

Equazioni e sistemi non lineari

Lucia Gastaldi

Dipartimento di Matematica,
<http://dm.ing.unibs.it/gastaldi/>

Lezione 6 - 27 febbraio 2009

Indice

1 Ricerca degli zeri di una funzione

- Problema e definizioni
- Metodo di Newton-Raphson
- Test d'arresto
- Metodo delle secanti
- Function di Matlab

2 Soluzione di sistemi non lineari

- Il metodo di Newton-Raphson per sistemi
- Problemi di minimo

3 Appendice

Zeri di funzione

Problema

Data $f : [a, b] \rightarrow \mathbb{R}$ si cerca $x \in [a, b]$ tale che $f(x) = 0$.

Indichiamo con α uno zero di f .

Teorema

Supponiamo che la funzione $f : [a, b] \rightarrow \mathbb{R}$ sia **continua in $[a, b]$** e che $f(a) \cdot f(b) < 0$; allora esiste $\alpha \in (a, b)$ tale che $f(\alpha) = 0$.

Ordine di convergenza di un metodo iterativo

Definizione

Si dice che un **metodo iterativo** è **convergente di ordine** $p > 1$ se vale

$$\lim_{k \rightarrow \infty} \frac{|\alpha - x_{k+1}|}{|\alpha - x_k|^p} = C \neq 0.$$

Si dice che un **metodo iterativo** **converge linearmente** se esiste un numero positivo $0 < C < 1$ tale che

$$\lim_{k \rightarrow \infty} \frac{|\alpha - x_{k+1}|}{|\alpha - x_k|} = C.$$

Si dice che un **metodo iterativo** **converge superlinearmente** se vale

$$\lim_{k \rightarrow \infty} \frac{|\alpha - x_{k+1}|}{|\alpha - x_k|} = 0.$$

Metodo di Newton-Raphson

Supponiamo di avere calcolato il valore x_k .

La migliore approssimazione lineare della funzione f nel punto x_k è data dalla retta tangente

$$t_k(x) = f(x_k) + f'(x_k)(x - x_k).$$

Ponendo $t_k(x) = 0$, si ricava il nuovo punto della successione x_{k+1} .

Iterata di Newton-Raphson

Dato x_0 ,

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

Teorema di convergenza locale quadratica

Teorema

Sia $f : [a, b] \rightarrow \mathbb{R}$ una funzione di classe \mathbf{C}^2 . Sia α tale che

$$f(\alpha) = 0, \quad f'(\alpha) \neq 0.$$

Allora esiste $\eta > 0$ tale che se il punto iniziale x_0 soddisfa $|\alpha - x_0| \leq \eta$ allora si ha:

1. Per ogni $k \in \mathbb{N}$, $|\alpha - x_k| \leq \eta$;
2. $\lim_{k \rightarrow \infty} x_k = \alpha$;
3. $\lim_{k \rightarrow \infty} \frac{\alpha - x_{k+1}}{(\alpha - x_k)^2} = -\frac{f''(\alpha)}{2f'(\alpha)}$.

Algoritmo di Newton-Raphson

1. Dato x_0 .
2. valuta $y = f(x_0)$ e la derivata $dy = f'(x_0)$;
3. Se il **test d'arresto** è verificato, x_0 è la soluzione cercata; **stop**.
4. Altrimenti:
 - 3.1 calcola $\delta = -y/dy$;
 - 3.2 aggiorna $x_0 = x_0 + \delta$.
 - 3.3 valuta $y = f(x_0)$ e la derivata $dy = f'(x_0)$;
5. Ripeti da 3.

Test d'arresto

Si deve trovare un modo per imporre che l'errore sia inferiore ad una tolleranza prestabilita, ossia tale che

$$\frac{|\alpha - x_k|}{|\alpha|} \leq \text{toll}, \quad \text{oppure } |\alpha - x_k| \leq \text{toll se } \alpha = 0.$$

Due possibilità:

$$|f(x_k)| \leq \text{toll}$$

non efficiente se $|f'(\alpha)| \approx 0$ oppure $|f'(\alpha)| \gg 1$.

$$|x_{k+1} - x_k| \leq \text{toll}$$

efficiente se il metodo converge **superlinearmente**.

