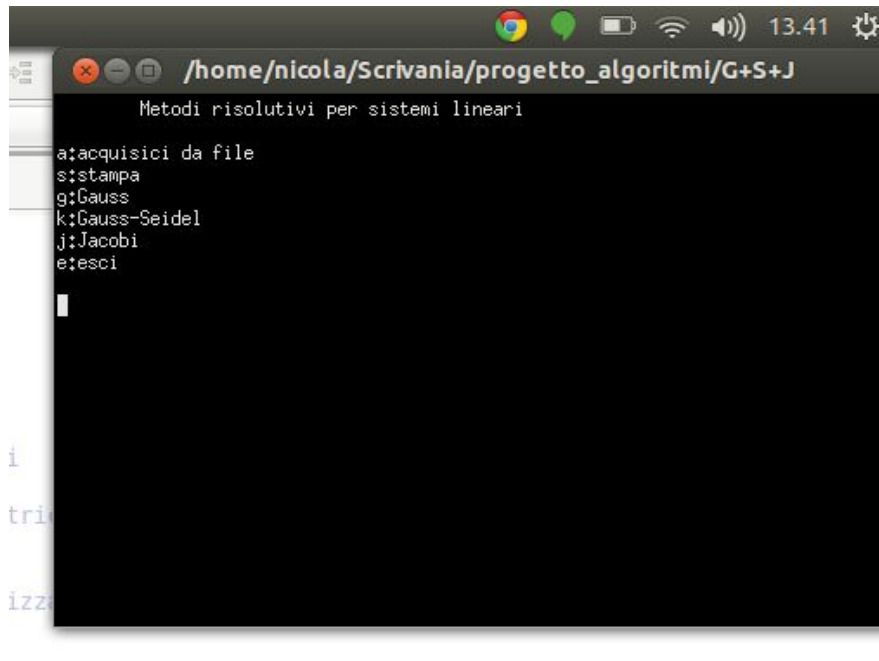


ANALISI DI TECNICHE RISOLUTIVE PER SISTEMI LINEARI



```
/home/nicola/Scrivania/progetto_algoritmi/G+S+J

Metodi risolutivi per sistemi lineari

a:acquisisci da file
s:stampa
g:Gauss
k:Gauss-Seidel
j:Jacobi
e:esci

i
tri
izz
```

Il progetto consiste nell'implementazione delle tre principali tecniche risolutive per sistemi lineari, il **metodo diretto** di **Gauss** e i due **metodi iterativi Jacobi e Gauss-Seidel**, in linguaggio c++ e Matlab.

Sfruttando la potenza di questi due linguaggi di programmazione abbiamo steso questo piccolo documento dove riportiamo le considerazioni, le valutazioni e i relativi risultati sperimentali ottenuti dal programma sviluppato.

IL PROGRAMMA

Il programma consiste in un unico sorgente con un *main* e una serie di *sottofunzioni* che offrono all'utente un'interfaccia essenziale e funzionale con un menù a selezione e la possibilità di acquisire le matrici comodamente da un file di testo, a patto che sia presente nella directory da dove viene lanciato l'eseguibile del programma.

Un possibile miglioramento del programma potrebbe consistere nell'inserire i prototipi delle sottofunzioni in un Header-File e le sottofunzioni stesse in un file linkato al sorgente precompilato, diminuendo così il tempo di compilazione del sorgente in caso di revisioni e, grazie alla tecnica di incapsulamento, prevenire errori in fase di revisione del codice.

```

1      #include <iostream>
2      #include <fstream>
3      #include <stdlib.h>
4      using namespace std;
5
6      //variabili globali
7
8      //Gauss
9      int n, i, j, k, ind=0; //dimensione dell'input e contatori
10     double alfa, rmax, temp, ck, s=0;
11     //alfa=variabile di appoggio che contiene il valore di matrice[i][k]
12     //rmax=elemento massimo della colonna
13     //temp=variabile di appoggio
14     //ck=coefficiente che moltiplica le righe per la diagonalizzazione
15     //variabile per il calcolo delle soluzioni all'indietro
16
17     //AcquisisciMatrice
18     fstream y;//variabile di tipo fstream
19     const int massimo=100;
20     double matrice[massimo][massimo];
21
22     int SO,q;//serve per selezionare la sintassi per i comandi della shell
23
24
25
26     // F U N Z I O N I-----
27
28     void pausa(){
29         if(SO)system("pause");
30         else{
31             cout<<"\nQualsiasi lettera per continuare...\n";
32             getchar();//inserisco due get char in quanto uno
33             getchar();//prende l'invio del comando precedente e uno
34                 //prende l'invio per 34continuare
35             }
36         }
37
38     void stampa(){
39         cout<<endl;
40
41         //stampa
42         for(int i=1;i<=n;i++){//scorrimento delle righe
43             for(int j=1;j<=n+1;j++){//scorrimento delle colonne
44                 cout<<matrice[i][j]<<"\t";
45             }
46             cout<<endl;
47         }
48         cout<<endl<<endl;
49
50         //pausa
51         pausa();
52     }
53
54     void inizializza(int k){ //serve a inizializzare la matrice completa
55         for (int i=0; i<k;i++)
56             for(int j=0; i<k+1;i++)
57                 matrice[i][j]=0;

