Matlab

sign	*segno*	abs	*valore assoluto*
sqrt	*radice quadrata*	sin	*seno*
cos	*coseno*	tan	*tangente*
cotan	*cotangente*	asin	*arcoseno*
acos	*arcocoseno*	exp	*esponenziale*
log	*logaritmo naturale*	log10	*logaritmo in base 10*
log2	*logaritmo in base 2*	round	*arrotondamento*
floor	*parte intera*	ceil	*parte intera + 1*

+Inf	*+ infinito (es. 5/0)*	-Inf	*- infinito (es. -5/0)*
NaN	*numero indefinito (es. 0/0)*	eps	*precisione di macchina*
pi	*pi greco*	i	*unità immaginaria*

Comandi di base

- a = 0 → assegno ad a il valore 0
- clc → cancella il prompt dei comandi
- clear all → cancella prompt e variabili salvate
- disp(nome_var) → stampa nome variabile
- %_____ → commento
- s = ‘ciao’; → stringa ciao assegnata alla variabile s
- fprintf(’testo’, variabile); → stampa una stringa in output e una variabile dopo la virgola
- \n → a capo
- \t → tabulazione
- format long → permette di mostrare più cifre decimali rispetto alle impostazioni standard di matlab
- format short → meno cifre decimali rispetto allo standard

Operazioni sui vettori

- v = [1,3,4] → vettore riga
- v = [1;3;4] → vettore colonna
- v = [1,2,3; 4,5,6] → vettore 2 righe di 3 elementi

- u = v’ → trasforma v da vettore riga a vettore colonna
- x = 1:10 → crea un vettore con step = 1 da 1 a 10
- linespace(a,b,n) → crea un vettore di n elementi spaziati da a a b tipo (0,10,10) → 0,1,2,3,4….
- linespace(a:s:b) → vettore che inizia con a, incrementa ogni volta di s, e termina quando il valore è ≤ di b
- u = [v,w] → concatena i vettori riga v e w
- u = [v;w] → concatena i vettori colonna v e w
- zeros(m,n) crea una matrice m × n di zeri
- ones(m,n) crea una matrice m × n di uni
- eye(n) crea una matrice identità n × n
- length(v = vettore) → restituisce la dimensione del vettore v
- size(a) → ritorna la dimensione di a
- size(a,n) → ritorna la dimensione di a se a è multidimensionale, n infatti serve per specificare la seconda dimensione del vettore
- v(i,j) → accede all’elemento i,j del vettore v

Operazioni su vettori con le stesse dimensioni

- v+w restituisce la somma dei due vettori
- v-w restituisce la differenza dei due vettori
- v.*w restituisce il prodotto puntuale dei vettori (infatti c’è il punto)
- v./w restituisce la divisione puntuale dei vettori (infatti c’è il punto)
- v.^w restituisce l’elevamento a potenza puntuale dei vettori

Funzioni personalizzate

- nome_variabile = @(Variabili) Funzione(Variabili)

```
f = @(x) (sin(x)).^2 + 2.*log(x.^2+1)
```

Plot

Permette di sovrapporre più grafici contemporaneamente

- plot(x,y,x,z); → permette di sovrapporre più grafici contemporaneamente
- figure → crea un grafico della funzione in una nuova finestra

