SISTEMI LINEARI

CHOL TRIDIAG

```
function [d,m] = chol_tridiag(d,c)
% Sia A tridiagonale, simmetrica e definita positiva:
% INPUT
% d: vettore contenente gli elementi della diagonale principale di A
% c: vettore contenente gli elementi della codiagonale superiore
%   (e inferiore) di A
% OUTPUT
% d: vettore contenente gli elementi della diagonale principale di L_1
n = length(d);
m = zeros(n-1,1);
for k = 1:n-1
    m(k) = c(k)/d(k);
    d(k+1) = d(k+1)-m(k)*c(k);
end
```

GAUSS TRIDIAG NOPIV

```
function x = gauss_tridiag_nopiv(d,c,b)
% Sia Ax=b con A tridiagonale, simmetrica e a diagonale dominante oppure
% definita positiva:
% INPUT
% d: vettore contenente gli elementi della diagonale principale di A
% c: vettore contenente gli elementi della codiagonale superiore
%   (e inferiore) di A
% b: vettore contenente gli elementi del termine noto
% OUTPUT
% x: vettore contenente la soluzione del sistema Ax=b
n = length(d);
x = zeros(n,1);
m = zeros(n-1,1);
for k = 1:n-1
    m(k) = c(k)/d(k);
    d(k+1) = d(k+1)-m(k)*c(k);
    b(k+1) = b(k+1)-m(k)*b(k);
end
x(n) = b(n)/d(n);
for i = n-1:-1:1
    x(i) =(b(i)-c(i)*x(i+1))/d(i);
end
```

LAB 01

```
%soluzioni esercizi Laib01
clear all
clc
disp('***********************************************')
disp('****************esercizio_1_1****************')
disp('***********************************************')
format short e
n = 10;
d = 4*ones(n,1);
c= -ones(n-1,1);
A = diag(d,0)+diag(c,1)+diag(c,-1);
% verifica A definita positiva
```

```
autovalori_A = eig(A)
b = sum(A,2);
x = gauss_tridiag_nopiv(d,c,b);
% verifica correttezza risultato
x_m =A\b;
err_sol = norm(x-x_m,inf)
pause
clear all
clc
disp('************************************************')
disp('*****************esercizio_1_2*****************')
disp('************************************************')
format short e
n = 10;
d = 4*ones(n,1);
c= -ones(n-1,1);
A = diag(d,0)+diag(c,1)+diag(c,-1);
%verifica A definita positiva
autovalori_A = eig(A)
[d,m] = chol_tridiag(d,c);
L_1 = (diag(ones(n,1))+diag(m,-1))*diag(sqrt(d));
% verifica correttezza risultato
R = chol(A);
err_chol = norm(L_1-R',inf)
pause
clear all
clc
disp('************************************************')
disp('*****************esercizio_1_3*****************')
disp('************************************************')
n = 2500;
d = 4*ones(n,1);
c= -ones(n-1,1);
A = diag(d,0)+diag(c,1)+diag(c,-1);
b = sum(A,2);
tic
[L,U,P] = lu(A);
y = L\(P*b);
x = U\y;
tempo_PALU = toc
pause
tic
R = chol(A);
y = R'\b;
x = R\y;
tempo_Chol = toc
pause
tic
[Q,R] = qr(A);
x = R\(Q'*b);
tempo_QR = toc
pause
tic
[U,S,V] = svd(A);
y = S\(U'*b);
x = V*y;
tempo_svd = toc
```

SISTEMI LINEARI ITERATIVI

GAUSS-SEIDEL

```
function [x,k,ier] = gauss_seidel(A,b,x,tol,kmax)
%
% [x,k,ier] = gauss_seidel(A,b,x,tol,kmax)
%
%   A   :   matrice dei coefficienti del sistema lineare
%   b   :   vettore dei termini noti del sistema lineare
%   x   :   vettore colonna, in input soluzione iniziale, in output soluzione
approssimata
%   tol :   tolleranza relativa
%   kmax:   numero massimo di iterazioni consentite
%   k   :   numero di iterate calcolate
%   ier :   indicatore del criterio d'arresto utilizzato, vale 1 se raggiunte
kmax iterazioni,
%           vale 0 se soddisfatta la tolleranza relativa tol
%

% verifica che A sia quadrata
[n,m] = size(A);
if n ~= m
   error('la matrice A non è quadrata');
end

% verifica che x sia un vettore colonna
if ~iscolumn(x)
   error('x deve essere un vettore colonna');
end

% verifica che b sia un vettore colonna
if ~iscolumn(b)
   error('b deve essere un vettore colonna');
end

% verifica che la dimensione di x sia compatibile con quella di A
if n ~= length(x)
   error('A e x devono avere le dimensioni compatibili');
end

% verifica che la dimensione di b sia compatibile con quella di A
if n ~= length(b)
   error('A e b devono avere le dimensioni compatibili');
end

% algoritmo di Gauss-Seidel
for k = 1:kmax
    y = x(1);
    x(1) = (b(1)-A(1,2:end)*x(2:end))/A(1,1);
    xmax = abs(x(1));
    emax = abs(y-x(1));
    for i = 2:n
        y = x(i);
        x(i) = (b(i)-A(i,1:i-1)*x(1:i-1)-A(i,i+1:end)*x(i+1:end))/A(i,i);
        e = abs(y-x(i));
        if abs(x(i)) > xmax
            xmax = abs(x(i));
        end
        if e > emax
            emax = e;
```

```
        end
    end
    if emax <= tol*xmax
        ier = 0;
        return
    end
end
ier = 1;
```

GAUSS-SEIDEL-MATRICI

```
function [x,k,ier] = gauss_seidel_mat(A,b,x,tol,kmax)
%
% [x,k,ier] = gauss_seidel_mat(A,b,x,tol,kmax)
%
%   A   :   matrice dei coefficienti del sistema lineare
%   b   :   vettore dei termini noti del sistema lineare
%   x   :   vettore colonna, in input soluzione iniziale, in output soluzione
approssimata
%   tol :   tolleranza relativa
%   kmax:   numero massimo di iterazioni consentite
%   k   :   numero di iterate calcolate
%   ier :   indicatore del criterio d'arresto utilizzato, vale 1 se raggiunte
kmax iterazioni,
%           vale 0 se soddisfatta la tolleranza relativa tol
%

% verifica che A sia quadrata
[n,m] = size(A);
if n ~= m
   error('la matrice A non è quadrata');
end

% verifica che x sia un vettore colonna
if ~iscolumn(x)
   error('x deve essere un vettore colonna');
end

% verifica che b sia un vettore colonna
if ~iscolumn(b)
   error('b deve essere un vettore colonna');
end

% verifica che la dimensione di x sia compatibile con quella di A
if n ~= length(x)
   error('A e x devono avere le dimensioni compatibili');
end

% verifica che la dimensione di b sia compatibile con quella di A
if n ~= length(b)
   error('A e b devono avere le dimensioni compatibili');
end

% algoritmo di Gauss-Seidel
D = tril(A);
C = A-D;
for k = 1:kmax
    y = x;
    x = D\(b-C*y);
    if norm(x-y,inf) <= tol*norm(x,inf)
        ier = 0;
```

```
        return
    end
end
ier = 1;
```

GAUSS-SEIDEL-SPARSE

```
function [x,k,ier] = gauss_seidel_sparse(A,b,x,tol,kmax)
%
% utilizza il ciclo while invece di nz_ir =  nnz(ir == i):
% avanza nel vettore ir fintantoché il numero di riga non cambia!!!
% [x,k,ier] = gauss_seidel_sparse(A,b,x,tol,kmax)
%
%   A   :   matrice dei coefficienti del sistema lineare in formato sparso
%   b   :   vettore dei termini noti del sistema lineare
%   x   :   vettore colonna, in input soluzione iniziale, in output soluzione
approssimata
%   tol :   tolleranza relativa
%   kmax:   numero massimo di iterazioni consentite
%   k   :   numero di iterate calcolate
%   ier :   indicatore del criterio d'arresto utilizzato, vale 1 se raggiunte
kmax iterazioni,
%           vale 0 se soddisfatta la tolleranza relativa tol
%

% verifica che A sia sparsa
if ~issparse(A)
   warning('la matrice A non è in formato sparso');
   A = sparse(A);
end

% verifica che A sia quadrata
[n,m] = size(A);
if n ~= m
   error('la matrice A non è quadrata');
end

% verifica che x sia un vettore colonna
if ~iscolumn(x)
   error('x deve essere un vettore colonna');
end

% verifica che b sia un vettore colonna
if ~iscolumn(b)
   error('b deve essere un vettore colonna');
end

% verifica che la dimensione di x sia compatibile con quella di A
if n ~= length(x)
   error('A e x devono avere le dimensioni compatibili');
end

% verifica che la dimensione di b sia compatibile con quella di A
if n ~= length(b)
   error('A e b devono avere le dimensioni compatibili');
end

% algoritmo di Gauss-Seidel per matrici in formato sparso
d = diag(A);
[c,r,a] = find(A'-diag(diag(A)));
for k = 1:kmax
```

```
    xmax = 0;
    emax = 0;
    count = 0;
    for i = 1:n
        y = x(i);
        count0 = count;
        while count+1 <= length(r) && r(count+1) == i
              count = count+1;
        end
        s = sum(a(count0+1:count).*x(c(count0+1:count)));
        x(i) = (b(i)-s)/d(i);
        e = abs(y-x(i));
        if abs(x(i)) > xmax
            xmax = abs(x(i));
        end
        if e > emax
            emax = e;
        end
    end
    if emax <= tol*xmax
        ier = 0;
        return
    end
end
ier = 1;
```

