Elaborato di

Calcolo Numerico

Anno Accademico 2019–2020

Andre Cristhian Barreto Donayre - 5364116
andre.barreto@stud.unifi.it

June 30, 2020

Contents

1 Traccia degli esercizi

1.1 Formule di Quadratura: a.a 2019-2020

1.1.1 Esercizio 21

Costruire una function Matlab che, dato in input n, restituisca i pesi della Quadrattura della formula di Newton-Cotes di grado n.
Tabulare, quindi, i pesi delle formule di grado 1 , 2 , ... , 7 (come numeri razionali).

1.1.2 Esercizio 22

Utilizzare la function del precedente esercizio per graficare, in forma **semilogy**, il rapporto $\frac{k_n}{k}$ rispetto ad n, essendo k il numero di condizionamento di un integrale definito, e $k_n$quello della formula di Newton-Cotes utilizzata di grado n per approssimarlo.
Riportare i risultati per n = 1,...,50.

1.1.3 Esercizio 23

Tabulare le approssimazioni dell'integrale

$$I(f) = \int_{-1}^{1.1} \tan(x)\mathrm{d}x \equiv \log \frac{\cos(1)}{\cos(1.1)},$$

ottenute mediale le formule di Newton-Cotes di grado n, n = 1,...,9. Tabulare anche il relativo errore (in notazione scientifica con 3 cifre significative).

1.1.4 Esercizio 24

Confrontare le formule Composite dei Trapezi e di Simpson per approssimare l'integrale del precedente esercizio, per valori crescenti del numero dei sottointervalli dell'intervallo di integrazione. Commentare i risultati ottenuti, in termini di costo computazionale.

1.1.5 Esercizio 25

Confrontare le formule adattive dei Trapezi e di Simpson, con tolleranze $tol = 10^-2, 10^-3, ..., 10^-6$, per approssimare l'integrale definito:

$$I(f) = \int_{-1}^{1} \frac{1}{1 + 10^2x^2}\mathrm{d}x.$$

Commentare i risultati ottenuti, in termini di costo computazionale.

1.2 Calcolo del Google Pagerank: a.a 2018-2019

1.2.1 Esercizio 26

Scrivere una function che implementi efficientemente il metodo delle potenze.

1.2.2 Esercizio 27

Sia data la matrice di *Toeplitz* simmetrica in cui le extra-diagonali piú esterne sono le none.

$$A_N = \begin{pmatrix} 4 & -1 & & -1 & & \\ -1 & \ddots & \ddots & & \ddots & \\ & \ddots & \ddots & \ddots & & -1 \\ -1 & & \ddots & \ddots & \ddots & \\ & \ddots & & \ddots & \ddots & -1 \\ & & -1 & & -1 & 4 \end{pmatrix} \in \mathbb{R}^{N \times N}, \qquad N \geq 10, \qquad (1)$$

Partendo dal vettore $\mathbf{u}_0 = (1, \cdots, 1)^T \in R^N$, applicare il metodo delle potenze con tolleranza $tol = 10^{-10}$ per $N = 10 : 10 : 500$, utilizzando la function dell'esercizio 26. Graficare il valore dell'autovalore dominante, e del numero di iterazioni necessarie per soddisfare il criterio di arresto, rispetto ad N. Utilizzare la funzione spdiags di Matlab per creare la matrice e memorizzarla come matrice sparsa.

1.2.3 Esercizio 28

Scrivere una function che implementi efficientemente un metodo iterativo, per risolvere un sistema lineare, definito da un generico splitting della matrice dei coefficienti.

1.2.4 Esercizio 29

Scrivere le function ausiliarie, per la function del precedente esercizio, che implementano i metodi iterativi di Jacobi e Gauss-Seidel.

1.2.5 Esercizio 30

Con riferimento alla matrice A_N definita in (1), risolvere il sistema lineare

$$A_N \boldsymbol{x} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^N,$$

con i metodi di Jacobi e Gauss-Seidel, per $N = 10 : 10 : 500$, partendo dalla approssimazione nulla della soluzione, ed imponendo la norma del residuo sia minore di 10^{-8}. Utilizzare, a tale fine, la function dell'esercizio 28, scrivendo function ausiliare *adhoc* (vedi esercizio 29) che sfruttino convenientemente la struttura di sparsità (nota) della matrice A_N. Graficare il numero delle iterazioni richieste dai due metodi iterativi, rispetto ad N, per soddisfare il criterio di arresto prefissato.

