Università degli studi di Cagliari

Corso di Laura Specialistica in Ingegneria Elettrica ed Elettronica

TESINA DI CALCOLO NUMERICO 2

Oggetto:
ANALISI DEI METODI DI RISOLUZIONE DEI SISTEMI LINEARI

Docente
GIUSEPPE RODRIGUEZ

Studenti
ROBERTO SECHI matr. 36202
MAURO SANNAIS matr. 35793

1. Introduzione

Nel seguente elaborato vengono analizzati diversi metodi di risoluzione dei sistemi lineari. L'analisi valuta gli errori commessi nella risoluzione mediante l'utilizzo di metodi diretti e iterativi, e il tempo impiegato da ciascun algoritmo per il calcolo della soluzione. Verranno riportati inoltre grafici ottenuti mediante l'utilizzo del software Matlab, nei quali si evidenziano le differenze tra le soluzioni degli algoritmi implementati.

1.1 Sistemi lineari perturbati

Si consideri un sistema lineare $Ax = b$ perturbato sia nei dati che nei termini noti:

$$(A + \delta A)(x + \delta x) = b + \delta b$$

L'eccessiva perturbazione della matrice A potrebbe renderla singolare, pertanto si ha un vincolo sul δA: $\quad \|\delta A\| < \dfrac{1}{\|A^{-1}\|}$.

Il condizionamento del sistema nel caso più generale è:

$$\frac{\|\delta x\|}{\|x\|} \le \frac{K(A)}{1 - K(A)\dfrac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right)$$

Con la presenza di δA il condizionamento peggiora.

Per risolvere sistemi lineari perturbati esistono diversi metodi di risoluzione:

- diretti
- iterativi.

I metodi diretti analizzati in questa tesina sono quello di Gauss, di Householder e di Givens, mentre i metodi iterativi analizzati sono quelli di Jacobi e di Gauss-Seidel.

2. Metodi diretti

2.1 Metodo di Gauss

Il metodo di Gauss consente di trasformare un generico sistema lineare Ax=b in un sistema triangolare superiore ad esso equivalente, che può essere risolto con l'algoritmo di sostituzione all'indietro. L'algoritmo di Gauss è applicabile solo se tutti gli elementi pivot a_{kk} per $k = 1,...,n-1$ risultano essere non nulli. Questo è vero per matrici simmetriche definite positive o diagonalmente dominanti non singolari, ma non per tutte le matrici invertibili. Infatti si può dimostrare che se A è non singolare, esiste al passo k almeno un indice l compreso tra k ed n tale che $a_{lk}^{(k)} \neq 0$. Sarà quindi sufficiente permutare le righe k ed l per evitare che l'algoritmo si blocchi.

In un generico passo k dell'algoritmo di Gauss la matrice elementare di Gauss è strutturata in questo modo: $M_k = I - m_k e_k^T$

$$M_k = \begin{bmatrix} 1 & 0 & . & . & . & . & . & . & 0 \\ 0 & 1 & . & & & & & & . \\ . & . & . & 0 & & & & & . \\ . & & . & 1 & 0 & & & & . \\ . & & & 0 & 1 & 0 & & & . \\ . & & & & -m_{k+1,k} & 1 & . & & . \\ . & & & & . & 0 & . & . & . \\ . & & & & . & & . & 1 & 0 \\ 0 & . & . & . & -m_{n,k} & . & . & 0 & 1 \end{bmatrix}$$

La matrice elementare di Gauss M_k moltiplicata per A costituisce il passo k-esimo dell'algoritmo di Gauss: M_k lascia invariate tutte le prime k righe della matrice A, le successive le sottrae tramite un opportuno coefficiente che varia per ogni riga dalla k+1-esima alla n-esima.

Esprimendo A come un vettore colonna i cui elementi a^i sono dei vettori riga:

$$\underline{a^i} = (a_{i1}, a_{i2}, ..., a_{in})$$

$$M_k \cdot A = \begin{bmatrix} & & \\ & M^{(k)} & \\ & & \end{bmatrix} \cdot \begin{bmatrix} \underline{a^1} \\ \underline{a^2} \\ \\ \underline{a^k} \\ \\ \\ \underline{a^n} \end{bmatrix} = \begin{bmatrix} \underline{a^1} \\ \underline{a^2} \\ \\ \underline{a^k} \\ \underline{a^{k+1}} - m_{k+1,k}\underline{a^k} \\ \\ \underline{a^n} - m_{n,k}\underline{a^k} \end{bmatrix}$$

Se al posto dei coefficienti $m_{k+1,k}$ si sostituiscono i coefficienti $m_{ik} = \frac{a_{ik}}{a_{kk}}$, si ottiene la generalizzazione dell'algoritmo di Gauss.

Prendendo in considerazione tutte le n-1 matrici elementari di Gauss il loro prodotto è una matrice triangolare inferiore:

$$U = M_{n-1} \cdot M_{n-2} \cdot \ldots \cdot M_2 \cdot M_1 \cdot A$$

$$U = L^{-1} \cdot A$$

$$L \cdot U = A$$

ciò significa che l'algoritmo di Gauss ammette una fattorizzazione LU, quindi $Ax = b$ diventa $Ux = c$.

Per il condizionamento del sistema vale

$$K_\infty(U) \le 4^{n-1} K_\infty(A),$$

questo in generale cresce, ma con l'ipotesi di applicare il pivoting, non va oltre il valore dettato dalla precedente maggiorazione. Qualora $K_\infty(U)$ fosse proprio pari a $4^{n-1} K_\infty(A)$ sarebbe un disastro, dunque il pivoting si applica sempre anche perché applicare Gauss senza pivoting non da risultati attendibili.

La fattorizzazione LU non è adatta a problemi mal condizionati.

Nell'algoritmo di Gauss con pivoting totale:

- applicare il pivoting può accrescere la stabilità numerica, le due strategie di pivoting (parziale e totale) producono risultati sostanzialmente equivalenti;
- il pivoting totale è consigliato quando l'algoritmo di Gauss è applicato a sistemi di grandi dimensioni.

