

# Metodo di Newton

## Laboratorio di Calcolo Numerico

Annalisa Buffa Fausto Cavalli

Corso di Laurea in Chimica

A.A. 2009-2010

# Function

Oltre ai file di tipo `script` in Matlab si possono creare anche **M-file** di tipo **Function**. Mentre un file di tipo `script` non ha nessuna intestazione, un file `function` deve iniziare con una dichiarazione del tipo

```
function [out1, out2, ..., outn] = nomefun(in1, in2, ..., inp)
```

dove:

- `function` è una parola chiave che fa parte dell'intestazione
- `[out1, out2, ..., outn]` è l'elenco delle variabili che vengono restituite dalla funzione
- `nomefun` è il nome della funzione
- `(in1, in2, ..., inp)` è l'elenco delle variabili che la funzione si aspetta in ingresso.

Le variabili che vengono definite in una `function` sono locali, nel senso che non possono essere lette nel `Workspace` della `Command Window`; analogamente una `function` non può leggere le variabili del `Workspace` della `Command Window`.

# Function

L'unico modo con cui si può comunicare con la Command Window sono le variabili in input ed in output.

## Attenzione

Un file di tipo `function` deve essere salvato con lo stesso nome dato alla `function` nell'intestazione, ovviamente con l'estensione `.m`

Per richiamare una `function`, basta richiamarne la sintassi (**SENZA** la parola chiave `function`). Per la `function` di prima si può scrivere

```
>> [out1, out2, ..., outn] = nomefun(in1, in2,...,inp)
```

## Attenzione

Quando lavoro dentro una `function`, le variabili che sono in input sono a disposizione del codice, per cui le posso utilizzare come se le avessi già a disposizione. Devo provvedere invece a creare tutte le variabili in uscita. Inoltre è bene che in una `function` le variabili in input non siano ridefinite o modificate all'interno della `function`, ma solo utilizzate.

# Function

Esempio: si voglia scrivere una funzione che si occupi di calcolare la somma, la media e la deviazione degli elementi di un vettore. Una soluzione può essere

```
function [somma,media,deviazione] = statistica(x)
n=length(x); % calcolo il numero di elementi
somma=sum(x);
media=somma/n;
deviazione=sqrt(sum((x - media).^2)/n); % ricordarsi il .^
```

Ora dopo averla salvata (con il nome di statistica.m) può essere richiamata nel seguente modo:

```
>> x=[-1 0 10 4 -5];
>> [a,b,c]=statistica(x)
```

dove a,b,c contengono rispettivamente la somma, la media e la deviazione degli elementi di x.

# Metodo di Newton

## Metodo di Newton

Il metodo di Newton è un metodo per la soluzione delle equazioni non lineari basato su una iterazione del tipo

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

## Esercizio

Si scriva la `function` di nome `newton` che, prendendo una stima  $x_0$  della soluzione, la funzione  $f$ , la sua derivata  $f'$ , una tolleranza ed un numero massimo di iterazioni, restituisca l'approssimazione della soluzione dell'equazione  $f(x) = 0$  data dal metodo di Newton, il numero delle iterazioni effettuate dal metodo ed un vettore che contenga le approssimazioni di volta in volta generate dal metodo. Il metodo andrà arrestato se  $|x_{k+1} - x_k| < toll$  oppure se il numero di iterazioni fatte supera  $nmax$ .

# Metodo di Newton

## Suggerimento

→ Si usino le seguenti variabili in input

- `f`, `df` per la funzione e la sua derivata
- `x0` per la stima  $x_0$
- `toll` per la tolleranza
- `nmax` per il numero massimo di iterazioni

ed in output

- `alfa` per l'approssimazione della soluzione
- `nit` per il numero di iterazioni
- `xk` per il vettore delle approssimazioni successive

→ L'intestazione della `function` sarà quindi

```
[alfa,nit,xk]=newton(f,df,x0,toll,nmax)
```

A questo punto conviene salvare la `function` con il nome di `newton.m`.

## Metodo di Newton

- Si inizializzano la variabile `nit=0`, che conterrà le iterazioni fatte, e `x=x0` che conterrà l'ultimo valore generato dal metodo di Newton
- Dato che Newton è definito solo se  $f'(x_k) \neq 0$  si effettui il controllo
  - Se  $df(x) == 0$  allora messaggio di errore (con `error`)
- si definisca `inc=1`, che poi controllerà l'incremento  $|x_{k+1} - x_k|$ .
- La parte principale sarà costituita da un ciclo del tipo:
  - ripetere finchè `inc>toll` e `nit<nmax`
    - Aggiornare `x`, calcolando l'iterazione del metodo di Newton  $x - f(x)/df(x)$
    - Ripetere il controllo su  $f'(x_k) \neq 0$ 
      - Se  $df(x) == 0$  allora fornire un messaggio di errore
    - aggiornare il valore di `inc`: si noti che  $|x_{k+1} - x_k| = |f(x_k)/f'(x_k)|$ , per cui si può porre `inc=abs(f(x)/df(x))`;
    - aggiornare `nit` incrementandone il valore di 1
    - aggiungere a `xk` il nuovo valore `x` con `xk(nit)=x`
  - Definire `alfa=x`

# Metodo di Newton

## Applicazione

Si utilizzi la function `newton` appena scritta per calcolare le soluzioni dell'equazione  $x^4 - 2 = 0$ .

