Data l'equazione di secondo grado x^2 - 0.5001x + 5*10^-5, calcolare le radici e confrontare con il risultato ottenuto con i comandi MatLab.

```
%Esercizio
clear
clc
%radici esatte trovate analiticamente (calcolando a mano)
x1=0.5;
x2=0.0001;
%verifica che l'equazione si annulli nelle radici
f=@(x) x.^2-0.5001*x+5*10^-5;
f(x1)
f(x2)
%confronto con i risultati della function "roots"
coeff=[1 -0.5001 5*10^-5];
s=roots(coeff)
abs(s(1)-x1)
abs(s(2)-x2)
%confronto con i risultati della function "fzero"
sol_1=fzero(f,0)
f=@(x) (x.^2-0.5001*x+5*10^-5)./(x-sol_1);
sol_2=fzero(f,0)
```

Approssimare analiticamente, con la formula del punto medio, l'integrale int(5.2, 3.1)exp(6*t)
SOLUZIONE: (5.2 – 3.1) f [6*(3.1+5.2)/2]

Data la Matrice e il termine noto b, far sì che la soluzione sia x = [1, 0, 1, 0, 1, 0, 1]’
con metodo di Gauss-Seidel e metodo di Jacobi.

```
%Esercizio
format long
close all
clear
clc
%Inserimento dati
A=diag(15*ones(7,1))+diag(1*ones(6,1),1)+diag(1*ones(6,1),-1);
x=[1 0 1 0 1 0 1]';
b=A*x;
%Metodo di Jacobi
[n,n]=size(A);
x0=zeros(n,1);
iter=0;
nmax=40;
while iter<nmax
    iter=iter+1;
    D=diag(diag(A));
    C=A-D;
    B_Jacobi=-inv(D)*C;
    x_J(:,iter)=-inv(D)*C*x0+inv(D)*b;
    x0=x_J(:,iter);
    err_J(iter,1)=norm(x-x_J(:,iter),inf);
end
disp('norma della matrice di Jacobi')
norm(B_Jacobi,inf)
semilogy([1:iter],err_J)
xlabel('numero di iterazioni')
ylabel('errore in norma Inf sulla soluzione esatta')
```

```
%Metodo di Gauss-Seidel
x0=zeros(n,1);
iter=0;
x=[];
x_old=x0;
while iter<nmax
    iter=iter+1;
    D=diag(diag(A));
    E=tril(A,-1);
    F=triu(A,1);
    B_Gauss=-inv(D+E)*F;
    x(:,iter)=-inv(D+E)*F*x_old+inv(D+E)*b;
    x_old=x(:,iter);
end

for iter=1:nmax
    err_GS(iter,1)=norm(x_old-x(:,iter),inf);
end
disp('norma della matrice di Gauss-Seidel')
norm(B_Gauss,inf)
hold on
semilogy([1:iter],err_GS,'r')
legend('Jacobi','Gauss-Seidel')
disp('raggi spettrali Jacobi e Gauss-Seidel')
[max(eig(B_Jacobi)) max(eig(B_Gauss))]
```

Siano a = 10^110 e b = 10^200. Calcolare a ∗ b con comandi Matlab:
In aritmetica di macchina a ∗ b = Inf poichè il massimo numero floating point rappresentabile
è realmax= 1.797693134862316e + 308 e a ∗ b è maggiore di realmax.

Siano p = 10−25 e q = 10−300. Calcolare p ∗ q con comandi Matlab:
In aritmetica di macchina p ∗ q = 0 poichè il più piccolo numero floating point in virgola mobile normalizzato rappresentabile è realmin= 2.225073858507201e − 308.
Rinunciando alla normalizzazione si possono rappresentare anche alcuni numeri più piccoli
(eps(realmin)= 4.940656458412465e − 324) ma p ∗ q è molto minore di realmin.

