

# Equazioni e sistemi non lineari

Lucia Gastaldi

DICATAM - Sezione di Matematica,  
<http://lucia-gastaldi.unibs.it>

# Indice

- 1 Ricerca degli zeri di una funzione
  - Problema e definizioni
  - Bisezione
  - Metodo di Newton-Raphson
  - Test d'arresto
  - Algoritmo ed esercizi
  - Metodo delle secanti
  - Function di Matlab
- 2 Soluzione di sistemi non lineari
  - Il metodo di Newton-Raphson per sistemi
  - Problemi di minimo
- 3 Gestione dell'output
- 4 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  $\alpha$  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  $f(\alpha) = 0$ .

## Metodo di bisezione

Data una funzione  $f : [a, b] \rightarrow \mathbb{R}$  continua, con  $f(a) \cdot f(b) < 0$ , il metodo di bisezione individua un intervallo di data ampiezza nel quale è contenuto lo zero  $\alpha$  della funzione.

### Algoritmo di bisezione

Fintanto che l'ampiezza dell'intervallo è superiore alla tolleranza desiderata si eseguono i seguenti passi:

1. si calcola il punto medio  $c = (a + b)/2$  dell'intervallo corrente;
2. si valuta  $f(c)$ ;
3. se  $f(c) = 0$ , allora  $c$  è lo zero cercato; **stop**;
4. altrimenti se  $f(a) \cdot f(c) < 0$  allora  $b = c$ ; si torna all'inizio del ciclo;
5. altrimenti  $a = c$ ; si torna all'inizio del ciclo.

### Stima dell'errore

Nell'intervallo corrente l'errore  $|\alpha - c| \leq (b - a)/2$ .

Dopo  $k$  iterazioni l'ampiezza dell'intervallo corrente è data dall'ampiezza dell'intervallo di partenza diviso  $2^k$ .

# Function bisezione

Scrivere un **M-file** di tipo function per trovare uno zero di una funzione con il metodo delle bisezioni. La riga di definizione è la seguente:

```
function [zero,fval,iter]=bisezione(f,a,b,tol)
```

## Input

`f` nome della funzione;  
`a,b` estremi dell'intervallo;  
`tol` tolleranza desiderata.

## Output

`zero` lo zero della funzione trovato;  
`fval` valore di  $f$  nello zero calcolato;  
`iter` iterazioni eseguite.

## Esercizio 1

Usare la function `bisezione` per trovare uno zero delle seguenti funzioni con tolleranza `tol=eps`. Rappresentare la funzione e marcare sul grafico lo zero trovato.

$$f(x) = x^2 - 2, \quad x \in [0, 2]$$

$$f(x) = 3x - 1, \quad x \in [0, 1]$$

$$f(x) = \arctan(x), \quad x \in [-1, 1]$$

$$f(x) = \sin(x) - \cos(2x), \quad x \in [-\pi, \pi]$$

$$f(x) = \sin(x) - \cos(2x), \quad x \in [0, 10004]$$

### Risultati

1. zero=1.4142, fval= 8.8818e-16, iter=52
2. zero=0.3333, fval=4.4409e-16, iter=51
3. zero=0, fval=0, iter=1
4. La funzione assume negli estremi lo stesso valore. Se si cambia l'intervallo di partenza in  $[-2, 2]$  si trova zero=0.5236, fval=-4.9960e-16, iter=53
5. zero=4.6438e+03, fval=-2.0687e-12, iter=65

## Metodo di bisezione con test sull'errore relativo

Modificare la function bisezione in modo di considerare il seguente test d'arresto

$$\frac{b - a}{\max(|a|, |b|)} < \text{tol}$$

Ripetere le prove dell'esercizio precedente.

