IV° Esercitazione – Secondo semestre
Analisi Numerica – Calcolo Numerico
A.A. 2014– 2015

Esercizio 1.

- Assegnata la funzione $f(x) = e^x + \frac{1}{3}x^2 - 3$ si approssimi il suo zero positivo tramite il metodo di bisezione (eseguire almeno 10 iterazioni).
- Approssimare le soluzioni dell'equazione non lineare: $x^4 - 5x = 0, \ \ x \in \mathbf{R}$.
- Descrivere un metodo numerico per il calcolo di un autovalore di una matrice reale tridiagonale simmetrica.

Esercizio 2.

Approssimare l'integrale

$$\int_0^{\pi} sin(x)\, e^x \, dx$$

– utilizzando il metodo dei trapezi composito su due sottointervalli;
– utilizzando il metodo di Cavalieri-Simpson composito su due sottointervalli.
Nel primo caso si dia anche una stima dell'errore commesso.

Esercizio 3.

- Risolvere il sistema $A^tAx = b$ con

$$A = \begin{pmatrix} 1 & 2 & 3 \\ -1 & 1 & 0 \\ 2 & 1 & -1 \end{pmatrix} \qquad b = \begin{pmatrix} 10 \\ 14 \\ 16 \end{pmatrix}$$

e soluzione unitaria x, utilizzando una fattorizzazione della matrice A.
- Costruire la fattorizzazione QR della matrice A.

Esercizio 4.

Dato l'integrale

$$\int_0^{\pi} \frac{1}{\sqrt{x}} e^x \, dx$$

– Applicare una formula di Newton-Cotes aperta composita.
– Dopo una opportuna trasformazione per togliere la *singolarità* applicare una formula di Newton-Cotes chiusa composita e confrontare, in una tabella, i risultati con quelli ottenuti nel punto precedente.