Di seguito viene riportato l'algoritmo utilizzato per le simulazioni effettuate su Matlab:

2.1.1 Algoritmo di Gauss con Pivoting totale

```
function [t,q,A1,b1]=GaussSIM(A,b)

n=length(b);
q=[1:n];

tic
for k=1:n-1
    % Localizzazione max matrice A indice riga (iv(jv)) indice colonna (jv)
    [v,iv]=max(abs(A(k:n,k:n)));
    [u,jv]=max(v);
    % Scambio righe e termini noti
    Aperm=A(iv(jv)+k-1,:);
    A(iv(jv)+k-1,:)=A(k,:);
    A(k,:)=Aperm;
    bperm=b(iv(jv)+k-1);
    b(iv(jv)+k-1)=b(k);
```

```
    b(k)=bperm;
    % Scambio colonne e memorizzazione scambi
    Aperm=A(:,jv+k-1);
    A(:,jv+k-1)=A(:,k);
    A(:,k)=Aperm;
    qperm=q(jv+k-1);
    q(jv+k-1)=q(k);
    q(k)=qperm;
    % Algoritmo di Gauss
    m=0;
    for i=k+1:n
        m=A(i,k)/A(k,k);
        for j=k+1:n
            A(i,j)=A(i,j)-m*A(k,j);
        end
        A(i,k)=0;
        b(i)=b(i)-m*b(k);
    end
end
t=toc;

A1=A;
b1=b;
```

2.2 Metodo di Householder

Esistono casi in cui la fattorizzazione LU non si può utilizzare (matrici singolari, rettangolari) o è preferibile non usare (come nel caso di sistemi mal condizionati). In questi casi si utilizza un altro tipo di fattorizzazione: la fattorizzazione QR che trasforma un sistema lineare in un uno triangolare superiore ad esso equivalente.

La fondamentale differenza con la fattorizzazione LU è che il nuovo sistema è dotato dello stesso numero di condizionamento del sistema originale.

Nel presente documento verranno analizzati diversi metodi che permettono di ottenere una fattorizzazione QR:

- Householder
- Givens.

Il metodo di Householder è il più famoso ed è anche il migliore, infatti confrontando la complessità dell'algoritmo di Gauss con la complessità dei metodi di fattorizzazione QR, si ha:

Metodo	Ordine
Gauss	$O\left(\frac{n^3}{3}\right)$
Householder	$O\left(2\cdot\frac{n^3}{3}\right)$
Givens	$O\left(4\cdot\frac{n^3}{3}\right)$

Nonostante l'algoritmo di Householder abbia complessità superiore a Gauss, come già accennato precedentemente, il vantaggio nell'utilizzarlo si evidenzia in termini di condizionamento.
Il sistema Ax=b, a seguito della fattorizzazione QR diventa

$$\begin{cases} Qy = b \\ Rx = y \end{cases}$$

e per il suo condizionamento si può scrivere

$$K_2(R) = K_2(A)$$

Nell'analizzare la matrice elementare di Householder si nota la stessa costruzione della matrice elementare di Gauss.

$$H = I - 2w \cdot w^T \text{ con } w \in \Re^n, \ \|w\|_2 = 1$$

La differenza con quest'ultima consiste nella validità delle seguenti proprietà:

- $H = H^T$ è simmetrica
- $H^T H = I$ è ortogonale

È possibile determinare il vettore w tale che sia:

$$Hx = ke_1$$

dove x è un vettore dato in ingresso, k è una costante $k \in \Re$ ed e_1 è il primo versore della base canonica di $\Re^n$.
Per le proprietà di H è possibile scrivere:

$$\|Hx\|_2 = \|ke_1\|_2 \quad \rightarrow \quad \|x\|_2 = |k| = \sigma$$

e per l'ipotesi di $\|w\|_2 = 1$, applicando la definizione di H, si trova il vettore w:

$$w = \frac{x - ke_1}{2(w^T x)} = \frac{x - ke_1}{\|x - ke_1\|}$$

Mediante opportuni passaggi si ottiene:

$$\|x - ke_1\|_2 = \sqrt{2\sigma(\sigma + |x_1|)}$$

Noto il vettore w si può costruire la matrice di Householder. In realtà la matrice H può essere usata senza essere calcolata, serve soltanto per essere moltiplicata a dei vettori, non si tiene quindi in memoria; ciò che si conserva è soltanto il vettore w, questo rende l'algoritmo veloce.
Applicando il metodo di Householder si moltiplica la matrice H per una A scritta in questa forma: $A = [\underline{a}_1 \quad \underline{a}_2 \quad . \quad . \quad \underline{a}_n]$, i cui elementi sono vettori colonna.
Al passo 1 si ha:

$$H_1 : H_1 a_1^{(1)} = k_1 e_1$$

$$A^{(2)} = H_1 A^{(1)} = \begin{bmatrix} k_1 e_1 & H_1 a_2^{(1)} & . & . & H_1 a_n^{(1)} \end{bmatrix} \rightarrow \begin{bmatrix} k_1 & * & . & * \\ 0 & * & . & * \\ . & . & . & . \\ 0 & * & . & * \end{bmatrix} \rightarrow \begin{bmatrix} k_1 & \underline{v}^T \\ \underline{0} & \hat{A}^{(2)} \end{bmatrix}$$

Al secondo passo si vogliono mettere degli zeri nella colonna 2 di A senza però alterare il resto della matrice già modificata. Il trucco è passare anziché alla 2° colonna di $A^{(2)}$, passare alla 1° di $\hat{A}^{(2)}$ e orlare la matrice H_2 che assumerà questa forma:

$$H_2 = \begin{bmatrix} 1 & \underline{0}^T \\ \underline{0} & \hat{H}_2 \end{bmatrix}$$

$$A^{(3)} = H_2 A^{(2)} = \begin{bmatrix} 1 & \underline{0}^T \\ \underline{0} & \hat{H}_2 \end{bmatrix} \cdot \begin{bmatrix} k_1 & \underline{v}^T \\ \underline{0} & \hat{A}^{(2)} \end{bmatrix} = \begin{bmatrix} k_1 & \underline{v}^T \\ \underline{0} & \hat{H}_2 \hat{A}^{(2)} \end{bmatrix}$$