Dati 3 punti, implementare in Matlab il polinomio interpolante in forma di Lagrange e farne il grafico nell'intervallo [0.05, 1.5] evidenziando i punti di interpolazione.
Aggiungere al grafico precedente il grafico della funzione f(x) = -cos(x)./x e la legenda.
Costruire il polinomio interpolante in Matlab attraverso il comando polyfit.
Costruire in Matlab i polinomi di interpolazione di Lagrange della funzione f con numero di nodi crescente n = 5 : 5 : 50 equispaziati [nell'intervallo [0.05, 1.5]. Per ogni valore n memorizzare l'errore commesso nell'approssimazione di f su una griglia uniforme di 1000 punti.Fare la stessa cosa con polyfit.

```
% Esercizio
close all
clear
clc
format long e
% costruzione del polinomio interpolatore relativa a 3 punti di
% interpolazione con i polinomi fondamentali di Lagrange
```

```
x_interp=[0.05,1.0,1.5];
y_interp=[-1.997500520789933*10,-5.403023058681398*10^-1,-4.715813444513527*10^-2];
a=0.05;
b=1.5;
x=linspace(a,b,1000);
y=zeros(size(x));
for i=1:3
    y=y+y_interp(i)*Lagrange(i,x_interp,x);
end
%grafico
plot(x,y,'b')
hold on
plot(x_interp,y_interp,'ob')
axis square
grid on
xlabel('x')
ylabel('y')
% grafico della funzione
f=@(x) -cos(x)./x;
plot(x,f(x),'k')
% grafico dell'interpolante ottenuto con polyfit
P=polyfit(x_interp,y_interp,length(x_interp)-1);
yy=polyval(P,x);
plot(x,yy,'r')
legend('interpolazione a 3 punti','nodi','funzione','polyfit')
% costruzione del polinomio interpolatore con polyfit e nodi equispaziati
% figure
% plot(x,f(x),'k')
% hold on
for n=5:5:50
    x_interp=linspace(a,b,n);
    y_interp=f(x_interp);
    P=polyfit(x_interp,y_interp,n-1);
    yy=polyval(P,x);
%     plot(x,yy)
    err(n/5)=norm(yy-f(x),inf);
end
% costruzione del polinomio interpolatore Lagrange e nodi equispaziati
% figure
% plot(x,f(x),'k')
% hold on
L=ones(size(x));
for j=1:length(x_interp)
    if j~=i
        L=L.*(x-x_interp(j))/(x_interp(i)-x_interp(j));
    end
end
for n=5:5:50
    y=0;
    x_interp=linspace(a,b,n);
    y_interp=f(x_interp);
    for i=1:n
        y=y+y_interp(i)*L;
    end
%     plot(x,y)
%     plot(x_interp,y_interp,'o')
    err_L(n/5)=norm(y-f(x),inf);
end
```

Approssimare la derivata seconda ff (x) = ((-2+x.^2).*cos(x)-2*x.*sin(x))./x.^3;
della funzione f(x) = -cos(x)./x negli 8 nodi interni di una griglia di 10 nodi
equispaziati
nell'intervallo [0.05, 1.5]. Generare una tabella di errore relativo.

```
% Esercizio
close all
clear
clc
format long e
% inserimento funzione e della sua derivata seconda
f=@(x) -cos(x)./x;
ff=@(x) ((-2+x.^2).*cos(x)-2*x.*sin(x))./x.^3;
a=0.05;
b=1.5;
n=10;
x=linspace(a,b,n);
h=abs(x(2)-x(1));
yy=0;
for i=2:n-1
    f_xx(i-1)=(f(x(i+1))+f(x(i-1))-2*f(x(i)))/(h^2);
    yy(i-1)=ff(x(i));
    err(i-1)=abs((f_xx(i-1)-yy(i-1))/yy(i-1));
end
disp('errore')
[x(2:n-1)' err']
```

Costruire la matrice di Hilbert di ordine 5 e calcolarne il condizionamento. Scambiare
la prima
e la quinta riga. Poi scambiare la seconda e la quarta colonna. Calcolare il
condizionamento
della matrice così ottenuta. Costruire la matrice di Hilbert di ordine 10 e calcolarne
il
condizionamento.

```
% Esercizio
clear all
clc
format long
% matrice di Hilbert di ordine 5
A=hilb(5);
disp(A)
cond(A)
%scambio prima e quinta riga
a=A(1,:);
A(1,:)=A(5,:);
A(5,:)=a;
disp(A)
cond(A)
%scambio seconda e quarta colonna
a=A(:,2);
A(:,2)=A(:,4);
A(:,4)=a;
disp(A)
cond(A)
% matrice di Hilbert di ordine 10
A=hilb(10);
disp(A);
cond(A);
```