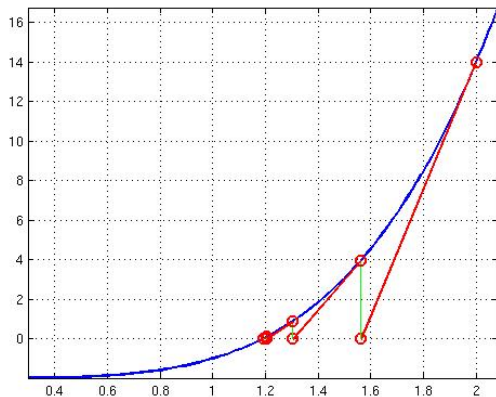
Si usino per esempio `toll=1e-10` e `kmax=100`. L'equazione ha due soluzioni pari a  $\pm\sqrt[4]{2}$ . Il metodo ha il seguente comportamento:

- se la stima iniziale è  $x_0 > 0$  ho che il metodo converge sempre alla soluzione positiva  $+\sqrt[4]{2}$ ,
- se parto con  $x_0 < 0$  ho che il metodo converge sempre alla soluzione negativa  $-\sqrt[4]{2}$ .
- il valore  $x_0 = 0$  non è accettabile dato che  $f'(0) = 0$ .

Per questa particolare equazione il metodo converge quindi per qualunque valore accettabile di  $x_0$ .

# Metodo di Newton

La figura seguente mostra graficamente il comportamento delle iterazioni del metodo di Newton



# Metodo di Newton

## Applicazione

Si utilizzi la function `newton` appena scritta per calcolare le soluzioni dell'equazione  $x^3 - x + 3 = 0$ .

Si usino per esempio `toll=1e-10` e `nmax=100`. L'equazione ha una soluzione vicino a  $-1.6$  Il metodo ha il seguente comportamento:

- se la stima iniziale è vicina alla soluzione ho che il metodo converge
- se parto con una stima iniziale lontana dalla soluzione, per esempio con  $x_0 = 0$ , potrei non avere convergenza

Per questa particolare equazione il metodo converge solo localmente, cioè la convergenza è garantita solo partendo sufficientemente vicino alla soluzione.

# Metodo di Newton

Come posso scegliere una stima iniziale vicina alla soluzione se la soluzione non la conosco? Innanzitutto è opportuno uno studio grafico della funzione, per esempio con i seguenti comandi

```
>> x=linspace(-2,2);  
>> f=inline('x.^3-x+3');  
>> plot(x,f(x));  
>> grid on    %aggiunge una griglia sullo sfondo
```

Poi è possibile, una volta individuato un intervallo in cui è applicabile, usare il metodo di bisezione con una tolleranza relativamente alta (tipo  $\text{toll}=1e-1$ ) in modo da ottenere una stima sufficientemente vicina alla soluzione. Questa stima può essere quindi usata per innescare il metodo di Newton e convergere in poche iterazioni. Perché non utilizzare solo il metodo di bisezione? Perché è un metodo che converge lentamente, la cui utilità si recupera proprio se usato in sinergia con metodi più veloci come quello di Newton.

# Metodo di Newton

## Applicazione

Usando la function `newton` confrontino le velocità di convergenza del metodo di Newton relativamente all'approssimazione delle due soluzioni dell'equazione  $x^3 - x^2 - x + 1 = 0$  (soluzione  $x_1 = -1$ ,  $x_2 = 1$ ).

Si usino per esempio `toll=1e-8` e `nmax=100`. Per approssimare le soluzioni posso preparare uno script (per esempio con il nome `zeri_newton.m`) che contenga i seguenti comandi

```
f=inline('x.^3-x.^2-x+1');  
df=inline('3*x.^2-2*x-1');  
x0=-2;      %cerco la prima soluzione  
toll=1e-8;  
nmax=100;  
[alfa1,it1,xk1]=newton(x0,f,df,toll,nmax);  
x0=2;      %cerco la seconda soluzione  
[alfa2,it2,xk2]=newton(x0,f,df,toll,nmax);
```

# Metodo di Newton

Per calcolare gli errori che via via commetto posso definire le due variabili

```
>> err1=abs(xk1+1);  
>> err2=abs(xk2-1);
```

dove ho fatto la differenza fra il vettore delle approssimazioni intermedie e la soluzione esatta. A questo punto posso provare a disegnare i due andamenti con il comando `plot`, dove sulle ascisse vado a mettere le iterazioni. Posso disegnare più grafici con lo stesso comando `plot` semplicemente elencandoli tutti

```
>> plot(1:it1,err1,1:it2,err2);
```

Il grafico che si ottiene non è molto chiaro, dato che sulle ordinate i valori cambiano di diversi ordini di grandezza. Quando ciò succede si può scegliere una scala diversa per le ordinate, di tipo logaritmico. In matlab questo si può ottenere con il comando **semilogy**, la cui sintassi ed uso è identica al comando `plot`. Provando con

```
>> semilogy(1:it1,err1,1:it2,err2);
```

# Metodo di Newton

si può vedere che per la prima soluzione ho una convergenza più rapida (curva blu che decresce come una parabola), mentre per la seconda soluzione, la curva di convergenza è una retta (curva verde). Disegnando la funzione si capisce il motivo:

```
>> x=linspace(-2,2);  
>> f=inline('x.^3-x.^2-x+1');  
>> plot(x,f(x));  
>> grid on
```

si vede che la funzione ha uno zero semplice in  $x_1 = -1$  mentre ha uno zero multiplo in  $x_2 = 1$  (la curva è tangente all'asse  $x$ ). Quando una radice è multipla allora il metodo di Newton, che solitamente è del secondo ordine, decade al primo ordine.