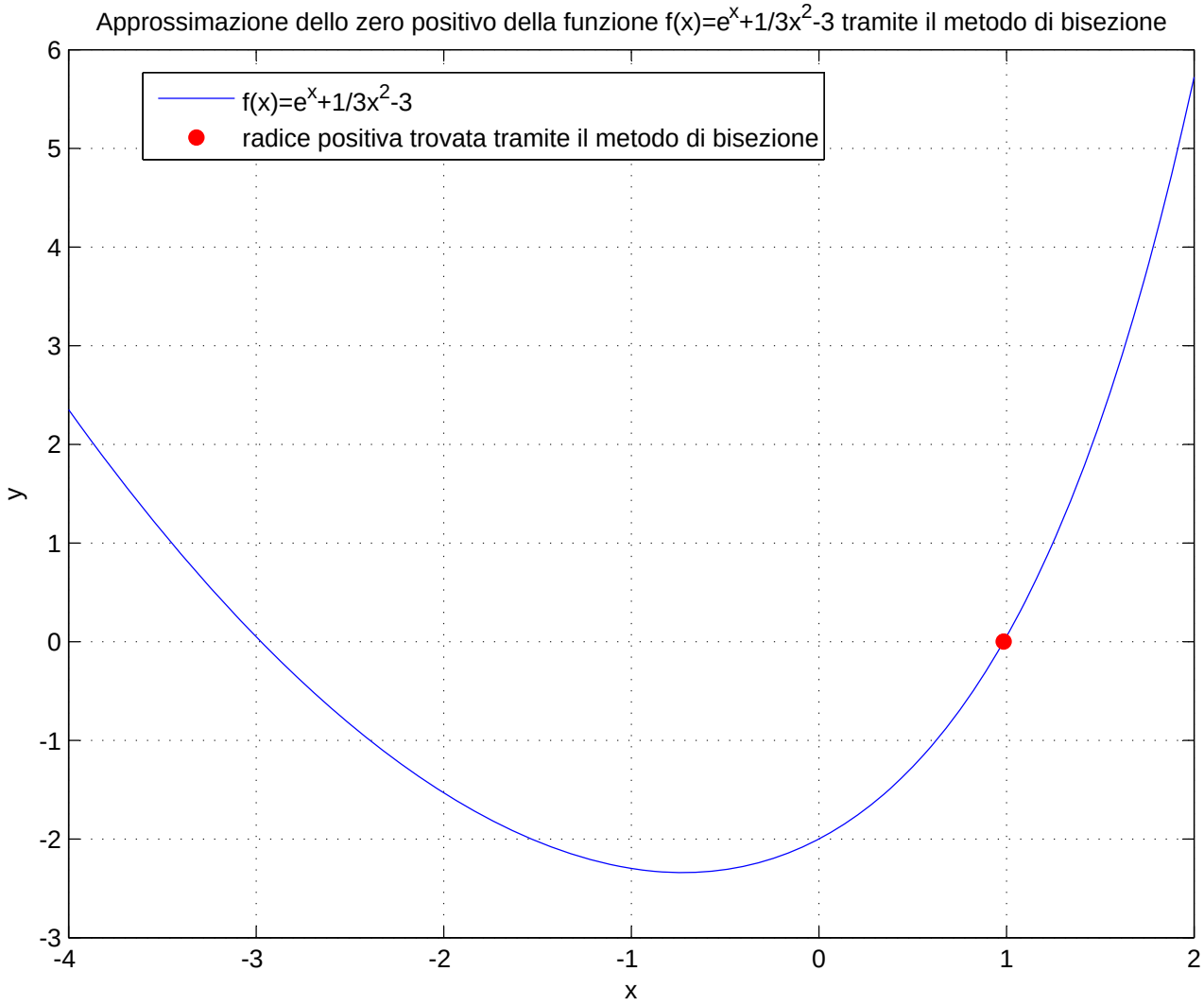
```
%*****************************************************************
% File: BISECT.M
%
% Scopo: Calcolo zero di una funzione con bisezione
%
% Uso:    [x,fx,n]=bisect(f,x1,x2,toll)
%
% Input: f     macro contenente la funzione in x
%    x1,x2 estremi sx e dx dell'intervallo
%    toll  tolleranza sull'intervallo
% Output: n    numero iterazioni
%     x    approssimazione dello zero
%     fx   valore della funzione in x.
%
%*****************************************************************
%
function [x,fx,n]=bisect(f,x1,x2,toll)
x=x1;
f1=eval(f);
if f1==0;
x=x1;
fx=f1;
n=0;
return;
end
x=x2;
f2=eval(f);
if f2==0;
x=x2;
fx=f2;
n=0;
return;
end
if sign(f1)*sign(f2)> 0
disp('** ERROR ** f(x1)*f(x2) > 0 '),
return,
end;
% n= numero iter. neces. per prec. toll
n=fix(log(abs(x2-x1)/toll)/log(2)+1); % +1 perche' int tronca
for i=1:n;
x=x1+(x2-x1)/2; % piu' accurato di (x1+x2)/2
fx=eval(f);
if sign(f1)*sign(fx)>0;
xn=x2;
x1=x;
else;
xn=x1;x2=x;
end; end;
return
```

```
% Esercizio 1a - Esercitazione 10
clear all
close all
clc
diary esercizio_1a.txt
% utilizziamo le macro, per poter passare la funzione alla funzione bisect
func ='exp(x)+(x.^2)/3-3;';
% punti di interpolazione
x=linspace(-4,2,101);
% eval : valuta la stringa come un comando matlab.
plot(x,eval(func))
% utilizziamo la funzione bisect per trovare la radice positiva della funzione
[xx yy n_iter]=bisect(func,0,2,0.0001);

% mettiamo nel plot la radice trovata
hold on
plot(xx,yy,'-xr')
disp('la radice positivo è in ');
xx
disp('il numero di iterazioni per trovare la radice positiva con tolleranza = 0,0001 è ');
n_iter
```

COMMAND WINDOW

```
la radice positivo è in

xx =

    0.9847

il numero di iterazioni per trovare la radice positiva con tolleranza = 0,0001 è

n_iter =

    15
```