```

```

58     }
59
60     void acquisisciMatrice(){
61         char o[15]; //creo una stringa di 15 caratteri
62
63         cout<<"\nFile disponibili...\n\n";
64         if(SO) system("dir");
65         else system("ls");
66         cout<<"_____ "<<endl;
67         cout<<"\nInserisci il nome del file contenente la matrice da analizzare
68         \n[specifica l'estensione .txt]\n";
69         cin>>o;
70         y.open(o,ios::in); //nel file di testo vanno inseriti in testa due caratteri
71                                //indicanti le dimensioni della matrice
72
73
74         //ACQUISIZIONE
75         if(y.good()){
76             //recupera il numero delle equazioni
77             y>>n;
78
79             for(int i=1;i<=n;i++){ //scorrimento delle righe
80                 for(int j=1;j<=n+1;j++){ //scorrimento delle colonne
81                     y>>matrice[i][j]; //assegnamento
82                 }
83             }
84
85
86         }
87         y.close();
88         cout<<"Acquisizione avvenuta...";
89
90         pausa();
91         cout<<endl<<endl;
92
93         return;
94     }
95
96
97     void Gauss(){ //ok
98         cout<<"\nGauss\n\n";
99         double soluzione[n]; //inizializzo il vettore soluzione
100         for(k=1;k<=n-1;k++){
101             rmax=abs(matrice[k][k]);
102             ind=k; //la variabile ind indica la riga sulla quale si sta operando
103             for(i=k+1;i<=n;i++){
104                 alfa=abs(matrice[i][k]);
105                 if(alfa>rmax){
106                     rmax=alfa;
107                     ind=i; //indica la riga che 108contiene il massimo
108                 }
109             }
110
111
112
113             if(rmax==0){
114                 cout<<"\n Il sistema non ammette soluzione\n\n";
115                 //pausa
116                 pausa();
117             }
118
119             if(ind!=k){
120                 for(j=k;j<=n+1;j++){ //per tutti gli elementi della riga

```

```

121             temp=matrice[ind][j];
122             matrice[ind][j]=matrice[k][j];
123             matrice[k][j]=temp;
124         }
125     }
126     //Diagonalizzazione della matrice dei coefficienti
127     for(i=k+1;i<=n;i++){//ok
128         ck=matrice[i][k]/matrice[k][k];
129         for(j=k/*+1*/;j<=n+1;j++) matrice[i][j]-=ck*matrice[k][j];
130     }
131 }//chiusura for k
132
133 //-----
134 //Calcolo del vettore delle soluzioni
135 if(matrice[n][n]==0){//ok
136     cout<<"\n Il sistema e' indeterminato\n\n";
137     //pausa
138     pausa();
139 }
140 soluzione[n]=matrice[n][n+1]/matrice[n][n];
141 for(i=n-1;i>=1;i--){//ok
142     for(j=i+1;j<=n;j++){
143         s+=matrice[i][j]*soluzione[j];
144     }
145     soluzione[i]=(matrice[i][n+1]-s)/matrice[i][i];
146 }
147 //-----
148 //Stampa del vettore soluzione
149 cout<<"\n La soluzione e':\n\n";
150 for(i=1;i<=n;i++) cout<<soluzione[i]<<endl<<endl;
151 //-----
152
153     //pausa
154     pausa();
155     inizializza(n);
156 }
157
158 void Jacobi(){//ok
159     cout<<"\nJacobi\n\n";
160     //variabili
161     int i=1, j=1;
162
163     //controllo che la matrice sia a diagonale dominante
164     for (i=1; i<=n; i++){
165         double s = 0;
166         for (j=1; j<=n; j++){
167             s += abs(matrice[i][j]);
168         }
169         s -= abs(matrice[i][i]);
170         if (abs(matrice[i][i])<s){
171             cout <<"\nLa matrice non e' a diagonale dominante.\n\n";
172             pausa();
173             return ;
174         }
175     }
176     int m;
177     cout << "\nInserire il numero di iterazioni\n\n";
178     cin >>m;
179     //creo il vettore tentativo
180     double tentativo[n], tentativo2[n];
181     for(i=1;i<=n;i++){
182         tentativo[i]=matrice[i][n+1]/matrice[i][i];

