//ALGORITMO BISEZIONE
//faccio una function che chiamo bisezione

----------SCRIPT – bisezione----------

//
OUTPUT(come per gs e jacobi):
il vettore delle soluzioni(x),
il numero di iterazioni fatte dall'algoritmo(k),
INPUT:
funzione dell'equazione non lineare(fun),
l'intervallo in cui la funz dovrebbe cambiare di segno(a,b),
una tolleranza per vedere quando l'intervallo si rimpicciolisce abbastanza(tau)
il numero massimo di iterazioni(kmax)

```
function [x,k] = bisezione(fun,a,b,tau,kmax)
```
//verifichiamo che la funzione cambi di segno agli estremi facendo la moltiplicazione del valore della funzione all'estremo a per il valore che prende all'estremo b. Se è positivo, significa che i segni ai due estremi sono rimasti gli stessi

```
fa = fun(a);
fb = fun(b);

if fa*fb > 0
    error('La radice non e'' contenuta in [a,b]') //il messaggio di errore interrompe la
                                                    funzione
end
```
//se superiamo questo if, significa che la radice della funzione è contenuta in questo intervallo.
ora calcoliamo il punto medio
```
c = (a+b)/2;
fc = fun(c); //calcoliamo il valore della funzione nel punto medio
k = 1;

if abs(fc) < tau //CASO 1: f(c) è la radice della funzione -> è molto vicino a 0,
                                quindi inferiore della soglia imposta da noi tau
    x = c; //quindi c è la nostra soluzione, la restituisco
    return //con il return interrompo l'esecuzione e restituisce gli output
end
```

//se neanche c è la soluzione, allora devo iterare il metodo:

```
flag = 1;
while flag
```
//devo controllare se fc ha lo stesso segno di fa o di fb
```
if fa*fc < 0 //se fc ha segno discorde da fa, allora c prende il posto dell'estremo b
    b = c;
    fb = fc;
else //se invece fc ha lo stesso segno di a, allora c prende il posto dell'estremo a
    a = c;
    fa = fc;
end
k = k+1;
c = (a+b)/2;
fc = fun(c);

flag = (abs((b-a))>tau) && (abs(fc)>tau) && (k<kmax);
```

//criteri di arresto:
-l'intervallo diventa più piccolo di una certa tolleranza
-troviamo la radice della funzione in un punto medio
-raggiungiamo il max delle iterazioni
end

```
x = c; //devo aggiornare l'output della soluzione con un ulteriore punto medio per ottenere l'approssimazione
```

//inoltre, se la radice era presente nell'intervallo ma raggiungo il numero max di iterazioni raggiunto, posso inserire un messaggio per avvisare l'utente del motivo per cui si è usciti dal while

```
if abs(fc)>tau
    if k>=kmax
        fprintf('Numero max di iterazioni raggiunto')
    end
end
```

//creiamo un test per poter testare il metodo di bisezione

----------SCRIPT: test bisezione----------

//utilizziamo un function handle: mi serve la funzione

Il function handle si inizializza in questo modo:
- -fun: nome della function handle
- -@(variabili da cui dipende la funzione)
- - sin(3.*x) - x.^2 + 2.*x + 3 : funzione che ci interessa calcolare
  N.B.: ogni operazione va scritta con il punto, perché se andasse applicata su vettore senza il punto, farebbe operazioni su vettori e non sugli elementi

```
fun = @(x) sin(3.*x) - x.^2 + 2.*x + 3;
a = 3; //estremi dell'intervallo in cui è contenuta la radice
b = 4;
tau = 1e-7; //imposto anche i criteri di stop
kmax = 100;

%[x,k] = bisezione(fun,a,b,tau,kmax);
```

```
//al RUN di test_bisezione ottengo:

k=17

x=3.05
//quindi in 17 passaggi è riuscito ad ottenere un valore distante da quello reale meno di 10^-7
```

----------SCRIPT: newton----------

//questa volta non abbiamo bisogno di un intervallo iniziale, ma di:

-la funzione(fun)
-la derivata della funzione(fund)
-un test iniziale (x0)
-i soliti criteri di stop(tau, kmax)

```
function [x,k] = newton(fun,fund,x0,tau,kmax)

// la prima cosa che controllo è che il punto x0 non sia la radice
if abs(fun(x0))<tau
    x = x0;
    return;
end
//verifico che la derivata non si annulli nel punto x0 e do errore
if abs(fund(x0))<tau
    error('la derivata si annulla in x0!')
end

x_new = x0-(fun(x0)/fund(x0)); //faccio il primo calcolo di x nuovo con Newton prima dell'inizio dell'iterazione
k = 1;
flag = 1;
//utilizzo la formula di newton iterata per trovare il nuovo punto approssimato
while flag
    k = k+1;
    x0 = x_new; //utilizzo x nuovo come x di riferimento
//calcoliamo la formula di newton, che è xk-fxk/f'xk quindi mi serve solo il punto precedente, e non tutti i punti della sequenza che andiamo a produrre. Lavoreremo quindi solo con 2 punti, l'x0 iniziale, l'x nuovo che calcoliamo con la formula, e se quest'ultimo non va bene allora sostituiamo xo con x nuovo e ricalcoliamo un x nuovo con la formula, e così via.
    x_new = x0-(fun(x0)/fund(x0)); //calcoliamo x nuovo
    flag = (abs(x_new-x0) > abs(x0)*tau) && (abs(fun(x_new)) > tau) && (k<kmax);
end
//per Newton è sufficiente che due iterazioni successive diano risultati molto vicini, quindi lo usiamo come criterio di uscita tramite lo scarto relativo abs(x_new-x0)/abs(x0)<tau(ricorda che il criterio del while è per il rientro nel ciclo). Porto abs(x0) dall'altra parte per evitare di avere la frazione
if k>kmax
    fprintf('Num max di iterazioni raggiunto')
end

x = x_new; //infine inserisco nel parametro in output il valore di x nuovo trovato
```

//modifico il test di bisezione per poterlo utilizzare anche per Newton

----------SCRIPT: test bisezione----------

```
fun = @(x) sin(3.*x) - x.^2 + 2.*x + 3;
fund = @(x) 3.*cos(3.*x) - 2.*x + 2; //inserisco la function handle per la derivata di fun
a = 3;
b = 4;
x0 = 4; //inserisco il punto di partenza x0
tau = 1e-7;
kmax = 100;

[x,k] = newton(fun,fund,x0,tau,kmax); //applico la funzione newton
```

//ESERCIZIO CONSIGLIATO: metodo delle corde e metodo delle secanti:

//cosa cambia:
-per le corde non serve più la derivata in ingresso, ma al suo posto si usa un denominatore dato in chiamata(fuori dal test)
-per le secanti servono 2 punti di innesco(x0 e x1), e m si calcola dentro la funzione con la formula m=(f(x1)-f(x0))/(x1-x0)

```
function [x,k] = corde(fun,m,x0,tau,kmax)
function [x,k] = secanti(fun,x0,x1,tau,kmax)
```

//dentro il while ci sarà (e fuori, alla prima iterazione) per entrambe le funzioni
```
x_new = x0 - (fun(x0)/m)
```

//inoltre, ricordare che per le secanti ci sono più scambi da fare tra x0,x1 e x_new:

```
x0=x1;

x1=x_new
```

//e poi calcolare x_new con la formula sopra

//INTERPOLAZIONE: ho delle coppie di punti e vogliamo il polinomio che passa per questi punti in forma canonica o di lagrange
Dato che non è stato fatto a lezione il calcolo simbolico con matlab per rappresentare il polinomio (le x per matlab sono dei contenitori di valori, non dei simboli),viene fatto un campionamento (con il comando linspace()) al posto delle x e verrà poi plottato.

//Per poter usare questo metodo, la funzione deve:
-avere una griglia di punti in input per fare il plot
-deve restituire quanto vale il polinomio in quella griglia di punti, sempre per il plot

//la mia funzione di interpolazione mi restituirà i valori che mi servono per il grafico, che chiamo yy.
quindi mi restituisce i pn(x) calcolati in una griglia abbastanza fitta dell'intervallo

----------SCRIPT: canint----------
//per la funzione servono in input:
-i dati da interpolare (x e y)
-il campionamento dell'asse x per fare il grafico (xx), ovvero un vettore ottenuto con il comando linspace
```
function yy = canint(x,y,xx)
```

//per trovare il polinomio in forma canonica,bisogna risolvere un sistema lineare che ha come matrice dei coefficienti una matrice di Vandermonde (la prima colonna =1, la seconda ha le x^1, la terza x^2 etc) e i termini noti sono le y di interpolazione

$$\begin{vmatrix} 1 & x & x^2 & x^3 \\ 1 & y & y^2 & y^3 \\ 1 & z & z^2 & z^3 \\ 1 & t & t^2 & t^3 \end{vmatrix}$$

//devo assicurarmi che:

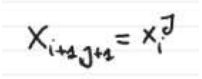
-x e y siano dei vettori: utilizzo la forma x: e y:, che se applicato ad un vettore colonna non fa nulla, se applicato ad un vettore riga lo trasforma in un vettore colonna, e se lo applico ad una matrice prende le colonne della matrice e le mette una sopra l'altra.