### Risultati

1. zero=1.4142, fval=4.4409e-16, iter=53
2. zero=0.3333, fval=0, iter=53
3. zero=0, fval=0, iter=1
4. La funzione assume negli estremi lo stesso valore. Se si cambia l'intervallo di partenza in  $[-2, 2]$  si trova zero=0.5236, fval=-1.6653e-16, iter=55
5. zero=4.6438e+03, fval=2.9432e-13, iter=53

## 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  $\alpha$  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.  $\lim_{k \rightarrow \infty} \frac{x_{k+1} - \alpha}{(x_k - \alpha)^2} = \frac{f''(\alpha)}{2f'(\alpha)}$ .

## 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**.

## Algoritmo di Newton-Raphson

1. Dati  $f$ ,  $f'$ ,  $x_0$ ,  $toll$  e  $nmax$ ;
2. valuta  $y = f(x_0)$  e la derivata  $dy = f'(x_0)$ ;
3. inizializza  $\delta = 1$  e  $iter = 0$ ;
4. Se  $|\delta| \leq toll$  il test d'arresto è verificato,  $x_0$  è la soluzione cercata; **stop**.
5. Se  $iter > nmax$ , è stato raggiunto il numero massimo di iterazioni senza arrivare a convergenza; **stop**.
6. Altrimenti:
  - 6.1 calcola  $\delta = -y/dy$ ;
  - 6.2 aggiorna  $x_0 = x_0 + \delta$ .
  - 6.3 valuta  $y = f(x_0)$  e la derivata  $dy = f'(x_0)$ ;
  - 6.4 incrementa l'indice di iterazione  $iter = iter + 1$ .
7. Ripeti da 4.

## Function newton

Scrivere un programma di tipo function per implementare il metodo di Newton, con la seguente riga di definizione:

```
function [zero,fz,iter]=newton(f,df,x0,toll,Nmax)
```

### 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>Nmax</code>	numero massimo di iterazioni da eseguire;

### Output

---

<code>zero</code>	soluzione cercata;
<code>fz</code>	valore di $f$ nello zero calcolato;
<code>iter</code>	numero di iterazioni utilizzate.

## Esercizi

### Esercizio 2

Testare la function `newton` per determinare gli zeri delle funzioni dell'Esercizio 1.

### Esercizio 3

La funzione

$$f(x) = e^x - 2x^2$$

ha tre zeri,  $\alpha_1 < 0$ ,  $\alpha_2$  e  $\alpha_3$  positivi.

- Fare il grafico della funzione.
- Per  $i = 1, 2, 3$  trovare un valore di  $x_0$  in modo che il metodo di Newton implementato converga a  $\alpha_i$ .
- Marcare gli zeri trovati sul grafico della funzione.

## Esercizio 4

Si consideri la funzione

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

La funzione e la sua derivata sono implementate nelle function `sin3` e `dsin3` rispettivamente.

Usare la function `newton` 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  $x_0 = 3$ .