Function grafnewt

grafnewt

La **function** `grafnewt.m` implementa l'algoritmo di Newton-Raphson e riporta in un grafico il comportamento della soluzione:

```
[zero,fz,iter,xk,fk]=grafnewton (f,fd,a,b,x0,toll,Niter)
```

Input

| | |
|--------------------|---|
| <code>f</code> | function che contiene l'espressione della funzione f ; |
| <code>df</code> | function che contiene l'espressione della derivata f' ; |
| <code>a,b</code> | intervallo in cui viene rappresentata la funzione; |
| <code>x0</code> | punto iniziale per l'iterazione; |
| <code>toll</code> | tolleranza desiderata; |
| <code>Niter</code> | numero massimo di iterazioni da eseguire; |

Output

| | |
|-------------------|--|
| <code>zero</code> | soluzione cercata; |
| <code>fz</code> | valore di f nello zero calcolato; |
| <code>xk</code> | vettore contenente le successive approssimazioni x_k ; |
| <code>fk</code> | vettore contenente i valori $f(x_k)$; |
| <code>iter</code> | numero di iterazioni necessarie per arrivare alla soluzione. |

Esercizio

Si consideri la funzione

$$f(x) = x - \frac{3\sin(3x)}{x} \quad x \in [0, 6].$$

Usare la function `grafnewt` per calcolare lo zero di f e rappresentare il procedimento iterativo con le seguenti scelte del dato iniziale $x_0 = 0.1$, $x_0 = 2$, $x_0 = 2.1$, $x_0 = 2.2$, $x_0 = 2.4$, $x_0 = 2.5$ e $x_0 = 3$.

Function newton

`newton`

La **function** `newton.m` implementa l'algoritmo di Newton-Raphson con la seguente riga di dichiarazione:

```
[zero,fz,iter]=newton (f,fd,x0,toll,Niter,alfa)
```

Input

| | |
|--------------------|---|
| <code>f</code> | function che contiene l'espressione della funzione f ; |
| <code>df</code> | function che contiene l'espressione della derivata f' ; |
| <code>x0</code> | punto iniziale per l'iterazione; |
| <code>toll</code> | tolleranza desiderata; |
| <code>Niter</code> | numero massimo di iterazioni da eseguire; |
| <code>alfa</code> | valore esatto della soluzione se conosciuto. |

Output

| | |
|-------------------|--|
| <code>zero</code> | soluzione cercata; |
| <code>fz</code> | valore di f nello zero calcolato; |
| <code>iter</code> | numero di iterazioni necessarie per arrivare alla soluzione. |

Tabella dei risultati

La function `newton` produce una tabella che contiene in ciascuna riga:

- numero dell'iterazione k ;
- soluzione approssimata x_k ;
- valore della funzione $f(x_k)$;
- differenza $x_k - x_{k-1}$;
- rapporto $\frac{\alpha - x_k}{(\alpha - x_{k-1})^2}$

Nota bene

Per calcolare l'ultima colonna della tabella viene usato il valore esatto se conosciuto.

Se non si conosce il valore esatto dello zero, la variabile `alfa` non viene assegnata e l'ordine di convergenza viene calcolato usando `alfa=zero`.

Test

Esercizio 1

Usare la function `newton` per calcolare gli zeri delle seguenti funzioni, con x_0 assegnato:

$$\begin{array}{ll} f(x) = \sin x - \cos 2x & x_0 = 1 \\ f(x) = x^3 - 7x^2 + 11x - 5 & x_0 = 2 \text{ e } x_0 = 7 \\ f(x) = x^4 - 12x^3 - 47x^2 - 60x + 24 & x_0 = 0 \text{ e } x_0 = 2 \end{array}$$

Fare un grafico di ciascuna funzione e commentare i risultati ottenuti.

Esercizio 2

Si consideri l'equazione

$$2x^4 - 11x^3 + 21x^2 - 16x + 4 = 0.$$

1. Plottare la funzione nell'intervallo $[0, 3]$.
2. Trovare gli zeri mediante il metodo di Newton usando i seguenti valori iniziali: $x_0 = 0.75, 1.25, 1.75, 2.25, 2.75$.
3. Per ciascuna radice trovata dire l'ordine di convergenza.

Metodo delle secanti

Supponiamo di avere calcolato il valore x_k .

Nel caso in cui non si disponga della derivata di f oppure il costo del calcolo sia eccessivo si può considerare la secante che passa per $(x_k, f(x_k))$ e $(x_{k-1}, f(x_{k-1}))$:

$$s_k(x) = f(x_k) + \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}(x - x_k).$$

Ponendo $s_k(x) = 0$, si ricava il nuovo punto della successione x_{k+1} .