Dati i punti di coordinate P1 = (1, 2); P2 = (3, 1); P3 = (7, 0); determinare analiticamente il
polinomio interpolatore di grado 2. Ricostruire il polinomio in forma di Newton definito,
attraverso la tabella delle differenze divise. Poi aggiungere il punto P4 = (12, 0) e costruire
il relativo polinomio interpolatore di grado 3.

```
% Esercizio
clear
clc
format rat
f=@(x) 3*log(1./x)+sqrt(5*x);
x=[1 3 7];
y_0=f(x);
y=round(y_0);
%costruisco il polinomio interpolatore con 3 punti
p=polyfit(x,y,2)
xx=linspace(min(x),max(x));
yy_0=f(xx);
yy=polyval(p,xx);
plot(xx,yy_0,'k',x,y_0,'k*',xx,yy,'b',x,y,'bo')
pause
close all
%costruisco il polinomio interpolatore con 4 punti
x=[1 3 7 12];
y_0=f(x);
y=round(y_0);
p=polyfit(x,y,3)
xx=linspace(min(x),max(x));
yy_0=f(xx);
yy=polyval(p,xx);
plot(xx,yy_0,'k',x,y_0,'k*',xx,yy,'b',x,y,'bo')
```

Approssimare un integrale con la formula del punto medio semplice e trovare il suo valore esatto.
Rifare il calcolo cambiando gli estremi a [−π/2, π/2].
Implementare la formula del punto medio composto per approssimare l'integrale con una decomposizione uniforme dell'intervallo. [−π/2, π/2]. Costruire una tabella con il decadimento
dell'errore all'aumentare dei sottointervalli di decomposizione.
Costruire una tabella con il decadimento dell'errore ottenuto con la formula dei trapezi
composita implementata con il comando trapz di Matlab all'aumentare dei sottointervalli di decomposizione. Confrontare le due tabelle e commentare.

```
clear
clc
format long
f=@(x) cos(x);
a=0;
b=pi;
% formula del punto medio semplice
I_ms=f((a+b)/2)*(b-a)
% valore esatto
I_esatto=sin(b)-sin(a)
% errore relativo
Err_rel=abs((I_ms-I_esatto)/I_esatto)
% formula del punto medio composita
% errore punto medio composita
err_mc=[];
```

```
H_mc=[];
for m=10:10:100 %numero di sottointervalli della decomposizione
    h=(b-a)/m;
    x=a+h/2:h:b;
    y=f(x);
    int=h*sum(y);
    err_mc=[err_mc; abs(I_esatto-int)];
end
% errore formula trapezi composita Matlab
err_tc=[];
H_tc=[];
for m=10:10:100 %numero di sottointervalli della decomposizione
    h=(b-a)/m;
    x=a:h:b;
    y=f(x);
    int=h/2*(y(1)+2*sum(y(2:m))+y(m+1));
    err_tc=[err_tc; abs(I_esatto-int)];
end
format short e
[[10:10:100]' err_mc err_tc]
```

Data la funzione, implementa la formula dei trapezi semplice. Implementa la formula dei trapezi
composita con decomposizione uniforme dell'intervallo -1, 1. Verifica la correttezza del risultato
con il comando trapz. Approssima l'integrale con il comando quad. Costruire una tabella con il
decadimento dell'errore all'aumentare dei sottointervalli di decomposizione.

```
% Esercizio 3
close all
clear
clc
format long
f=@(x) 1+exp(-x.^2)/2;
a=-1;
b=1;
% formula trapezi semplice
I_ts=(f(a)+f(b))*(b-a)/2
% formula trapezi composita
m=10;
h=(b-a)/m;
x=a:h:b;
y=f(x);
int=h/2*(y(1)+2*sum(y(2:m))+y(m+1));
x=linspace(a,b,m+1);
I_trapz=trapz(x,f(x)) %formula trapezi composita Matlab
I_trapz-int; %verifica correttezza
I_quad=quad(f,a,b) %quadratura adattiva Matlab
% errore
err=[];
H=[];
for m=10:10:100
    h=(b-a)/m;
    x=a:h:b;
    y=f(x);
    int=h/2*(y(1)+2*sum(y(2:m))+y(m+1));
    err=[err; abs(I_quad-int)];
    H=[H;(b-a)/m]%ampiezza sottointervalli
end
```

```
format short e
[[10:10:100]' H err err./H.^2]
```