Programmazione varia

If - else

```
if condizione
  corpo
else
  corpo
end
```

for

```
for c = 1:10
  corpo
end
```

Esercizi vari

```
%Scriviamo uno script che assegna il numero 2 alla variabile a ed il numero
%3π alla variabile b e che calcola successivamente il seno del prodotto di
%queste variabili, stampando poi a schermo il risultato.

a = 2;
b = 3*(pi);
c = sin(a*b);
disp(c);
```

```
%{
Produrre uno script dove vengono definiti tre vettori u, v e w di dimensione
1 × 5. Si definisca v con componenti equispaziate in [0, 1] (gli altri possono
essere definiti a piacere). Si calcoli poi prima il prodotto scalare p tra u e v
e poi si definisca un nuovo vettore z ottenuto moltiplicando w per lo scalare
p trovato. Infine, data A=[0,2,-1,2,0;1,1,1,0,0] si calcoli e stampi a
schermo il prodotto matrice-vettore tra A e z.
}%
u = zeros(1,5);
w = ones(1,5);
v = linspace(0,1,5);
```

```
p = u*v';
z = p .*w;
A = [0,2,-1,2,0;1,1,1,0,0];

disp(A*z');
disp(p);
disp(z);
```

```
%{Sia f (x) = sin(x) nel dominio I = [0, 2π]. Consideriamo il polinomio di
Taylor di grado 3 p(x) = x − x
3/6. È noto che tale polinomio approssima
f in un intorno di 0. Vogliamo descrivere l'errore assoluto |f (x) − p(x)| per
x ∈ I. Rappresentiamo graficamente nel dominio I.
1) Le funzioni f e p, nello stesso grafico.
2) La funzione err(x) = |f (x) − p(x)|.
3) Ancora err(x) ma in scala semilogaritmica
}%

f = @(x) sin(x);
p = @(x) x - (x .^3) ./6;

err =  @(x) abs(f(x) - p(x));

x = linspace(0,2*pi,300);

y_1 = f(x);
y_2 = p(x);
y_3 = err(x);

plot(x,y_1,x,y_2);
plot(x,y_3);
semilpgy(x,y_3);
grid on
```

Laboratorio - comandi utili

Formulazione di successioni con formule a partire da n+1

el ricorrenza

$$x_2 = 2, \qquad x_{n+1} = 2^{n-1/2}\sqrt{1-\sqrt{1-4^{1-n}x_n^2}}, \ n = 2, 3, \ldots$$

```
metodo2(1) = 2;
for i = 3:n
    metodo2(i-1) = 2^((i-1)-1/2)*sqrt(1-sqrt(1-4^(1-(i-1))*metodo2(i-2)^2));
end
```

$$x_{n+1} = \frac{\sqrt{2}x_n}{\sqrt{1+\sqrt{1-4^{1-n}x_n^2}}},\ n = 2, 3, \ldots.$$

```
metodo3(1) = 2;
for i = 3:n
    metodo3(i-1) = sqrt(2) * metodo3(i-2) / sqrt(1 + sqrt(1 - 4^(1 - (i-1))*metodo3(i-2)^2));
end
```

Equazioni starane e dove trovarle

$$\frac{1}{\sqrt{\lambda}} = -2\log_{10}\left(\frac{e}{3.51 \cdot d} + \frac{2.52}{N_R\sqrt{\lambda}}\right)$$

```
% risolviamo x=-2*log10( (e/3.51*d) + 2.52*x/NR ) (per x ≥ 0) e poi
% poniamo x=1/sqrt(lambda) ovvero lambda=1/x^2.
phi=@(x) -2*log10( (e/3.51*d) + 2.52*x/NR );
[solF, xvF, stepF] = puntofisso (phi, x0, toll, maxit);
```

Plottare funzione senza @(x) in un intervallo e creare la relativa figura

```
f = x +1 ;
x = linspace(..);

figure(1)
plot(x,f);
```

plottare funzione con @(x)

```
x = linspace(..)
f = @(x) ....

err_rel = abs(f_esatta - f(x)) / abs(f_esatta)

figure(1)
plot(x, f(x))
```

Creare figura con leggenda e nomi assi

```
f = x +1 ;
x = linspace(..);

figure(1)
plot(x,f);
title('grafico f(x)');
xlabel('x');
ylabel('valore di f(x)');
```

Stampare risultati valutazioni con formati specifici

```
fprintf('..... %10.19f, ris) %floating 10 cifre mantissa, 19 parte frazionaria
fprintf('..... %2.2e, ris) %esponenziale 2 cifre mantissa, 2 parte frazionaria
```

Ricerca zeri funzioni

Residuo non pesato bisezione

|f(xk-1) | ≤ toll → il valore assoluto di f(x) dell’iterazione precedente deve essere ≤ della tolleranza

Metodo di bisezione

Il metodo di bisezione genera una successione di intervalli $[a_k, b_k]$ con

- $f(a_0) \cdot f(b_0) < 0$
- $[a_k, b_k] \subset [a_{k+1}, b_{k+1}]$
- $|b_k - a_k| = \frac{1}{2}|b_{k-1} - a_{k-1}|$.

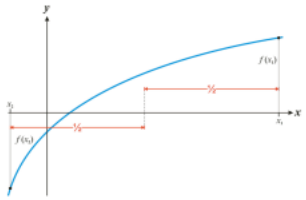

Figure: Bisezione di un intervallo.