JACOBI

```
function [x,k,ier] = jacobi_mat(A,b,x,tol,kmax)
%
% [x,k,ier] = jacobi_mat(A,b,x,tol,kmax)
%
%   A   :   matrice dei coefficienti del sistema lineare
%   b   :   vettore dei termini noti del sistema lineare
%   x   :   vettore colonna, in input soluzione iniziale, in output soluzione
approssimata
%   tol :   tolleranza relativa
%   kmax:   numero massimo di iterazioni consentite
%   k   :   numero di iterate calcolate
%   ier :   indicatore del criterio d'arresto utilizzato, vale 1 se raggiunte
kmax iterazioni,
%           vale 0 se soddisfatta la tolleranza relativa tol
%

% verifica che A sia quadrata
[n,m] = size(A);
if n ~= m
   error('la matrice A non è quadrata');
end

% verifica che x sia un vettore colonna
if ~iscolumn(x)
   error('x deve essere un vettore colonna');
end

% verifica che b sia un vettore colonna
if ~iscolumn(b)
   error('b deve essere un vettore colonna');
end

% verifica che la dimensione di x sia compatibile con quella di A
if n ~= length(x)
   error('A e x devono avere le dimensioni compatibili');
end
```

```
% verifica che la dimensione di b sia compatibile con quella di A
if n ~= length(b)
   error('A e b devono avere le dimensioni compatibili');
end

% algoritmo di Jacobi
D = diag(diag(A));
C = A-D;
for k = 1:kmax
    y = x;
    x = D\(b-C*y);
    if norm(x-y,inf) <= tol*norm(x,inf)
        ier = 0;
        return
    end
end
ier = 1;
```

LAB 02

```
% Laib02: sistemi lineari - metodi iterativi
clear all
clc
disp('***********************************************')
disp('*****************esercizio_2_1*****************')
disp('***********************************************')

n = 5;
A = magic(n);
disp('determinante di A ')
determinante_A = det(A);
fprintf('%e\n',determinante_A);

B = A'*A;
disp('verifica simmetria di B, calcola max(max(abs(B''-B)))')
M = max(max(abs(B'-B)));
fprintf('%e\n',M);

disp('verifica B definita positiva, calcola min e max di eig')
fprintf('%e\t %e\n',min(eig(B)),max(eig(B)));

% genera termine noto
b = sum(B,2);
x = zeros(n,1);
tol = 1.0e-05;
kmax = 100;
disp('raggio spettrale matrice di iterazione')
rho=max(abs(eig(eye(n)-inv(tril(B))*B)));
fprintf('rho = %e\n',rho);
[x,k,ier] = gauss_seidel(B,b,x,tol,kmax);
disp('errore relativo associato alla soluzione calcolata')
errore_relativo = norm(ones(n,1)-x,inf)/norm(ones(n,1),inf);
fprintf('%e\n',errore_relativo);

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause
```

```
clear all
clc
disp('***********************************************')
disp('*****************esercizio_2_2*****************')
disp('***********************************************')

% A = sprandsym(n,isp,a) genera una matrice A
% di ordine n,
% simmetrica,
% sparsa,
% pseudocasuale,
% con indice di sparsità isp(=nnz(A)/(n^2)) e
% con autovalori, memorizzati in a, reali e positivi
% (A è dunque simmetrica e definita positiva).

fprintf('%10s\t%10s\t%10s\t%10s\r\n','iter','gauss_seidel_sparse','iter','gauss_
seidel');
for i = 1:20
    n = 400;
    isp = 0.01;
    a = rand(1,n)*10; % autovalori desiderati
    A = sprandsym(n,isp,a);
    condizionamento = condest(A);
    indice_sparsita_eff = nnz(A)/n^2;
%
    b = sum(A,2);
    x = zeros(n,1);
    tol = 1.0e-6;
    kmax = 10000;

    x = zeros(n,1);
    tic
    [x,k,ier] = gauss_seidel_sparse(A,b,x,tol,kmax);
    time_GS_sparse(i) = toc;
    iter(i) = k;

    x = zeros(n,1);
    B = full(A);
    tic
    [x,k,ier] = gauss_seidel(B,b,x,tol,kmax);
    time_GS_full(i) = toc;
    iter_full(i) = k;


fprintf('%10.0d\t%6.4e\t\t\t%10.0d\t%6.4e\n',iter(i),time_GS_sparse(i),iter_full
(i),time_GS_full(i));
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
clc
disp('***********************************************')
disp('*****************esercizio_2_3*****************')
disp('***********************************************')

format long e
```

```
kmax = 100;
tol = 1.e-7;

A1 = [1 -2 2; -1 1 -1; -2 -2 1];
b1 = sum(A1,2);
n = length(b1);
x0 = zeros(n,1);
disp('Jacobi')
[x,k,ier] = jacobi_mat(A1,b1,x0,tol,kmax)
disp('Gauss Seidel')
[x,k,ier] = gauss_seidel_mat(A1,b1,x0,tol,kmax)
pause
%
A2 = [4 0 2/5; 0 5 2/5; 5/2 2 1];
b2 = sum(A2,2);
n = length(b2);
x0 = zeros(n,1);
disp('Jacobi')
[x,k,ier] = jacobi_mat(A2,b2,x0,tol,kmax)
disp('Gauss Seidel')
[x,k,ier] = gauss_seidel_mat(A2,b2,x0,tol,kmax)
pause
%
A3 = [2 -1 1; 2 2 2; -1 -1 2];
b3 = sum(A3,2);
n = length(b3);
x0 = zeros(n,1);
disp('Jacobi')
[x,k,ier] = jacobi_mat(A3,b3,x0,tol,kmax)
disp('Gauss Seidel')
[x,k,ier] = gauss_seidel_mat(A3,b3,x0,tol,kmax)
pause
%
A = [3 -1 0;-1 3 -1;0 -1 3];
B = [0 0 -1;0 -1 0;-1 0 0];
A4 = [A B;B A];
b4 = sum(A4,2);
n = length(b4);
x0 = zeros(n,1);
disp('Jacobi')
[x,k,ier] = jacobi_mat(A4,b4,x0,tol,kmax)
disp('Gauss Seidel')
[x,k,ier] = gauss_seidel_mat(A4,b4,x0,tol,kmax)

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
clc
disp('*********************************************')
disp('****************esercizio_2_4****************')
disp('*********************************************')

format long e
tol = 1.0e-10;
kmax = 100;
X = [4 -1 0;-1 4 -1;0 -1 4];
Y = -eye(3);
```

```
A = [X Y;Y X];
b = sum(A,2);
n = length(b);
x0 = zeros(n,1);
disp('Jacobi')
[x,k,ier] = jacobi_mat(A,b,x0,tol,kmax)
rho_J = max(abs(eig(eye(n)-inv(diag(diag(A)))*A)))
disp('Gauss Seidel')
[x,k,ier] = gauss_seidel_mat(A,b,x0,tol,kmax)
rho_GS = max(abs(eig(eye(n)-inv(tril(A))*A)))

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
clc
disp('***********************************************')
disp('*****************esercizio_2_5*****************')
disp('***********************************************')

format long e
tol= 1.0e-12;
kmax = 100;
X = [4 -1 0;-1 4 -1;0 -1 4];
Y = -eye(3);
A = [X Y;Y X];
b = sum(A,2);
n = length(b);
x0 = zeros(n,1);
for k = 0:5
    disp('SOR')
    omega = 1+0.2*k;
    [x,k] = SOR_mat(A,b,x0,tol,kmax,omega)
    disp('valore di omega e raggio spettrale matrice di iterazione')
    rho_SOR = max(abs(eig(inv(diag(diag(A))+omega*tril(A,-1))*((1-
omega)*diag(diag(A))-omega*triu(A,1)))));
    fprintf('omega = %e, rho = %e\n',omega,rho_SOR);
    disp('*********************************************************')
    pause
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

SISTEMI LINEARI GRADIENTE

GRADIENTE

```
function [x,k,ier] = gradiente(A,b,x,tol,kmax)
%
% [x,k,ier] = gradiente(A,b,x,tol,kmax)
%
%   A   :   matrice dei coefficienti del sistema lineare simmetrica e definita
positiva
%   b   :   vettore dei termini noti del sistema lineare
%   x   :   vettore colonna, in input soluzione iniziale, in output soluzione
approssimata
%   tol :   tolleranza relativa
%   kmax:   numero massimo di iterazioni consentite
%   k   :   numero di iterate calcolate
%   ier :   indicatore del criterio d'arresto utilizzato, vale 1 se raggiunte
kmax iterazioni,
%           vale 0 se soddisfatta la tolleranza relativa tol
%

% verifica che A sia quadrata
[n,m] = size(A);
if n ~= m
   error('la matrice A non è quadrata');
end

% verifica che x sia un vettore colonna
if ~iscolumn(x)
   error('x deve essere un vettore colonna');
end

% verifica che b sia un vettore colonna
if ~iscolumn(b)
   error('b deve essere un vettore colonna');
end

% verifica che la dimensione di x sia compatibile con quella di A
if n ~= length(x)
   error('A e x devono avere le dimensioni compatibili');
end

% verifica che la dimensione di b sia compatibile con quella di A
if n ~= length(b)
   error('A e b devono avere le dimensioni compatibili');
end

if ~issymmetric(A)
   error('A deve essere simmetrica');
end

if min(eig(A)) <= 0
   error('A deve essere definita positiva');
end

% algoritmo del gradiente o di discesa più ripida
r = b-A*x;
for k = 1:kmax
    s = A*r;
    alpha = r'*r/(r'*s);
    x = x+alpha*r;
```

```
    r = r-alpha*s;
    if norm(r) <= tol * norm(b)
       ier = 0;
       return
    end
end
ier = 1;
```

LAB 03