Fare il grafico delle funzioni in un intervallo a e b. Calcolare, inoltre, un'approssimazione delle due curve con comandi MatLab.

```
% Esercizio 3
close all
clear
clc
format long e
% grafico intersezione
r=@(s) 2*s;
z=@(s) sqrt(s.^2+1);
a=0;
b=2;
s=linspace(a,b);
plot(s,r(s),'k',s,z(s),'r')
xlabel('s')
axis square
grid on
hold on
% dati
f=@(s) 2*s-sqrt(s.^2+1);%2*s-sqrt(s.^2+1);
a=0;
b=2;
% intersezione approssimata con il comando fzero
disp('ascissa intersezione calcolata con il comando FZERO')
root=fzero(@(s) f(s),1)
plot(root,r(root),'*r')
legend('r(s)','z(s)','approssimazione intersezione')
```

Data una funzione, due estremi a e b, una tolleranza di 10^-10 e punto iniziale uno=1. Approssimare col metodo delle corde e successivamente con il metodo delle secanti. Stampare entrambe le tabelle e confrontare con il risultato di fzero.

```
clear
clc
f=@(s) 2*s-sqrt(s.^2+1);
a = 0;
b = 2;
uno = 1;
% approssimazione con il metodo delle corde
nmax=100;
toll=10^-10;
root=fzero(@(s) f(s),1)
fa=f(a);
fb=f(b);
r=(fb-fa)/(b-a);
err=b-a;
nit=0;
xvect=[];
x=uno;
fx=f(x);
xdif=[];
while nit<nmax && abs(fx)>toll
    nit=nit+1;
    x=x-fx/r;
```

```
    xvect=[xvect;x];
    fx=f(x);
end
disp('tabella metodo corde')
[[1:nit]' xvect abs(f(xvect))]
disp('confronto corde-FZERO')
abs(root-xvect(end))
% approssimazione con il metodo delle secanti
fa=f(a);
fb=f(b);
xvect=[];
nit=0;
while nit<nmax && abs(fb)>toll
    nit=nit+1;
    x=b-fb*(b-a)/(fb-fa);
    xvect=[xvect;x];
    a=b;
    fa=fb;
    b=x;
    fb=f(b);
end
disp('tabella metodo secanti')
[[1:nit]' xvect abs(f(xvect))]
% confronto con FZERO
disp('confronto secanti-FZERO')
abs(root-xvect(end))
```

Dati 3 punti, attraverso la matrice di Vandermonde, costruisci il polinomio interpolante 3 punti
e nell'intervallo definito

```
% Esercizio 3
close all
clear
clc
format long e
% costruzione del polinomio interpolatore attraverso la matrice di
% Vandermonde relativa a 3 punti di interpolazione
x_interp=[-1.4,0,0.1];
y_interp=[1.450850792298967,1,1.110719891073862]';
matrix=vander(x_interp);
coeff=matrix\y_interp;
%grafico
a=-1.5;
b=1.5;
x=linspace(a,b,1000);
y=polyval(coeff,x);
plot(x,y,'b')
hold on
plot(x_interp,y_interp,'ob')
axis square
grid on
xlabel('x')
ylabel('y')
%aggiunto il nuovo punto
x_interp=[x_interp,1];
y_interp=[y_interp;5.031038733198447];
matrix=vander(x_interp);
coeff=matrix\y_interp;
y=polyval(coeff,x);
```

```
hold on
plot(x,y,'b')
plot(x_interp,y_interp,'ob')
```