Dato $a_{k-1} < b_{k-1}$ e $x_{k-1} = \frac{a_{k-1}+b_{k-1}}{2}$, allora il nuovo intervallo $[a_k, b_k]$ è definito come

$$a_k = a_{k-1}, b_k = x_{k-1} \text{ se } f(a_{k-1}) \cdot f(x_{k-1}) < 0,$$
$$a_k = x_{k-1}, b_k = b_{k-1} \text{ se } f(x_{k-1}) \cdot f(b_{k-1}) < 0.$$

Utilizzeremo quindi il residuo non pesato come criterio di arresto

$$|f(x_{k-1})| \leq \texttt{tol}.$$

Per la convergenza ad una radice è necessario individuare un intervallo $[a, b]$ della funzione f tale che

- $f(a) \cdot f(b) < 0$,
- f sia strettamente monotona in $[a, b]$.

Calcolare errore relativo bisezione al variare delle iterate

```
[x_bis,xall,iter] = bisezione(f,0,1,toll,1000);

x_esatta = 0.4428544010023885;

err_rel = abs(xall-x_esatta) / abs(x_esatta);

figure(2)
semilogy(1:iter, err_rel);
```

Residuo non pesato bisezione

```
while (abs(f(x)) > tol)
```

Residuo pesato bisezione

```
w = (b-a) / (f(b) - f(a));
while (abs(w*f(x))> tol) && (iter <  max_iter
    w = (b-a) / (f(b) - f(a));
```

Trovare radice x* funzione con metodo del punto fisso

```
x = linspace(0,1,100);
f = @(x) sin(x) + x -1;

figure(1)
plot(x,f(x));
grid on;
title('grafico della funzione in corrispondenza dello zero');

x0 = 0.5; %lo vedo dal grafico
```

Trovare equazione g da f con il metodo del punto fisso

Devo isolare la x

es:

$\cos(x) - x = 0 \rightarrow - x = - \cos(x) \rightarrow x = \cos(x)$

$f(x) = \sin(x) + x -1 \rightarrow - x = \sin(x) -1 \rightarrow x = -\sin(x) +1$

Metodo Newton

Metodo di Newton

Il metodo di Newton è un metodo di punto fisso con funzione di iterazione

$$g(x) = x - \frac{f(x)}{f'(x)}$$

per la ricerca dello zero x^* di una funzione f.
Ovvero, l'iterazione è

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

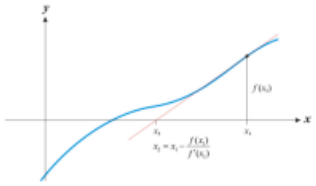

Figure: Interpretazione geometrica dell'iterazione di Newton.

La funzione g ha derivata

$$g'(x) = \frac{f(x)f''(x)}{f'(x)^2},$$

e essa si annulla nello zero x^*, infatti la convergenza di questo metodo è quadratica, ovvero ha ordine di convergenza $p = 2$ (se $f'(x^*) \neq 0$, ovvero se x^* è uno zero semplice di f).

Iterata normale Newton

```
%Prima iterazione del metodo di Newton
dx = -f(x0)/Df(x0);    % primo incremento
x = x0+dx;             % prima iterata
iter = 1;
xall(iter) = x;

while (abs(dx) > tol) && (iter < max_iter)          % ciclo iterativo
  x0 = x;
  if Df(x0) == 0
      break
  end
  dx = -f(x0)/Df(x0);                                % nuovo incremento
  x = x0 + dx;                                       % nuova iterazione
  iter = iter + 1;                                   % nuovo numero di iterazione
  xall(iter) = x;
end
```

Iterata con molteplicità Newton

```
% Prima iterazione del metodo di Newton
dx = -molt*f(x0)/Df(x0);    % primo incremento
x = x0+dx;             % prima iterata
iter = 1;
xall(iter) = x;
```

```
while (abs(x-x0) > tol*abs(x)) && (iter < max_iter)           % ciclo iterativo
  x0 = x;
  dx = -molt*f(x0)/Df(x0);                                  % nuovo incremento
  x = x0 + dx;
  iter = iter + 1;
  xall(iter) = x;% nuova iterazione
                                    % nuovo numero di iterazione

end
```