```
% Laib03: sistemi lineari - metodi del gradiente
clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_3_1*****************')
disp('***********************************************')

for n = [100 196 400]
    for i = 1:2
        if i == 1
           A = gallery('toeppd',n,n,linspace(0,1,n),linspace(0,2*pi,n));
           disp('****************** A = gallery(''toeppd'',n,...)
******************')
           fprintf('n = %e\n',n);
           disp('condizionamento spettrale di A')
           condizionamento_A = cond(A);
           fprintf('%e\n',condizionamento_A);
           A = sparse(A);
        elseif i == 2
           m = sqrt(n);
           A = delsq(numgrid('S',m)); n = (m-2)^2;
           disp('****************** A = delsq(numgrid(''S'',n))
******************')
           fprintf('n = %e\n',n);
           disp('condizionamento spettrale di A')
           condizionamento_A = condest(A);
           fprintf('%e\n',condizionamento_A);
        end
        b = sum(A,2);
        x = zeros(n,1);
        tol = 1.0e-5;
        kmax = 1000;
        disp(' ')
        disp('%%%%%%%%%%%%%%%%%% GAUSS SEIDEL %%%%%%%%%%%%%%%%%%')
        [x,k,ier] = gauss_seidel_mat(A,b,x,tol,kmax);
        disp(['metodo di Gauss-Seidel: eseguite ' num2str(k) ' iterazioni'])
        errore = norm(ones(n,1)-x)/norm(ones(n,1));
        disp(['errore relativo associato alla soluzione calcolata = '
num2str(errore)])
        residuo = norm(b-A*x)/norm(b);
        disp(['residuo associato alla soluzione calcolata diviso ||b||_2 = '
num2str(residuo)])
        disp(' ')

        disp('%%%%%%%%%%%%%%%%%% GRADIENTE %%%%%%%%%%%%%%%%%%')
        x = zeros(n,1);
        [x,k,ier] = gradiente(A,b,x,tol,kmax);
        disp(['metodo del gradiente: eseguite ' num2str(k) ' iterazioni'])
        errore = norm(ones(n,1)-x)/norm(ones(n,1));
```

```
        disp(['errore relativo associato alla soluzione calcolata = '
num2str(errore)])
        residuo = norm(b-A*x)/norm(b);
        disp(['residuo associato alla soluzione calcolata diviso ||b||_2 = '
num2str(residuo)])
        disp(' ')
        disp('%%%%%%%%%%%%%%%%% GRADIENTE CONIUGATO %%%%%%%%%%%%%%%%%')
        x = zeros(n,1);
        [x,ier,rel_res,k] = pcg(A,b,tol,kmax,[],[],x);
        disp(['metodo del gradiente coniugato: eseguite ' num2str(k) '
iterazioni'])
        errore = norm(ones(n,1)-x)/norm(ones(n,1));
        disp(['errore relativo associato alla soluzione calcolata = '
num2str(errore)])
        residuo = norm(b-A*x)/norm(b);
        disp(['residuo associato alla soluzione calcolata diviso ||b||_2 = '
num2str(residuo)])
        disp(' ')
        disp('%%%%%%%%%%%%%%%%% GRADIENTE CONIUGATO PRECONDIZIONATO
%%%%%%%%%%%%%%%%%')
        x = zeros(n,1);
        L1 = ichol(A);
        [x,ier,rel_res,k] = pcg(A,b,tol,kmax,L1,L1',x);
        disp(['metodo del gradiente coniugato precondizionato: eseguite  '
num2str(k) ' iterazioni'])
        errore = norm(ones(n,1)-x)/norm(ones(n,1));
        disp(['errore relativo associato alla soluzione calcolata = '
num2str(errore)])
        residuo = norm(b-A*x)/norm(b);
        disp(['residuo associato alla soluzione calcolata diviso ||b||_2 = '
num2str(residuo)])
        disp(' ')
        pause
    end
end
disp('********************************************FINE
ESERCIZIO********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('************************************************')
disp('*****************esercizio_3_2******************')
disp('************************************************')

N = 10;
A = delsq(numgrid('S',N));
condizionamento_A = condest(A)
n = (N-2)^2;
b = sum(A,2);
tol = 1.0e-12;
kmax = 1000;
x = zeros(n,1);
% metodo del gradiente coniugato
[x0,ier0,rel_res0,k0,resvec0] = pcg(A,b,tol,kmax,[],[],x);
x = zeros(n,1);
L1 = ichol(A);
% metodo del gradiente coniugato precondizionato
```

```
[x1,ier1,rel_res1,k1,resvec1] = pcg(A,b,tol,kmax,L1,L1',x);
semilogy(0:k0,resvec0,'r*',0:k1,resvec1,'b*','linewidth',3)
legend('senza precondizionatore','con precondizionatore')
disp(' ')
disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

INTERPOLAZIONE POLINOMIALE

```
% Laib04: approssimazione, interpolazione
clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_4_1*****************')
disp('***********************************************')

a = 0;
b = 1;
ord = 2:20;
for n = 1:length(ord)
    x = linspace(a,b,ord(n));
    A = vander(x);
    K1(n) = cond(A,1);
    K2(n) = cond(A,2);
    Kinf(n) = cond(A,inf);
end
figure(1)
semilogy(ord,K1,'*',ord,K2,'o',ord,Kinf,'+','linewidth',3)
legend('K_1','K_2','K_\infty')
xlabel('ordine sistema')
ylabel('condizionamento')
for n = 1:length(ord)
    x = linspace(a,b,ord(n));
    A = vander(x);
    t = sum(A,2);
    z = A\t;
    err(n) = norm(z-ones(ord(n),1))/norm(ones(ord(n),1));
end
figure(2)
semilogy(ord,err,'o','linewidth',3)
xlabel('ordine sistema')
ylabel('errore')

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_4_2*****************')
disp('***********************************************')

a = 0;
b = 1;
f = @(x) sin(pi*x);
z = linspace(a,b);
ord = 2:20;
for n = 1:length(ord)
    x = linspace(a,b,ord(n));
    y = f(x);
    c = polyfit(x,y,ord(n)-1);
```

```
    p = polyval(c,z);
    plot(z,f(z),'r',x,y,'g*',z,p,'b','linewidth',3)
    axis([0 1 0 1])
    pause
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause
clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_4_3*****************')
disp('***********************************************')

a = -5;
b = 5;
f = @(x) 1./(1+x.^2);
z = linspace(a,b);
for n = 5:5:20
    x = linspace(a,b,n+1);
    y = f(x);
    c = polyfit(x,y,n);
    p = polyval(c,z);
    err_p = max(abs(f(z)-p));

    fprintf('n = %d, err_p = %e\n',n,err_p);
    plot(x,y,'g*',z,f(z),'r',z,p,'linewidth',3)
    hold on
    legend('dati','funzione di Runge','polinomio interpolante')
    pause
end

disp('**************************************************************************
************')
pause
close all

a = 1;
b = 2;
f = @(x) 1./(1+x.^2);
z = linspace(a,b);
for n = 5:5:20
    x = linspace(a,b,n+1);
    y = f(x);
    c = polyfit(x,y,n);
    p = polyval(c,z);
    err_p = max(abs(f(z)-p));

    fprintf('n = %d, err_p = %e\n',n,err_p);
    plot(x,y,'g*',z,f(z),'r',z,p,'linewidth',3)
    hold on
    legend('dati','funzione di Runge','polinomio interpolante')
    pause
end

disp('**************************************************************************
************')
```

```
pause
close all
a = -5;
b = 5;
f = @(x) 1./(1+x.^2);
z = linspace(a,b);
for n = 5:5:20
    u = -cos((2*[1:n+1]-1)*pi/(2*(n+1)));
    x = (b-a)/2*u+(b+a)/2;
    y = f(x);
    c = polyfit(x,y,n);
    p = polyval(c,z);
    err_p = max(abs(f(z)-p));
    fprintf('n = %d, err_p = %e\n',n,err_p);
    plot(x,y,'g*',z,f(z),'r',z,p,'linewidth',3)
    hold on
    legend('dati','funzione di Runge','polinomio interpolante')
    pause
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_4_4*****************')
disp('***********************************************')

x = 0:3;
y = [1 4 8 16];
c = polyfit(exp(x),y,3);
z = linspace(0,3);
p = polyval(c,exp(z));
plot(x,y,'g*',z,p,'b','linewidth',2)

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

INTERPOLAZIONE – SPLINE

```
% Laib05: approssimazione, funzioni polinomiali a tratti
clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_5_1*****************')
disp('***********************************************')

a = 0;
b = 1;
n = 20;
x = linspace(a,b,n);
f = @(x) sin(x);
y = f(x);
fd = @(x) cos(x);
di = fd(a);
df = fd(b);
z = linspace(a,b,100);
s = spline_vincolata_partizione_uniforme(x,y,di,df,z);
plot(x,y,'g*',z,f(z),'r',z,s,'b','linewidth',2)
hold on
s_matlab = spline(x,[di y df],z);
plot(z,s_matlab,'c','linewidth',2)
err_rel = norm(s-s_matlab,inf)/norm(s_matlab,inf);
disp(['L''errore relativo in norma infinito in 100 punti è pari a
',num2str(err_rel)])


disp('*********************************************FINE
ESERCIZIO*********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_5_2*****************')
disp('***********************************************')

a = -5;
b = 5;
f = @(x) 1./(1+x.^2);
z = linspace(a,b);
for n = 5:5:15
    x = linspace(a,b,n);
    y = f(x);
    s = spline(x,y,z);
    err_s = max(abs(f(z)-s));
    fprintf('n = %d, err_s = %e\n',n,err_s);
    plot(x,y,'g*',z,f(z),'r',z,s,'linewidth',3)
    legend('dati','funzione di Runge','spline interpolante')
    pause
end
```

```
disp('********************************************FINE
ESERCIZIO********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('************************************************')
disp('*****************esercizio_5_3******************')
disp('************************************************')

f = @(x) (1-x.^2).^(5/2);
fd = @(x) (5/2)*(1-x.^2).^(5/2-1).*(-2*x);
di = fd(-1);
df = fd(1);
z = linspace(-1,1);
fz = f(z);
for k = 2:5
    n = 2^k;
    x = -1+2*(0:n)/n;
    y = f(x);
    s = spline(x,y,z);
    s1 = spline(x,[di y df],z);
    figure(1)
    plot(x,y,'ko',z,fz,'r',z,s,'b',z,s1,'g','linewidth',3)
    legend('dati','f(x)','spline not-a-knot','spline vincolata')
    pause
    figure(2)
    semilogy(z,abs(s-fz),'b',z,abs(s1-fz),'g','linewidth',3)
    legend('errore spline not-a-knot','errore spline vincolata')
    err = norm(fz-s,inf);
    err1 = norm(fz-s1,inf);
    fprintf('n = %d, err = %e, err1 = %e\n',n,err,err1);
    pause
end
% la spline vincolata fornisce un'approssimazione più
% accurata della spline not-a-knot in quanto, a differenza di
% quest'ultima, oltre alle condizioni di interpolazione, soddisfa
% due ulteriori condizioni che stabiliscono un legame
% con la funzione f

disp('********************************************FINE
ESERCIZIO********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('************************************************')
disp('*****************esercizio_5_4******************')
disp('************************************************')

a = 0;
b = 1;
n = 20;
```

```
x = linspace(a,b,n);
f = @(x) sin(x);
y = f(x);
z = linspace(a,b);
p = zeros(1,length(z));
for k = 1:length(z)
    [p(k),ier] = poligonale(x,y,z(k));
end
plot(x,y,'g*',z,f(z),'r',z,p,'b','linewidth',3)
hold on
p_matlab = interp1(x,y,z);
plot(z,p_matlab,'c','linewidth',3)
err_rel = norm(p-p_matlab,inf)/norm(p_matlab,inf);
disp(['L''errore relativo in norma infinito in 100 punti è pari a
',num2str(err_rel)])

