html>



Mathworks, Return,

<https://www.mathworks.com/help/matlab/ref/return.html>



Wikipedia, Ciclo For,

https://it.wikipedia.org/wiki/Ciclo_for



Wikipedia, Metodo della Bisezione,

https://it.wikipedia.org/wiki/Metodo_della_bisezione



Wikipedia, Metodo delle Tangenti,

https://it.wikipedia.org/wiki/Metodo_delle_tangenti