Equazioni nonlineari in Matlab

Alvise Sommariva

Università degli Studi di Padova

3 aprile 2025

Si supponga di dover risolvere l'equazione $f(x) = 0$ con $f : [a, b] \subset \mathbb{R} \to \mathbb{R}$ 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 `toll`, concluderemo le iterazioni se $f(c) = 0$ oppure $|w \cdot f(c)| \leq$ toll.

  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 `toll` 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 immagazzinate 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=@(x) exp(x)-2+x;
        a=0; b=1; sol=0.4428544010023885;
    case 2
        f=@(x) 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))';
semilogy(indici,wr);
```

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 immagazzinate nell'ultima componente del vettore `aa` cioè in `aa(end)`;
- di seguito dopo aver scritto su monitor alcune statistiche, il codice in scala semilogaritmica disegna le coppie $(k, \mathtt{wr}_k)$ dove $\mathtt{wr}_k$ è il residuo pesato alla k-sima iterazione.

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 di iterazione.

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
>>
```

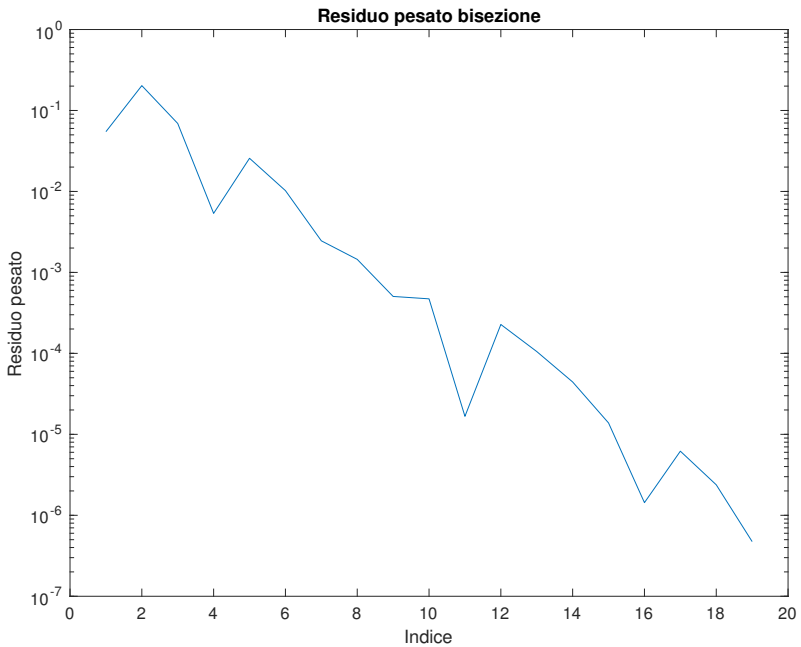

Figura: Residuo pesato delle iterazioni fornite dal metodo di bisezione, relativamente al metodo di bisezione per lo studio dello zero di $f(x) = \exp(x) - 2 + x$, con $a = 0$, $b = 1$.

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
>>
```

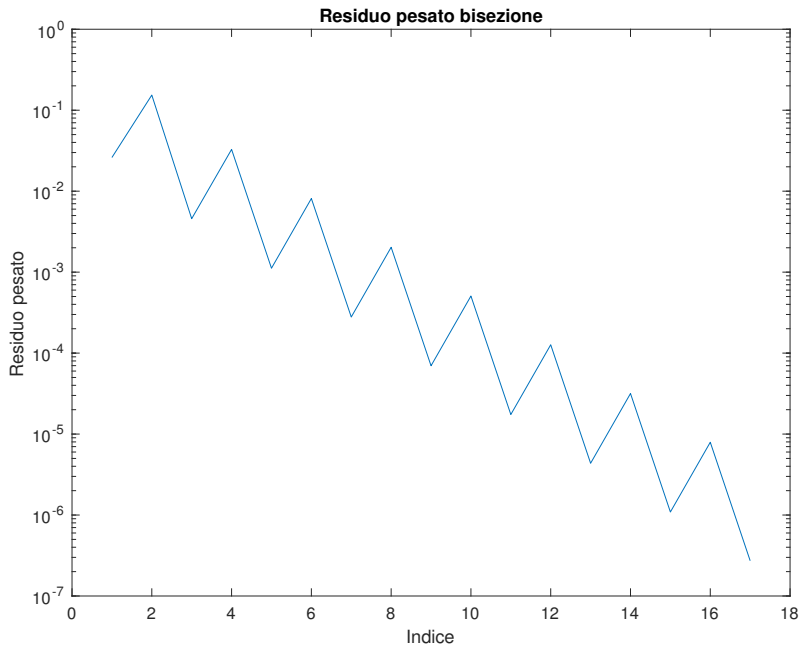

Figura: Residuo pesato delle iterazioni fornite dal metodo di bisezione per lo studio dello zero di $f(x) = \sin(x) - x$, con $a = -2$, $b = 3$.

Definizione

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

$$x_{k+1} = x_k + s_k, \;\; s_k = -\frac{f(x_k)}{f^{(1)}(x_k)}, \;\; k = 0, 1, \ldots \tag{1}$$

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

Un codice del metodo di Newton che si arresta quando il modulo della differenza tra due iterate successive (cioé lo step) è inferiore alla soglia di tolleranza è la seguente