GRAFICO Esercizio 1a - Esercitazione 10

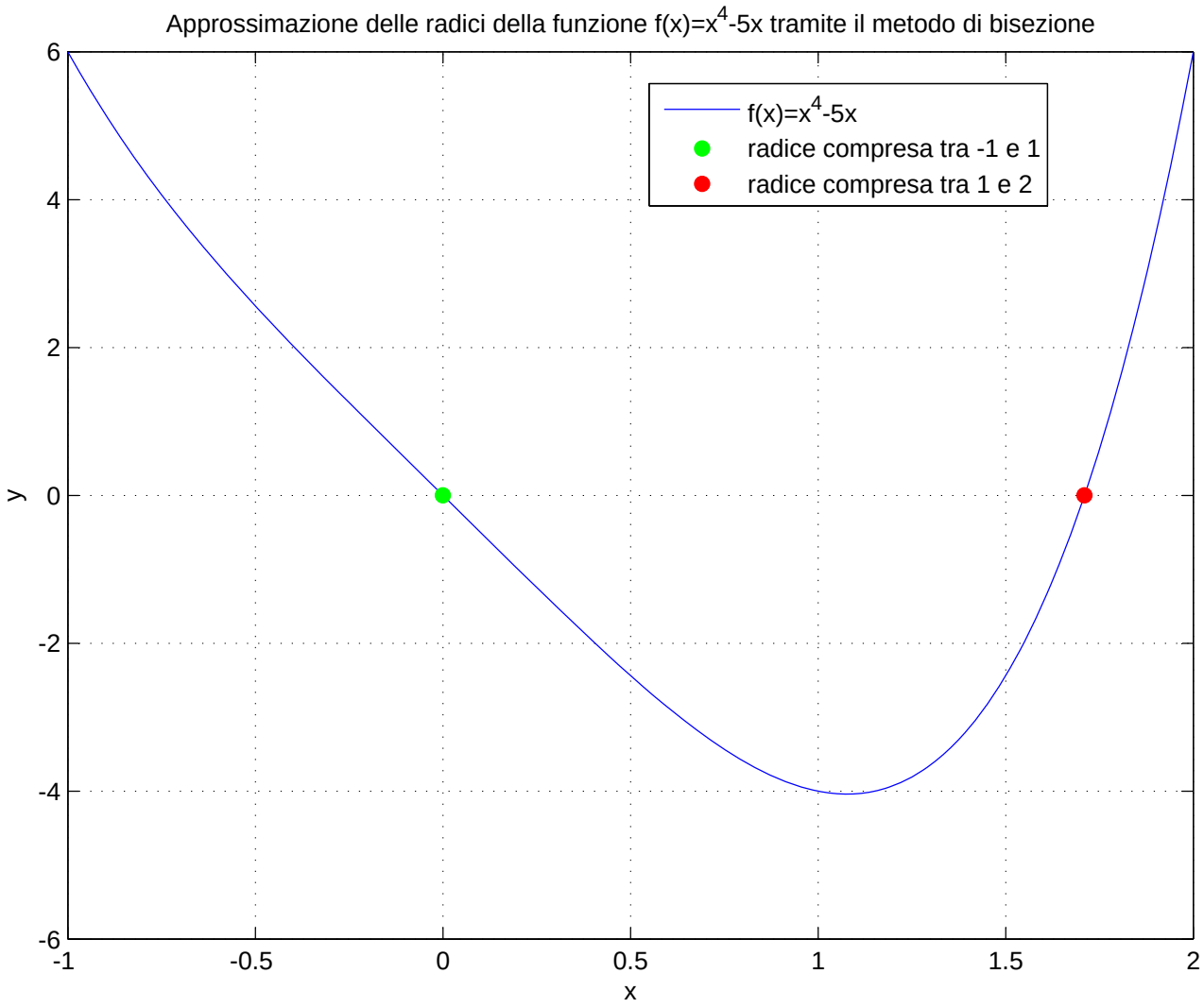


```
% Esercizio 1b - Esercitazione 10
clear all
close all
clc
diary esercizio_1b.txt
% utilizziamo le macro, per poter passare la funzione alla funzione bisect
func ='x.^4-5*x;';
% punti di interpolazione
x=linspace(-1,2,101);
% eval : valuta la stringa come un comando matlab.
plot(x,eval(func))

% utilizziamo la funzione bisect per trovare la radice della funzione
% nell'intervallo -1 e 1
[xx yy n_iter]=bisect(func,-1,1,0.0001);
% mettiamo nel plot la radice trovata
hold on
plot(xx,yy,'-xr')
disp('la radice nel intervallo -1 e 1 è ')
xx
disp('il numero di iterazioni per trovare la radice positiva, nel intervallo -1 e 1, con tolleranza = 0,0001 è ');
n_iter

% utilizziamo la funzione bisect per trovare la radice della funzione
% nell'intervallo 1 e 2
[xx yy n_iter]=bisect(func,1,2,0.0001);
% mettiamo nel plot la radice trovata
hold on
plot(xx,yy,'-xr')
disp('la radice nel intervallo -1 e 1 è ')
xx
disp('il numero di iterazioni per trovare la radice positiva, nel intervallo 1 e 2 con tolleranza = 0,0001 è ');
n_iter
% si trascurano le 2 radici complesse della funzione x.^4-5*x, come da
% richiesto dal testo la x appartiene all'insieme dei numeri reali.
```