$\hat{H}_2 \hat{A}^{(2)}$ avrà una colonna di zeri

$$A^{(3)} = \begin{bmatrix} k1 & * & * & . & * \\ 0 & k2 & * & . & * \\ . & 0 & * & . & * \\ . & . & . & & . \\ 0 & 0 & * & . & * \end{bmatrix} = \begin{bmatrix} A_{11}^{(3)} & A_{12}^{(3)} \\ \underline{0} & \hat{A}^{(3)} \end{bmatrix}$$

Al passo i-esimo dell'algoritmo si ha:

$$H_i = \begin{bmatrix} I_{i-1} & \underline{0}^T \\ \underline{0} & \hat{H}_i \end{bmatrix}$$

$$A^{(i+1)} = H_i A^{(i)} = \begin{bmatrix} I_{i-1} & 0^T \\ 0 & \hat{H}_i \end{bmatrix} \cdot \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ 0 & \hat{A}_{22}^{(i)} \end{bmatrix} = \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ 0 & \hat{H}_i \hat{A}^{(i)} \end{bmatrix}$$

infine si arriva all'ultimo passo per i=n-1, dove la matrice $A^{(n)}$ è una matrice triangolare superiore:

$$R = A^{(n)}$$

$$A^{(n)} = H_{n-1} \cdot A^{(n-1)} = H_{n-1} \cdot H_{n-2} \cdot A^{(n-2)} = H_{n-1} \cdot H_{n-2} \cdot \ldots \cdot H_1 \cdot A$$

il prodotto $H_{n-1} \cdot H_{n-2} \cdot \ldots \cdot H_1 = Q^T$ è un prodotto tra matrici ortogonali. Moltiplicando per Q entrambi i membri si ottiene:

$$QR = A$$

che è proprio la fattorizzazione QR cercata.

Di seguito vengono riportati gli algoritmi utilizzati per le simulazioni effettuate su Matlab:

- il primo è l'algoritmo di Householder;
- il secondo è l'algoritmo per la costruzione della matrice di Householder.

2.2.1 Algoritmo di Householder

```
function [t,Q,R]=HouseSIM(A,b)
% Esegue la fattorizzazione QR di una matrice A fornita in ingresso
n=length(b);
Q=eye(n);

tic
H1=Householder(A(:,1));
A1=H1*A;
Q=Q*H1;
for i=2:n-1
    Horl=Householder(A1(i:n,i));
    H=[eye(i-1),zeros(i-1,n-i+1);zeros(n-i+1,i-1),Horl];
    A1=H*A1;
    Q=Q*H;
end
t=toc;
R=A1;
```

2.2.2 Matrice di Householder

```
function Horlata=Householder(x);

sigma=norm(x,2);
k=-sign(x(1))*sigma;
beta=sigma*(sigma+abs(x(1)));
v=x-k*eye(size(x,1),1);
H1=1/beta*v*v';
Horlata=eye(size(x,1))-H1;
```

2.3 Metodo di Givens

Il metodo di Givens è un altro metodo che permette di ottenere la fattorizzazione QR. A differenza del metodo di Householder, che utilizza una matrice di riflessione, quello di Givens ne utilizza una di rotazione piana su $\Re^n$ fatta in questo modo:

- matrice elementare di Givens in dimensione $\Re^n$

$$G_{ij} = \begin{bmatrix} 1 & & & & & & \\ & . & & & & & \\ & & c & . & s & & \\ & & . & & . & & \\ & & -s & . & c & & \\ & & & & & . & \\ & & & & & & 1 \end{bmatrix}$$

Essenzialmente è una matrice identità con due righe diverse: la riga i e la riga j. I coefficienti della matrice godono della seguente proprietà: $c^2 + s^2 = 1$

L'operazione elementare che compie su un vettore è quella di cambiare due componenti.

$$y = G_{ij}x = \begin{bmatrix} 1 & & & & & & \\ & . & & & & & \\ & & c & . & s & & \\ & & . & & . & & \\ & & -s & . & c & & \\ & & & & & . & \\ & & & & & & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_i \\ x_j \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 \\ cx_i + sx_j \\ -sx_i + cx_j \\ x_n \end{bmatrix}$$

L'obiettivo è annullare selettivamente la componente j-esima e modificarne un'altra, lasciando le restanti invariate: $y_j = 0$

$$\begin{cases} -sx_i + cx_j = 0 \\ c^2 + s^2 = 1 \end{cases}$$ in questo modo si ricavano i coefficienti c ed s.

$$s = \frac{x_j}{\sqrt{x_i^2 + x_j^2}} \qquad c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}$$ sono simmetrici

Esiste un modo per ottimizzare la stabilità:

$$\begin{cases} \text{se } x_i < x_j \qquad t : 1 + \frac{x_i}{x_j} \qquad \text{e ricavo c da s} \\ \text{se } x_i > x_j \qquad t : 1 + \frac{x_j}{x_i} \qquad \text{e ricavo s da c} \end{cases}$$

La matrice elementare G_{ij} viene moltiplicata sistematicamente per la matrice di partenza A in modo da ottenere elementi nulli nel triangolo inferiore di A.

È essenziale procedere in ordine:

- a partire dalla prima riga della prima colonna, si usa il primo elemento per cambiare il secondo, il primo per cambiare il terzo fino ad aver fatto comparire lo zero nell'ultima riga della prima colonna
- successivamente si passa alla seconda colonna, la matrice G_{ij} agisce anche sulla prima, ma agisce su due zeri.