```

```

183     }
184
185     while(m>0){
186         for (i=1; i<=n; i++){
187             double s=0;
188             for (j=1; j<=n; j++){
189                 if(j!=i){
190                     s+=matrice[i][j]*tentativo[j];
191                 }//chiusura if
192             }//chiusura for j
193             tentativo2[i]=(matrice[i][n+1]-s)/matrice[i][i];
194         }
195         for(i=1; i<=n; i++){
196             tentativo[i]=tentativo2[i];
197         }
198         m--;
199     }
200
201
202
203     cout << "\n La soluzione e': \n\n";
204     for (i=1; i<=n; i++){
205         tentativo[i]=tentativo2[i];
206         cout << tentativo[i] << endl;
207     }
208     cout<<endl<<endl;
209
210     //pausa
211     pausa();
212     inizializza(n);
213 }
214
215 void GSeidel(){//ok
216     cout<<"\nGauss-Seidel\n\n";
217     //Controllo diagonale dominante
218     for(i=1;i<=n;i++){
219         float s=0;
220         j=1;
221         for(j=1;j<=n;j++){
222             s+=abs(matrice[i][j]);
223         }
224         s-=abs(matrice[i][i]);
225         if(abs(matrice[i][i])<s){
226             cout << " La matrice non soddisfa la condizione di dominanza della diagonale\n\n";
227             //pausa
228             pausa();
229         }
230     }
231
232     //Creazione vettore tentativo
233     double tentativo[n], tentativo1[n];
234     for(i=1;i<=n;i++) tentativo[i]=(matrice[i][n+1])/(matrice[i][i]);
235
236     //Calcolo del vettore soluzione
237     int m;
238     cout << " Inserire il numero di iterazioni \n\n";
239     cin >> m;
240     cout << endl << endl;
241     while(m>0){
242         for(i=1;i<=n;i++){
243             tentativo1[i]=matrice[i][n+1]/matrice[i][i];
244             for(j=1;j<=n;j++){

```

```

245             if(j==i) continue;
246             tentativo1[i]-=(matrice[i][j]/matrice[i][i])*tentativo[j];
247             tentativo[i]=tentativo1[i];
248         }
249     }
250     m--;
251 }
252
253     cout << " Il vettore soluzione e':\n\n";
254     for(i=1;i<=n;i++){
255         tentativo[i]=tentativo1[i];
256         cout << tentativo[i] << endl;
257     }
258     cout << endl;
259
260     //pausa
261     pausa();
262     inizializza(n);
263
264     return ;
265 }
266
267
268
269 //  M A I N -----
270 int main(){
271
272
273     char c;
274     int h=1;
275     cout<<"Quale Sistema Operativo stai utilizzando?\n0 per Unix-Linux-OSX\n1 per
Microsoft 276 Windows\n\n";
277     cin>>S0;
278     while(1) {
279
280         if(S0)system("cls");
281         else system("clear");
282
283         cout << "\tMetodi risolutivi per sistemi lineari\n\n";
284
285         cout<<"a:acquisisci da
file\ns:stampa\ng:Gauss\nk:Gauss-Seidel\nj:Jacobi\ne:esci\n\n";
286         cin>>c;
287         switch(c){
288             case 'a': acquisisciMatrice();break;
289             //case 'i': inserisci();break;
290             case 's': stampa();break;
291             case 'g': Gauss();break;
292             case 'j': Jacobi();break;
293             case 'k': GSeidel();break;
294             case 'e': return 0;
295             default:cout<<"\tCARATTERE NON VALIDO!\n";
296         }//chiusura dello switch
297
298     }//chiusura del while
299 }
300 //Fine del programma

```

COMPLESSITÀ COMPUTAZIONALE

La complessità computazionale è un indice di efficienza di un algoritmo, essa può essere suddivisa in complessità spaziale e complessità temporale. In questa discussione ci occuperemo della seconda, in quanto il tempo rappresenta una risorsa *non riutilizzabile*, mentre la quantità di memoria occupata risulta trascurabile poiché *riutilizzabile* e soprattutto perché grazie alle odierne tecnologie, volte a risolvere problemi ben più complessi della semplice risoluzioni di sistemi lineari, disponiamo di una vastissima capacità di memoria-dati.