COMMAND WINDOW

```
 la radice nel intervallo -1 e 1 è

 xx =
   -6.1035e-05

 il numero di iterazioni per trovare la radice positiva, nel intervallo -1 e 1, con tolleranza = 0,0001 è

 n_iter =
    15

la radice nel intervallo -1 e 1 è

xx =
    1.7100

il numero di iterazioni per trovare la radice positiva, nel intervallo 1 e 2 con tolleranza = 0,0001 è

n_iter =
    14
```

GRAFICO Esercizio 1b - Esercitazione 10



FUNZIONE METODO DI BISEZIONE(DICOTOMICA)

```
function [x,fx,n]=bisect(f,x1,x2,toll)
% Input:
%       f    macro contenente la funzione in x
%       x1,x2   estremi sx e dx dell'intervallo
%       toll    tolleranza sull'intervallo
% Output:
%       n   numero iterazioni
%       x   approssimazione dello zero
%       fx  valore della funzione in x
x=x1;
f1=eval(f);
if f1==0;
   x=x1;
   fx=f1;
   n=0;
   return;
end
x=x2;
f2=eval(f);
if f2==0;
   x=x2;
   fx=f2;
   n=0;
   return;
end
if sign(f1)*sign(f2)> 0
   disp('** ERRDR ** f(x1)*f(x2) > 0 '),
   return,
end
% n= numero iter. neces. per prec. toll
n=fix(log(abs(x2-x1)/toll)/log(2)+1); % +1 perche' int tronca
for i=1:n
   x=x1+(x2-x1)/2; % piu' accurato di (x1+x2)/2
   fx=eval (f);
   if sign(f1)*sign(fx)>0;
      xn=x2;
      x1=x;
   else
      xn=x1;
      x2=x;
   end
end
return
```

```
% Esercizio 1b - Esercitazione 10
clear all
close all
clc
diary esercizio_1c.txt

% matrice casuale tridiagonale simmetrica
diag_inf_sup =rand(3);
rand_mat= zeros(4)+diag(diag(rand(4)))+diag(diag(diag_inf_sup),1)+diag(diag(diag_inf_sup),-1)
% calcoliamo l'autovalore massimo della matrice tramite funzione
% funzione che calcola l'autovalore massimo della matrice
[autoval, num_iter] = newton_mat_trid_sim( rand_mat , 0.00001 )
% confrontiamo l'autovalore trovato con gli autovalori calcolati da matlab
% tramite la funzione eig
eig(rand_mat)
```

COMMAND WINDOW

```
rand_mat =

    0.0971    0.7577         0         0
    0.7577    0.0344    0.1712         0
         0    0.1712    0.1869    0.0462
         0         0    0.0462    0.7547

autoval =

    0.8466


num_iter =

     6


ans =

   -0.7098
    0.1789
    0.7575
    0.8466
```