Si fanno tante matrici G_{ij} per tutti gli elementi del triangolo inferiore di A:

$$G_{n-1,n} \cdot \ldots \cdot G_{3,n} \cdot \ldots \cdot G_{3,4} \cdot G_{2,n} \cdot \ldots \cdot G_{2,3} \cdot G_{1,n} \cdot \ldots \cdot G_{1,2} \cdot A = R$$

ma $G_{n-1,n} \cdot \ldots \cdot G_{3,n} \cdot \ldots \cdot G_{3,4} \cdot G_{2,n} \cdot \ldots \cdot G_{2,3} \cdot G_{1,n} \cdot \ldots \cdot G_{1,2} = Q^T$ e moltiplicando entrambi i termini per Q si ha: $A = QR$.

Il numero di moltiplicazioni è pari a $\left(4 \cdot \frac{n^3}{3}\right)$, maggiore rispetto al numero di operazioni svolte dall'algoritmo di Householder, ma il vantaggio si nota quando la matrice di partenza A è sparsa, infatti non servono le G_{ij} degli elementi nulli.

Applicando l'algoritmo di Givens in maniera opportuna, si può arrivare a ridurre la complessità in modo sostanziale, con matrici sparse o strutturate.

Di seguito vengono riportati gli algoritmi utilizzati per le simulazioni effettuate su Matlab:

- il primo è l'algoritmo di Givens;
- il secondo è l'algoritmo per il calcolo dei coefficienti della matrice elementare di Givens.

2.3.1 Algoritmo di Givens

```
function [t,RG,C,S]=GivensSIM(A,b)
% Esegue la fattorizzazione QR di una matrice A fornita in ingresso
n=length(b);

tic
for k=1:n
    for i=k+1:n
        [c,s]=GIVROT(A(k,k),A(i,k));
        for j=k:n
            t=c*A(k,j)+s*A(i,j);
            A(i,j)=-s*A(k,j)+c*A(i,j);
            A(k,j)=t;
        end
        C(k,i)=c;
        S(k,i)=s;
    end
end
t=toc;
```

```
RG=A;

function QG=GiVMatriceQ(C,S,b)
% Genera la matrice Q con i coefficienti recuperati dalle matrici C ed S
n=length(b);
QG=eye(n);
for k=1:n
    for i=k+1:n
        G = eye(n);
        G(k,k) = C(k,i);
        G(i,i) = C(k,i);
        G(k,i) = -S(k,i);
        G(i,k) = S(k,i);
        QG = QG*G;
    end
end
```

2.3.2 Coefficienti di Givens

```
function [c,s]=GIVROT(xi,xj);

if(xj==0)
    c=1;
    s=0;
elseif(abs(xj)>abs(xi))
    t=xi/xj;
    z=sqrt(1+t^2);
    s=1/z;
    c=t*s;
else
    t=xj/xi;
    z=sqrt(1+t^2);
    c=1/z;
    s=t*c;
end
```

3. Metodi Iterativi

I metodi iterativi per la risoluzione di un sistema lineare

$$Ax = b$$

generano, a partire da un vettore iniziale $x(0)$, una successione di vettori $x(k)\ k = 1,2,.....$,che sotto opportune ipotesi converge alla soluzione del problema.

A differenza dei metodi diretti che modificano la matrice del sistema, quelli iterativi non richiedono una sua modifica e nemmeno la sua effettiva memorizzazione, è solo necessario poter accedere in qualche modo ai suoi elementi. Risultato: una convenienza per matrici di grandi dimensioni, specialmente se strutturate o sparse.

Nei metodi iterativi agli errori sperimentali e di arrotondamento si aggiungono gli errori di troncamento, derivanti dal fatto che il limite cercato deve essere necessariamente approssimato troncando la successione per un indice sufficientemente grande. Infine, è possibile ridurre di molto il tempo di elaborazione, eseguendo un minor numero di iterazioni in quei casi in cui non sia richiesta un'elevata accuratezza modificando il criterio di arresto.

I metodi che verranno analizzati in questo documento sono Jacobi e Gauss-Seidel.

Con i metodi lineari stazionari del 1° ordine $x^{(k+1)} = Bx^{(k)} + f$ si ha la risoluzione del sistema lineare

$$Ax = b$$

Di seguito sono riportate alcune definizioni utili inerenti il metodo in esame.

Per convergenza di un metodo si intende:

$$\left\|x^{(k+1)} - x^{(k)}\right\| = \left\|e^{(k)}\right\| \to 0 \qquad \forall\ x^{(0)} \text{ il metodo è globalmente convergente}$$

Al tendere di $k \to \infty$, $\left\|x^{(k+1)} - x^{(k)}\right\| \to 0$ definizione di Cauchy

Per consistenza di un metodo si intende:

$x^{(k)} = x \Rightarrow x^{(k+1)} = x$ se il metodo arriva alla soluzione, rimane li.

Un metodo può essere consistente ma non convergere: $x^{(k+1)} = x^{(k)}$; viceversa la convergenza implica la consistenza. La non consistenza implica la non convergenza.

La convergenza del metodo iterativo dipende esclusivamente dalla scelta della matrice B:

- **Th.1** Un metodo converge se in una qualsiasi norma considerata si ha: $\|B\| < 1$
- **Th.2** C.N.S. perché un metodo converga è che il raggio spettrale di B: $\rho(B) < 1$

Una strategia per costruire metodi iterativi lineari è quella di scomporre la matrice A del sistema $Ax = b$ in tre matrici:

$$A = D - E - F = \begin{bmatrix} a_{11} & 0 & . & 0 \\ 0 & a_{22} & . & . \\ . & . & . & 0 \\ 0 & . & 0 & a_{nn} \end{bmatrix} - \begin{bmatrix} 0 & . & . & 0 \\ -a_{2,1} & . & & . \\ & & . & . \\ -a_{n,1} & & -a_{n,n-1} & 0 \end{bmatrix} - \begin{bmatrix} 0 & -a_{1,2} & & -a_{1,n} \\ . & . & & \\ . & & . & -a_{n-1,n} \\ 0 & . & . & 0 \end{bmatrix}$$

$$A = P - N$$

con $\det(P) \neq 0$

$$(P - N)x = b \qquad Px = Nx + b$$

questo poi si trasforma arbitrariamente in un metodo iterativo:

$x^{(k+1)} = P^{-1}Nx^{(k)} + P^{-1}b$ che è pari a $x^{(k+1)} = Bx^{(k)} + f$

Si può scrivere che il metodo converge se $\rho(P^{-1}N) < 1$, inoltre il metodo è sicuramente consistente perché se si pone $x^{(k+1)} = x^{(k)} = x$ si ritrova la definizione vista precedentemente.