Basandoci sul modello ideale di *Macchina RAM* e considerando i seguenti “costi” (in termini di tempo macchina) per le istruzioni

- *UNITARIO* per lettura, scrittura, assegnamento, operazioni elementari, return, accesso all'array e valutazioni booleane.
- *NON UNITARIO* per istruzioni composte, cicli, valutazioni condizionali e chiamate a sottofunzioni.

calcoleremo le complessità computazionali delle sotto-funzioni presenti nel sorgente del nostro programma:

pausa()

Una funzione che arresta i processi permettendo all'utente di visualizzare i risultati delle precedenti operazioni. E' volta esclusivamente a ottimizzare l'esperienza dell'utente ed è composta da una semplice istruzione condizionale contenente operazioni a costo unitario. Ciò permette di ottenere una complessità computazionale pari a $\theta(1)$, pertanto una tale procedura, sebbene non abbia alcuno scopo funzionale, non pregiudica assolutamente la tempistica dei sottoprogrammi in cui viene richiamata.

stampa()

Funzione di stampa a video della matrice completa precedentemente acquisita.

Essendo la matrice inizializzata come Array bidimensionale (con doppio indice), questa subroutine produce una complessità computazionale pari a $\theta(n^2)$.

inizializza()

Metodo che inizializza a zero tutti gli elementi della matrice. Come la funzione di stampa, anche essa è legata alla tecnica implementativa della matrice e quindi ha complessità computazionale $\theta(n^2)$.

acquisisciMatrice()

Funzione che acquisisce da file di testo una matrice bidimensionale di dimensione massima $n=100*101$.

E' una funzione molto comoda che agevola l'inserimento delle matrici ed evita errori di battitura da parte dell'utente, piuttosto frequenti quando si lavora con sistemi molto grandi.

I file di testo devono presentarsi nella seguente forma:

EsempioDiMatrice.txt

```
2          nella prima riga è inserita la dimensione della matrice incompleta
3      33      36      nelle righe successive sono inseriti i termini della matrice completa
2      22      24
```

Per mezzo di tale procedura è inoltre possibile - sia utilizzando sistemi UnixLike, come le varie distro Linux o Sistemi OsX, sia le varie versioni di Microsoft Windows - visualizzare l'elenco dei file disponibili nella directory del programma.

Il tutto è implementato tramite un ciclo *if* a costo unitario con due cicli *for* annidati al suo interno e la chiamata alla funzione *pausa()*, con una conseguente complessità computazionale pari a $\theta(n^2)$.

Gauss()

Questa funzione implementa il metodo numerico per il calcolo del vettore soluzione di un sistema rappresentato da una matrice completa del tipo $[A|c]$, dove A è la matrice dei coefficienti e c è il vettore dei termini noti.

Il sottoprogramma inizia con una prima valutazione della matrice per mezzo di due cicli *for* annidati e una condizione *if* per valutare se effettivamente il sistema ammetta soluzioni, tale operazione comporta una complessità $\theta(n^2)$.

Successivamente il codice provvede a triangolarizzare la matrice (complessità $\theta(n^3)$) e alla risoluzione del sistema all'indietro (complessità $\theta(n^2)$).

Oltre a ciò sono presenti una semplice funzione di stampa del vettore soluzione, avente complessità $\theta(n)$, e le chiamate alle funzioni *pausa()* ($\theta(1)$) e alla funzione *inizializza()* ($\theta(n^2)$).

La formula implementata è la seguente

$$x_i = \frac{[b_i - \sum_{j=i+1}^n a_{ij}x_j]}{a_{ii}}$$

Caso migliore = Caso medio = Caso peggiore:

Uno degli svantaggi di questo metodo è che, essendo di tipo diretto, il numero di passaggi che esegue è predeterminato, questo fatto non ne ottimizza né la precisione (errore di arrotondamento), né il tempo di esecuzione. La complessità computazionale complessiva della routine è pari a $\theta(n^3)$.

Jacobi()

Implementa l'algoritmo iterativo di Jacobi.

Questo procedimento richiede un'ulteriore condizione affinché il programma restituisca un valore corretto: la matrice deve essere a **diagonale dominante**, ossia deve valere la seguente relazione:

$$\forall i = 1, \dots, n \quad |a_{ii}| \geq \sum_{j=1}^n |a_{ij}|$$

Il codice inizia con due cicli *for* annidati che controllano la diagonale dominante del sistema (complessità $\theta(n^2)$).