FUNZIONE RADICE MATRICE TRIDIAGONALE SIMMETRICA

```
function [ x_new, num_iter ] = newton_mat_trid_sim( M , toll )
%RAD_MAT_TRID_SIM calcola l'autovalore massimo(destro) della matrice
%------------------INPUT
% M : matrice tridiagonale simmetrica
% toll : tolleranza
%-----------------OUTPUT
% x_new : autovalore massimo della matrice
% num_iter: numero iterazione
%----------------------
   % troviamo il punto iniziale dove applicare newton con il metodo di
   % Gerschgorin. dal secondo teorema di Gerschgorin, in ogni cerchio si  trova esattamente un autovalore.
   n= size(M,1);
   % R: vettore di lunghezza n, che contiene i raggi dei cerchi di Gerschgorin
   % R(i) = somma dei valori assoluti degli elementi non diagonali della riga i-esima
   R = sum(abs(M-diag(diag(M))));
   % punto estremo del cerchio di Gerschgorin, punto del centro + punto del raggio
   x0 = max(diag(M)'+R);
   x_new = x0;
   % valore che varia per ogni iterazione
   x_old = x_new + toll +1;
   num_iter=0;
   while (abs(x_new - x_old) > toll)
      % VALUTAZIONE DEL POLINOMIO CARATTERISTICO
      % y_old : valore del polinomio carattatteristico in x_old, funzione
      % dove valutiamo la x_old
      y_old = 0;
      D=zeros(n+1,1);
      % determinante della matrice di dimensione 0
      D(1)=1;
      % determinante della matrice di dimensione 1
      D(2)=M(1,1)-x_old;
      % determinante della matrice di dimensione da 2 a n
      for i = 2:n
         %D_n = det(J-lambda*I)
         D(i+1)=det(M(1:i,1:i)-x_new*eye(i));
      end
      y_old = D(n+1);
      % VALUTAZIONE DERIVATA DEL POLINOMIO CARATTERISTICO
      % Dy_old : valore della derivata prima del polinomio caratteristico.
      Dy_old = 0;
      D_deriv = zeros(n+1,1);
      % derivata determinante della matrice di dimensione 0
      D_deriv(1) = 0;
      % derivata determinante della matrice di dimensione 1
      D_deriv(2) = -1;
      % derivata determinante della matrice di dimensione da 2 a n
      for i = 2:n
         % D'_n=D_(n-1)(lambda) + (alfa-lambda)D'_(n-1)-(beta_n)^n*D'_(n-2)(lambda)
         % alpha(M(i,i)): elem diag principale, betaM(i-1,i): elem diag sup e ing
         D_deriv(i+1)=-D(i) + (M(i,i)-x_new)*D_deriv(i) - M(i,i-1)^2 *D_deriv(i-1);
      end
      Dy_old = D_deriv(n+1);
      % Calcolo del nuovo punto
      x_old=x_new;
      x_new=x_new - y_old/Dy_old;
      num_iter=num_iter+1;
   end
end
```

```
% Esercizio 2 - Esercitazione 10
clear all
close all
clc
diary esercizio_2.txt
funz = @(x) sin(x).*exp(x);
funzD2 = @(x) (2.*exp(x).*cos(x));

% TRAPEZI COMPOSITO su 2 sottointervalli
x_t=linspace(0,pi,3); % 3 punti della decomposizione con 2 intervalli.
% primo intervallo
passo_sot_trap = x_t(2)-x_t(1);  % passo sottointervallo trapezi
val_trap_comp=(passo_sot_trap/2)*(sin(x_t(1))*exp(x_t(1))+sin(x_t(2))*exp(x_t(2)));
%secondo intervallo
passo_sot_trap = x_t(3)-x_t(2);
val_trap_comp=val_trap_comp + (passo_sot_trap/2)*(sin(x_t(2))*exp(x_t(2))+sin(x_t(3))*exp(x_t(3)))

% CAVALIERI-SIMPSON COMPOSITO su 2 sottointervalli
x_cv=linspace(0,pi,5); % 6 punti della decomposizione con 2 intervalli.
% primo intervallo
h_sot_cavsim = (x_cv(3)-x_cv(1))/2;  % h=(b-a/n) sottointervallo cavalieri-simpson
val_cavsim_comp=(h_sot_cavsim/3)*(sin(x_cv(1))*exp(x_cv(1)) + 4*sin(x_cv(2))*exp(x_cv(2)) +
sin(x_cv(3))*exp(x_cv(3)));
% secondo intervallo
h_sot_cavsim = (x_cv(5)-x_cv(3))/2;  % h=(b-a/n) sottointervallo cavalieri-simpson
val_cavsim_comp=val_cavsim_comp+(h_sot_cavsim/3)*(sin(x_cv(3))*exp(x_cv(3)) +
4*sin(x_cv(4))*exp(x_cv(4)) + sin(x_cv(5))*exp(x_cv(5)))

%valore esatto dell'integrale da 0 a pi di sin(x)*exp(x)
val_esa=integral(funz, 0, pi)

% ERRORE DI INTEGRAZIONE NUMERICA TRAPEZI COMPOSITA
% primo intervallo
xx=linspace(0,pi/2,500);
% norma inf. per ottenere il massimo valore della derivata seconda nei punti
R_trap_comp=-1/12 * norm(funzD2(xx),inf) * (pi/2-0)^3;
% secondo intervallo
xx=linspace(pi/2,pi,500);
% norma inf. per ottenere il massimo valore della derivata seconda nei punti
R_trap_comp=abs(R_trap_comp + (-1/12 * norm(funzD2(xx),inf) * (pi-pi/2)^3))
```