2 Formule di Quadratura: Svolgimento

2.1 Esercizio 21 - Descrizione

L'esercizio richiede di costruire una funzione Matlab che calcoli i coefficienti $C_{i,n}$ della formula di Quadratura di Newton-Cotes di cui si considera la sua espressione generale : $I_n(f) = h\sum_{i=0}^{n} \mathbf{C}_{i,n} f_i$ dove i coefficienti $\mathbf{C}_{i,n}$ seguono la formula:

$$\mathbf{C}_{i,n} = \int_0^n \prod_{j=0}^{n} \frac{(\mathbf{t}-j)}{(i-j)} d\mathbf{t}$$

con $j \neq i$, $i = 1, ..., n$.
Inoltre dobbiamo Tabulare i pesi per $n = 1, 2, .., 7$, essendo i più significativi, perché per $n \geq 7$ i coefficienti iniziano ad assumere valori negativi e andrebbero ad influire sul condizionamento del problema.

2.1.1 Svolgimento:

```
function [cin] = pesiNC(n)
% Funzione che ritorna i pesi della formula di Newton-Cotes
% Input:
%   n = numero di sottointervalli sui quali applicare la formula.
prod= @(t) t;
cin= zeros(1,n+1);
for i=0:n
    func{i+1}=@(t) 1;
    for j=0:n
        if j~=i
            func{i+1}= @(t) func{i+1}(t).*(prod(t)-j)/(i-j);
        end
    end
    cin(i+1)= integral(func{i+1},0,n);
end
return

end
```

Tabulazione di risultati significativi:

n	$C_{i,n}$
1	$\frac{1}{2}$ $\frac{1}{2}$
2	$\frac{1}{3}$ $\frac{4}{3}$ $\frac{1}{3}$
3	$\frac{3}{8}$ $\frac{9}{8}$ $\frac{9}{8}$ $\frac{3}{8}$
4	$\frac{14}{45}$ $\frac{64}{45}$ $\frac{8}{15}$ $\frac{64}{45}$ $\frac{14}{45}$
5	$\frac{95}{288}$ $\frac{125}{96}$ $\frac{125}{144}$ $\frac{125}{144}$ $\frac{125}{96}$ $\frac{95}{288}$
6	$\frac{41}{140}$ $\frac{54}{35}$ $\frac{27}{140}$ $\frac{68}{35}$ $\frac{27}{140}$ $\frac{54}{35}$ $\frac{41}{140}$
7	$\frac{1073}{3527}$ $\frac{810}{559}$ $\frac{343}{640}$ $\frac{649}{536}$ $\frac{649}{536}$ $\frac{343}{640}$ $\frac{810}{559}$ $\frac{1073}{3527}$

Table 1: Valori significativi $n = 1, ..., 7$ con $i = 0, ..., n$

2.2 Esercizio 22 - Descrizione

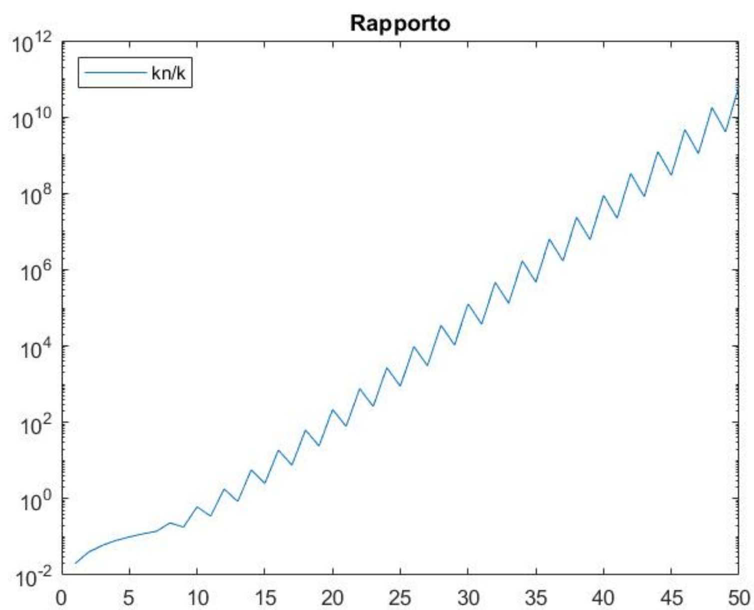
L'esercizio richiede di rappresentare in formato **semilogy** il rapporto tra *Numero di Condizionamento* k_n definito come $\frac{(b-a)}{n}\sum_{i=0}^{n}|\mathbf{C}_{i,n}|$ e k definita come $(b-a)$ *distanza* dell'intervallo di valutazione.
Date le definizioni è possibile ricondurre il rapporto alla seguente formula:

$$\frac{k_n}{k} = \frac{1}{(b-a)}\frac{(b-a)}{n}\sum_{i=0}^{n}|\mathbf{C}_{i,n}| = \frac{1}{n}\sum_{i=0}^{n}|\mathbf{C}_{i,n}|$$

2.2.1 Svolgimento:

```
n= 50;
rapp= zeros(1,n);
for i=1:n
    pe= pesiNC(i);
    absP= 0;
    for j=1:i+1
        absP=absP + abs(pe(j));
    end
    rapp(i)=absP*1/n;
end

n=1:50;
figure
semilogy(n,rapp);
title( 'Rapporto');
legend('kn/k','location','northwest');
```



2.3 Esercizio 23 - Descrizione

L'esercizio richiede di tabulare le approssimazioni dell'integrale dato, valori di n pari a $n = 1, ..., 9$ considerando anche l'errore di approssimazione nel processo.Si è voluto favorire la generalità e riutilizzo del

codice, perciò l'esercizio è stato strutturato come una funzione. La funzione da studiare è stata definita `f=@(x) tan(x)` e le chiamate sono state eseguite `intAp(f,-1,1.1,n)` per $n = 1, ..., 9$.

2.3.1 Svolgimento:

```
function [aprox] = intAp(func,a,b,n)
% Funzione che ritorna l'approssimazione della funzione in ingresso
% nell'intervallo a,b considerato con un numero di suddivisioni pari ad n
% Input:
%       func= funzione da aprossimare
%       a,b= estremi di integrazione
%       n = numero di sottointervalli sui quali applicare la formula dei pesi.
format short;
if a==b
   error("impossibile esseguire approssimazione");
else
   s= 0;
   for i=1:n
       x= linspace(a,b,i+1);
       f= feval(func,x);
       pe= pesiNC(i);
       h= (b-a)/i;
       for j=1:i+1
           s= pe(j)*f(j);
       end
       app= s*h;
   end
   int= integral(func,a,b);
   errAp= int-app;
   aprox= [app errAp];
   return


end
```

Tabulazione delle Approssimazioni :

n	Approssimazione	Errore
1	2.0630	-1.8881
2	$6.877 * 10^{-1}$	$-5.127 * 10^{-1}$
3	$5.157 * 10^{-1}$	$-3.408 * 10^{-1}$
4	$3.209 * 10^{-1}$	$-1.460 * 10^{-1}$
5	$2.722 * 10^{-1}$	$-9.73 * 10^{-2}$
6	$2.014 * 10^{-1}$	$-2.65 * 10^{-2}$
7	$1.793 * 10^{-1}$	$-4.4 * 10^{-3}$
8	$1.439 * 10^{-1}$	$3.10 * 10^{-2}$
9	$1.316 * 10^{-1}$	$4.34 * 10^{-2}$

2.4 Esercizio 24 - Descrizione

Le formule Composite dei Trapezi e di Simpson hanno come obiettivo quelle di approssimare un integrale definito, dato un intervallo suddiviso in n punti. Entrambe le funzioni avranno un costo pari ad n+1 perché vengono eseguita in ogni sotto intervallo. Questo fatto evidenzia che il costo dipende dal numero di suddivisioni che si sta considerando. Sappiamo che per definizione la formula Composita di Simpson è applicabile solo su una suddivisione pari d'intervalli. La formula dei Trapezi sarà sempre applicabile. A parità di costo bisogna considerare che sebbene la formula di Simpson sia un po' più lenta a eseguire, offre una maggiore precisione rispetto a quella dei Trapezi.