3.1 Metodo di Jacobi

Il metodo di Jacobi pone:

$$\begin{cases} P = D \\ N = E + F \end{cases}$$ da cui si ha: $x^{(k+1)} = D^{-1}(E + F)x^{(k)} + D^{-1}b$

che espresso in coordinate diventa:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)} \right], \qquad i = 1,...,n$$

Questa formula permette di calcolare le componenti di $x_i^{(k+1)}$ a partire da quelle di $x_i^{(k)}$ in qualsiasi ordine e indipendentemente l'una dall'altra. Il metodo di Jacobi è parallelizzabile.

Di seguito viene riportato l'algoritmo utilizzato per le simulazioni effettuate su Matlab:

3.1.1 Algoritmo di Jacobi

```
function [xn,nit]=JacobiOttimizzato(A,b)
%  JACOBI(A,b): Calcola la soluzione x del sistema
%        Ax=b, usando il metodo iterativo di Jacobi
% Estrae la diagonale principale di A
dd=diag(A);
% Costruisce la matrice diagonale come matrice sparsa
n=length(A);
dd=spdiags(dd,0,n,n);
% Usa b come stima iniziale
xvecchio=b;
```

```
% Stima del numero massimo di iterazioni
nmax=length(dd)^2;
% Algoritmo J
for n=1:nmax
    xn=dd\((dd-A)*xvecchio+b);
    % Criterio d'arresto
    residuo=norm(A*xn-b);
    differenza=norm(xn-xvecchio);
    if (residuo<=eps*norm(b)|differenza<=eps*norm(xvecchio))
        nit=n;
        return
    else
        xvecchio= xn;
    end
end
nit=nmax;
```

3.2 Metodo di Gauss-Seidel

Il metodo di Gauss-Seidel pone:

$$\begin{cases} P = D - E \\ N = F \end{cases} \qquad \text{da cui si ha:} \qquad x^{(k+1)} = \left((D-E)^{-1} \cdot F\right)x^{(k)} + (D-E)^{-1}b$$

che espresso in coordinate diventa:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} \right], \qquad i = 1,...,n$$

Questa formula permette di calcolare le componenti di $x_i^{(k+1)}$ risolvendo un sistema triangolare inferiore. Il metodo di Gauss-Seidel a differenza del metodo di Jacobi non è parallelizzabile, però in molte occasioni converge più velocemente di quest'ultimo e con un minor numero di iterazioni.

Non è detto che i metodi di Gauss-Seidel e Jacobi siano direttamente applicabili a una matrice A, perché potrebbe avere degli zeri in diagonale, quindi si dovrebbe fare un preventivo scambio di qualche riga.

Per la convergenza valgono i seguenti:

- **Th.1** Se A è strettamente diagonale dominante oppure irriducibilmente diagonalmente dominante, allora i due metodi J e G-S convergono.
- **Th.2** Se A è definita positiva il metodo G-S converge.

Di seguito viene riportato l'algoritmo utilizzato per le simulazioni effettuate su Matlab:

3.2.1 Algoritmo di Gauss-Seidel

```
function [xn,nit]=GaussSeidelOtt(A,b)
%  GaussSeidelOtt (A,b): Calcola la soluzione x del sistema
%        Ax=b, usando il metodo iterativo di Gauss-Seidel

% Costruisce la matrice B ed N
B=tril(A);
N=-triu(A,1);
% Usa b come stima iniziale
xvecchio=b;
% Stima del numero massimo di iterazioni
nmax=length(b)^2;
% Algoritmo GS
for n=1:nmax
    xn=B\(N*xvecchio+b);
    % Criterio d'arresto
    residuo=norm(A*xn-b);
    differenza=norm(xn-xvecchio);
    if (residuo<=eps*norm(b)|differenza<=eps*norm(xvecchio))
        nit=n;
        return
    else
        xvecchio= xn;
    end
end
nit=nmax;
```

4. Simulazioni

In questo paragrafo sono riportati i risultati sperimentali delle simulazioni condotte col software MATLAB, in particolare si è proceduto differenziando l'analisi per i metodi diretti e per quelli iterativi:

- per i primi si è deciso di considerare come classi di matrici delle random di ordine n (con n variabile da 10 a 500) e delle Hilbert con ordine variabile da 2 a 14;
- per i metodi iterativi invece si è optato per un'unica classe di matrici, random diagonalmente dominanti, stavolta però studiando ciò che accade qualora
    - l'ordine della matrice n sia fisso e sia variabile un parametro f che rende la matrice più o meno diagonalmente dominante a seconda che, rispettivamente, il valore di f sia molto maggiore/minore dell'unità;
    - viceversa sia fisso il parametro f e vari la dimensione della matrice.