COMMAND WINDOW

```
 val_trap_comp =

    7.5563

 val_cavsim_comp =

   11.9554

 val_esa =

   12.0703

R_trap_comp =

   15.9499
```

```
% Esercizio 3 - Esercitazione 10
clear all
close all
clc
diary esercizio_3.txt

A= [ 1 2 3; -1 1 0; 2 1 -1];
x=ones(3,1);
b=[10 14 16]';
% Per la risoluzione dei sistemi lineari nel caso in cui la matrice
% associata sia triangolare inferiore o superiore utilizzaremo
% rispettivamente gli algoritmi di sostituzione in avanti e all'indietro.


% Fattorizzazione LU della matrice A
[L U] = Fatt_LU_NOPIV(A)
% A'Ax=b
% (LU)'LUx=b
% U'L'LUx=b

% Risolviamo il sistema lineare U'y1=b,   y1=(L'LUx)
y1= RSL_SA(U',b);
% Risolviamo il sistema lineare L'y2=y1,   y2=(LUx)
y2= RSL_SI(L',y1);
% Risolviamo il sistema lineare Ly3=y2,   y3=(Ux)
y3= RSL_SA(L,y2);
% Risolviamo il sistema lineare Uy4=b
xlu= RSL_SI(U,y3)

% Fattorizzazione QR della matrice A
[Q R] = Fatt_QR(A)
% A'Ax=b   --- Sostituiamo la matrice A con QR
% (QR)'QRx=b
% R'Q'QRx=b   --- dato che Q è una matrice ortogonale, Q'Q=Id
% R'Rx=b

% Risolviamo il sistema lineare R'y=b,    y=(Rx)
y= RSL_SA(R',b);
% Risolviamo il sistema lineare Rx=y
xqr= RSL_SI(R,y)
```

COMMAND WINDOW

L =

1	0	0
-1	1	0
2	-1	1

U =

1	2	3
0	3	3
0	0	-4

xlu =

1
1
1

Q =

-0.4082	-0.7071	-0.5774
0.4082	-0.7071	0.5774
-0.8165	-0.0000	0.5774

R =

-2.4495	-1.2247	-0.4082
0.0000	-2.1213	-2.1213
0.0000	-0.0000	-2.3094

xqr =

1.0000
1.0000
1.0000

FATTORIZZAZIONE LU

```
function [ L, U ] = Fatt_LU( A )
% Fattorizzazione LU
% ------------INPUT----------------
% A: matrice da fattorizzare
% --------------------------------
   % dimensioni matrice n righe, m colonne
   [n,m] = size(A);
   % iterazioni sulle n righe
   L=zeros(n,m);
   U=A;
   % inizializziamo il pivot con valori di default, e quando U(i,i)=0 non
   % vengono scambiate le righe
   pivot=1:length(A);
   for i = 1:n
      % la condizione serve per evitare pivoting non necessari
      if U(i,i) ==0
         % PIVOTING
         % inizializziamo max con la prima riga i
         max = i;
         % impostiamo max uguale alla riga con il massimo valore , cioe
         % scegliamo la riga del pivot
         for j = i+1 : n
            if (abs(U(j,i)) > abs(U(max,i)))
               max = j;
            end
         end
         % scambio della riga i con la riga max(del pivot)
         L([i max],:) = L([max i],:);
         % memorizziamo il pivot per la permutazione
         pivot(i) = max;
         U([i max],:) = U([max i],:);
      end
      % SCALING
      for j=i+1 :n
         if U(i,i) ~=0
         % Costruice matrice valori coefficienti di gauss, -U(j,i) e tmp
         % per ottenere il segno corretto rispettivamente sulla matrice L e U
         tmp=-U(j,i)/U(i,i);
         % costruisce matrice riduzione di gauss
         U(j,:)=U(i,:)*tmp + U(j,:);
         L(j,i)=-tmp;
         end
      end
   end
   % Permutazione per ordinare la matrice L, cioè la matrice P è la
   % permutazione che applico su L
    P = diag(ones(size(U,1),1));
    for i = 1:n-1
       P([i pivot(i)],:) = P([pivot(i) i],:);
    end
   L = L + diag(ones(size(U,1),1));
    L = P\L;
end
```