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_5_5*****************')
disp('***********************************************')

format short e
a = 0;
b = 1;
n = 21;
x = linspace(a,b,n);
f = @(x) sin(x);
y = f(x);
z = linspace(a,b);

p1 = zeros(1,length(z));
p2 = zeros(1,length(z));
p4 = zeros(1,length(z));
for k = 1:length(z)
    [p1(k),ier] = polinomiale_a_tratti(x,y,1,z(k));
    [p2(k),ier] = polinomiale_a_tratti(x,y,2,z(k));
    [p4(k),ier] = polinomiale_a_tratti(x,y,4,z(k));
end
plot(x,y,'g*',z,f(z),'r',z,p1,'c',z,p2,'b',z,p4,'k','linewidth',3)
legend('dati','funzione','d=1','d=2','d=4')
err1 = norm(f(z)-p1,inf);
err2 = norm(f(z)-p2,inf);
err3 = norm(f(z)-p4,inf);
disp(['L''errore in norma infinito in 100 punti per d=1 è pari a
',num2str(err1)])
disp(['L''errore in norma infinito in 100 punti per d=2 è pari a
',num2str(err2)])
disp(['L''errore in norma infinito in 100 punti per d=4 è pari a
',num2str(err3)])

disp('**********************************************FINE
ESERCIZIO**********************************************')
```

EQUAZIONI NON LINEARI

NEWTON

```
function [x,ier] = newton(f,fd,x0,nmax,tol)
%
% [x,ier] = newton(f,fd,x0,nmax,tol)
%
%   f   :   funzione non lineare di cui si vuole
%           calcolare uno zero con il metodo di Newton
%   fd  :   derivata prima di f
%   x0  :   approssimazione iniziale
%   nmax:   numero massimo di iterazioni consentite
%   x   :   vettore contenente le approssimazioni calcolate
%   ier :   indicatore del criterio d'arresto utilizzato, vale 0 se raggiunte
nmax iterazioni,
%           vale 1 se soddisfatta la tolleranza relativa tol
%

ier = 0;
x(1) = x0;
for n = 1:nmax
    x(n+1) = x(n)-f(x(n))/fd(x(n));
    if abs(x(n+1)-x(n)) <= tol*abs(x(n+1))
        ier = 1;
        break
    end
end
```

PUNTO FISSO

```
function [x,ier] = punto_fisso(g,x0,nmax,tol)
%
% [x,ier] = punto_fisso(g,x0,nmax,tol)
%
%   g   :   funzione non lineare di cui si vuole
%           calcolare un punto fisso
%   x0  :   approssimazione iniziale
%   nmax:   numero massimo di iterazioni consentite
%   x   :   vettore contenente le approssimazioni calcolate
%   ier :   indicatore del criterio d'arresto utilizzato, vale 0 se raggiunte
nmax iterazioni,
%           vale 1 se soddisfatta la tolleranza relativa tol
%

ier = 0;
x(1) = x0;
for n = 1:nmax
    x(n+1) = g(x(n));
    if abs(x(n+1)-x(n)) <= tol*abs(x(n+1))
        ier = 1;
        break
    end
end
```

LAB 06

```
% Laib06: equazioni e sistemi non lineari
clear all
close all
clc
disp('**********************************************')
disp('****************esercizio_6_1****************')
disp('**********************************************')

syms x % variabile x simbolica
% costruzione simbolica del polinomio
% p(x) = (x-1)(x-2)...(x-20)
e = prod(x-[1:20]);
e = expand(e);
c = coeffs(e);

% in c i coefficienti sono ordinati inversamente
% rispetto alla rappresentazione polinomiale di Matlab
% (c(1)x^20 + c(2)x^19 + ...)
c = flip(c);

% calcolo degli zeri del polinomio p(x) con
% i coefficienti ''esatti''
x_exact = roots(c);
x_exact = flip(x_exact);

% ritorno da simbolico ad aritmetica reale
c = double(c);

% calcolo degli zeri del polinomio p(x) con
% i coefficienti ''perturbati''
x = roots(c);

% errore relativo commesso su ciascuna radice
x_exact_d = double(x_exact);
err_rel = max((abs(x-x_exact_d))./((abs(x_exact_d))));
disp(['Il max errore relativo associato agli zeri del polinomio perturbato è
pari a ',num2str(err_rel)])

disp('**********************************************FINE
ESERCIZIO*********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('**********************************************')
disp('****************esercizio_6_2****************')
disp('**********************************************')

nmax = 100;
tol = 1.0e-10;
%
disp('***************funzione 1***************')
f = @(x) x.^2-2;
fd = @(x) 2*x;
figure
```

```
x = linspace(-2,2);
plot(x,f(x),'b',x,0*x,'r','linewidth',3)
grid on
x0 = input('fornisci x0 = ');
[x,ier] = newton(f,fd,x0,nmax,tol);
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)

% calcolo dell'ordine sperimentale di convergenza
osc = [0 log(e(2:end))./log(e(1:end-1))];
% stampa delle iterate calcolate, dei corrispondenti
% errori e dell'ordine sperimentale di convergenza
fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end
pause
%
disp('***************funzione 2***************')
f = @(x) x.^3-x-1;
fd = @(x) 3*x.^2-1;
figure
x = linspace(-2,2);
plot(x,f(x),'b',x,0*x,'r','linewidth',3)
grid on
x0 = input('fornisci x0 = ');
[x,ier] = newton(f,fd,x0,nmax,tol);
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end
pause
%
disp('***************funzione 3***************')
f = @(x) (x-2.^(-x)).^3;
fd = @(x) 3*(x-2.^(-x)).^2*(1+2.^(-x).*log(2));
figure
x = linspace(-2,2);
plot(x,f(x),'b',x,0*x,'r','linewidth',3)
grid on
x0 = input('fornisci x0 = ');
[x,ier] = newton(f,fd,x0,nmax,tol);
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end
pause
%
disp('***************funzione 4***************')
```

```
f = @(x) exp(x)-2*x.^2;
fd = @(x) exp(x)-4*x;
figure
x = linspace(-2,2);
plot(x,f(x),'b',x,0*x,'r','linewidth',3)
grid on
x0 = input('fornisci x0 = ')
[x,ier] = newton(f,fd,x0,nmax,tol);
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_6_3*****************')
disp('***********************************************')

nmax = 100;
tol = 1.0e-10;
disp('****************funzione 1****************')
g = @(x) -sqrt(exp(x)/2);
figure
x = linspace(-2,2);
plot(x,g(x),'b',x,x,'r','linewidth',3)
grid on
x0 = input('fornisci x0 = ')
[x,ier] = punto_fisso(g,x0,nmax,tol);
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end
pause
%
disp('****************funzione 2****************')
g = @(x) (2*x.^3+4*x.^2+10)./(3*x.^2+8*x);
figure
x = linspace(1,2);
plot(x,g(x),'b',x,x,'r','linewidth',3)
grid on
x0 = input('fornisci x0 = ')
```

```
[x,ier] = punto_fisso(g,x0,nmax,tol);
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_6_4*****************')
disp('***********************************************')

close all
clear all
nmax = 100;
tol = 1.0e-10;
%
disp('****************funzione 1****************')
g = @(x) -log(x);
figure
x = linspace(0.2,0.8);
plot(x,g(x),'b',x,x,'r','linewidth',3)
grid on
x0 = input('fornisci x0 = ')
[x,ier] = punto_fisso(g,x0,nmax,tol);
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end
% non converge |g'(csi)|>1 (le iterate sono complesse)
pause
%
disp('****************funzione 2****************')
g = @(x) exp(-x);
figure
x = linspace(0.2,0.8);
plot(x,g(x),'b',x,x,'r','linewidth',3)
grid on
x0 = input('fornisci x0 = ');
[x,ier] = punto_fisso(g,x0,nmax,tol)
figure
n = length(x);
```

```
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end
% converge |g'(csi)| è circa 0.607
pause
%
disp('****************funzione 3 (k=2) ****************')
g = @(x) (x+exp(-x))/2;
figure
x = linspace(0.2,0.8);
plot(x,g(x),'b',x,x,'r','linewidth',3)
grid on
x0 = input('fornire x0 = ');
[x,ier] = punto_fisso(g,x0,nmax,tol);
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end
% converge |g'(csi)| è circa 0.197
pause
%
disp('****************funzione 4 (k=3/2) ****************')
g = @(x) (x+2*exp(-x))/3;
figure
x = linspace(0.2,0.8);
plot(x,g(x),'b',x,x,'r','linewidth',3)
grid on
x0 = input('fornire x0 = ');
[x,ier] = punto_fisso(g,x0,nmax,tol);
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end
% converge |g'(csi)| è circa 0.071
pause
%
disp('****************funzione 5****************')
% newton
f = @(x) x+log(x);
fd = @(x) 1+(1/x);
figure
x = linspace(0.2,0.8);
plot(x,f(x),'b',x,0*x,'r','linewidth',3)
grid on
x0 = input('fornire x0 = ');
[x,ier] = newton(f,fd,x0,nmax,tol);
```