## Convergenza del metodo di Newton

Modificare la function `newton` che implementa l'algoritmo di Newton-Raphson con la seguente riga di dichiarazione:

```
[zero,fz,iter,xk]=newton(f,df,x0,tol,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>x0</code>	punto iniziale per l'iterazione;
<code>tol</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>iter</code>	numero di iterazioni utilizzate;
<code>xk</code>	vettore delle approssimazioni successive ( <b>opzionale</b> ).

► nargin e nargout



## Verifica dell'ordine di convergenza

Se il metodo di Newton converge quadraticamente si verifica che

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x_k|}{|x_k - x_{k-1}|^2} = \frac{|f''(\alpha)|}{|2f'(\alpha)|}.$$

Quindi per verificare l'andamento dell'errore si può riportare in un grafico in scala semilogartimica ([semilogy](#)) la differenza fra due iterate successive. Se l'ordine di convergenza è quadratico si potrà vedere che l'errore decresce rapidamente.

Si può verificare questo comportamento anche dai valori della frazione nel limite scritto sopra.

# Test

## Esercizio 5

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

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

Fare un grafico di ciascuna funzione. Riportare in scala semilogaritmica la differenza fra due successive iterate e dedurre se la convergenza è quadratica o lineare.

## Esercizio 6

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.

## Soluzione Esercizio 5

`tol=1e-10, nmax=20`

$$f(x) = \sin x - \cos 2x \quad x_0 = 1$$

Il metodo converge in 5 iterazioni `zero = 5.235988e-01 fval = -1.665335e-16` Convergenza quadratica

$$f(x) = x^3 - 7x^2 + 11x - 5$$

Se  $x_0 = 2$  il metodo non converge nel massimo numero di iterazioni

Residuo finale = `-1.869616e-12` Convergenza lineare

Se  $x_0 = 7$  il metodo converge in 7 iterazioni `zero =`

`5.000000e+00 fval = -7.105427e-15` Convergenza quadratica

$$f(x) = x^4 - 12x^3 + 47x^2 - 60x + 24$$

Se  $x_0 = 0$  il metodo converge in 9 iterazioni `zero =`

`8.883058e-01 fval = 0.000000e+00` Convergenza quadratica

Se  $x_0 = 2$  il metodo converge in 8 iterazioni `zero =`

`1.000000e+00 fval = 0.000000e+00` Convergenza quadratica

## Soluzione Esercizio 6

$f(x) = 2x^4 - 11x^3 + 21x^2 - 16x + 4$   $\text{tol}=1e-10$ ,  $\text{nmax}=20$

Se  $x_0 = 0.75$  il metodo converge in 10 iterazioni  $\text{zero} =$

$5.000000e-01$   $\text{fval} = 0.000000e+00$  Convergenza quadratica

Se  $x_0 = 1.25$  Il metodo converge in 5 iterazioni  $\text{zero} =$

$1.000000e+00$   $\text{fval} = 0.000000e+00$  Convergenza cubica

Se  $x_0 = 1.75$ ,  $x_0 = 2.25$ ,  $x_0 = 2.75$  il metodo non converge nel massimo numero di iterazioni con residui finali =  $4.973799e-14$ ,  $3.339551e-13$ ,  $8.292034e-12$  Convergenza lineare

## Esempi

### Un esempio perverso

Usare la function `newton` per calcolare lo zero della seguente funzione:

$$f(x) = \text{sign}(x)\sqrt{|x|}, \quad x \in \mathbb{R}.$$

Osservare che per qualunque scelta del dato iniziale  $x_0$  la successione è oscillante.

### `arctan(x)`

Applicare il metodo di Newton alla ricerca dello zero della funzione  $f(x) = \arctan(x)$  con  $x_0 = 0.3, 2$ .

Usare il metodo di Newton per trovare il valore critico di  $x_0$  per cui  $x_1 = -x_0$ . Chiamare  $x_c$  tale valore.

Verificare, usando la function `newton`, che il metodo di Newton applicato alla funzione  $f(x) = \arctan(x)$ :

- converge per  $x_0 \leq x_c$ ;
- diverge per  $x_0 \geq x_c$ ;
- è oscillante per  $x_0 = x_c$ .

## 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  $\alpha$  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

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