Iterata newton con rapporto incrementale(secante)

```
%Primo incremento
dx = -f(x1)*(x1-x0)/(f(x1)-f(x0));
x = x1+dx;          %Prima iterazione
iter = 1;
xall(iter) = x;

while (abs(dx) > tol) && (iter < max_iter)           % ciclo iterativo
  x0 = x1;
  x1 = x;
  dx = -f(x1)*(x1-x0)/(f(x1)-f(x0));                  % nuovo incremento
  x = x1 + dx;                                        % nuova iterazione
  iter = iter + 1;                                    % nuovo numero di iterazione
  xall(iter) = x;
end
```

Interpolazione

Comandi lagrange

repmat(x,1,n-1) → copia il vettore x, dalla riga 1 alla colonna n-1

repmat(z([1:i-1,i+1:n],m,1) → copia il vettore z (1, i-1, i+1,n) dalla riga m alla colonna 1

prod(z(i)-z([1:i-1,i+1:n]) → fa il prodotto della differenza fra (z(i)-…)

polyfit(x,y,m) → genera i coefficienti del polinomio interpolatore tramite i vettori di ascisse x e y in cui si associa il polinomio p_n, m è la dimensione di x -1

a:h:b : genera n+1 punti equispaziati in [a,b] con passo h = b-a/n

polyval(coeff, s) → valuta i coefficienti generati con polyfit sul vettore di ascisse s su cui valutare il polinomio interpolatore

generare nodi per interpolazione

```
%equispaziati
x = linspace(-1,1,n+1) %n = massimo grado del polinomio interpolante

%chebyshev-lobatto
x = -cos([0:n]*pi / n)

%dentro un ciclo

for i = 1:n
  %equispaziati
    x = linspace(-1,1,i+1) %n = massimo grado del polinomio interpolante

  %chebyshev-lobatto
  x = -cos([0:i]*pi / i)
end
```

Generare vettore y i cui elementi sono f(xi)

```
y = f(x)
```

errore assoluto polinomio interpolante

```
s = linspace(-1,1,500)
x = linspace(-1,1,n+1)
x_cheb = -cos([0:n] * pi / n);

t = interpol....
%errori vari
abs(f - t(s)) / abs(f)

%valutare funzione esatta
plot(s,f(s))
%valutare polinomio interpolatore
plot(s,t)
```

Integrazione

Integrazione: trapezi e cavalieri-simpson normali

Integrazione numerica

Due tipiche regole per approssimare integrali definiti, del tipo

$$I = \int_a^b f(x)\mathrm{d}x$$

sono quelle dei trapezi e di Simpson

- Trapezi
  La formula dei trapezi, esatta per polinomi di grado al più 1, corrisponde ad approssimare l'integrale I con

$$S^T = \frac{b-a}{2}(f(a)+f(b)).$$

- (Cavalieri-)Simpson o metodo della parabola
  La formula di Simpson, esatta per polinomi di grado al più 3, corrisponde ad approssimare l'integrale I con

$$S^{CV} = \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)$$

Trapezi composito

Metodo trapezi composito

È possibile applicare tali formule suddividendo l'intervallo $[a, b]$ in N subintervalli aventi la stessa ampiezza e applicando le formule in ognuno di loro.
In tale modo l'integrale I è approssimabile attraverso la formula dei trapezi composta

$$S_N^T = \frac{h}{2} f(x_0) + h f(x_2) + \cdots + h f(x_{N-1}) + \frac{h}{2} f(x_N),$$

dove

- $x_k = a + k \cdot h, k = 0, \ldots, N,$
- $h = \frac{b-a}{N}$.

Per cui la formula è del tipo $\sum_{k=0}^{N} w_k f(x_k)$, e i valori w_k sono detti pesi e i punti x_K sono detti nodi.
In particolare i pesi sono

$$w_0 = w_N = \frac{h}{2}, \qquad w_i = h, i = 1, \ldots, N-1$$

Integrale approssimato = somma(k= 0, n) wk*f(xk);

Simpson composito

Similmente anche per il metodo di Simpson si può suddividere l'intervallo in subintervalli dove applicare il metodo.
In tale modo l'integrale I è approssimabile attraverso la formula di Simpson (delle parabole) composta

$$S_N^{CS} = \frac{h}{3} f(x_0) + \sum_{s=1}^{N-1} \frac{2h}{3} f(x_{2s}) + \sum_{s=0}^{N-1} \frac{4h}{3} f(x_{2s+1}) + \frac{h}{3} f(x_{2N}),$$

dove

- $x_k = a + k \cdot h, k = 0, \ldots, 2N,$
- $h = \frac{b-a}{2N}$.