```
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_6_5*****************')
disp('***********************************************')

nmax = 100;
tol = 1.0e-10;
f = @(x) sqrt(pi)/2*erf(x)-1/2;
fd = @(x) exp(-x.^2);
figure
x = linspace(-2,2);
plot(x,f(x),'b',x,0*x,'r','linewidth',3)
grid on
x0 = input('fornisci x0 = ')
[x,ier] = newton(f,fd,x0,nmax,tol);
figure
n = length(x);
e = abs(x(2:n)-x(1:n-1));
semilogy(1:n-1,e,'linewidth',3)
pause
osc = [0 log(e(2:end))./log(e(1:end-1))];

fprintf('%10s\t%10s\t\t%10s\t%10s\r\n','n','approssimazione','errore','p');
for i = 1:n-1
    fprintf('%10.0d\t%6.16e\t%6.4d\t%6.4e\n',i,x(i),e(i),osc(i));
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

SISTEMI NON LINEARI

NEWTON SISTEMI

```
function [x,ier] = newton_system(F,J,x0,tol,nmax)
%
% Uso:     [x,ier] = newton_system(F,J,x_0,toll,nmax)
% Scopo :  calcola a partire da x_0 una soluzione x del sistema di
%          equazioni non lineari F(x)=0 con tolleranza relativa toll
%          mediante il metodo di Newton
% Input:   F = sistema di equazioni
%          J = matrice Jacobiana del sistema
%         x0 = vettore colonna iniziale
%        tol = tolleranza relativa desiderata
%       nmax = massimo numero di iterazioni da eseguire
% Output:  x = approssimazione della soluzione del sistema
%        ier = indicatore del criterio d'arresto soddisfatto (vale 0
%              se si raggiunge nmax, vale 1 se si raggiunge toll)
%
x = x0;
ier = 0;
for n = 1:nmax
  e_n = -feval(J,x)\feval(F,x);
  x = x + e_n;
  if norm(e_n) <= tol*norm(x)
     ier = 1;
     break
  end
end
```

LAB 07

```
% Laib07: sistemi di equazioni non lineari
clear all
close all
clc
disp('************************************************')
disp('*****************esercizio_7_1*****************')
disp('************************************************')

% dati sistema e metodo
F1 = @(x) [x(1)^2+2*x(1)*x(2)+x(3); x(2)^3+x(3)^2; x(1)*x(3)-1];
J1 = @(x) [2*(x(1)+x(2)) 2*x(1) 1; 0 3*x(2)^2 2*x(3); x(3) 0 x(1)];
n_max = 20;
tol = 1.0e-10;
exact = [1;-1;1];
x0 = [0.5;-0.5;0.1];
[x,ier] = newton_system(F1,J1,x0,tol,n_max);
err = norm(exact-x);
disp(['err = ',num2str(err)])

% il metodo di Newton è sensibile alla scelta del punto iniziale
x0 = [-0.5;-0.5;-0.1]; %converge a un'altra soluzione
[x,ier] = newton_system(F1,J1,x0,tol,1000)

x0 = [-5;0;-5]; %converge a un'altra soluzione diversa dalle precedenti
[x,ier] = newton_system(F1,J1,x0,tol,1000)

disp('*********************************************FINE
ESERCIZIO*********************************************')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('*********************************************')
disp('****************esercizio_7_2****************')
disp('*********************************************')

d1 = 4.5;
d2 = 6;
r1 = 5;
r2 = 9;
syms a b
f1 = d1*cos(b)+d2*cos(a+b)-r1;
f2 = d1*sin(b)+d2*sin(a+b)-r2;
figure(1)
fimplicit(f1,'b','linewidth',2)
hold on
fimplicit(f2,'r','linewidth',2)
grid on

F2 = @(x) [d1*cos(x(2))+d2*cos(x(1)+x(2))-r1;
           d1*sin(x(2))+d2*sin(x(1)+x(2))-r2];
J2 = @(x) [-d2*sin(x(1)+x(2))  -d1*sin(x(2))-d2*sin(x(1)+x(2));
            d2*cos(x(1)+x(2))   d1*cos(x(2))+d2*cos(x(1)+x(2))];

x10 = [-0.4; 1.3];
x20 = [0.25; 0.75];

options = optimoptions('fsolve','OptimalityTolerance',1.0e-10);
[x1_m,FVAL1,EXITFLAG1,OUTPUT1] = fsolve(F2,x10,options);
[x2_m,FVAL2,EXITFLAG2,OUTPUT2] = fsolve(F2,x20,options);

nmax = 20;
tol = 1.0e-4;
[x1,ier1] = newton_system(F2,J2,x10,tol,nmax);
[x2,ier2] = newton_system(F2,J2,x20,tol,nmax);

figure(2)
plot([0 d1*cos(x1(2)) d1*cos(x1(2))+d2*cos(x1(1)+x1(2))],[0 d1*sin(x1(2))
d1*sin(x1(2))+d2*sin(x1(1)+x1(2))],'b','linewidth',3)
hold on
plot([0 d1*cos(x2(2)) d1*cos(x2(2))+d2*cos(x2(1)+x2(2))],[0 d1*sin(x2(2))
d1*sin(x2(2))+d2*sin(x2(1)+x2(2))],'g','linewidth',3)
grid on
legend('soluzione 1','soluzione 2')

err1 = norm(x1_m-x1)/norm(x1_m);
err2 = norm(x2_m-x2)/norm(x2_m);
disp('errori relativi associati a due soluzioni in [0,2 pi] calcolate con il
metodo di Newton')
disp(['err1 = ',num2str(err1)])
disp(['err2 = ',num2str(err2)])
disp(' ')
h = 2^(-30);
J2_app = @(x) [d2*(cos(x(1)+h+x(2))-cos(x(1)+x(2)))/h   d1*(cos(x(2)+h)-
cos(x(2)))/h + d2*(cos(x(1)+h+x(2))-cos(x(1)+x(2)))/h;
```

```
              d2*(sin(x(1)+h+x(2))-sin(x(1)+x(2)))/h    d1*(sin(x(2)+h)-
sin(x(2)))/h + d2*(sin(x(1)+h+x(2))-sin(x(1)+x(2)))/h];

[x1app,ier1app] = newton_system(F2,J2_app,x10,tol,nmax);
[x2app,ier2app] = newton_system(F2,J2_app,x20,tol,nmax);

err1app30 = norm(x1_m-x1app)/norm(x1_m);
err2app30 = norm(x2_m-x2app)/norm(x2_m);

disp('errori relativi associati a due soluzioni in [0,2 pi] calcolate con il
metodo di Newton')
disp('ove si usa un''approssimazione della matrice jacobiana')
disp(['h = 2^-30, err1app = ',num2str(err1app30)])
disp(['h = 2^-30, err2app = ',num2str(err2app30)])
disp('si ottengono errori aventi lo stesso ordine di grandezza di quelli
precedenti')
disp(' ')

h = 2^(-50);
J2_app = @(x) [d2*(cos(x(1)+h+x(2))-cos(x(1)+x(2)))/h   d1*(cos(x(2)+h)-
cos(x(2)))/h + d2*(cos(x(1)+h+x(2))-cos(x(1)+x(2)))/h;
               d2*(sin(x(1)+h+x(2))-sin(x(1)+x(2)))/h    d1*(sin(x(2)+h)-
sin(x(2)))/h + d2*(sin(x(1)+h+x(2))-sin(x(1)+x(2)))/h];

[x1app,ier1] = newton_system(F2,J2_app,x10,tol,nmax);
[x2app,ier2] = newton_system(F2,J2_app,x20,tol,nmax);

err1app50 = norm(x1_m-x1app)/norm(x1_m);
err2app50 = norm(x2_m-x2app)/norm(x2_m);

disp('errori relativi associati a due soluzioni  in [0,2 pi] calcolate con il
metodo di Newton')
disp('ove si usa un''approssimazione della matrice jacobiana teoricamente più
accurata!!!')
disp(['h = 2^-50, err1app = ',num2str(err1app50)])
disp(['h = 2^-50, err2app = ',num2str(err2app50)])
disp('si ottengono errori aventi ordine di grandezza maggiore a causa della
cancellazione numerica')
disp(' ')

% formule di prostaferesi per eliminare la cancellazione numerica
% che si verifica al numeratore dei rapporti incrementali.

cmc1 = @(x) -2/h*sin(h/2)*sin((2*x(1)+h+2*x(2))/2);
cmc2 = @(x) -2/h*sin(h/2)*sin((h+2*x(2))/2);
sms1 = @(x) 2/h*cos((2*x(1)+h+2*x(2))/2)*sin(h/2);
sms2 = @(x) 2/h*cos((h+2*x(2))/2)*sin(h/2);

J2_appc = @(x) [d2*cmc1(x)   d1*cmc2(x) + d2*cmc1(x);
                d2*sms1(x)   d1*sms2(x) + d2*sms1(x)];

[x1app,ier1] = newton_system(F2,J2_appc,x10,tol,nmax);
[x2app,ier2] = newton_system(F2,J2_appc,x20,tol,nmax);

err1app50 = norm(x1_m-x1app)/norm(x1_m);
err2app50 = norm(x2_m-x2app)/norm(x2_m);

disp('errore relativo associato a due soluzioni  in [0,2 pi] calcolate con il
metodo di Newton')
disp('ove si usa un''approssimazione della matrice jacobiana teoricamente e
numericamente più accurata')
disp(['h = 2^-50, err1appc = ',num2str(err1app50)])
disp(['h = 2^-50, err2appc = ',num2str(err2app50)])
```

```
disp('la cancellazione numerica è stata eliminata e si ottengono errori aventi
lo stesso ordine di grandezza di quelli che si ottengono con la matrice
jacobiana')

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('************************************************')
disp('*****************esercizio_7_3******************')
disp('************************************************')

x = -3:.01:3;
y = x;
[X,Y] = meshgrid(x,y);
Z = peaks(X,Y);
meshc(X,Y,Z)
axis([-3 3 -3 3 -10 10]);
syms x y
z =  3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) - 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-
y.^2) - 1/3*exp(-(x+1).^2 - y.^2);
zx = diff(z,x);
zy = diff(z,y);
% tentativo (che fallisce!) di ottenere una soluzione simbolica del sistema
M = solve(zx,zy);

% si procede quindi numericamente
F3 = @(x) [(exp(- (x(1) + 1)^2 - x(2)^2)*(2*x(1) + 2))/3 + 3*exp(- (x(2) + 1)^2
- x(1)^2)*(2*x(1) - 2) + exp(- x(1)^2 - x(2)^2)*(30*x(1)^2 - 2) - 6*x(1)*exp(-
(x(2) + 1)^2 - x(1)^2)*(x(1) - 1)^2 - 2*x(1)*exp(- x(1)^2 - x(2)^2)*(10*x(1)^3 -
2*x(1) + 10*x(2)^5);
      (2*x(2)*exp(- (x(1) + 1)^2 - x(2)^2))/3 + 50*x(2)^4*exp(- x(1)^2 - x(2)^2)
- 3*exp(- (x(2) + 1)^2 - x(1)^2)*(2*x(2) + 2)*(x(1) - 1)^2 - 2*x(2)*exp(- x(1)^2
- x(2)^2)*(10*x(1)^3 - 2*x(1) + 10*x(2)^5)];

x0 = [0.4;1.5];
options = optimoptions('fsolve','OptimalityTolerance',1.0e-10);
[sol,FVAL,EXITFLAG,OUTPUT] = fsolve(F3,x0,options);

z = @(x,y) 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) - 10*(x/5 - x.^3 - y.^5).*exp(-
x.^2-y.^2) - 1/3*exp(-(x+1).^2 - y.^2);
hold on
% rappresentazione grafica del massimo
plot3(sol(1),sol(2),z(sol(1),sol(2)),'b*','linewidth',3)
MAX = [sol(1),sol(2),z(sol(1),sol(2))]

% la matrice Jacobiana è stata calcolata con il comando simbolico jacobian!!
J3 = @(x) [ (2*exp(- (x(1) + 1)^2 - x(2)^2))/3 + 6*exp(- (x(2) + 1)^2 - x(1)^2)
+ 60*x(1)*exp(- x(1)^2 - x(2)^2) - 6*exp(- (x(2) + 1)^2 - x(1)^2)*(x(1) - 1)^2 -
2*exp(- x(1)^2 - x(2)^2)*(10*x(1)^3 - 2*x(1) + 10*x(2)^5) - (exp(- (x(1) + 1)^2
- x(2)^2)*(2*x(1) + 2)^2)/3 + 12*x(1)^2*exp(- (x(2) + 1)^2 - x(1)^2)*(x(1) -
1)^2 + 4*x(1)^2*exp(- x(1)^2 - x(2)^2)*(10*x(1)^3 - 2*x(1) + 10*x(2)^5) -
12*x(1)*exp(- (x(2) + 1)^2 - x(1)^2)*(2*x(1) - 2) - 4*x(1)*exp(- x(1)^2 -
x(2)^2)*(30*x(1)^2 - 2),  4*x(1)*x(2)*exp(- x(1)^2 - x(2)^2)*(10*x(1)^3 - 2*x(1)
+ 10*x(2)^5) - (2*x(2)*exp(- (x(1) + 1)^2 - x(2)^2)*(2*x(1) + 2))/3 -
2*x(2)*exp(- x(1)^2 - x(2)^2)*(30*x(1)^2 - 2) - 100*x(1)*x(2)^4*exp(- x(1)^2 -
```

```
x(2)^2) - 3*exp(- (x(2) + 1)^2 - x(1)^2)*(2*x(1) - 2)*(2*x(2) + 2) +
6*x(1)*exp(- (x(2) + 1)^2 - x(1)^2)*(2*x(2) + 2)*(x(1) - 1)^2; 4*x(1)*x(2)*exp(-
x(1)^2 - x(2)^2)*(10*x(1)^3 - 2*x(1) + 10*x(2)^5) - (2*x(2)*exp(- (x(1) + 1)^2 -
x(2)^2)*(2*x(1) + 2))/3 - 2*x(2)*exp(- x(1)^2 - x(2)^2)*(30*x(1)^2 - 2) -
100*x(1)*x(2)^4*exp(- x(1)^2 - x(2)^2) - 3*exp(- (x(2) + 1)^2 - x(1)^2)*(2*x(1)
- 2)*(2*x(2) + 2) + 6*x(1)*exp(- (x(2) + 1)^2 - x(1)^2)*(2*x(2) + 2)*(x(1) -
1)^2, (2*exp(- (x(1) + 1)^2 - x(2)^2))/3 - 6*exp(- (x(2) + 1)^2 - x(1)^2)*(x(1)
- 1)^2 - 2*exp(- x(1)^2 - x(2)^2)*(10*x(1)^3 - 2*x(1) + 10*x(2)^5) +
200*x(2)^3*exp(- x(1)^2 - x(2)^2) - 200*x(2)^5*exp(- x(1)^2 - x(2)^2) -
(4*x(2)^2*exp(- (x(1) + 1)^2 - x(2)^2))/3 + 4*x(2)^2*exp(- x(1)^2 -
x(2)^2)*(10*x(1)^3 - 2*x(1) + 10*x(2)^5) + 3*exp(- (x(2) + 1)^2 -
x(1)^2)*(2*x(2) + 2)^2*(x(1) - 1)^2];
tol = 1.0e-10;
nmax = 700;
% il metodo di Newton è sensibile alla scelta del valore iniziale x0
x0 = [0.5;1.5];
[x3,ier] = newton_system(F3,J3,x0,tol,nmax)
x3
```