```
[zero,fz,iter]=secanti (f,x0,x1,tol,Nit)
```

## Input

---

<code>f</code>	function che contiene l'espressione della funzione $f$ ;
<code>x0,x1</code>	dati iniziali per l'iterazione;
<code>tol</code>	tolleranza desiderata;
<code>Nit</code>	numero massimo di iterazioni da eseguire;

## Output

---

<code>zero</code>	soluzione cercata;
<code>fz</code>	valore di $f$ nello zero calcolato;
<code>iter</code>	numero di iterazioni utilizzate;

## Esercizio 7

Usare la function `secanti` per risolvere gli esercizi 5 e 6, scegliendo lo stesso valore di  $x_0$  e  $x_1 = x_0 + 0.5$ . In caso di non convergenza, modificare opportunamente il valore di  $x_1$ .

## Soluzione Esercizio 7 - Funzioni Es. 5

$tol=1e-10$ ,  $nmax=20$   $p = (1 + \sqrt{5})/2$

$f(x) = \sin x - \cos 2x$   $x_0 = 1$  e  $x_1 = 1.5$

Il metodo converge in 8 iterazioni  $zero = 5.235988e-01$   $fval = 1.110223e-16$  Convergenza di ordine  $p$

$f(x) = x^3 - 7x^2 + 11x - 5$

Se  $x_0 = 2$  e  $x_1 = 2.5$  il metodo non converge nel massimo numero di iterazioni Residuo finale =  $-2.068380e-08$  Convergenza lineare

Se  $x_0 = 7$  e  $x_1 = 7.5$  il metodo converge in 10 iterazioni  $zero = 5.000000e+00$   $fval = -2.131628e-14$

Convergenza di ordine  $p$

$f(x) = x^4 - 12x^3 + 47x^2 - 60x + 24$

Se  $x_0 = 0$  e  $x_1 = 0.5$  il metodo converge in 12 iterazioni  $zero = 8.883058e-01$   $fval = 7.105427e-15$

Convergenza di ordine  $p$

Se  $x_0 = 2$  e  $x_1 = 2.5$  dopo 11 iterazioni si trova la soluzione esatta  $zero = 1.000000e+00$   $fval = 0.000000e+00$

## Soluzione Esercizio 7 - Funzioni Es .6

$f(x) = 2x^4 - 11x^3 + 21x^2 - 16x + 4$   $tol=1e-10$ ,  $nmax=20$

Se  $x_0 = 0.75$  e  $x_1 = x_0 + 0.5$  dopo 6 iterazioni si trova la soluzione esatta  $zero = 1.000000e+00$   $fval = 0.000000e+00$

Per individuare la radice  $\alpha = 0.5$  scelgo  $x_0 = 0.4$  e  $x_1 = 0.6$

Il metodo converge in 9 iterazioni  $zero = 5.000000e-00$   $fval = 8.881784e-16$

Se  $x_0 = 1.25$  Il metodo converge in 5 iterazioni  $zero = 1.000000e+00$   $fval = 0.000000e+00$

Se  $x_0 = 1.75$ , il metodo non converge nel massimo numero di iterazioni Residuo finale =  $2.599424e-01$

Se  $x_0 = 2.25$  e  $x_1 = x_0 + 0.5$  dopo 34 iterazioni si trova la soluzione esatta  $zero = 2.000000e+00$   $fval = 0.000000e+00$

Se  $x_0 = 2.75$  e  $x_1 = x_0 + 0.5$  dopo 42 iterazioni si trova la soluzione esatta  $zero = 2.000000e+00$   $fval = 0.000000e+00$

## 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)` se `fun` è una function

### Input

---

<code>fun</code>	function 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'))
```

# fzero

## Algoritmo di Dekker-Brent

- Cerca un intervallo  $[a, b]$  in modo che  $f(a)f(b) < 0$ .
- Usa un passo delle secanti per trovare  $c$ .
- Ripete i passi seguenti finché  $|b - a| < \varepsilon|b|$  o  $f(b) = 0$ .
  - Ordina  $a$ ,  $b$  e  $c$  in modo tale che:

$$f(a)f(b) < 0, \quad |f(b)| < |f(a)|,$$

$c$  è il valore precedente di  $b$ .

- Se  $c \neq a$ , usa un passo IQI (Inverse Quadratic Interpolation).
- Se  $c = a$ , usa il passo delle secanti.
- Se IQI o le secanti forniscono un valore interno a  $[a, b]$ , lo accetta.
- Altrimenti, usa il metodo delle bisezioni.

## Utilizzo di fzero

`[x,fval]=fzero(fun,x0)` fornisce il valore della funzione `fun` nello zero `x`.