Successivamente viene richiesto all'utente il numero di iterazioni desiderate. Bisogna considerare che più alto sarà tale valore e più basso sarà l'*errore di troncamento*, pertanto il vettore soluzione convergerà ad un risultato sempre più preciso; al contrario però aumenterà l'*errore di arrotondamento*, con un conseguente allontanamento dal risultato esatto. È dunque fondamentale scegliere con criterio il numero di iterazioni da eseguire tenendo conto di questi due fattori e cercando un compromesso che minimizzi l'influenza di entrambi gli errori.

Successivamente viene applicato iterativamente il metodo per m-volte, ogni iterazione contiene due cicli *for* annidati con un'istruzione condizionale a costo unitario e un'ulteriore ciclo *for* che copia i valori dal vettore *tentativo2* al vettore *tentativo*.

Chiudono come sempre le funzioni *stampa del vettore soluzione*, *pausa()* e *inizializza()* rispettivamente $\theta(n^2)$, $\theta(1)$ e $\theta(n^2)$.

La formula implementata è la seguente

$$(x_i)^m = \frac{[b_i - \sum_{j=1}^m a_{ij} \cdot (x_j)^{m-1}]}{a_{ii}}$$

Caso Migliore:

La matrice non è diagonale Dominante quindi, non essendo applicabile il metodo di Jacobi, il programma esegue solo i due cicli *for* iniziali con conseguente complessità $\theta(n^2)$.

Caso Medio:

Per m sufficientemente piccolo rispetto ad n ($m \ll n$) il programma esegue poche iterazioni includenti i due cicli *for* annidati ($\theta(n^2)$), pertanto la complessità computazionale risultante è $\theta(m \cdot n^2) = m \cdot \theta(n^2)$.

Caso Peggior:

Per m sufficientemente grande il programma esegue molte iterazioni comprendenti i due cicli *for* annidati con conseguente complessità che tende a $\theta(n^3)$.

GSeidel()

Questo algoritmo è un perfezionamento di quello di Jacobi. Il vantaggio principale consiste nell'utilizzo di un unico vettore tentativo, con un conseguente abbassamento della complessità computazionale spaziale ma non di quella temporale.

La formula risolutiva è la seguente:

$$(x_i)^m = \frac{[b_i - \sum_{j=1}^{i-1} a_{ij} \cdot (x_j)^m + \sum_{j=i+1}^n a_{ij} \cdot (x_j)^{m-1}]}{a_{ii}}$$

ERRORI

Dalla teoria dell'analisi numerica sappiamo che nei metodi numerici esistono due tipi di errore:

- Errore di Troncamento
- Errore di Arrotondamento

Il primo è causato da un numero insufficiente di iterazioni del metodo numerico che non permettono al vettore soluzione di avvicinarsi abbastanza al vettore soluzione "algebrico"

Il secondo, al contrario, è provocato da un numero troppo elevato di iterazioni che comportano un eccessivo arrotondamento dei numeri considerati, con una conseguente perdita di qualità del metodo risolutivo e un calo della precisione dei valori ottenuti.

A dimostrazione della veridicità delle nozioni teoriche appena esposte abbiamo testato i tre metodi implementati, ottenendo i risultati sotto riportati.

Al fine di stimare lo scarto tra il vettore soluzione (Chiamato v) e il vettore Soluzione Algebrica (Chiamato $v0$) calcoleremo mediante lo script *Norma.m* la norma del vettore differenza dei due, ossia la distanza che intercorre tra v e $v0$. La norma è direttamente proporzionale all'errore del vettore V .