data la funzione e l'intervallo, costruire i polinomi di interpolazione con numero di nodi
crescente n = 10 : 10 : 100 equispaziati nell'intervallo. Per ogni valore n, memorizzare il
numero di condizionamento della matrice di Vandermonde e l'errore commesso nell'approssimazione
di f su una griglia uniforme di 1000 punti. Al variare di n, interpolare la funzione f nei nodi
definiti, attraverso il comando interp1 e memorizzare l'errore commesso.

```
close all
clear
clc
% grafico della funzione
f=@(x) exp(x)./cos(x);
a=-1.5;
b=1.5;
x=linspace(a,b,1000);
plot(x,f(x),'k')
% costruzione del polinomio interpolatore attraverso la matrice di
% Vandermonde con un numero crescente di punti di interpolazione
for n=10:10:100
    x_interp=linspace(a,b,n);
    y_interp=f(x_interp)';
    matrix=vander(x_interp);
    condiz(n/10)=cond(matrix);
    coeff=matrix\y_interp;
    y=polyval(coeff,x);
    y_bis=interp1(x_interp,y_interp,x);
    err(n/10)=norm(y-f(x),inf);
    err_bis(n/10)=norm(y_bis-f(x),inf);
end
disp('condizionamento')
```

Data una funzione e i due intervalli, genera il grafico. Aggiungere il grafico della funzione
costante che interseca il grafico della funzione nel punto medio dell'intervallo.

```
g = @(x) 2.*x./(sqrt(x.^2+1));
a = sqrt(3);
b = 2;
x = linspace(a, b);
plot(x, g(x), 'blue');
hold on
pm = g((a+b)/2);
plot(x, pm*ones(size(x)), 'Red');
```

Implementare l'algoritmo delle tangenti(Newton) e applicarlo alla funzione f con dato iniziale
s0 = 2 e test di arresto basato sul controllo dell']incremento con tolleranza pari a 10^-10 .

```
f=@(x) x.^8-2;
a=0.5;
```

```
b=2;
s=linspace(a,b);
f_der=@(x) 8*x.^7;
s0=2;
nmax=100;
toll=10^-10;
err=1;
nit_New=0;
s_New=s0;
x=s0;
fx=f(x);
xdif=[];
while nit_New<nmax && err>toll
    nit_New=nit_New+1;
    x=s_New(nit_New);
    dfx=f_der(x);
    if dfx==0
        disp('arresto per azzeramento dfun')
    else
    xn=x-fx(nit_New)/dfx;
    err=abs(xn-x);
    xdif=[xdif;err];
    x=xn;
    s_New=[s_New;x];
    fx=[fx;f(x)];
    end
end
plot(s_New,f(s_New),'ob')
```

```
Approssimare una funzione con il metodo di bisezione
f=@(x) x.^8-2;
a=0.5;
b=2;
nmax=100;
toll=10^-10;
err=abs(b-a);
nit_bis=1;
s_bis=b;
fx=[];
xdif=[];
while nit_bis<nmax && err>toll
    nit_bis=nit_bis+1;
    c=(a+b)/2;
    x=c;
    fc=f(x);
    s_bis=[s_bis;x];
    fx=[fx;fc];
    x=a;
    if fc*f(x)>0
        a=c;
    else
        b=c;
    end
    err=abs(s_bis(nit_bis)-s_bis(nit_bis-1));
    xdif=[xdif;err];
end
```

Data la matrice A, effettuarne la fattorizzazione di Cholesky attraverso chol e verificare
che attraverso la fattorizzazione si possa ricostruire la matrice A. Costruire un vettore termine
noto b ∈ R n tale che il sistema Ax = b abbia come soluzione. Risolvere il sistema lineare
attraverso l'operatore \ di Matlab e calcolare l'errore commesso rispetto alla soluzione esatta.
Dopodiché implementare la funzione di sostituzione all indietro e sostituzione in avanti. Tabella degli errori.

```
function x=Backward(U,b)
[n,m]=size(U);
x(n)=b(n)/U(n,n);
for i=n-1:-1:1
    x(i)=(b(i)-U(i,i+1:n)*(x(i+1:n))')/U(i,i);
end
x=x';
function x=Forward(L,b)
[n,m]=size(L);
x(1)=b(1)/L(1,1);
for i=2:n
    x(i)=(b(i)-L(i,1:i-1)*(x(1:i-1))')/L(i,i);
end
x=x';
% Esercizio
close all
clear
clc
format long
Tab=[];
for n=10:10:100
    %assegnazione della matrice
    A=2*eye(n)-diag(ones(n-1,1),-1)-diag(ones(n-1,1),1);
    %fattorizzazione di Cholesky
    B=chol(A);
    %verifica della correttezza della fattorizzazione
    prova = norm(A-B'*B,inf);
    %costruzione del termine noto affinchè la soluzione sia un vettore di
    %elementi pari ad 1
    x=ones(n,1);
    b=A*x;
    %risoluzione del sistema lineare con function Matlab
    x_1=A\b;
    % Metodo sostituzione in avanti per risolvere B'y=b
    y = Forward(B', b);
    err_1=norm(x-x_1,inf);
    % Metodo sostituzione all'indietro per risolvere By=b
    x_2=Backward(B,y);
    err_2=norm(x-x_2,inf);
    %confronto degli errori
    Tab=[Tab; err_1 err_2];
end
```