```
1  function [x,n,flag]=newton(f,f1,x0,toll,nmax)
2
3  n=1; x=x0; flag=1; % settings iniziali
4  while (n < nmax)
5     if f1(x(n)) == 0 then % problemi: derivata prima nulla!
6          return; % non serve mettere flag=1 (e' gia il valore attuale)
7     else % calcola iterazione e suo step abs(x(n+1)-x(n))
8        s=-f(x(n))/f1(x(n));
9        x(n+1)=x(n)+s;
10       step=abs(s);
11       n=n+1;
12       if step < tol
13          flag=0; return;
14        end
15    end
16 end
17
```

Alcune osservazioni.

- Il codice inizialmente assegna le possibili variabili di output, scrivendo

  `n=1; x=x0; flag=1;`

  Mettiamo `flag=1` perché nonostante vada tutto bene, il risultato non sia stato raggiunto.
- L'istruzione del ciclo `while` (cf. [2])

```
while (n < nmax)
    ...
end
```

  in qualche senso dice continua a iterare se non si e' raggiunto il risultato e il numero di iterazioni non sono troppe.
- 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.
  - se lo step é sotto la tolleranza esce immediatamente dalla routine altrimenti reitera se il test `n < nmax` é soddisfatto.

Esercizio (1)

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

  Gli input siano gli stessi di `newton.m`*, mentre quali output si forniscano*
  - *il vettore delle iterate* `xv`*,*
  - *il vettore degli step* `step` *al variare delle iterazioni,*
  - *il vettore* `flag` *che indica il comportamento del metodo alla sua terminazione.*

  A tal proposito si esca per `return` *(cf. [3])*
  - *se la derivata prima si annulla in una iterazione si pongano l'ultima componente dello step pari a* `NaN` *e* `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*.*

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

xv(1)=x0;                       % inizializzazioni
flag=1;

for k=1:maxit
    f1xv(k)=feval(f1,xv(k));
    if f1xv(k) == 0          % uscita per derivata nulla
       step(k)=NaN; return; % flag a questo livello e' uguale a 1
    end
    fxv(k)=feval(f,xv(k));

    step(k)=-fxv(k)/f1xv(k); % calcolo step
    xv(k+1)=xv(k)+step(k);    % aggiornamento
    if abs(step(k)) <= toll  % uscita per test di arresto verificato
        flag=0; return;
    end
end

flag=2; % troppe iterazioni
```

Esercizio (2)

Utilizzando quale base `demo_bisezione`*, 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* `x0` *uguale a* 1,
2. $f(x) = \sin(x) - x$ *partendo dal valore iniziale* `x0` *uguale a* 1.

utilizzando `newtonfun_for.m`*.*

Si pongano in particolare:

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

Si testi la performance in entrambi gil esempi, variando il valore della variabile relativa all'esempio.

```
function demo_newton_for

toll=10^(-6); nmax=1000; esempio=1;
% esempi.
switch esempio
    case 1
        f=@(x) exp(x)-2+x;
        f1=@(x) exp(x)+1;
        x0=1; sol=0.4428544010023885;
    case 2
        f=@(x) sin(x)-x;
        f1=@(x) cos(x)-1;
        x0=1; sol=0;
end

[xv,step,flag] = newton_for(f,f1,x0,toll,nmax);

fprintf('\n \t soluzione : %1.15e',xv(end));
fprintf('\n \t tolleranza: %1.15e',toll);
fprintf('\n \t numero it.: %4d',length(xv));

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

indici=(1:length(step))';
semilogy(indici,step);
```

Bibliografia I

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