4.1 Metodi diretti

4.1.1 Matrici random con dimensione n variabile

Con questa classe di matrici la norma dell'errore ottenuto come differenza tra una soluzione campione e quella fornita dagli algoritmi di Gauss, Householder e Givens è stata calcolata in questa maniera:

Gauss

```
%Soluzione sistema dopo l'applicazione del nostro algoritmo di Gauss
xg1=A1\b1;
%Vettore soluzione riordinato con il vettore permutazione colonne q
xg1(q)=xg1;
%Norma dell'errore della differenza tra le due soluzioni
EG=norm(sol-xg1);
```

Householder

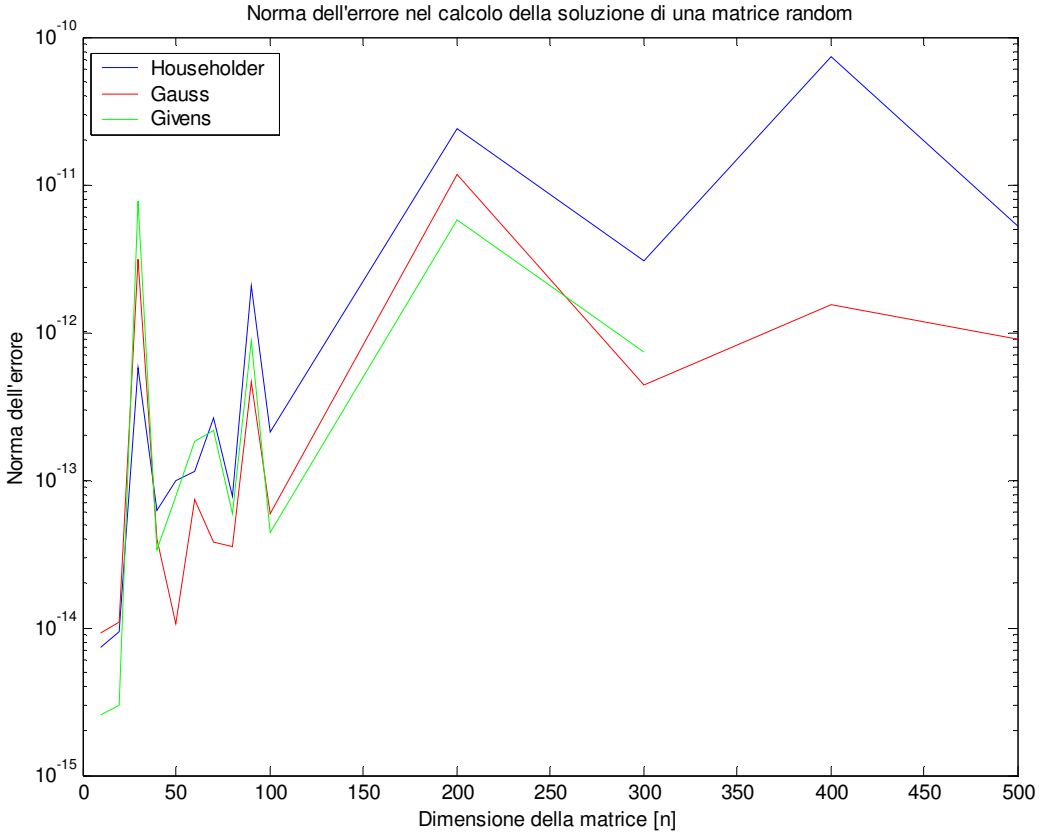
```
%Soluzione del sistema dopo l'applicazione del nostro algoritmo di Householder
xh1=R\(Q'*b);
%Norma dell'errore
EH=norm(sol-xh1);
```

Givens

```
%Soluzione del sistema dopo l'applicazione del nostro algoritmo di Givens
xgi1=RG\(QG'*b);
```

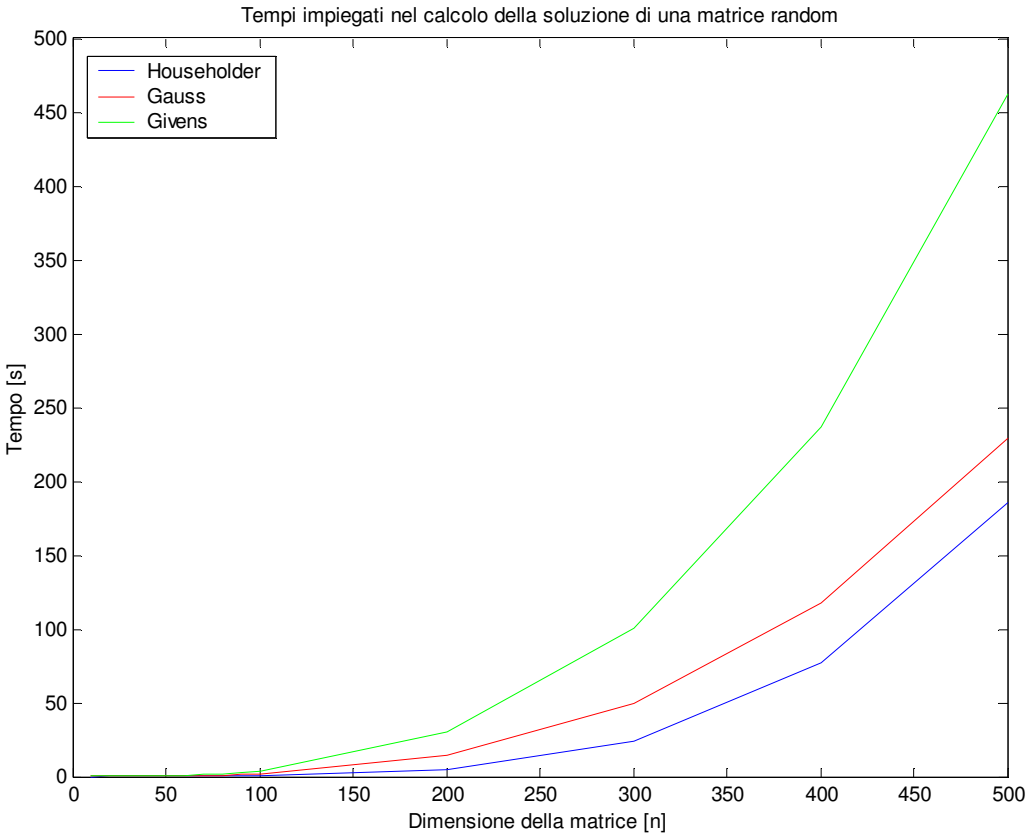
```
%Norma dell'errore
EGiv=norm(sol-xgi1);
```