Iterata delle secanti

Dati x_0 e x_1

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

Teorema di convergenza locale superlineare

Teorema

Sia $f : [a, b] \rightarrow \mathbb{R}$ una funzione di classe \mathbf{C}^2 . Sia α tale che

$$f(\alpha) = 0, \quad f'(\alpha) \neq 0.$$

Allora esiste $\eta > 0$ tale che se il punto iniziale x_0 soddisfa $|\alpha - x_0| \leq \eta$ allora si ha:

1. Per ogni $k \in \mathbb{N}$, $|\alpha - x_k| \leq \eta$;
2. $\lim_{k \rightarrow \infty} x_k = \alpha$;
3. $\lim_{k \rightarrow \infty} \frac{\alpha - x_{k+1}}{(\alpha - x_k)(\alpha - x_{k-1})} = -\frac{f''(\alpha)}{2f'(\alpha)}$
4. $\lim_{k \rightarrow \infty} \frac{\alpha - x_{k+1}}{|\alpha - x_k|^p} = M^{-(1+1/p)}$

con $p = (1 + \sqrt{5})/2$ e $M = |f''(\alpha)|/(2|f'(\alpha)|)$.

Algoritmo delle secanti

1. Dato x_0 e x_1 ;
2. valuta $f_0 = f(x_0)$ e $f_1 = f(x_1)$.
3. Se il **test d'arresto** è verificato, x_1 è la soluzione cercata; **stop**.
4. Altrimenti:
 - 3.1 calcola $dy = (f_1 - f_0)/(x_1 - x_0)$;
 - 3.2 calcola $\delta = -f_1/dy$;
 - 3.3 aggiorna $x_0 = x_1$, $x_1 = x_1 + \delta$;
 - 3.4 aggiorna $f_0 = f_1$;
 - 3.5 valuta $f_1 = f(x_1)$.
5. Ripeti da 3.

Function secanti

secanti

La **function** `secanti.m` implementa l'algoritmo delle secanti con la seguente riga di dichiarazione:

```
[zero,fz,iter]=secanti (f,x0,x1,toll,Niter,alfa)
```

Input

| | |
|--------------------|--|
| <code>f</code> | function che contiene l'espressione della funzione f ; |
| <code>x0,x1</code> | dati iniziali per l'iterazione; |
| <code>toll</code> | tolleranza desiderata; |
| <code>Niter</code> | numero massimo di iterazioni da eseguire; |
| <code>alfa</code> | valore esatto della soluzione se conosciuto. |

Output

| | |
|-------------------|--|
| <code>zero</code> | soluzione cercata; |
| <code>fz</code> | valore di f nello zero calcolato; |
| <code>iter</code> | numero di iterazioni necessarie per arrivare alla soluzione. |

Esercizio

Usare la function `secanti` per risolvere gli esercizi 1 e 2, scegliendo lo stesso valore di x_0 e $x_1 = x_0 + 0.5$. In caso di non convergenza, modificare opportunamente il valore di x_1 .

fzero

Calcola gli zeri di una funzione reale di variabile reale con la seguente sintassi

```
[x,fval]=fzero(@fun,x0)
[x,fval]=fzero(fun,x0)
```

Input

| | |
|------------------|---|
| <code>fun</code> | nome della function o della funzione <code>inline</code> che contiene la funzione f |
| <code>x0</code> | dato iniziale |

Output

| | |
|-------------------|--------------------------------------|
| <code>x</code> | approssimazione dello zero calcolato |
| <code>fval</code> | valore di f in x . |

Si possono ottenere delle informazioni complete sulle iterazioni usando il comando

```
[x,fval]=fzero(@fun,x0,optimset('disp','iter'))
```

Il metodo di Newton-Raphson per sistemi

Consideriamo una funzione a valori vettoriali $F : A \rightarrow \mathbb{R}^n$ con $A \subseteq \mathbb{R}^n$:

$$F(\mathbf{x}) = \begin{cases} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \dots\dots\dots \\ f_n(x_1, x_2, \dots, x_n) \end{cases}$$

Problema

Trovare $\mathbf{x} = (x_1, x_2, \dots, x_n) \in A$ tale che $F(\mathbf{x}) = 0$.