FATTORIZZAZIONE QR

```
function [Q,R] = Fatt_QR (A)
% FATTORIZZAZIONE QR
%------------------INPUT
% A : matrice da fattorizzare
%-----------------OUTPUT
% Q : Matrice prodotto di tutti i riflettori elementari
% R : Matrice triangolare superiore
%----------------------
% Applichiamo l'algoritmo di fattorizzazione QR alla matrice A. iteriamo
% per ottenere i riflettori elementari di ogni sottomatrice di A per poi
% moltiplicarli alla A precendente, infatti a ogni iterazione A varierà a
% seconda del riflettore elementare a cui è stata moltiplicata, ed infine
% nell'ultimo passo del ciclo la matrice A sarà la matrice triangolare
% superiore R.
   [m n]=size(A);
   % matrice rispetto alla base canonica, utilizzata per ricavare il
   % vettore v
   Me=zeros(m,n)+diag(diag(ones(m,n)));
   for i=1:m-1
       % vettore contenente prima riga di A
       x=A(i:m,i);
       % A non è univocamete determinata, quindi è possibile scegliere
       % la norma di x di segno positivo o negativo, la scelta piu
       % opportuna per calcolare v è che x+norm_x sia di segno concorde
       % in modo che la somma non sia 0.
       if x(1) >0
         nor_x=norm(x,2);
       else x(1) <0
         nor_x=-norm(x,2);
       end
       v=x + nor_x.*Me(i:m,i);
       % riflettore elementare: matrice di dimensione della i-esima
       % sottomatrice ( è matrice di hauseholder: simm,ortog,idempoten),
       % cioè costruiamo una matrice che trasformi la matrice A in una
       % triangolare superiore.
       P=eye(m+1-i,n+1-i) - ((2*v*v')/(norm(v,2))^2);
       % Costruiamo la matrice P di dimensione sempre m,n
       tmpMe=Me;
       tmpMe(i:m,i:n)=zeros(m+1-i,n+1-i) + P;
       P=tmpMe;
       % Modifichiamo la matrice A per l'iterata successiva
       A = P*A;
       % Costruisce la matrice Q facendo il prodotto tra matrici del
       % riflettore elementare precedente(Q)(mantiene ortogononalità
       % e il riflettore elementare secondo la i sottomatrice. utilizziamo
       % il costrutto if per %inizializzare nella prima iterata Q con il
       % valore del primo riflettore
       if i == 1
          Q=P;
       else
          Q=Q*P;
       end
   end
   R = A;
end
```

FUNZIONI RISOLUZIONE SISTEMA LINEARE

```
function x = RSL_SA(L,b)
%   RISOLUZIONE SISTEMA LINEARE DI UNA MATRICE TRIANGOLARE INFERIORE
%   SOSTITUZIONE IN AVANTI

   n = length(b);
   % ricaviamo l'incognita della prima riga
   x(1) = b(1)/L(1,1);
   % ricaviamo le i incognite con i=2,...,n
   for i=2:n
   % ricaviamo l'elemento i dell'incognita
   x(i) = (b(i)-L(i,1:i-1)*x(1:i-1)') / L(i,i);
   end
   x=x';
end
```

```
function x = RSL_SI(U,b)
%   RISOLUZIONE SISTEMA LINEARE DI UNA MATRICE TRIANGOLARE SUPERIORE
%   SOSTITUZIONE ALL'INDIETRO

      n = length(b);
      % ricaviamo l'incognita dell'ultima riga
      x(n) = b(n)/U(n,n);
      % ricaviamo le i incognite con i=n-1,...,1
      for i=n-1: -1 :1
      % ricaviamo l'elemento i dell'incognita
      x(i) = (b(i)-U(i,i+1:n)*x(i+1:n)') / U(i,i);
      end
      x=x';
   end
```

```
% Esercizio 4 - Esercitazione 10
clear all
close all
clc
diary esercizio_4.txt

%function handle
fun= @(x) (1./sqrt(x).*exp(x));
%valore esatto
val_esa=ones(20,1)*integral(fun,0,pi);

%Newton-Cotes aperta composita, grado n=0, Formula del punto medio
NCac_0 = zeros(20,1);
for i=1:20
   NCac_0(i) = NewCot_aper_comp(fun,0,pi,0,i);
end

% Per rimuovere la singolarità viene effettuata una sostituzione per poi
% poter utilizzare una formula di Newton-Cotes chiusa composita
% sostituiamo t=sqrt(x), x=t^2, dx=2t*dt ottenendo integral(@exp(t^2),0,sqrt(pi))
fun_tras= @(t) (2.*exp(t.^2));
%valore esatto
val_esa_tran=ones(20,1)*integral(fun_tras,0,sqrt(pi));
NCcc_1 = zeros(20,1);
for i=1:20
   NCcc_1(i) = NewCot_chiu_comp(fun_tras,0,sqrt(pi),1,i);
end

% Tabella contfronto risultati
% TABELLE ERRORI (con comando matlab fprintf
tabella = table(val_esa, NCac_0, val_esa_tran, NCcc_1)
```