Per cui la formula è del tipo $\sum_{k=0}^{N} w_k f(x_k)$, e i pesi sono

$$w_0 = w_{2N} = \frac{h}{3}, \quad w_i = \frac{2h}{3}, i \text{ è pari} \quad w_i = \frac{4h}{3}, i \text{ è dispari}$$

Integrale approssimato = somma(k= 0, n) wk*f(xk);

```
for i = 1:n
    [x_s,w_s] = simpson_composta(a,b,i);
    [x_t,w_t] = trapezi_composta(a,b,i);

    Int_s(i) = f(x_s)'*w_s;
    Int_t(i) = f(x_t)' * w_t;
end
```

Punto medio composto

Partendo dalla funzione `trapezi_composta` si generi una funzione `ptomed_composta` che generi i punti e i pesi per la formula composita del metodo del punto medio per l'integrazione, ovvero il metodo che divide l'intervallo in `N` subintervalli e in ciascuno di essi approssima l'integrale con l'area del rettangolo costruito nel punto medio, cioè

$$\int_a^b f(x)\mathrm{d}x \approx (b-a) f\left(\frac{a+b}{2}\right).$$

```
function [x,w] = ptomed_composta(a,b,N)

% Formula dei trapezi composta
% ----------------------------
% ---- input ----
% a,b : estremi di integrazione
% N : numero di subintervalli
%
% ---- output ----
% x : nodi di integrazione (vettore colonna)
% w : pesi di integrazione (vettore colonna)

% passo di integrazione
h = (b-a)/N;
% nodi di integrazione
x_ex = a:h:b;
x = (x_ex(1:end-1)+x_ex(2:end))/2;
x = x';
% pesi di integrazione
w = h*ones(N,1);

end
```

Calcolare numero intervalli e punti usati per l'approssimazione dell'integrale

```
int_trap = 2^(length(I1)-1); %I1 = vettore approssimazioni integrale
num_punti_trap = int_trap +1;

int_simpson = 2^(length(I2)-1);
num_punti_simps = 2*int_simpson -1;
```

Algebra e sistemi

controllare dimensioni vettore/matrice e controllare che se è vettore o matrice

```
[n1,n2] = size(V);

if(n1 == 1) || (n2 == 2) % allora è un vettore, altrimenti è una matrice
```

norma vettoriale/matriciale

```
norm(x,n); % x = matrice/vettore, n = 1,2,Inf
```

calcolare condizionamento matrice

```
k(A) = ||A|| * ||A'||
oppure
cond(A)
```

calcolare soluzione sistema

```
// sistema nella forma Ax = b

x = A\b;
```

errore relativo sistema perturbato

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1-\kappa(A)\frac{\|\delta A\|}{\|A\|}}\left(\frac{\|\delta A\|}{\|A\|}+\frac{\|\delta b\|}{\|b\|}\right),$$

scomposizione pa=lu di un sistema

```
%serve in input (A,b)
[P,L,U] = lu(A);

y = L \ (P*b)
x = U \ y
```

minimi quadrati

```
% si usa su matrici A[m,n], con m > n
x = (A'*A) \ (A'*B)
```

soluzione esercizi con minimi quadrati

```
x = linspace(...);
y = .... % funzione data
s = linspace(a,b,100);
V = fliplr(vander(x));

A = V(:,[colonne]);

sol = (A'*A) \ (A'*B)
%generiamo la soluzione
y_approssimata = (sol(1) + sol(2)) * s;
```

decomposizione QR di un sistema

```
[Q,R] = qr(A);

d = 8; % d è un grado che viene fornito dal testoF
%per risolvere il sistema ci servono solo Q1 e E1
Q1 = Q(:,[1:d+1]);
R1 = R([1:d+1],[1:d+1]);

b = Q1' * y';
coeffs_qr = R1\b; % Calcolo i coefficienti con QR

rec_qr = polyval(coeffs_qr(end:-1:1),s); % Si veda sopra
```