$$\| v0 - v \| = \sqrt{(v0_1 - v_1)^2 + \dots + (v0_n - v_n)^2}$$

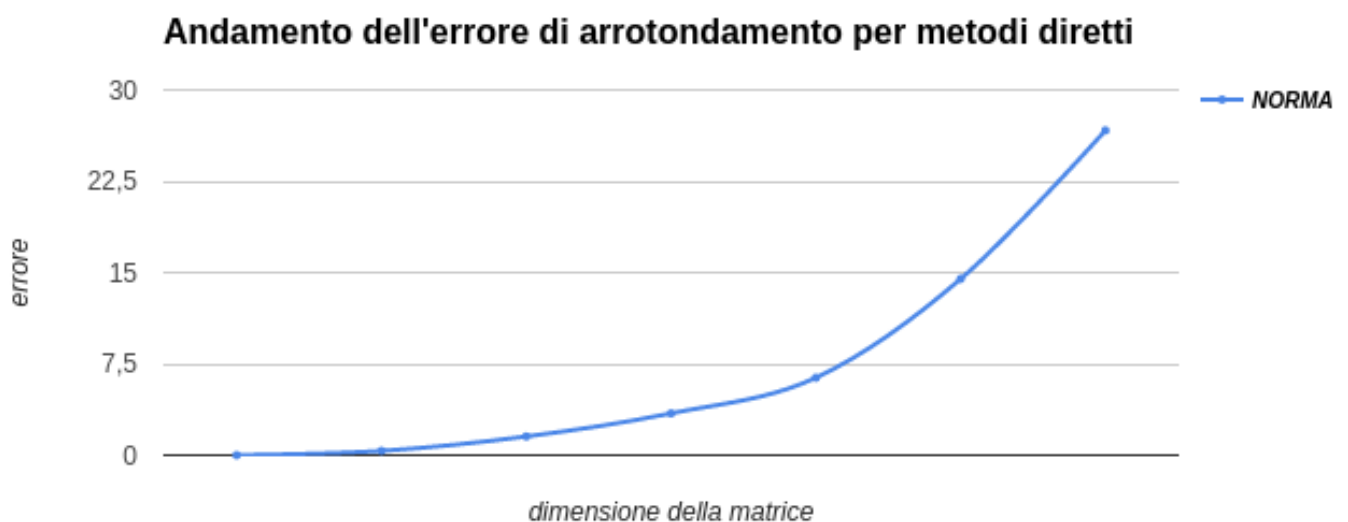
Gauss

Il metodo di Gauss è un metodo **diretto** in quanto il numero di passaggi che esegue è funzione della dimensione della matrice. Di conseguenza è un metodo soggetto al solo **errore di arrotondamento**.

Considerando le matrici *m2.txt* ... *m20.txt* abbiamo ottenuto i seguenti risultati sperimentali...

DIMENSIONE MATRICE	NORMA
2	0
4	0,3525
6	1,5362
8	3,4352
10	6,3607
15	14,476
20	26,6734

L'andamento dell'errore è quindi il seguente...



Jacobi e Gauss-Seidel

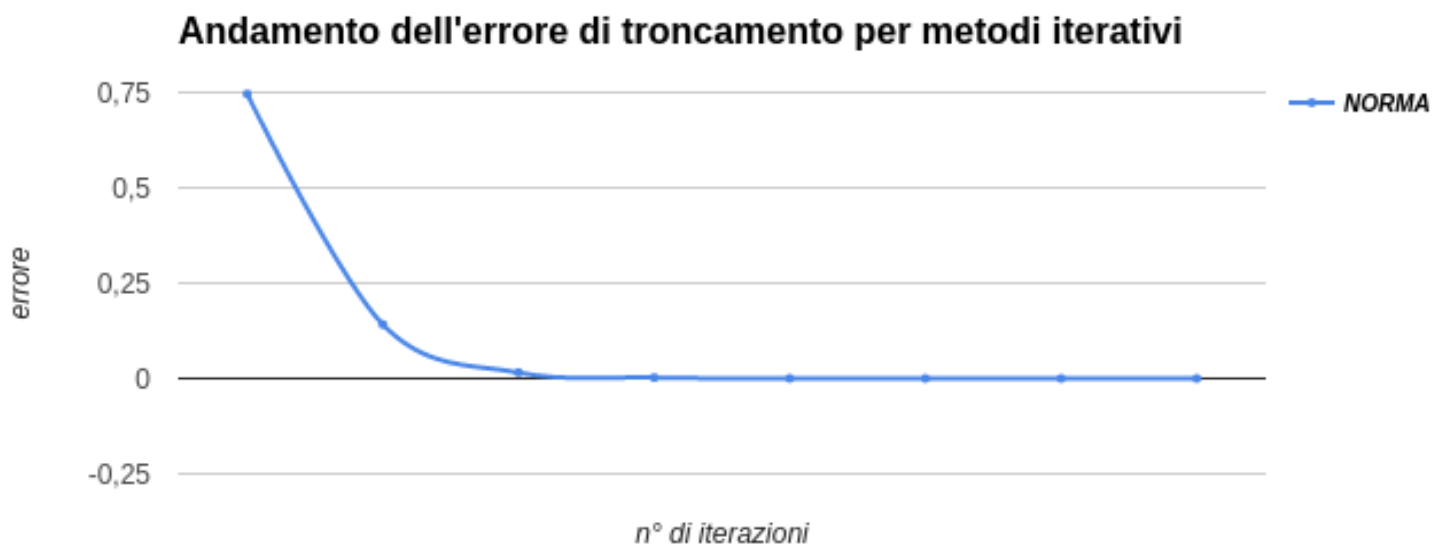
I metodi di Jacobi e Gauss-Seidel sono di tipo **iterativo** e, oltre all'*errore di arrotondamento*, presentano anche l'**errore di troncamento**.