2.4.1 Svolgimento:

Trapezi Composita: Consideriamo la formula dei Trapezi espressa in questo modo, più facile da codificare.

$$\mathbf{I} = h * (\frac{1}{2}[f(a) + f(b)] + \sum_{i=2}^{n} f(x_i))$$

```
function [Int]= trapComp(func, a, b, n)
% Function che approssima lintegrale definito da f(x) con
% estremi a,b mediante la formula di Trapezi composita
% Input:
%       func= funzione da approssimare
%       a,b= estremi di integrazione
%       n= numero suddivisione
format long;
if a==b
    Int= 0;
else
    if n<1
        error('numero di ascisse non consistente');
    else
        h= (b-a)/n;
        x= linspace(a ,b ,n+1);
        f= feval(func, x);

        S= 0.5*(f(1)+f(n+1));
        for i=2:n
            S=S+f(i);
        end
        Int= h*S;
    end
   return

end
```

Simpson Composita: Consideriamo la formula di Simpson espressa in questo modo, più facile da codificare.

$$\mathbf{I} = \frac{1}{3}h * ([f(a) + f(b)] + \sum_{i=2}^{n} 4f(x_i) + \sum_{i=3}^{n-1} 2f(x_i))$$

```
function [Int]= simpComp(func,a,b,n)
% Function che approssima lintegrale definito da f(x) con estremi a, b
% mediante la formula di Simpson composita su n+1 ascisse equidistanti.
% Input:
%       func= funzione da approssimare
%       a,b= estremi di integrazione
%       n= numero suddivisione pari di intervalli
format long;
if a==b
    Int=0;
else
    if(n <2 || mod(n,2)~=0)
        error("numero di ascisse non consistente");
    else
        h= (b-a)/n;
        x= linspace(a,b,n +1);
        f= feval(func ,x);

        S=(f(1)+f(n+1));
        S= S+4*sum(f(2:2:n))+ 2*sum(f(3:2:n-1));
        Int= (h/3)*S;
    end
    return

end
```

n	Trapezi	Simpson
1	$4.27719529223438 * 10^{-1}$	$-$
2	$2.66403558406035 * 10^{-1}$	$2.12631568133567 * 10^{-1}$
3	$2.21993054797790 * 10^{-1}$	$-$
4	$2.03432804450016 * 10^{-1}$	$1.82442553131343 * 10^{-1}$
5	$1.93960608940975 * 10^{-1}$	$-$
6	$1.88498346613972 * 10^{-1}$	$1.77333443886033 * 10^{-1}$
7	$1.85073233223903 * 10^{-1}$	$-$
8	$1.82789408875225 * 10^{-1}$	$1.75908277016961 * 10^{-1}$
9	$1.81193075146052 * 10^{-1}$	$-$
10	$1.80034803521960 * 10^{-1}$	$1.75392868382289 * 10^{-1}$
11	$1.79168480904422 * 10^{-1}$	$-$
12	$1.78504015707472 * 10^{-1}$	$1.75172572071972 * 10^{-1}$
13	$1.77983464586677 * 10^{-1}$	$-$
14	$1.77568218195411 * 10^{-1}$	$1.75066546519247 * 10^{-1}$
15	$1.77231763077838 * 10^{-1}$	$-$
16	$1.76955413111201 * 10^{-1}$	$1.75010747856527 * 10^{-1}$
17	$1.76725697384898 * 10^{-1}$	$-$
18	$1.76532709616469 * 10^{-1}$	$1.74979254439942 * 10^{-1}$
19	$1.76369035415410 * 10^{-1}$	$-$
20	$1.76229037552030 * 10^{-1}$	$1.74960448895386 * 10^{-1}$
21	$1.76108369095178 * 10^{-1}$	$-$
22	$1.76003635123329 * 10^{-1}$	$1.74948686529632 * 10^{-1}$
23	$1.75912153425968 * 10^{-1}$	$-$
24	$1.75831782461067 * 10^{-1}$	$1.74941038045599 * 10^{-1}$
25	$1.75760795825844 * 10^{-1}$	$-$
26	$1.75697789420194 * 10^{-1}$	$1.74935897698033 * 10^{-1}$
27	$1.75641611931910 * 10^{-1}$	$-$
28	$1.75591312187042 * 10^{-1}$	$1.74932343517586 * 10^{-1}$
29	$1.75546098850584 * 10^{-1}$	$-$
30	$1.75505309277079 * 10^{-1}$	$1.74929824676826 * 10^{-1}$