Data una funzione, i due estremi e 3 punti traccia i grafici dei polinomi interpolatori semplici
e compositi

```
clear
clc
hold on
f = @(x) 1./(1+9*x.^2);
a = -1;
b = 1;
i = 1;
for N = 3:2:7
    x_plot = linspace(a, b, 1000);
    y_plot = f(x_plot);
    x_equispaced = linspace(a, b, N+1);
    y_equispaced = f(x_equispaced);
    %serve solo per lineare semplice
    p_equispaced = polyfit(x_equispaced, y_equispaced, N);
    y_interp_equispaced = polyval(p_equispaced, x_plot);
    %serve solo per lineare composita
    y_interp_equispaced_linear = interp1(x_equispaced, y_equispaced, x_plot, 'linear');
    %grafico lineare semplice
    figure(1);
    subplot(1, 3, i);
    plot(x_plot, y_plot, 'b');
    hold on;
    plot(x_plot, y_interp_equispaced, 'r--');
    %grafico lineare composita
    figure(2);
    subplot(1, 3, i);
    plot(x_plot, y_plot, 'b');
    hold on;
    plot(x_plot, y_interp_equispaced_linear, 'r--');
    i = i +1;
end
```

Dato il sistema lineare e fissando un vettore iniziale con tolleranza, implementa il metodo
di Jacobi. Fare il grafico al variare dell'indice di iterazione.

```
clear
clc
close all
A = [8, 2, 3; 2, 4 ,0; 3, 0, 5];
b = [48;2;35];
x = [0 0 0]';
toll = 10^-6;
[n,n]=size(A);
x0=zeros(n,1);
iter=0;
nmax=40;
while iter<nmax
    iter=iter+1;
    D=diag(diag(A));
    C=A-D;
    B_Jacobi=-inv(D)*C;
    x_J(:,iter)=-inv(D)*C*x0+inv(D)*b;
    x0=x_J(:,iter);
    err_J(iter,1)=norm(x-x_J(:,iter),inf);
end
semilogy([1:iter],err_J);
```

```
hold on;
plot(err_J, err_J, '*');
xlabel('numero di iterazioni')
ylabel('errore in norma Inf sulla soluzione esatta')
```

```
Dati 12 valori dei quali 1 ignoto, trova e fai il grafico del valore ignoto attraverso
la polinomiale lineare a tratti interpolante, il polinomio di interpolazione, e
la parabola approssimante ai minimi quadrati
clear
clc
% Dati forniti
y = [12.51 13.05 11.7 9.26 7.755 6.25 5.34 4.59 5.14 6.36 10.31 13.88];
% Il valore mancante (per Maggio, posizione 5)
missing_index = 5;
% Rimuovere il valore mancante dai dati
x = 1:length(y);
x_nuovo = x;
x_nuovo(missing_index) = [];
y_nuovo = y;
y_nuovo(missing_index) = [];
% Creazione del piano cartesiano
figure;
hold on;
% Polinomiale lineare a tratti
linearFit = polyfit(x_nuovo, y_nuovo, 1);
linearCurve = polyval(linearFit, x);
plot(x, linearCurve, '--', 'DisplayName', 'Polinomiale Lineare a Tratti');
% Polinomio di interpolazione (grado n-2 poiché un dato è mancante)
interpolationPoly = polyfit(x_nuovo, y_nuovo, length(x) - 2);
interpolationCurve = polyval(interpolationPoly, x);
plot(x, interpolationCurve, '-', 'DisplayName', 'Polinomio di Interpolazione');
% Parabola approssimante
parabolicFit = polyfit(x_nuovo, y_nuovo, 2);
parabolicCurve = polyval(parabolicFit, x);
plot(x, parabolicCurve, '-.', 'DisplayName', 'Parabola Approssimante');
%Trovo i valori mancanti dei vari dei vari polinomi