TABELLA CONFRONTO FORMULE NEWTON-COTES COMPOSITE DA 1 A 20 INTERVALLI

tabella =

val_esa	NCac_0	val_esa_tran	NCcc_1
16.369	12.058	16.369	42.788
16.369	14.684	16.369	25.282
16.369	15.334	16.369	20.712
16.369	15.601	16.369	18.907
16.369	15.744	16.369	18.025
16.369	15.832	16.369	17.532
16.369	15.892	16.369	17.229
16.369	15.936	16.369	17.03
16.369	15.969	16.369	16.893
16.369	15.996	16.369	16.795
16.369	16.018	16.369	16.721
16.369	16.037	16.369	16.666
16.369	16.053	16.369	16.622
16.369	16.066	16.369	16.587
16.369	16.078	16.369	16.559
16.369	16.089	16.369	16.536
16.369	16.099	16.369	16.517
16.369	16.107	16.369	16.501
16.369	16.115	16.369	16.488
16.369	16.122	16.369	16.476

FORMULA NEWTON COTES APERTE SEMPLICE

```
function Q=NewCot_aper_semp(fun,a,b,n)
% NEWTON COTES APERTE SEMPLICE (su un intervallo)
%------------------INPUT
% fun : funzione integranda
% a,b : estremi di integrazione
% n+1 : numero di nodi
%-----------------OUTPUT
% Q : quadratura risultante
%----------------------

   h=(b-a)/(n+2);
   % intervallo su cui costruire la quadratura
   x=linspace(a+h,b-h,n+1)';

   switch n % grado
      case 0  % formula dei rettangoli o del punto medio
         alpha=2;
      case 1
         alpha=[3/2 3/2];
   end
   Q=h*alpha*fun(x);
end
```

FORMULA NEWTON COTES APERTE COMPOSITE

```
function Q=NewCot_aper_comp(fun,a,b,n,N)
%------------------INPUT
% NEWTON COTES APERTE COMPOSITE
% fun : funzione integranda
% a,b : estremi di integrazione
% n+1 : numero di nodi
% N numero di suddivisioni di [a,b]
%-----------------OUTPUT
% Q : quadratura risultante
%----------------------

    H=(b-a)/N;
    % intervallo su cui costruire la quadratura
    X=linspace(a,b,N+1)';
   Q=0;
   for i=1:N
      Q=Q + NewCot_aper_semp(fun,X(i),X(i+1),n);
   end
end
```

FORMULA NEWTON COTES CHIUSE SEMPLICE

```
function Q=NewCot_chiu_semp(fun,a,b,n)
% NEWTON COTES CHIUSA SEMPLICE (su un intervallo)
%------------------INPUT
% fun : funzione integranda
% a,b : estremi di integrazione
% n+1 : numero di nodi
%-----------------OUTPUT
% Q : quadratura risultante
%----------------------

   h=(b-a)/n;
   % intervallo su cui costruire la quadratura
   x=linspace(a,b,n+1)';

   switch n % grado
      case 1  % formula dei trapezi
         alpha=[1/2 1/2];
      case 2  % formula di Cavalieri-Simpson
         alpha=[1/3 4/3 1/3];
      case 3  % formula dei tre ottavi
         alpha=[3/8 9/8 9/8 3/8];
      case 4
         alpha=[14/45 64/45 24/45 64/45 14/45];
      case 5
         alpha=[95/288 375/288 250/288 250/288 375/288 95/288];

   end
   Q=h*alpha*fun(x);
end
```

FORMULA NEWTON COTES CHIUSE COMPOSITE

```
function Q=NewCot_chiu_comp(fun,a,b,n,N)
% NEWTON COTES CHIUSE COMPOSITE
%------------------INPUT
% fun : funzione integranda
% a,b : estremi di integrazione
% n+1 : numero di nodi
% N numero di suddivisioni di [a,b]
%-----------------OUTPUT
% Q : quadratura risultante
%----------------------

   H=(b-a)/N;
   % intervallo su cui costruire la quadratura
   X=linspace(a,b,N+1)';
  Q=0;
  for i=1:N
     Q=Q + NewCot_chiu_semp(fun,X(i),X(i+1),n);
  end
end
```