L'errore di troncamento è *inversamente proporzionale* al numero di iterazioni applicate dal metodo numerico.

Considerando la matrice *matrice.txt* abbiamo ottenuto i seguenti risultati sperimentali...

NUMERO DI ITERAZIONI	NORMA
2	0,746
4	0,1416
6	0,0153
8	0,0022
10	3,28E-04
12	4,68E-05
14	3,87E-06
16	0

l'andamento dell'errore è il seguente...



qualche screen del codice in esecuzione e dell'analisi con matlab...

The screenshot shows a computer screen with a Google Sheet and a terminal window. The Google Sheet, titled 'ErroreTroncamentoGSidel', has columns A, B, C, and D. Column B is labeled 'DIMENSIONE MATRICE' and column C is labeled 'NORMA'. The data in column C shows values increasing with the dimension of the matrix: 0, 0.9728, 5.5551, 16.1542, and 260.95... for dimensions 2, 4, 6, 8, and 20 respectively. The terminal window, titled 'Metodi risolutivi per sistemi lineari', shows a list of methods: a:acquisisci da file, s:stampa, g:Gauss, k:Gauss-Seidel, j:Jacobi, e:esci. It then displays a matrix of values for different methods and dimensions, with the value 16.1542 highlighted in the terminal output.

Dimensione Matrice	Norma
2	0
4	0.9728
6	5.5551
8	16.1542
10	
15	
20	

The screenshot shows the MATLAB IDE with a script named 'norma.m' in the Editor. The script calculates the norm of a vector 'a' using the formula $\sqrt{\sum_{i=1}^n (a(i,1) - r(i,1))^2}$. The Command Window shows the execution results, including the value of 'norm' (16.1542) and the value of 'sum' (260.95...). The Workspace window shows the variables 'a', 'ans', 'b', 'i', 'n', 'nor', 'r', and 'sum' with their respective values.

```

1 %calcolo la norma (ovvero la distanza) tra due vettori 'a' e 'r' dove
2
3 % a=soluzioni ottenute dall'applicazione di tot iterazioni per risolvere
4 % un sistema lineare
5
6 % r=soluzione ALGEBRICA di riferimento
7
8 sum=0;%inizializzo sum
9 for i=1:n
10     sum=sum+((a(i,1)-r(i,1))^2);
11 end
12 nor=sqrt(sum);
13 disp(nor);
  
```

Name	Value	Min	Max
a	[-5.95... -5.9...]	-5.9...	10
ans	4	4	4
b	[1;2;3]...	1	6
i	6	6	6
n	6	6	6
nor	16.1542	16.1...	16.1...
r	[1;2;3]...	1	10
sum	260.95...	260...	260...

SCRIPT MATLAB

norma.sh

%calcolo la norma (overo la distanza) tra due vettori "a" e "r" dove

%a=soluzioni ottenute dall'applicazione ti tot iterazioni per risolvere
% un sistema lineare

% r=soluzione ALGEBRICA di riferimento

```
sum=0;%inizializzo sum
for i=1:n
    sum=sum+((a(i,1)-r(i,1))^2);
end
nor=sqrt(sum);
disp(nor);
```

sparsa.sh

%Genera la matrice sparsa n*n a diagonale dominante
function a=sparsa(n)

```
e=ones(n,1);
```

% per avere una matrice a diagonale dominante,
diag>=5

```
diag=6;
```

```
b=[e, -e, diag*e, -e, 2*e];
```

```
d=[-n/2, -1, 0, 1, n/2];
```

```
a=spdiags(b,d,n,n);
```