INTEGRALI

```
% Laib08: integrali
clear all
close all
clc
disp('***********************************************')
disp('****************esercizio_8_1*****************')
disp('***********************************************')

kmax = 128;
tol = 1.0e-10;
for j = 1:6
    disp(['integrale numero ',num2str(j)])
    switch j
      case 1 % esempio 1
           clearvars -except j kmax tol
           a = 0;
           b = 1;
           f = @(x) exp(x);
           exact = exp(1)-1;
           % funzione di classe C^inf
      case 2 % esempio 2
           clearvars -except j kmax tol
           a = 0;
           b = 1;
           f = @(x) cos(x);
           exact = sin(1);
           % funzione di classe C^inf
      case 3 % esempio 3
           clearvars -except j kmax tol
           a = 0.01;
           b = 1.1;
           f = @(x) 1./x.^4;
           exact = 1/3*(10^6-1.1^(-3));
           % funzione di classe C^inf ma con singolarità vicina
      case 4 % esempio 4
           clearvars -except j kmax tol
           a = 0;
           b = 1;
           f = @(x) sqrt(x);
           exact = 2/3;
           % funzione di classe C^0
      case 5 % esempio 5
           clearvars -except j kmax tol
           a = 0;
           b = 1;
           f = @(x) sin(99*pi*x);
           x = linspace(a,b,1000);
           plot(x,f(x),'linewidth',3)
           exact = 2/(99*pi);
           % funzione di classe C^inf ma molto oscillante
      case 6 % esempio 6
           clearvars -except j kmax tol
           a = 0;
           b = 1;
           f = @(x) sin(100*pi*x);
           x = linspace(a,b,1000);
           plot(x,f(x),'linewidth',3)
           exact = 0;
           % funzione di classe C^inf ma molto oscillante
```

```
           % dispari rispetto a x =1/2
    end
    k = 1;
    for n = 2:2:kmax
        num(k) = n;
        [nodi,pesi] = gaussq(1,num(k),0,0,0,[0 0]');
        nodi_s = (b-a)/2*nodi+(b+a)/2;
        pesi_s = (b-a)/2*pesi;
        int(k) = pesi_s*f(nodi_s)';
        err(k) = abs(exact-int(k));%/abs(exact);
        if err(k)<= tol
           break
        elseif n < kmax
           k= k+1;
        end
    end
    fprintf('%10s\t%10s\t\t%10s\r\n','n','approssimazione','errore');
    for i = 1:k
        fprintf('%10.0d\t%6.16e\t%6.4d\t\n',num(i),int(i),err(i));
    end
    pause
end
disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('*************************************************')
disp('*****************esercizio_8_2*******************')
disp('*************************************************')

kmax = 128;
tol = 1.0e-10;
f = @(x) x.^2.*sin(x.^2);
exact = 2^(1/4)/16*sqrt(pi*(sqrt(2)+2));

k = 1;
for n = 2:2:kmax
    num(k) = n;
    % formula di quadratura di Gauss-Hermite
    [nodi,pesi] = gaussq(4,n,0,0,0,[0 0]');
    int(k) = 1/2*pesi*f(nodi)';
    err(k) = abs(exact-int(k))/abs(exact);
    if err(k)<= tol
       break
    elseif n < kmax
       k= k+1;
    end
end

fprintf('%10s\t%10s\t\t%10s\r\n','n','approssimazione','errore');
for i = 1:k
    fprintf('%10.0d\t%6.16e\t%6.4d\t\n',num(i),int(i),err(i));
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_8_3*****************')
disp('***********************************************')

kmax = 128;
tol = 1.0e-10;
f = @(x) exp(-1)*(x+1).^(3/2);
exact = 5/(2*exp(1)) + 3/4*sqrt(pi)*(1-erf(1));

k = 1;
for n = 2:2:kmax
    num(k) = n;
    % formula di quadratura di Gauss-Laguerre
    [nodi,pesi] = gaussq(6,n,0,0,0,[0 0]');
    int(k) = pesi*f(nodi)';
    err(k) = abs(exact-int(k))/abs(exact);
    if err(k)<= tol
       break
    elseif n < kmax
       k= k+1;
    end
end

fprintf('%10s\t%10s\t\t%10s\r\n','n','approssimazione','errore');
for i = 1:k
    fprintf('%10.0d\t%6.16e\t%6.4d\t\n',num(i),int(i),err(i));
end

disp('***********************************************FINE
ESERCIZIO*********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_8_4*****************')
disp('***********************************************')

f = @(x) x.^9+x.^8+x.^7+x.^2+9;
exact = (1251*pi)/128;
% formula di quadratura di Gauss-Chebichev
n = 5;
int = pi/n*sum(f(cos((2*[1:n]-1)*pi/(2*n))));
err = abs(int-exact)/abs(exact);
disp(['L''errore relativo per n = ' num2str(n),' è ',num2str(err)])

disp('***********************************************FINE
ESERCIZIO*********************************************')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_8_5*****************')
disp('***********************************************')

clear all
close all
clc
f = @(x) (cos(x)).^2.*exp(sin(2*x));
for j = 1:2
    disp(['integrale numero ',num2str(j)])
    switch j
      case 1
           clearvars -except f
           a = 0;
           b = 2*pi;
           exact = 3.9774632605064206e+00;
      case 2
           clearvars -except f
           a = 0;
           b = 1;
           exact = 1.429777221309004;
    end
    k = 0;
    for r = 2:10
        k = k+1;
        n(k) = 2^r;
        % formula dei trapezi
        x = linspace(a,b,n(k));
        y = f(x);
        int_T(k) = (b-a)/(2*(n(k)-1))*(y(1)+2*sum(y(2:n(k)-1))+y(n(k)));
        err_T(k) = abs(exact-int_T(k))/abs(exact);
        % formula di quadratura Gauss-Legendre
        [nodi,pesi] = gaussq(1,n(k),0,0,0,[0 0]');
        nodi_s = (b-a)/2*nodi+(b+a)/2;
        pesi_s = (b-a)/2*pesi;
        int_GL(k) = pesi_s*f(nodi_s)';
        err_GL(k) = abs(exact-int_GL(k))/abs(exact);
    end
    fprintf('%5s\t%15s\t\t\t\t%15s\r\n','n','trapezi','Gauss-Legendre');
    for i = 1:k
        fprintf('%5.0d\t%6.16e\t%6.16e\t\n',n(i),int_T(i),int_GL(i));
    end
    semilogy(n,err_T,'r',n,err_GL,'b','linewidth',3)
    legend('trapezi','Gauss-Legendre')
    pause
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause
```

```
clear all
close all
clc
disp('**********************************************')
disp('****************esercizio_8_6****************')
disp('**********************************************')

clear all
close all
clc
f = @(x,y) exp(x+y);
a = 0;
b = 2*pi;
k = 0;
for r = 2:10
    k = k+1;
    n(k) = 2^r;
    % formula dei trapezi
    t = linspace(a,b,n(k));
    % equazioni parametriche del dominio di integrazione
    x = 2*cos(t);
    y = 3*sin(t);
    jac = sqrt((-2*sin(t)).^2+(3*cos(t)).^2);
    z = f(x,y).*jac;
    int_T(k) = (b-a)/(2*(n(k)-1))*(z(1)+2*sum(z(2:n(k)-1))+z(n(k)));
end
fprintf('%5s\t%15s\r\n','n','trapezi');
for i = 1:k
    fprintf('%5.0d\t%6.16e\t\n',n(i),int_T(i));
end

disp('*********************************************FINE
ESERCIZIO*********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



clear all
close all
clc
disp('**********************************************')
disp('****************esercizio_8_5****************')
disp('**********************************************')

clear all
close all
clc
f = @(x) (cos(x)).^2.*exp(sin(2*x));
for j = 1:2
    disp(['integrale numero ',num2str(j)])
    switch j
      case 1
           clearvars -except f
           a = 0;
           b = 2*pi;
           exact = 3.9774632605064206e+00;
```

```
      case 2
           clearvars -except f
           a = 0;
           b = 1;
           exact = 1.429777221309004;
    end
    k = 0;
    for r = 2:10
        k = k+1;
        n(k) = 2^r;
        % formula dei trapezi
        x = linspace(a,b,n(k));
        y = f(x);
        int_T(k) = (b-a)/(2*(n(k)-1))*(y(1)+2*sum(y(2:n(k)-1))+y(n(k)));
        err_T(k) = abs(exact-int_T(k))/abs(exact);
        % formula di quadratura Gauss-Legendre
        [nodi,pesi] = gaussq(1,n(k),0,0,0,[0 0]');
        nodi_s = (b-a)/2*nodi+(b+a)/2;
        pesi_s = (b-a)/2*pesi;
        int_GL(k) = pesi_s*f(nodi_s)';
        err_GL(k) = abs(exact-int_GL(k))/abs(exact);
        % formula di quadratura Gauss-Legendre a 2 nodi composta
        x = linspace(a,b,n(k));
        h = (b-a)/(n(k)-1);
        y1 = f(h/2*(1-1/sqrt(3))+x(1:n(k)-1));
        y2 = f(h/2*(1+1/sqrt(3))+x(1:n(k)-1));
        int_GL_comp(k) = h/2*sum(y1+y2);
        err_GL_comp(k) = abs(exact-int_GL_comp(k))/abs(exact);
    end
    fprintf('%5s\t%15s\t\t\t\t%15s\r\n','n','trapezi','Gauss-Legendre');
    for i = 1:k

fprintf('%5.0d\t%6.16e\t%6.16e\t%6.16e\t\n',n(i),int_T(i),int_GL(i),int_G
L_comp(i));
    end
    fprintf('%5s\t%15s\t\t\t\t%15s\r\n','n','trapezi','Gauss-Legendre');
    for i = 1:k

fprintf('%5.0d\t%6.16e\t%6.16e\t%6.16e\t\n',n(i),err_T(i),err_GL(i),err_G
L_comp(i));
    end
    semilogy(n,err_T,'r-*',n,err_GL,'b-o',n,err_GL_comp,'k-
s','linewidth',3)
    legend('trapezi','Gauss-Legendre','Gauss-Legendre composta')
    pause
end
```