2.5 Esercizio 25 - Descrizione

Per entrambe le formule si ha che il costo computazionale dipenderà dal tipo di funzione che si vuole studiare e della tolleranza fissata. Inoltre il costo sarà direttamente proporzionale al numero di volte che viene chiamata ricorsivamente la funzione che si sta considerando.

Consideriamo l'integrale definito:

$$I(f) = \int_{-1}^{1} \frac{1}{1+10^2x^2}\mathrm{d}x.$$

2.5.1 Svolgimento:

Trapezi Adattiva

```
function [I2, points] = adapTrap( f, a, b, tol, fa, fb )
% [I2,points] = adapTrap( f, a, b, tol )
% Function che approssima lintegrale definito da func(x) con estremi a, b
% mediante la formula di Trapezi adattiva.
% Input:
%      f= Funzione Integranda
%      a,b= estremi di integrazione
%      tol= Accuratezza risultati
global points
delta = 0.5; % ampiezza minima intervalli
if nargin<=4
   fa = feval( f, a );
   fb = feval( f, b );
   if nargout==2
      points = [a fa; b fb];
   else
      points = [];
   end
end
h = b-a;
x1 = (a+b)/2;
f1 = feval( f, x1 );
if ~isempty(points)
   points = [points; [x1 f1]];
end
I1 = .5*h*( fa+fb );
I2 = .5*( I1 + h*f1 );
e = abs( I2-I1 )/3;
if e>tol || abs(b-a)>delta
   I2 = adapTrap( f, a, x1, tol/2, fa, f1 ) + adapTrap( f, x1, b, tol/2, f1, fb );
end
return
end
```

Simpson Adattiva

```
function [I2,points] = adapSim( func, a, b, tol, fa, f1, fb )
% [I2,points] = adapSim( f, a, b, tol )
% Function che approssima lintegrale definito da func(x) con estremi a, b
% mediante la formula di Simpson adattiva.
% Input:
%      f= Funzione Integranda
%      a,b= estremi di integrazione
%      tol= Accuratezza risultati
global points
delta = 0.5; % ampiezza minima intervalli
x1 = (a+b)/2;
if nargin<=4
   fa = feval( func, a );
   fb = feval( func, b );
   f1 = feval( func, x1 );
   if nargout==2
      points = [a fa;x1 f1; b fb];
   else
      points = [];
   end
end
h = (b-a)/6;
x2 = (a+x1)/2;
x3 = (x1+b)/2;
f2 = feval( func, x2 );
f3 = feval( func, x3 );
if ~isempty(points)
   points = [points; [x2 f2; x3 f3]];
end
I1 = h*( fa+4*f1+fb );
I2 = .5*h*( fa + 4*f2 + 2*f1 + 4*f3 +fb );
e = abs( I2-I1 )/15;
if e>tol || abs(b-a)>delta
   I2 = adapSim( func, a, x1, tol/2, fa, f2, f1 ) + adapSim( func, x1, b, tol/2, f1, f3, fb );
end
return
end
```

Risultati a Confronto:

Tol	Trapezi Adattativa	Simpson Adattativa
10^{-2}	$2.956 * 10^{-1}$	$2.813 * 10^{-1}$
10^{-3}	$2.946 * 10^{-1}$	$2.813 * 10^{-1}$
10^{-4}	$2.943 * 10^{-1}$	$2.943 * 10^{-1}$
10^{-5}	$2.942 * 10^{-1}$	$2.942 * 10^{-1}$
10^{-6}	$2.942 * 10^{-1}$	$2.942 * 10^{-1}$

3 Calcolo del Google Pagerank: Svolgimento

3.1 Esercizio 26 - Descrizione

Il Metodo delle potenze, nella sua definizione generale potrebbe avere problemi di *underflow* o *overflow*, questo problema può essere limitato impostando un limite alla approssimazione che si vuole ottenere del autovalore dominante e del corrispettivo autovettore.