MATRICI UTILIZZATE

Matrice.txt

50

[illegible]

m2.txt

2
50 5 105
5 50 60

m4.txt

4
50 5 4 2 225
5 50 5 4 184
4 5 50 5 136
1 4 5 50 76

m6.txt

6
50 5 4 2 1 7 356
5 50 5 4 1 4 318
4 5 50 5 4 1 273
1 4 5 50 5 4 210
1 8 4 5 50 5 182
3 1 2 4 5 50 103

m8.txt

8
50 5 4 2 1 7 1 1 497
5 50 5 4 1 4 1 5 463
4 5 50 5 4 1 2 1 416
1 4 5 50 5 4 1 3 353
1 8 4 5 50 5 1 1 331
3 1 2 4 5 50 1 1 236
1 1 2 1 1 1 50 5 144
1 2 1 0 1 3 5 50 101

m10.txt

10
50 5 1 1 1 2 1 0 1 2 584
5 50 5 1 3 1 1 1 3 1 584
1 5 50 5 3 1 2 1 0 1 525
1 4 5 50 5 1 1 1 1 6 486
2 1 1 5 50 5 1 2 1 1 410
1 1 2 1 5 50 5 1 4 1 354
2 1 0 3 1 5 50 5 1 1 299
2 0 0 2 4 1 5 50 5 0 243
1 1 2 4 0 1 0 5 50 5 188
2 1 0 2 4 3 3 1 5 50 157

m15.txt

15
80 1 1 5 1 1 7 1 1 1 0 1 1 1 1 1472
1 80 1 5 1 3 1 4 1 9 1 1 7 1 3 1 1444
1 1 80 1 1 1 1 1 9 1 1 1 20 1 1 1260
0 1 1 80 1 1 0 1 1 3 1 1 0 1 1 1053
1 22 1 1 80 1 1 1 1 0 5 1 1 0 1 1295
1 1 1 18 1 80 2 1 4 1 1 18 1 1 30 1241
1 1 8 1 1 1 80 1 1 1 9 1 1 13 1 986
1 7 1 1 1 1 1 80 1 1 0 1 1 1 1 831
1 1 1 13 1 1 1 1 80 1 1 22 1 1 5 905
1 5 1 1 1 4 1 1 1 80 1 1 1 4 1 686
1 1 1 1 1 1 1 1 1 1 80 1 1 1 1 515
1 1 9 1 1 1 1 8 1 3 1 80 1 1 3 610
0 1 5 1 1 1 5 1 1 1 1 1 80 1 1 430
1 1 1 1 1 1 1 1 1 1 1 1 80 1 278
1 1 1 1 1 1 1 1 1 1 1 1 1 80 199

m20.txt

20
80 1 0 1 3 1 4 1 9 1 18 1 3 0 1 2 1 3 1 1 2132
0 80 1 1 5 1 20 1 1 18 1 1 3 1 1 5 1 1 2 1 2246
1 0 80 1 1 0 1 1 0 1 1 21 1 1 4 1 3 1 1 1 3 1804
1 1 1 80 1 0 1 1 0 1 4 1 13 1 0 1 6 1 1 12 1677
1 1 1 1 80 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1474
4 1 2 1 1 80 1 0 1 1 5 1 4 1 8 1 1 0 1 1 1563
3 1 1 20 1 1 80 1 1 0 0 0 1 1 4 1 1 3 2 1 1675
3 4 1 1 1 1 1 80 1 1 1 1 1 1 1 1 1 1 1 1334
1 1 1 1 2 1 1 5 80 6 1 1 1 4 2 2 0 2 1 8 1319
1 1 5 1 1 1 1 1 1 80 1 1 1 1 1 1 1 1 1 1 1291
0 1 1 1 1 1 1 3 5 1 80 1 1 1 1 1 1 1 0 1 1052
1 0 1 4 1 0 1 2 1 0 1 80 1 1 1 1 6 1 1 1 960
1 1 1 20 1 3 1 1 1 1 1 1 80 1 1 1 1 1 1 1 1 1195
1 1 1 1 1 1 1 6 1 1 1 4 1 80 1 1 -4 1 1 4 1 861
1 1 1 6 1 1 1 1 1 1 1 1 1 1 80 1 1 1 1 0 768
1 1 1 1 7 1 1 1 1 2 1 6 1 1 4 80 1 1 1 0 774
1 1 1 6 1 1 0 1 1 0 0 1 1 1 3 1 80 1 1 1 588
1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 80 1 1 433

Quest'opera è il risultato del connubio di materiale proveniente dalle lezioni (appunti e materiale fornito dal docente), materiale proveniente dal web (pdf di corsi di studio di altri atenei e codici provenienti da Forum e Siti Web quali Unifacile Cplusplus) e di un' attenta revisione del codice sorgente da parte dei candidati.

Bianchini Giovanni

Carfagna Lorenzo

Sabino Nicola

Gjini Skerdi