EQUAZIONI DIFFERENZIALI

EULERO ESPLICITO

```
function [x,y] = Eulero_esp(f,x_0,y_0,x_N,N)
%
% Uso:     [x,y] = Eulero_esp(f,x_0,y_0,x_N,N)
% Scopo:   risolve l'equazione differenziale y'(x)=f(x,y(x))
%          con condizione iniziale y(x_0)=y_0,
%          mediante il metodo di Eulero esplicito
% Input:   f = funzione nota f(x,y) del problema
%        x_0 = punto iniziale
%        y_0 = valore iniziale
%        x_N = punto finale
%          N = controlla l'ampiezza del passo h
% Output:  x = vettore (colonna) contenente N+1 nodi dell'intervallo
%              [x_0,x_N], equidistanti con passo h=(x_N-x_0)/N
%          y = vettore colonna contenente le N+1 approssimazioni di y(x),
%              nei punti del vettore x
%
h = (x_N-x_0)/N;
x = (x_0:h:x_N)';
y = zeros(N+1,1);
y(1) = y_0;
for n = 1:N
   fn = f(x(n),y(n));
   y(n+1) = y(n)+h*fn;
end
```

EULERO ESPLICITO SISTEMI

```
function [x,y] = Eulero_esp_system(f,x_0,y_0,x_N,N)
%
% Uso:     [x,y] = Eulero_esp_sys(f,x_0,y_0,x_N,N)
% Scopo:   risolve un sistema di m equazioni differenziali
%          y'(x)=f(x,y(x)) con condizioni iniziali y(x_0)=y_0
%          mediante il metodo di Eulero esplicito
% Input:   f = funzione nota f(x,y) del problema
%        x_0 = punto iniziale
%        y_0 = vettore (colonna) contenente gli m valori iniziali
%        x_N = punto finale
%          N = controlla l'ampiezza del passo h
% Output:  x = vettore (colonna) contenente N+1 nodi dell'intervallo
%              [x_0,x_N], equidistanti con passo h=(x_N-x_0)/N
%          y = matrice N+1 x m contenente le approssimazioni di y_1(x),
%              y_2(x),...,y_m(x) nei punti del vettore x
%
h = (x_N-x_0)/N;
x = (x_0:h:x_N)';
m = length(y_0);
y = zeros(m,N+1);
y(:,1) = y_0;
for n = 1:N
   fn = f(x(n),y(:,n));
   y(:,n+1) = y(:,n)+h*fn;
end
y = y';
```

HEUN

```
function [x,y] = Heun(f,x_0,y_0,x_N,N)
%
% Uso:     [x,y] = Heun(f,x_0,y_0,x_N,N)
% Scopo:   risolve l'equazione differenziale y'(x)=f(x,y(x))
%          con condizione iniziale y(x_0)=y_0,
%          mediante il metodo di Heun
% Input:   f = funzione nota f(x,y) del problema
%        x_0 = punto iniziale
%        y_0 = valore iniziale
%        x_N = punto finale
%          N = controlla l'ampiezza del passo h
% Output:  x = vettore (colonna) contenente N+1 nodi dell'intervallo
%              [x_0,x_N], equidistanti con passo h=(x_N-x_0)/N
%          y = vettore N+1 x m contenente le approssimazioni di y(x),
%              nei punti del vettore x
%
h = (x_N-x_0)/N;
x = (x_0:h:x_N)';
y = zeros(N+1,1);
y(1) = y_0;
for n = 1:N
   k1 = f(x(n),y(n));
   k2 = f(x(n)+h,y(n)+h*k1);
   y(n+1) = y(n)+h/2*(k1+k2);
end
```

HEUN SISTEMI

```
function [x,y] = Heun_system(f,x_0,y_0,x_N,N)
%
% Uso:     [x,y] = Heun_system(f,x_0,y_0,x_N,N)
% Scopo:   risolve un sistema di m equazioni differenziali
%          y'(x) = f(x,y(x)) con condizioni iniziali y(x_0)=y_0
%          mediante il metodo di Heun
% Input:   f = funzione nota f(x,y) del problema
%        x_0 = punto iniziale
%        y_0 = vettore (colonna) contenente gli m valori iniziali
%        x_N = punto finale
%          N = controlla l'ampiezza del passo h
% Output:  x = vettore (colonna) contenente N+1 nodi dell'intervallo
%              [x_0,x_N], equidistanti con passo h=(x_N-x_0)/N
%          y = matrice N+1 x m contenente le approssimazioni di y_1(x),
%              y_2(x),...,y_m(x) nei punti del vettore x
%
h = (x_N-x_0)/N;
x = (x_0:h:x_N)';
m = length(y_0);
y = zeros(m,N+1);
y(:,1) = y_0;
for n = 1:N
    f1 = f(x(n),y(:,n));
    f2 = f(x(n)+h,y(:,n)+h*f1);
    y(:,n+1) = y(:,n)+h/2*(f1+f2);
end
y = y';
```