Linearizzazione

Per semplicità consideriamo $n = 2$ quindi abbiamo il sistema:

$$\begin{cases} f(x, y) = 0 \\ g(x, y) = 0 \end{cases}$$

Supponiamo di essere arrivati a calcolare un approssimazione (x_k, y_k) e consideriamo l'approssimazione di f e g con i piani tangenti nel punto (x_k, y_k) :

$$\begin{cases} f(x, y) \approx f(x_k, y_k) + f_x(x_k, y_k)(x - x_k) + f_y(x_k, y_k)(y - y_k) \\ g(x, y) \approx g(x_k, y_k) + g_x(x_k, y_k)(x - x_k) + g_y(x_k, y_k)(y - y_k) \end{cases}$$

Iterazione del metodo di Newton-Raphson

La nuova approssimazione si ottiene come

$$x_{k+1} = x_k + \delta_x \quad y_{k+1} = y_k + \delta_y$$

dove il vettore $\delta = (\delta_x, \delta_y)^T$ è la soluzione del sistema

$$J(x_k, y_k)\delta = -F(x_k, y_k)$$

e

$$J(x_k, y_k) = \begin{pmatrix} f_x(x_k, y_k) & f_y(x_k, y_k) \\ g_x(x_k, y_k) & g_y(x_k, y_k) \end{pmatrix} \quad F(x_k, y_k) = \begin{pmatrix} f(x_k, y_k) \\ g(x_k, y_k) \end{pmatrix}$$

Algoritmo di Newton-Raphson

Newton_sist.m

1. Dato \mathbf{x}_0 .
2. Se il **test d'arresto** è verificato, \mathbf{x}_0 è la soluzione cercata; **stop**.
3. Altrimenti:
 - 3.1 valuta $Y = F(\mathbf{x}_0)$ e lo Jacobiano $A = J(\mathbf{x}_0)$;
 - 3.2 risolvi $A\delta = -Y$;
 - 3.3 aggiorna $\mathbf{x}_0 = \mathbf{x}_0 + \delta$.
4. Ripeti da 2.

Nota bene

F è il nome di una function che fornisce il valore di F in un vettore colonna di dimensione n .

J è il nome di una function che fornisce il valore dello Jacobiano come array $n \times n$.

Function newtonsys

La function `newtonsys` risolve un sistema non lineare mediante il seguente comando:

```
[zero,fz,iter]=newtonsys (f,fd,x0,toll,Niter,alfa)
```

Input

| | |
|--------------------|--|
| <code>f</code> | function che contiene la funzione f (vettore colonna); |
| <code>df</code> | function che contiene lo Jacobiano J (matrice); |
| <code>x0</code> | punto iniziale per l'iterazione (vettore colonna); |
| <code>toll</code> | tolleranza desiderata; |
| <code>Niter</code> | numero massimo di iterazioni da eseguire; |
| <code>alfa</code> | valore esatto della soluzione se conosciuto. |

Output

| | |
|-------------------|--|
| <code>zero</code> | soluzione cercata; |
| <code>fz</code> | valore di f nello zero calcolato; |
| <code>iter</code> | numero di iterazioni necessarie per arrivare alla soluzione. |

fsolve

Risolve i sistemi di equazioni non lineari in più variabili.

```
[x,fval]=fsolve(@fun,x0)
```

Input

fun nome della function che contiene la funzione f
fun accetta in input un vettore x e dà in output il vettore dei valori di f valutata in x .

x0 dato iniziale

Output

x approssimazione dello zero calcolato

fval valore di **fun** in **x**.

Opzioni per `fsolve`

```
[x,fval]=fsolve(@fun,x0,options)
```

risolve il sistema con i parametri di default sostituiti da quelli dichiarati nella struttura `options`. `options` viene creato con il comando `optimset`. Vedere `optimset` per i dettagli. Le opzioni più usate sono: `Display`, `TolX`, `TolFun`, `DerivativeCheck`, `Diagnostics`, `Jacobian`, `MaxFunEvals`, `MaxIter`, `PlotFcns`, `OutputFcn`.

Per usare lo Jacobiano la function `FUN` deve dare come output sia il valore di f che quello del suo jacobiano.

```
[x,fval,exitflag,output]=fsolve(@fun,x0,options)
```

fornisce in output le seguenti informazioni:

`exitflag` ha valore da -4 a 4. Se l'algorithm è arrivato a convergenza correttamente `exitflag=1`.