mancante_lineare = polyval(linearFit, missing_index);
mancante_interpolante = polyval(interpolationPoly, missing_index);
mancante_parabolica = polyval(parabolicFit, missing_index);
% Aggiungere i valori stimati al grafico
y(missing_index) = mancante_interpolante;
plot(x, y, 'o', 'DisplayName', 'Dati Raccolti');
plot(missing_index, mancante_lineare, 'bx', 'DisplayName', 'Valore Stimato (Lineare)');
plot(missing_index, mancante_interpolante, 'rx', 'DisplayName', 'Valore Stimato
(Interpolante)');
plot(missing_index, mancante_parabolica, 'yx', 'DisplayName', 'Valore Stimato
(Parabola)');
% Aggiunta delle legende e del titolo
legend('show');
title('Grafico dei Dati e Curve Approssimanti');
xlabel('Mese');
ylabel('Portata (m^3)');
grid on;
hold off;
```

```
Controllo unicità della soluzione convergenza del metodo
for i = 1:3
    A = [1 1 i; -1 2 0.2; 1 -0.1 2];
    sum = 0;
    for j = 1:3
        if j != i
            sum = sum + A(i, j);
        end
    end
    if and(det(A) > 0, abs(A(i, i)) > sum)
        disp('trovato');
        break;
    end
end
```

```
METODO DI DOOLITTLE
n  = size(A, 1); L = eye(n); U = zeros(n); for k = 1 : n
   U(k, k : n) = A(k, k : n) - L(k, 1 : k-1) * U(1 : k-1, k : n);
   L(k+1 : n, k) = (A(k+1 : n, k) - L(k+1 : n, 1 : k-1) * U(1 : k-1, k)) / U(k, k); end
```

```
FORMULA DI CAVALIERI-SIMPSON SEMPLICE
dati f = @(x) che definisce la funzione, a e b gli intervalli di integrazione e sol =
il risultato esatto: c = (a + b) /2;
S_CS = ((b - a) /6) * (f(a) + 4 * f(c) + f(b)); integraleEsatto = quad(fun, a, b);
errore = abs(S_CS - integraleEsatto);
```

```
FORMULA DI CAVALIERI-SIMPSON COMPOSITA
dati N il numero di subintervalli, a e b gli estremi di integrazione: h = (b - a) / N;
x = a : h : b;
integrale_composita = h/6 * (f(a) + 4 * sum(f(x(2 : 2 : end-1))) + 2*sum(f(x(3 : 2 :
end-2))) + f(b)); integraleEsatto = quad(fun, a, b);
errore_composita = abs(integrale_composita - integraleEsatto);
```

```
%%% COMANDI GENERALI %%%
***COMDANI GENERALI***:
clc  -->  Cancella tutto il testo del log
clear  -->  Cancella tutte le variabili. Per toglierne una specifica aggiungerne il
nome format short  -->  Formato numeri corti format long  -->  Formato numeri bislunghi
format rat  -->  Formato razionali
size(x)  -->  Restituisce la dimensione dell'array o della matrice inf  -->  Infinito
exp(1)  -->  Costante e pi --> Pi greco
ALT+126 --> Simbolo NOT --> ~
CTRL + C --> Esci dal ciclo se è un loop --> In alternativa si scrive break disp('') --
> Stampa una stringa
input('inizio array ') --> Chiedo in input un numero
FUN1 = @(x) x^3 - 6 * x^2 + 11 * x - 6 --> Dichiaro una funzione
```

```
%%%OPERATORI MATEMATICI%%%:
sqrt(n)  -->  Radice quadrata di un numero abs(x)  -->  Valore assoluto di un
numero/array/matrice
roots(array)  -->  Dà le radici di un'equazione di n gradi    --> Array = [n+1 numeri]
con n uguale all'esponente di x più grande
```