-x e y abbiano la stessa lunghezza: si misura con length(), che restituisce il numero di elementi del vettore, al contrario di size() che restituisce il numero di righe e il numero di colonne

```
x = x(:);
y = y(:);

n = length(x);

if length(x) ~= length(y)  //tilde(~): alt+126 su windows
```

il simbolo ~= indica il simbolo di "diverso"

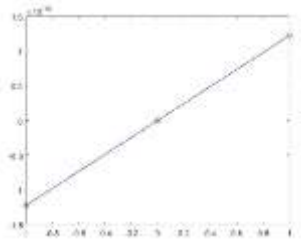
```
    error('x e y hanno lunghezza diversa')
end
//PASSO1:  creo la matrice di Vandermonde:
X = zeros(n);  //creo la matrice della stessa dimensione della lunghezza di x e y, popolata solo da zeri.
               //riempio la matrice di Vandermonde con il for che si riferisce alla formula :
```



esempio: per l'elemento(1,1) avrò che i=0 e j=0, e l'elemento sarà x_i^j=x_0^0=1

```
//creo la matrice colonna per colonna.
for j = 1:n //faccio variare j da 1 a n
    X(:,j) = x.^(j-1); //elevo ,per ogni elemento (:) di tutti i vettori colonna (da 1
a n) la soluzione(che cambia per ogni riga) il valore (j-1)
end
//PASSO2: risolvo il sistema con il backslash a=X^-1/y, dove a è la soluzione del
sistema lineare
a = X\y;
//Nota: il polinomio p viene scritto in questo modo nella teoria:
p = a0*x^0 + a1*x^1 + ... + an*x^n
ma gli indici in matlab non iniziano con 0, ma con 1. Quindi la scrittura del polinomio
sarà fatta così:
p = a(1)*x^0 + a(2)*x^1 + ... + a(n)*x^(n-1)
//quindi il pedice di a è esponente-1

//PASSO3: ricavo tutti gli y dei campionamenti contenuti nel vettore xx:

for i = 1:length(xx) //con questo for scorro il vettore campionamento xx

s = 0;
for k = 1:n //per ricavare tutte le ordinate corrispondenti alle ascisse del vettore
campionamento, devo scorrere tutte le componenti del polinomio (calcolato in quel
valore xx) e fare la somma tra loro
    s = s + a(k)*xx(i)^(k-1); //sommo ogni termine alla somma precedente

end
yy(i) = s; //la somma appena calcolata rappresenterà l'elemento i-esimo del vettore
delle ordinate campionate(yy)
end
//PASSO4: costruiamo il plot:
-xx campionamento fitto per fare il grafico con una forma accettabile
-yy ordinate ricavate dal campionamento xx
-x valori di ascissa di partenza in cui voglio che passi il polinomio
-y valori di ordinata di partenza in cui voglio che passi il polinomio
```

```
//nel plot imposto 2 curve: una blu con i dati interpolati e una rossa a pallini con i
dati x-y di partenza
plot(xx,yy,'b',x,y,'ro')

-----------------------------------------------------------------------

//affianchiamo un test alla funzione di interpolazione in forma canonica:

----------TEST: interpolazione----------
//ricaviamo i valori di x e y da interpolare E da passare alla funzione canint
NOTA: linspace() crea un vettore riga! Bisogna trasporlo per avere un vettore colonna
x = linspace(-1,1,3)';  %equispaziati
y = sin(pi*x);

//creiamo il vettore delle ascisse campionate da passare a canint
xx = linspace(-1,1,100)';

yy = canint(x,y,xx);

-----------------------------------------------------------------------
```

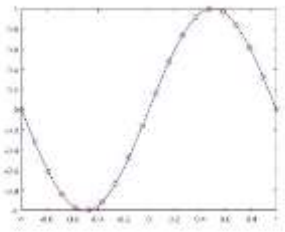


//il grafico che esce è una retta, perché ci sono troppe poche coppie di punti di partenza(x,y) assegnati (solo 3).