3.1.1 Svolgimento:

```
function [l1,x1]=metodoPotenze(M, tol, itmax)
% Function che implementa il metodo delle potenze per calcolare
%  lautovalore  dominante e il corrispettivo autovettore
% Input:
%      M= matrice
%      tol= tolleranza
%      itmax= numero massimo di iterazioni
[m,n]= size(M);
if m~=n, error('La matrice non  quadrata!'), end
if nargin<=2
    if nargin<=1
        tol= 1e-6;
    else
        if tol>=0.1 || tol<=0, error( 'Tolleranza non valida'), end
    end
    itmax= ceil(-log10(tol))*n;
end
x= rand(n,1);
l1= 0;
for k=1:itmax
    x1= x/norm(x);
    x= M*x1;
    l0= l1;
    l1= x'*x1;
    err= abs(l1-l0);
    if err<=tol*(1+ abs(l1)), break, end
end
    if err>tol *(1+abs(l1)), error('Convergenza non raggiunta'), end
return
```

3.2 Esercizio 27 - Descrizione

L'esercizio richiede di rappresentare il valore dell'autovalore dominante, e del numero di iterazioni necessarie per soddisfare il criterio di arresto, rispetto ad N. Per favorire la leggibilità del codice, la parte grafica è stata realizzata in uno script separato.

3.2.1 Svolgimento:

```
function[l1, q, k]=potenzeContatore(M, tol, itmax)
% Function che implementa i l metodo delle potenze per calcolare
%  lautovalore  dominante e il corrispettivo autovettore, avendo
% come vettore di partenza u=(1,...,1)^T
% Input:
%      M= matrice
%      tol= tolleranza
%      itmax= numero massimo di iterazioni
[m, n] = size(M) ;
if m~=n, error( 'Matrice non  quadrata' ), end
if nargin <= 2
    if nargin<=1, tol= 1e-6;
    else
        if tol>=0.1 || tol<=0, error('tolleranza non valida '), end
    end
itmax= ceil(-log10(tol))*n*n ;
end

u=ones(n,1);
l1 = 0 ;
for k=1:itmax
    q= u/norm(u);
    u= M *q;
    l0=(q'*u)/(q'* q ) ;
    err= abs(l0-l1);
    l1= l0;
    if err<=tol*(abs(l1)), break, end
end
if   err > tol*(abs(l1)), error('convergenza non ottenuta'), end

return
```

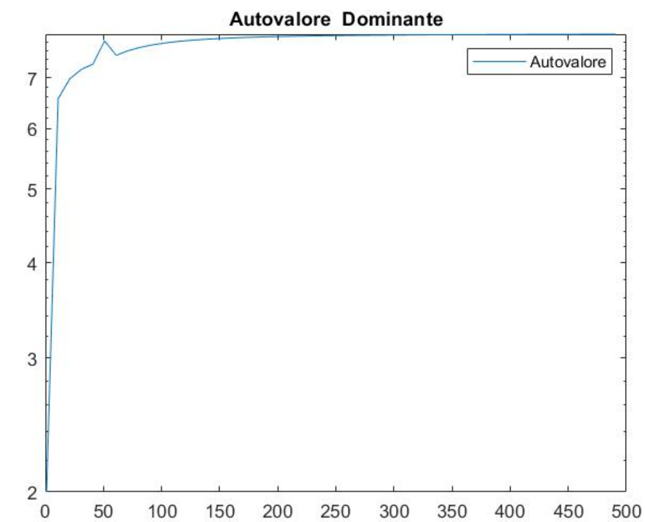
Script per l'esecuzione del Metodo con Grafici:

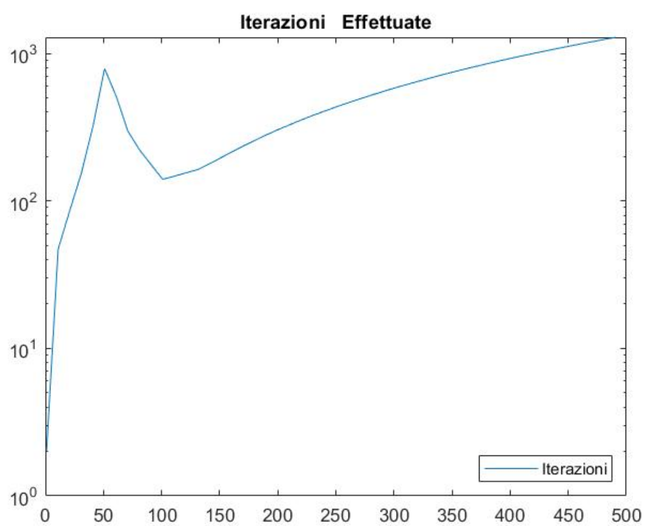
```
tol= 10e-10;
j= 1 ;
res= zeros(1,50);
iter= zeros(1,50);

for N=10:10:500
    B= -1*ones(N,5);
    B(:,3)= B(:,3)*(-4);
    A= spdiags(B, [-9 -1 0  1  9 ], N, N);
    [l1, x1, k]= potenzeContatore(A,tol);
    res(1,j)= l1;
    iter(1,j)= k;
    j= j+1 ;
end
```

```
figure
semilogy (1:10:500, res(:));
title('Autovalore Dominante');
legend('Autovalore');

figure
semilogy(1:10:500, iter(:)) ;
title('Iterazioni Effettuate');
legend('Iterazioni','location','southeast');
```





3.3 Esercizio 28 - Descrizione

L'esercizio richiede di costruire una Function che risolve il sistema lineare Ax=b tramite un generico splitting, che viene passato dal parametro Msolve.

3.3.1 Svolgimento:

```
function [x]=splitting(A, Msolve, b, tol)
% Function che risolve il sistema lineare Ax=b tramite un generico splitting,
% che viene passato dal parametro Msolve.
% Input:
%       b= vettore colonna dei termini noti
%       A= Matrice
%       Msolve= funzione di Jacobi o di GaussSeidel
%       tol= tolleranza
[n,m]= size(A);
if(n~= m || n~=length(b)), error('dati inconsistenti'), end

itmax= ceil(-log10(tol))*m;
x= zeros(n,1);
tolb= tol*norm(b,inf);
for i= 1:itmax
    r= A*x-b;
    nr= norm(r,inf);
        if nr <= tolb, break, end

    v= Msolve(r,A);
    x= x-v ;
end

if nr>tolb, error ('tolleranza non raggiunta'), end

return
```

3.4 Esercizio 29 - Descrizione

L'esercizio richiede di scrivere due Function ausiliarie, per la function del *precedente esercizio*, che implementano i metodi iterativi di Gauss-Seidel e Jacobi .

3.4.1 Svolgimento:

Gauss-Seidel:

```
function[y]=GaussSeidel(A,b)
% Function per la risoluzione del sistema tramite il metodo di Gauss-Seidel
% Input:
%       b= colonna vettore dei termini noti
%       A= matrice nxn
n= length(b);
y= b;

for i=1:n
    y(i)= y(i)/A(i,i);
    y(i+1:n)= y(i+1:n)- y(i)*A(i+1:n,i);
end

return
```

Jacobi:

```
function [y]= Jacobi(A,b)
% Function per la risoluzione del sistema tramite il metodo di Jacobi
% Input:
%       b= colonna vettore dei termini noti
%       A= matrice nxn
D= diag(A);
y= b./D;

return
```

3.5 Esercizio 30 - Descrizione

L'esercizio richiede di risolvere il sistema dato, e di graficare il numero delle iterazioni richieste dai due metodi iterativi, rispetto ad N, per soddisfare il criterio di arresto prefissato. Per favorire la leggibilità del codice, la parte grafica è stata realizzata in uno script separato.