`output` è una struttura del seguente tipo:

```
output =
```

```
    iterations: 5
```

```
    funcCount: 18
```

```
    algorithm: 'trust-region dogleg'
```

```
firstorderopt: 1.6919e-07
```

```
    message: [1x76 char]
```

Esercizi

Esercizio

Sono date le seguenti funzioni:

$$F_1(x, y) = \begin{pmatrix} x + y - 3 \\ x^2 + y^2 - 9 \end{pmatrix}$$

$$\begin{array}{l} \text{radici: } (0, 3) \ (3, 0) \\ x_0 = (1, 5), \ x_0 = (2, 3) \end{array}$$

$$F_2(x, y) = \begin{pmatrix} x^2 + y^2 - 2 \\ e^{x-1} + y^3 - 2 \end{pmatrix}$$

$$\begin{array}{l} \text{radice: } (1, 1) \\ x_0 = (1.5, 2), \ x_0 = (2, 3) \end{array}$$

Trovare la soluzione usando il metodo di Newton (`newtonsys`) e la function di Matlab `fsolve`. In questo caso confrontare cosa si ottiene fornendo anche il valore dello Jacobiano.

Problemi di minimo

Esercizio

Usare la function `newtonsys` per trovare il minimo delle seguenti funzioni di più variabili:

$$f(x, y) = 10x^2 + y^2 \quad \alpha = (0, 0) \quad x_0 = [1; 2]$$

$$f(x, y) = (x - 2)^4 + (x - 2)^2 y^2 + (y + 1)^2 \quad \alpha = (2, -1) \quad x_0 = [1; 1]$$

$$f(x, y) = x^4 + (x + y)^2 + (e^x - 1)^2 \quad \alpha = (0, 0) \quad x_0 = [1; 1], [-1; 3]$$

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2 \quad \alpha = (1, 1) \quad x_0 = [-1.9; 2]$$

Rappresentare le superfici corrispondenti alle funzioni date e le loro curve di livello.

Risolvere i precedenti problemi di minimo usando la function di Matlab `fminsearch` mediante il seguente comando

```
x = fminsearch(fun, x0)
```

essendo `fun` il nome della function che contiene la funzione e `x0` il punto iniziale. Per ulteriori dettagli sull'uso di questa function dare il comando `help fminsearch`.

Esercizio facoltativo

Radici terze dell'unità

- Risolvere in campo complesso l'equazione $z^3 = 1$. Tre radici: $z_0 = 1$, $z_1 = -0.5 + i \sin(2\pi/3)$ e $z_3 = -0.5 - i \sin(2\pi/3)$.
- Posto $z = x + iy$, trasformare l'equazione data in un sistema di due equazioni reali relative alla parte reale e al coefficiente dell'immaginario.
- Trovare le soluzioni mediante il metodo di Newton-Raphson.
- Considerare nel piano complesso un quadrato di lato L . Suddividere tale quadrato in n^2 quadratini. Utilizzare il centro di ciascun quadratino come x_0 per la risoluzione con il metodo di Newton-Raphson. Colorare il quadratino corrispondente
 - ▶ di rosso se la radice trovata è z_0 ,
 - ▶ di blu se la radice trovata è z_1 ,
 - ▶ di verde se la radice trovata è z_2 .

nargin

La function **nargin** conta i parametri in **input** di una function.
Può servire per rendere più flessibile l'uso di una function.

Esempio - nargin

Nella function `newton` si può passare come dato anche il valore dello zero esatto per fare l'analisi dell'errore e della convergenza.

Uso la seguente riga di dichiarazione:

```
function [zero,fz,iter,xk,fk]=newton (f,fd,x0,toll,Niter,alfa)
```

Nel programma per valutare l'ordine di convergenza uso le seguenti istruzioni:

```
if nargin==5  
alfa=zero  
end
```

e poi procedo con il calcolo del rapporto che dà la convergenza.

nargout

La function **nargout** conta i parametri in **output** di una function.
Può servire per rendere più flessibile l'uso di una function.

Esempio - nargout

Non sempre usando la function `newton` interessa conoscere tutta la successione dei valori ottenuti, più spesso interessano solo il valore dello zero, quello della f ed il numero delle iterazioni.

Uso comunque la seguente riga di dichiarazione:

```
function [zero,fz,iter,xk,fk]=newton (f,fd,x0,toll,Niter,alfa)
```

Se non interessa tutta la successione inserisco nel programma prima della costruzione dei vettori `xk` e `fk` il seguente controllo:

```
if nargout>=4  
xk=....  
fk=....  
end
```