Gli andamenti della norma dell'errore per Gauss, Householder e Givens sono i seguenti:



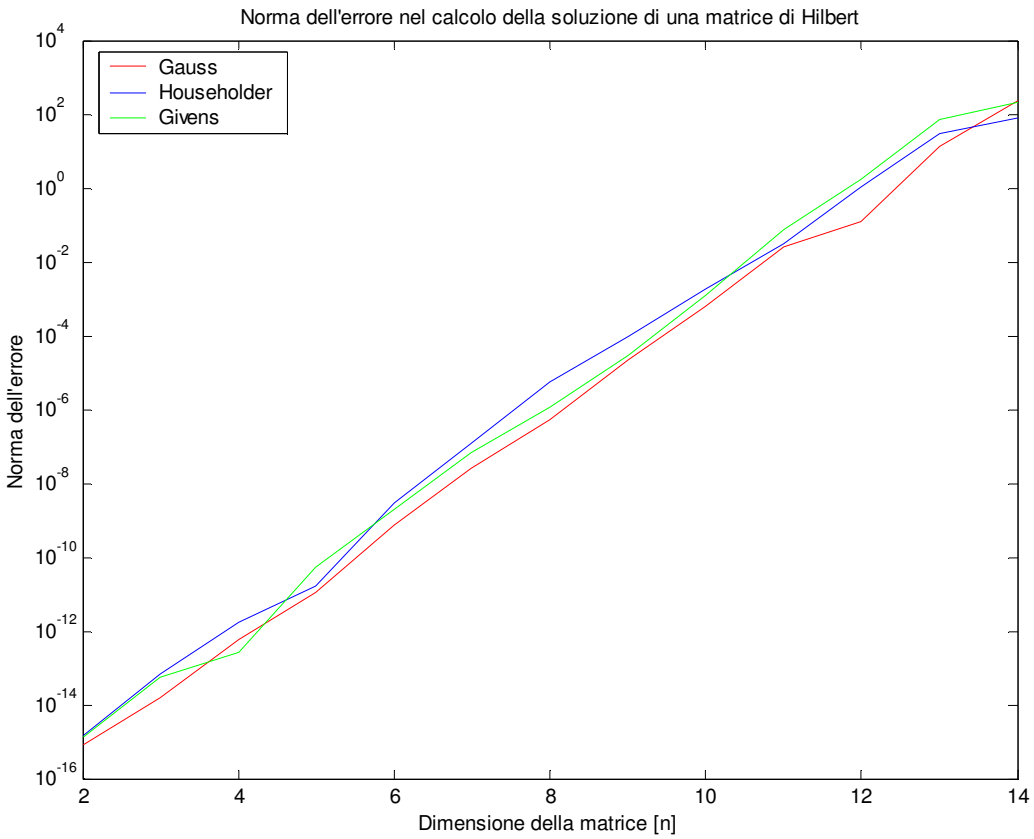
Dalla figura sovrastante si può notare che al crescere della dimensione della matrice la norma dell'errore cresce in maniera oscillatoria. Per l'algoritmo di Givens non è stato possibile effettuare la simulazione su matrici per n>300 a causa delle insufficienti risorse informatiche, tuttavia, sino a tale valore limite, segue l'andamento appena descritto ed è quindi abbastanza realistico pensare che farà lo stesso sino a n=500.

Sotto viene riportato il tempo impiegato dal calcolatore per implementare gli algoritmi di Gauss, Householder e Givens: il metodo più veloce risulta quello di Householder, seguito da Gauss e Givens.



4.1.2 Matrici di Hilbert

Con le matrici di Hilbert la situazione è ben diversa, come si sa esse hanno un condizionamento molto elevato che cresce esponenzialmente al crescere dell'ordine della matrice; di seguito sono riportati gli andamenti della norma dell'errore ottenuto come differenza tra una soluzione campione e quella fornita dagli algoritmi di Gauss, Householder e Givens riportati sopra.



Innanzitutto l'utilizzo di una scala logaritmica per le ordinate evidenzia la velocità di crescita della norma dell'errore, che cresce di un ordine di grandezza all'incirca per ogni incremento di un punto della dimensione della matrice. Questo è quindi l'effetto che ci si aspettava dal cattivo condizionamento delle matrici H.

4.2 Metodi iterativi

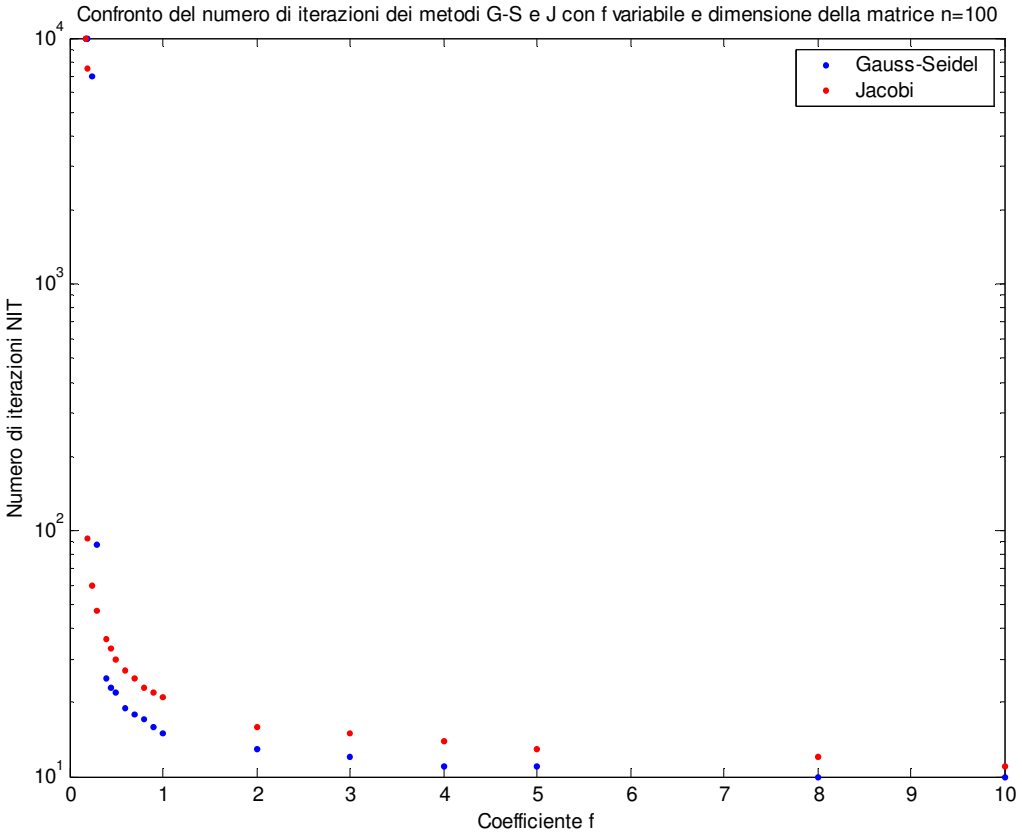
4.2.1 Matrici diagonalmente dominanti di dimensione n=100