3.5.1 Svolgimento:

```
function[x,i]=splitting(b, matvec, msolve, tol)
%  Funzione per la risoluzione adhoc di un sistema lineare con splitting
%  di  Jacobi  o  Gauss-Seidel
%  Input:
%        b= vettore dei termini noti
%        matvec= funzione per creare la matrice adhoc per  esercizio 26
%        msolve= funzione per risolvere il sistema lineare
%        tol= tolleranza desiderata

n= length(b);
itmax= ceil(-log10(tol))*(n*n);
x= zeros(n,1);
tolb= tol*norm(b,inf);

for i= 1:itmax
    r= matvec(x)-b;
    nr= norm(r,inf);
    if nr <= tolb, break, end

    u= msolve(r);
    x= x-u ;
end
if nr > tolb
    fprintf('Convergenza non raggiunta'), end

return
```

Funzione MatVec:

```
function [y] = matvec(x)
% Metodo che ritorna il prodotto tra la matrice A dell'esercizio 26 ed il vettore x .
% Input:
%       x= vettore da moltiplicare con la matrice A dell'esercizio 26
y= x*4;
y(9:end)= y(9:end)- x(1:end-8);
y(2:end)= y(2:end)- x(1:end-1);
y(1:end-1)= y(1:end-1)- x(2:end);
y(1:end-8)= y(1:end-8)- x(9:end);

return
```

Funzione Jacobi:

```
function [y]= Jacobi(x)
%  Risolve un sistema diagonale per la matrice dell'esercizio 26
%  Input:
%       x= vettore dei termini noti
y= x./4;

return
```

Funzione di Gauss-Seidel

```
function [y]= gaussSeidel(x)
%  Risolve un sistema triangolare inferiore per la matrice dell'esercizio 26
%  Input:
%        x= vettore dei termini noti
n= length(x);
y= x;
y(1)= y(1)/4;

for i= 2:n
    y(i)=(y(i)+ y(i-1))/4;
end

for i= 9:n
    y(i)=(y(i)+y (i-1))/4 ;
end

return
```

Scrip dedicato al plot del Grafico Richiesto:

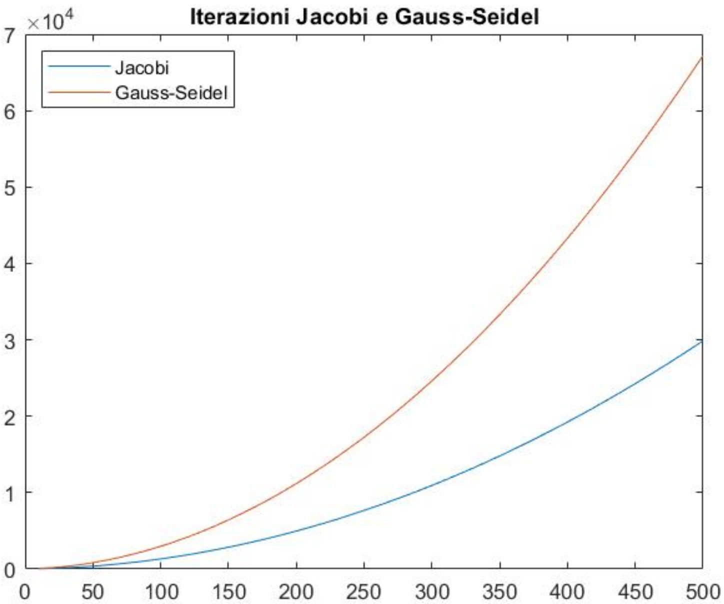
```
tol= 10^-8;
iter= zeros(50, 3);

for n= 10:10:500
    b= ones(n, 1);
    iter(n/10, 1)= n;
    [~, i]= splitting(b, @matvec, @Jacobi, tol);
    iter(n/10, 2)= i;
    [x, i]= splitting(b, @matvec, @gaussSeidel, tol) ;
    iter(n/10, 3)= i;
end

figure
plot((10:10:500), iter(:, 2));
hold on

plot((10:10:500), iter(:, 3));
hold on

legend('Jacobi','Gauss-Seidel','location','northwest');
title('Iterazioni Jacobi e Gauss-Seidel');
```


Iterazioni Jacobi e Gauss-Seidel
×10^4
Jacobi
Gauss-Seidel
0
1
2
3
4
5
6
7
0
50
100
150
200
250
300
350
400
450
500