`[x,fval,exitflag]=fzero(fun,x0)` fornisce un valore `exitflag` che indica l'esito di `fzero`

Valore	Esito
--------	-------

---

1	convergenza verso la soluzione <code>x</code>
-1	l'algoritmo è interrotto da un <b>output function</b>
-3	sono stati trovati valori NaN o Inf
-4	sono stati trovati valori complessi durante la ricerca di un intervallo con cambio di segno
-5	<code>fzero</code> potrebbe essere arrivata a convergenza in un punto singolare
-6	<code>fzero</code> non trova un intervallo con cambio di segno.

## Esercizio 8

Usare la function `fzero` per risolvere gli esercizi 5 e 6, scegliendo gli stessi valori iniziali per  $x_0$  e confrontare i risultati ottenuti con il metodo di Newton.



## Soluzione Esercizio 8 - Funzioni Es. 5

$$f(x) = \sin x - \cos 2x \quad x_0 = 1 \text{ e } x_1 = 1.5$$

$$z = 0.5236 \quad fz = 3.8858e-16$$

intervaliterations: 10 iterations: 6 funcCount: 26

$$f(x) = x^3 - 7x^2 + 11x - 5$$

$$\text{Se } x_0 = 2 \quad z = 5 \quad fz = 0$$

intervaliterations: 13 iterations: 9 funcCount: 36

$$\text{Se } x_0 = 7 \quad z = 5 \quad fz = 0$$

intervaliterations: 8 iterations: 7 funcCount: 23

$$f(x) = x^4 - 12x^3 + 47x^2 - 60x + 24$$

$$\text{Se } x_0 = 0 \quad z = 0.8883 \quad fz = 0$$

intervaliterations: 11 iterations: 11 funcCount: 34

Se  $x_0 = 2$  Exiting fzero: aborting search for an interval containing a sign change because NaN or Inf function value encountered during search. (Function value at  $-1.48214e+77$  is Inf.) Check function or try again with a different starting value.

intervaliter: 522 iterations: 0 funcCount: 1044

## Soluzione Esercizio 8 - Funzioni Es. 6

$f(x) = 2x^4 - 11x^3 + 21x^2 - 16x + 4$   $\text{tol}=1e-10$ ,  $\text{nmax}=20$

Se  $x_0 = 0.75$   $z = 0.5000$   $fz = 0$

intervaliterations: 9 iterations: 9 funcCount: 27

Se  $x_0 = 1.25$   $z = 1.0000$   $fz = 0$

intervaliterations: 7 iterations: 4 funcCount: 18

Se  $x_0 = 1.75$   $z = 1.0000$   $fz = 0$

intervaliterations: 9 iterations: 5 funcCount: 23

Se  $x_0 = 2.25$   $z = 1.0000$   $fz = 0$

intervaliterations: 10 iterations: 6 funcCount: 26

Se  $x_0 = 2.75$   $z = 1.0000$   $fz = 0$

intervaliterations: 10 iterations: 4 funcCount: 24

## 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

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

$$F(x_k, y_k) = \begin{pmatrix} f(x_k, y_k) \\ g(x_k, y_k) \end{pmatrix}$$

# 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:

```
[z,fz,iter]=newtonsys (@f,@fd,x0,tol,Nit)
```

## 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>tol</code>	tolleranza desiderata;
<code>Nit</code>	numero massimo di iterazioni da eseguire;

## Output

---

<code>zero</code>	soluzione cercata;
<code>fz</code>	valore di $f$ nello zero calcolato;
<code>iter</code>	numero di iterazioni utilizzate;

# fsolve

Risolve i sistemi di equazioni non lineari in più variabili.

Appartiene al toolbox `optim`.

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

## Input

---

<code>fun</code>	nome della function che contiene la funzione $f$ <code>fun</code> accetta in input un vettore $x$ e dà in output il vettore dei valori di $f$ valutata in $x$ .
<code>x0</code>	dato iniziale

## Output

---

<code>x</code>	approssimazione dello zero calcolato
<code>fval</code>	valore di <code>fun</code> in <code>x</code> .