Per generare questa classe di matrici sono state utilizzate le seguenti righe di comando:

```
function A=sparsap(A,b,f)
% Data una matrice A, un vettore di termini noti b e il fattore f in input, la
funzione restituisce una matrice fortemente diagonalmente dominante o no
n=length(b);
M=A-diag(diag(A));
s=abs(M)*ones(n,1);
A=M+diag(s)*f;
```

In esse come si desume dal titolo, n rimane costante e ciò su cui si agisce è il parametro f, a rigor di logica quindi si è deciso di mostrare l'andamento del numero di iterazioni al variare di f per i

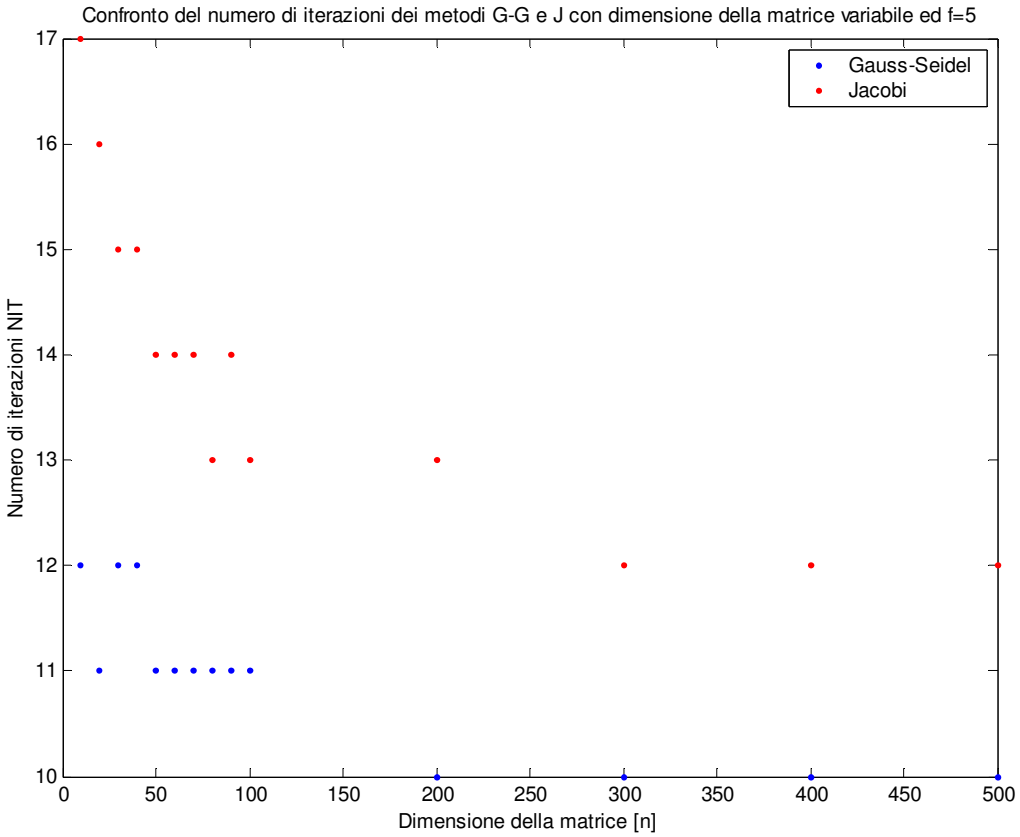
metodi di Jacobi e Gauss-Seidel. Il parametro f assume i seguenti valori: 10, 8, 5, 4, 3, 2, 1, 0.9, 0.8, 0.7, 0.6, 0.5, 0.45, 0.4, 0.3, 0.25, 0.2, 0.19, 0.18.



I risultati di sopra mostrano come al diminuire del fattore f (che corrisponde al rendere la matrice non diagonalmente dominante) aumenti il numero di iterazioni per giungere a convergenza di entrambi i metodi sino ad arrivare addirittura alla non convergenza per f sufficientemente piccoli.

4.2.2 Matrici diagonalmente dominanti con f=5

Nel presente paragrafo le simulazioni sono condotte facendo variare la dimensione n della matrice, è mostrato quindi l'andamento del numero di iterazioni al variare di n per i metodi di Jacobi e Gauss-Seidel. La dimensione n assume i seguenti valori: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500.



In questo caso i risultati evidenziano come all'aumentare di n diminuisca il numero di iterazioni per giungere a convergenza di entrambi i metodi: il metodo di Gauss-Seidel, a parità di n, converge in un numero di passi inferiore rispetto a Jacobi.