Se modifico il linspace di generazione di x con:

```
x = linspace(-1,1,20)';
```

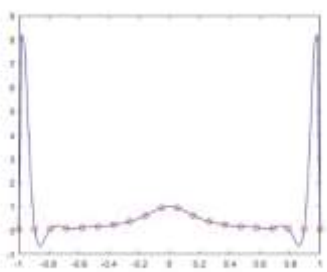
ottengo:



//se prendessi una funzione più irregolare, ad esempio y=1./(1+25.*x.^2):

```
----------TEST: interpolazione----------
x = linspace(-1,1,20)';
y = 1./(1+25.*x.^2); //funzione non regolare
xx = linspace(-1,1,100)';

yy = canint(x,y,xx);

-----------------------------------------------------------------------
```

Ottengo:



questa oscillazione segnata dalla linea interpolata non esiste nella linea della funzione reale. E' data dal fatto che l'errore non tende a 0.
E' una funzione di classe c0, la sua derivata non è continua,e quindi la distribuzione dei nodi equispaziata non fa tendere l'errore a 0.

Quindi, invece che usare i punti equispaziati, dobbiamo usare i nodi di Chebychev, e quindi calcolare le x che derivano dalle radici del polinomio di

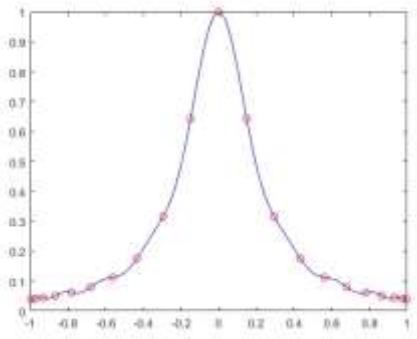
Chebychev:

----------TEST: interpolazione----------

```
%x = linspace(-1,1,20)';  %equispaziati
//calcoliamo un vettore colonna k con elementi che vanno da 0 ad un n prefissato
n = 20;
k = (0:n)';
x = cos((2*k+1)./(2*(n+1))*pi); //scrivo il polinomio di Chebychev
//Quindi, per ogni k che va da 0 ad n ho un'ascissa di interpolazione data dal
polinomio di Chebychev
%y = sin(pi*x);
y = 1./(1+25.*x.^2); //ora calcolo le ordinate con la funzione non regolare
xx = linspace(-1,1,100)';

yy = canint(x,y,xx);
```

```
//il nuovo plot avrà questa forma:
```

non abbiamo più l'oscillazione esagerata del grafico precedente



```
//La funzione Lagrint implementata qui sotto, fa la stessa cosa con il polimonio di
Lagrange. Il plot del polinomio è uguale, l'implementazione della funzione è
differente:
```

----------SCRIPT: Lagrint----------

```
function yy = lagrint(x,y,xx)

//ricordiamo che il polinomio caratteristico di Lagrange ha questa forma:
Lj(x) = ((x-x1)/(xj-x1))*((x-x2)/(xj-x2))*...*((x-xn)/(xj-xn))
Inoltre, per ogni Lj(x) calcolata, otteniamo un elemento del polinomio interpolatore
nella forma di Lagrange che ha questa forma:
p(x) = y(1)*L0(x) + y(2)*L1(x) + ... + y(n)*Ln-1(x)

x = x(:);//mi assicuro che x e y siano vettori colonna
y = y(:);

n = size(x,1); //size restituisce un vettore riga di un solo elemento contentente il
numero di righe del vettore x. Questo sarà anche il numero di L da calcolare.
den = zeros(n,1); //den sarà un vettore colonna di n elementi pari a 0

//calcolo questa formula:
```

$$L_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} = \frac{x - x_0}{x_j - x_0} \cdot \frac{x - x_1}{x_j - x_1} \cdot \dots \cdot \frac{x - x_n}{x_j - x_n} \qquad j = 0, \dots, n$$

```
//CALCOLO DEL DENOMINATORE:

for j = 1:n //scorro tutto il vettore den
```

```
//per ogni elemento di den, eseguo la seguente formula:

    den(j) = prod(x(j) - x([1:j-1,j+1:end])); //rappresenta il denominatore
end

//CALCOLO DEL NUMERATORE:
m = size(xx,1); //ricavo la lunghezza del vettore interpolante xx
yy = zeros(m,1); //creo il vettore dei valori delle ordinate ottenute tramite
interpolazione, popolato di zeri

for i=1:m //scorro il vettore xx
    num = prod(xx(i)-x)./(xx(i)-x); //questo rappresenta il numeratore
    L = num./den; //ricavo tutti gli L del polinomio
    yy(i) = y'*L; //moltiplico il vettore delle y fornite con il vettore L di lagrange,
elemento per elemento per ottenere le ordinate interpolate
end

plot(x,y,'or',xx,yy,'b-')
legend('dati da interpolare','polinomio interpolante')

-------------------------------------------------------------------
```