generare matrice di vandermonde

```
V = flipr(vander(x));
```

generare polinomio interpolatore che approssima ai minimi quadrati

```
f = @(x) ....
d = numero
x = linspace(a,b,numero_nodi)
coeffs_poly = polyfit(x,f_esatta,d); % Calcolo i coefficienti attraverso la funzione polyfit
rec_poly = polyval(coeffs_poly,t); % Valuto il pol in 1000 punti su [-5,5]
```

calcolare step metodo newton

|xn+1 - xn | = |f(xn)| / |f'(xn)|

calcolare autovettori matrice

$$\|A\|_2 = \sqrt{\max_{i=1,\dots,n} |\lambda_i(A^t A)|},$$

```
e = eig(x' * x)
s = sqrt(max(abs(e)));
```

controllare che una matrice sia invertibile

```
t = det(A)
% se t == 1 allora è invertibile, se t == 0 allora non lo è
```

controllare che una matrice sia quadrata

```
if size(A,1) ~= size(A,2) % matrice non quadrata

if size(A,1) == aize(A,2) % matrice è quadrata
```

soluzione del sistema con le equazioni normali

```
V = fliplr(vander(x));
A = V(:,[1,2]); % prende tutte le righe e le prime due colonne della matrice di Vandermonde

sol = (A'*A) \ (A'*y);
```

ricostruzione funzione con varie cose

```
f = @(x) sin(2.*x) - x.^2;
n = 100;
x = linspace(-5,5,n);

s = linspace(-5,5,1000);
f_dist =@(x) f + 0.5*rand(size(f));
y = f(x);
f_esatta = f(s);

d = 8;

%metodo poltfit
coeffs_poly = polyfit(x,y,d); % Calcolo i coefficienti attraverso la funzione polyfit
rec_poly = polyval(coeffs_poly,s); % Valuto il pol in 1000 punti su [-5,5]

figure(1)
plot(x,y, 'ob');
hold on;
plot(s,f_esatta);
plot(s, rec_poly);
legend('Nodi','Funzione','Ricostruzione')
title('Interpolazione ai minimi quadrati: polyfit/polyval')

%metodo equazioni normali
V = fliplr(vander(x));

A = V(:,1:d+1);

sol = (A'*A) \ (A'*y');

s = linspace(0,1,100);
ricostruzione_eq_norm = polyval(sol(end:-1:1),s);



figure(2)
plot(x,y);
hold on;
plot(s,y, 'b');
plot(s,ricostruzione_eq_norm);
legend('Nodi','Funzione','Ricostruzione');
title('Interpolazione ai minimi quadrati: equazioni normali');


%scomposizione qr
V = fliplr(vander(x));
```

```
A = V(:,1:d+1);

[Q,R] = qr(A);

Q1 = Q(:,[1:d+1]);
R1 = R([1:d+1],[1:d+1]);

b = Q1' * y';
coeffs_qr = R1\b; % Calcolo i coefficienti con QR

rec_qr = polyval(coeffs_qr(end:-1:1),s); % Si veda sopra

figure(3)
plot(x,y,'ob')
hold on
plot(s,y,'-r','LineWidth',2)
plot(s,rec_qr,'-b','LineWidth',2)
legend('Nodi','Funzione','Ricostruzione')
title('Interpolazione ai minimi quadrati: scomposizione QR')
```

Varie

Sommatoria funzione

```
f = 1/(k^2) da 1 a n

symsum(1/(k^2), k, 1 ,n);
```

calcolare derivata n-esima funzione

```
//da fare su matlab online oppure avere symbolic math toolbox installato
syms x;
y = diff(funzione, x,grado_derivata); %x è la variabile rispetto a cui fare la derivata
```

grafico in scala semilogaritmica errore relativo in un dato intervallo [1:n]

```
semilogy(1:n, err_rel)
```

Calcolo derivata n-esima senza fare calcoli

```
g = ..... % funzione data
diff(g) %derivata prima
diff(g,2) %derivata seconda
...
```

- **polyfit** per calcolare il polinomio interpolante su un insieme di nodi noti
- **polyval** per valutare il polinomio su punti in cui non si conosce la f(x)

sapere il segno di una funzione

```
sign(funzione);
```

errore relativo iterate metodo bisezione

```
[x, xall, iter] = bisezione(f,a,b,toll,iterazioni_max);
semilogy(1:iter, err_rel);
```

colore grafici

```
plot(s,f(s), 'b'); %blu
plot(s,valutazioni,'r--'); %rosso tratteggiato
```