log(x) --> Calcolo il logaritmo di un valore x abs(x) --> Restituisce il valore assoluto di un valore/array/matrice x \ --> Divisione inversa. --> se ho a=2 e b=6 allora a\b == 6/2 sum(x) --> Somma gli elementi di un array max(x) --> Trova il massimo in un array min(x) --> Trova il minimo di un array sum(A) --> Somma in colonna max(A) --> Trova il massimo in colonna max(max(A)) --> Trova il numero massimo in una matrice norm(array, tipo di norma) --> Trova la norma di un array prod(x) --> Prodotto di un array
sort(x) --> Riordina l'array in senso crescente
==, >, <, >=, <= --> Confronto tra due variabili, array o matrici
x&y --> Confronta i valori di due array x e y, restituisce 0 nella posizione dove il valore di uno dei due array ha come valore 0, nel restante restituisce 1 ~x --> Restituisce il valore 1 dove la matrice x ha il valore 0 fzero(FUN1, 0) --> non ho ben capito cosa faccia ma funziona così

%%%MATRICI o ARRAY%%%:
x = [... ... ...] --> Vettore riga x = [...; ...; ...; ...;] --> Vettore colonna
x = [inizio : incremento o decremento : fine] --> Vettore in un intervallo di valori A = [1 2 3; 4 5 6] --> Matrice 2x3 con quei valori x = [1 2 3 x] --> Aggiungo in testa i valori 1 2 3 all'array già esistente x = [x 1 2 3] --> Aggiungo in coda i valori 1 2 3 all'array già esistente x' --> Trasposizione di un array o matrice x(2) --> leggo il valore nella posizione 2 dell'array
x*A --> Prodotto riga per colonna tra un array e una matrice con un array x di colonne <= alle righe di A
A(3,:) = [] --> Cancello la terza riga A(:,3) = [] --> Cancello la terza colonna
x.*A --> Moltiplico per ogni elemento della matrice A --> Lo stesso vale con gli operatori / \ ^ rand(3) --> Genera una matrice 3x3 di numeri casuali eye(3) --> Genera una matrice identità 3x3 zeros(3) --> Genera una matrice di zeri 3x3 ones(3) --> Genera una matrice di uni 3x3 inv(A) --> Crea l'inversa di una matrice hilb(3) --> Crea la matrice 3x3 di razionali det(A) --> Trova il determinante di una matrice diag(A) --> Estrae la diagonale principale o crea una matrice con diagonale i valori dell'array tra parentesi tril(A) --> Estrae la matrice triangolare inferiore di una matrice triu(A) --> Estrae la matrice triangolare superiore di una matrice
linspace(inizio, fine, quantità di numeri) --> Vettore linearmente spaziato

%%%GRAFICI%%%:
plot(ascissa, ordinata) --> Crea il grafico con quelle coordinate grid on/off --> Mostra o toglie la grid nel grafico
xlabel('ascisse') o ylabel('ordinate') --> Dà il nome ascissa o ordinata agli assi x o y title('grafico') --> Dà il nome grafico al grafico
hold on --> Genera grafici sopra quello già attualmente attivo plot(x,y,'*k') --> Grafico, dati i punti x e y, mette i puntini neri nei punti di intersezione subplot(n, m, k) --> Divide una finestra in n righe e m colonne per vedere più grafici insieme. Lettera k per decidere il grafico da visualizzare
plot(x,y,'k--') --> Grafico a linea tratteggiata, dati i punti x e y figure --> legend('label 1', ... 'label n') --> Genera una legenda di stringhe scritte nella parentesi semilogy --> Mette in scala logaritmica i valori del grafico

%%%CICLO FOR%%%:
```
for i = 1:10
   x = x + i;
end
```

%%%CICLO WHILE***:
```
while k<=10;
   x(k) = k^2;    k = k+1;
end
```

%%%COMANDO IF%%%:
```
if a<0
   a = -1
```

```
elseif a>0
   a = 1
else
   a
end

%%%COMANDO SWITCH%%%:
switch z
   case 0
   disp('qui')    case 1
   disp('quo')    otherwise    disp('qua')
end
```