RUNGE – KUTTA 4

```
function [x,y] = Runge_Kutta_4(f,x_0,y_0,x_N,N)
%
% Uso:     [x,y] = Runge_Kutta_4(f,x_0,y_0,x_N,N)
% Scopo:   risolve l'equazione differenziale y'(x)=f(x,y(x))
%          con condizione iniziale y(x_0)=y_0,
%          mediante il metodo di Runge Kutta di ordine 4
% Input:   f = funzione nota f(x,y) del problema
%        x_0 = punto iniziale
%        y_0 = valore iniziale
%        x_N = punto finale
%          N = controlla l'ampiezza del passo h
% Output:  x = vettore (colonna) contenente N+1 nodi dell'intervallo
%              [x_0,x_N], equidistanti con passo h=(x_N-x_0)/N
%          y = vettore N+1 x m contenente le approssimazioni di y(x),
%              nei punti del vettore x
%
h = (x_N-x_0)/N;
x = (x_0:h:x_N)';
y = zeros(N+1,1);
y(1) = y_0;
for n = 1:N
   k1 = f(x(n),y(n));
   k2 = f(x(n)+h/2,y(n)+h/2*k1);
   k3 = f(x(n)+h/2,y(n)+h/2*k2);
   k4 = f(x(n)+h,y(n)+h*k3);
   y(n+1) = y(n)+h/6*(k1+2*k2+2*k3+k4);
end
```

RUNGE – KUTTA 4 – SISTEMI

```
function [x,y] = Runge_Kutta_4_system(f,x_0,y_0,x_N,N)
%
% Uso:     [x,y] = Runge_Kutta_4_system(f,x_0,y_0,x_N,N)
% Scopo:   risolve un sistema di m equazioni differenziali
%          y'(x)=f(x,y(x)) con condizioni iniziali y(x_0)=y_0
%          mediante un metodo Runge-Kutta del quarto ordine
% Input:   f = funzione nota f(x,y) del problema
%        x_0 = punto iniziale
%        y_0 = vettore (colonna) contenente gli m valori iniziali
%        x_N = punto finale
%          N = controlla l'ampiezza del passo h
% Output:  x = vettore (colonna) contenente N+1 nodi dell'intervallo
%              [x_0,x_N], equidistanti con passo h=(x_N-x_0)/N
%          y = matrice N+1 x m contenente le approssimazioni di y_1(x),
%              y_2(x),...,y_m(x) nei punti del vettore x
%
m = length(y_0);
h = (x_N-x_0)/N;
x = [x_0:h:x_N]';
m = length(y_0);
y = zeros(m,N+1);
y(:,1) = y_0;
for n = 1:N
    f1 = f(x(n),y(:,n));
    f2 = f(x(n)+h/2,y(:,n)+h/2*f1);
    f3 = f(x(n)+h/2,y(:,n)+h/2*f2);
    f4 = f(x(n+1),y(:,n)+h*f3);
    y(:,n+1) = y(:,n)+h/6*(f1+2*f2+2*f3+f4);
end
y = y';
```

```
% Laib09: equazioni differenziali
clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_9_1*****************')
disp('***********************************************')

% dati equazione e metodo
f = @(x,y) -y+x+1;
sol = @(x) x+exp(-x);
x0 = 0;
y0 = 1;
xN = 1;
z = linspace(x0,xN);
fprintf('%10s\t%8s\r\n','k','errore');
for k = 1:3
    N = 10^k;
    [x,y] = Eulero_esp(f,x0,y0,xN,N);
    figure(k)
    plot(x,y,'*','linewidth',2)
    hold on
    plot(z,sol(z),'k','linewidth',2)
    err(k) = abs(sol(x(end))-y(end));
    fprintf('%10.0d\t%6.4e\n',k,err(k));
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_9_2*****************')
disp('***********************************************')
% dati sistema e metodo
f = @(x,y) [y(2); 0.1*(1-y(1)^2)*y(2)-y(1)];
x_0 = 0;
y_0 = [1;0];
x_N = 1;
for k = 1:3
    N = 10^k;
    [x,y] = Eulero_esp_system(f,x_0,y_0,x_N,N);
    plot(x,y(:,1),'linewidth',2)
    hold on
    pause
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
```

```
close all
clc
disp('***********************************************')
disp('*****************esercizio_9_3*****************')
disp('***********************************************')
% dati sistema e metodo
% dati equazione e metodo
f = @(x,y) -y+x+1;
sol = @(x) x+exp(-x);
x0 = 0;
y0 = 1;
xN = 1;
fprintf('%10s\t%8s\t%8s\r\n','k','erroreRK2','erroreRK4');
for k = 3:8
    N = 2^k;
    [x,yRK2] = Heun(f,x0,y0,xN,N);
    [x,yRK4] = Runge_Kutta_4(f,x0,y0,xN,N);
    errRK2(k) = abs(sol(x(end))-yRK2(end));
    errRK4(k) = abs(sol(x(end))-yRK4(end));
    fprintf('%10.0d\t%6.4e\t%6.4e\n',k,errRK2(k),errRK4(k));
end

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
disp('*****************esercizio_9_4*****************')
disp('***********************************************')
% dati equazione e metodo
f = @(x,y) -y+x+1;
sol = @(x) x+exp(-x);
x0 = 0;
y0 = 1;
xN = 1;
z = linspace(x0,xN);
[x1,y1] = ode45(f,[x0 xN],y0);
err1 = abs(sol(1)-y1(end))
options = odeset('RelTol',1.0e-14,'AbsTol',1.0e-14);
[x2,y2] = ode45(f,[x0 xN],y0,options);
err2 = abs(sol(1)-y2(end))
semilogy(x1,abs(sol(x1)-y1),'b',x2,abs(sol(x2)-y2),'k','linewidth',2)
legend('without options','with options')

disp('**********************************************FINE
ESERCIZIO**********************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause

clear all
close all
clc
disp('***********************************************')
```

```
disp('****************esercizio_9_5****************')
disp('*******************************************')
% dati sistema e metodo
f = @(x,y) [y(2); 0.1*(1-y(1)^2)*y(2)-y(1)];
x_0 = 0;
y_0 = [1;0];
x_N = 1;
kmax = 7;
[x,ye] = Runge_Kutta_4_system(f,x_0,y_0,x_N,2^kmax);
fprintf('%10s\t%8s\t%8s\r\n','k','erroreRK2','erroreRK4');
for k = 1:kmax-1
    N = 2^k;
    [x,y2] = Heun_system(f,x_0,y_0,x_N,N);
    [x,y4] = Runge_Kutta_4_system(f,x_0,y_0,x_N,N);
    plot(x,y2(:,1),'b',x,y4(:,1),'k','linewidth',2)
    legend('RK2','RK4')
    errRK2(k) = abs(ye(end,1)-y2(end,1));
    errRK4(k) = abs(ye(end,1)-y4(end,1));
    fprintf('%10.0d\t%6.4e\t%6.4e\n',k,errRK2(k),errRK4(k));
    pause
end

disp('*******************************************FINE
ESERCIZIO*******************************************')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% esercizio_11_3
clear
close all
clc
% dati equazione e metodo
C1 = 1;
C2 = -1;
d1 = 1;
d2 = -1;
f = @(t,P) [C1*P(1)+d2*P(1)*P(2); C2*P(2)+d1*P(1)*P(2)];
t0 = 0;
P0 = [2;2];  % P0 = [1.2;1.2];
T = 10;
z = linspace(t0,T);
% punto di equilibrio stabile
PES = [-C2/d1,-C1/d2];
for N = 100:200:600
    [x,y] = Eulero_esp_system(f,t0,P0,T,N);
    figure(1)
    plot(x,y(:,1),'r',x,y(:,2),'b','linewidth',2)
    legend('P_1- preda','P_2- predatore')
    title('soluzioni approssimate')
    figure(2)
    for i = 1:length(x)

plot(PES(1),PES(2),'.r',y(i,1),y(i,2),'g.',y(1:i,1),y(1:i,2),'g','linewidth',2,'
markersize',40)
        m1 = min(y(:,1));
        M1 = max(y(:,1));
        m2 = min(y(:,2));
        M2 = max(y(:,2));
```

```
        axis([m1 M1 m2 M2])
        pause(0.01)
    end
    hold on
    % si rappresenta il campo vettoriale
    [X,Y] = meshgrid(m1:0.1:M1,m2:0.1:M2);
    u = C1*X+d2*X.*Y;
    v = C2*Y+d1*X.*Y;
    norma = 1;%sqrt(u.^2+v.^2);
    quiver(X,Y,u./norma,v./norma,'linewidth',2)
    title('piano delle fasi')
    pause
    hold off
end
options = odeset('AbsTol',1.0e-08,'RelTol',1.0e-08);
[x,y] = ode45(f,[t0 T],P0,options);
figure(3)
plot(x,y(:,1),'r',x,y(:,2),'b','linewidth',2)
legend('P_1- preda','P_2- predatore')
title('soluzioni approssimate')
figure(4)
for i = 1:length(x)

plot(PES(1),PES(2),'.r',y(i,1),y(i,2),'g.',y(1:i,1),y(1:i,2),'g','linewidth',2,'
markersize',40)
    m1 = min(y(:,1));
    M1 = max(y(:,1));
    m2 = min(y(:,2));
    M2 = max(y(:,2));
    axis([m1 M1 m2 M2])
    pause(0.01)
end
hold on
[X,Y] = meshgrid(m1:0.1:M1,m2:0.1:M2);
u = C1*X+d2*X.*Y;
v = C2*Y+d1*X.*Y;
norma = 1;%sqrt(u.^2+v.^2);
quiver(X,Y,u./norma,v./norma,'linewidth',2)
title('piano delle fasi')
hold off
```