## 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`, `Diagnostics`, `DerivativeCheck`, `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.

## Opzioni per fsolve

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

fornisce in output le seguenti informazioni:

`exitflag` ha valore da -4 a 4. Se l'algoritmo 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 9

Applicare il metodo di Newton-Raphson per trovare gli zeri delle seguenti funzioni:

$$F_1(x, y) = \begin{pmatrix} x + y - 3 \\ x^2 + y^2 - 9 \end{pmatrix} \quad \begin{array}{l} \text{radici: } (0, 3) \ (3, 0) \\ x_0 = (1, 5), \ x_0 = (2, 3) \end{array}$$

$$F_2(x, y) = \begin{pmatrix} x^2 + y^2 - 2 \\ e^{x-1} + y^3 - 2 \end{pmatrix} \quad \begin{array}{l} \text{radice: } (1, 1) \\ x_0 = (1.5, 2), \ x_0 = (2, 3) \end{array}$$

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 10

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

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

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

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

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

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

# Problemi di minimo

segue

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  $z_2$ .

## Gestione dell'output

`disp(stringa di caratteri)`

visualizza la **stringa di caratteri** sullo schermo

`fprintf(fid, formato, variabili)`

scrive in un file **fid** il valore delle **variabili** con il **formato** assegnato. Se **fid** manca l'output è inviato sullo schermo.

`stringa=sprintf(formato, variabili)`

indirizza su una stringa di testo **stringa** l'output. Per visualizzare poi si usa `disp(stringa)`.

# Formato

---

## Codice    Azione

---

<code>%s</code>	formato stringa
<code>%d</code>	formato senza parte frazionaria
<code>%f</code>	formato numero decimale
<code>%e</code>	formato esponenziale
<code>%g</code>	formato in forma compatta usando <code>%f</code> o <code>%e</code>
<code>\n</code>	inserisce carattere di ritorno a capo
<code>\t</code>	inserisce carattere di tabulazione



## Esempio

Valore	%6.3f	%6.3e	%6d
2	2.000	2.000e+00	2
0.02	0.020	2.000e-02	2.000000e-02
200	200.000	2.000e+02	200
sqrt(2)	1.414	1.414e+00	1.414214e+00
sqrt(2)/10	0.141	1.414e-01	1.414214e-01

## Esempio

```
% tabella.m
% Realizza una tabella di valori
% del seno e del coseno
%
n=input('Inserisci il numero di valori:');
x=linspace(0,pi,n);
c=cos(x);
s=sin(x);
disp('-----')
fprintf('k\t x(k)\t cos(x(k))\t sin(x(k))\n');
disp('-----')
fprintf('%d\t %3.2f\t %6.5f\t %6.5f\n',[1:n;x;c;s]);
```

## Uso del file `tabella.m`

```
>> tabella  
>> Inserisci il numero di valori: 5
```

```
-----  
k           x(k)           cos(x(k))           sin(x(k))  
-----  
1           0.00           1.00000           0.00000  
2           0.79           0.70711           0.70711  
3           1.57           0.00000           1.00000  
4           2.36           -0.70711          0.70711  
5           3.14           -1.00000          0.00000
```

Nell'esempio i valori del seno e del coseno sono contenuti in vettori riga.

## 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]=newton(f,fd,x0,toll,Niter,output,p)
```

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

```
if nargin==5  
output=0; p=2  
elseif nargin==6  
p=2  
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

Si può prevedere che usando la function `newton` si voglia 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)
```

Inserisco nel programma prima della costruzione dei vettori `xk` e `fk` il seguente controllo:

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

Se uso il comando `[zero,fz,iter]=newton(f,fd,x0,toll,Niter)` la successione dei valori ottenuti non verrà memorizzata.

**Attenzione** le variabili in output devono comunque essere assegnate.