UNIVERSITA' DEGLI STUDI DI CAGLIARI

Facoltà di Ingegneria

Corso di Laurea Specialistica in

Ingegneria per l'Ambiente e il Territorio

TESINA DI CALCOLO NUMERICO

Analisi dell'errore nei metodi di risoluzione dei sistemi lineari

Docente

Prof. Giuseppe Rodriguez

Studente

Pirroni Nicola matr. 34769

Anno accademico

2007/2008

Indice

Introduzione

Nel seguente elaborato vengono analizzati alcuni metodi diretti per la risoluzione dei sistemi lineari. Sono stati valutati gli errori commessi nella fattorizzazione e nella risoluzione, implementando gli algoritmi in ambiente Matlab ed analizzando irisultati.

Capitolo 1 Sistemi lineari

Un sistema di equazioni lineari (o sistema lineare) è un insieme di equazioni lineari, che devono essere verificate tutte contemporaneamente: in altre parole, una soluzione del sistema è tale se è soluzione di tutte le equazioni. La soluzione è, quindi, l'insieme di valori $x_1 \ldots x_n$ che, sostituiti alle incognite, rende le equazioni delle identità.

Un sistema di equazioni lineari è il dato di un certo numero m di equazioni lineari in n incognite, e può essere scritto nel modo seguente:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2 \\ \vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n = b_m \end{cases}$$

dove x_1, ... ,x_n sono le incognite e i numeri a_{ij}, detti i coefficienti, sono elementi di un campo, ad esempio dei numeri reali o complessi. Anche i termini noti b_i sono elementi del campo. Una n-upla $(x_1, \ldots, x_n)$ di elementi nel campo è una soluzione se soddisfa tutte le m equazioni.

Usando le matrici ed il prodotto fra matrici e vettori si possono separare agevolmente i coefficienti, le incognite ed i termini noti del sistema, e scriverlo nel modo seguente:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & & & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

In modo molto più schematico, si scrive

$$Ax = b$$

dove:

$A = (ai_j)_{i,j=1}^{n}$ è la matrice dei coefficienti,;

$b = (b_1, ...b_n)$ il vettore dei termini noti;

$x = (x_1, ..., x_n)$ la soluzione.

Con questa rappresentazione, la soluzione del sistema lineare è determinata dalla:

$$x = A^{-1}b$$

Calcolare l'inversa della matrice dei coefficienti, a dispetto della semplicità con cui la formula è scritta, non è mai banale nei problemi di ingegneria. Infatti si è in presenza di problemi di tipo numerico che devono poter essere trattati su un calcolatore.

Caratteristica fondamentale per la risoluzione, è che il sistema lineare abbia una soluzione unica che dipenda con continuità dai dati e dalle perturbazione che su di esso sono introdotte dalla matematica e dagli algoritmi: sia in sostanza ben posto.
Un problema è ben posto se esso possiede, in un prefissato campo di definizione, una e una sola soluzione e questa dipende con continuità dai dati. In caso contrario, viene detto mal posto.

Quando non c'è dipendenza tra la perturbazione sui dati e quella sulla soluzione con continuità il problema è instabile. L'influenza che le perturbazioni hanno sul risultato è detta condizionamento ed è misurabile attraverso il numero di condizionamento. Quindi il problema deve essere anche ben condizionato.

Capitolo 2 Risoluzione dei sistemi lineari

2.1 Metodi diretti per la risoluzione di sistemi lineari

I sistemi di equazioni lineari sono i problemi numerici che si incontrano più spesso nelle applicazioni della Matematica. Per la loro risoluzione, a differenza della quasi totalità dei problemi non lineari, sono disponibili algoritmi finiti, i cosiddetti metodi diretti. Esistono numerosi metodi diretti, spesso basati su idee molto diverse tra loro, ma che condividono la stessa strategia di base. Questa consiste nel trasformare, mediante un algoritmo con un numero finito di operazioni, un sistema lineare generico in un sistema equivalente, ma dotato di una struttura particolare che ne renda più semplice la risoluzione. L'applicazione dei metodi diretti per la risoluzione di un sistema lineare tramite calcolatore avviene attraverso l'implementazione di un algoritmo.

Un algoritmo è una sequenza univoca di un numero finito di operazioni elementari che stabilisce come calcolare la soluzione di un problema, assegnati certi dati iniziali.

E' importante che l'algoritmo sia ben strutturato al fine di garantire una bassa propagazione dell'errore. E' possibile sviluppare un algoritmo stabile solo in presenza di un problema ben condizionato.

Esistono alcuni sistemi lineari aventi una struttura particolare che li rende risolvibili tramite algoritmi con una complessità computazionale inferiore rispetto al caso generale:

- Sistemi diagonali;
- Sistemi ortogonali;
- Sistemi triangolari.

2.1 Sistemi diagonali

Il caso più semplice si ha in presenza di una matrice dei coefficienti che sia di tipo diagonale, cioè

$$D = (d_1,...d_n)_{i=1}^{n}$$

Il sistema si presenta sotto la forma

$$Dx = b$$

Di fatto non è un sistema, ma *n* equazioni non accoppiate, la cui risoluzione è data dalla:

$$x_i = \frac{b_i}{d_i}$$

La complessità computazionale è pari a $o(n)$, mentre il condizionamento è

$$K = \frac{\max(d_i)}{\min(d_i)}$$

Ciò implica che il problema può essere malcondizionato ed esserci amplificazione degli errori con rischio di *overflow* se $\min(d_i) \langle\langle \max(d_i)$.

2.2 Sistemi ortogonali

In questo caso, la matrice dei coefficienti è ortogonale, cioè la trasposta è uguale alla inversa:

$$Q^T = Q^{-1}$$

Quindi il vettore delle soluzioni è calcolabile con:

$$x_i = \sum_{j=1}^{n} q_{ij}^T b_j = \sum_{j=1}^{n} q_{ji} b_j$$

La complessità computazionale è pari a $o(n^2)$, ed ha un buon condizionamento, con K=1.

2.3 Sistemi triangolari

I sistemi di tipo triangolare hanno la matrice dei coefficienti che può essere triangolare superiore od inferiore. Rispettivamente, sono rappresentati così:

$$Ux = b$$
$$Lx = b$$

Si consideri un sistema triangolare di tipo inferiore: si può subito trovare l'unica incognita della prima equazione:

$$x_1 = \frac{b_1}{l_{11}}$$

Questo valore può quindi essere sostituito nella seconda equazione e così via: ad ogni passo è possibile calcolare la variabile x in diagonale, fino alla x_n. L'algoritmo in questo caso è detto di

discesa o forward substitution. In presenza di una matrice triangolare superiore si procede allo stesso modo ma partendo dal basso(algoritmo di risalita o backward substitution).

Per entrambi le situazioni, la complessità computazionale è pari a $o(\frac{1}{2}n^2)$, e sono dei problemi che possono essere malcondizionati.

2.4 Algoritmo di Gauss o di triangolarizzazione

Con l'algoritmo di Gauss si entra nel caso generale di sistema lineare. Si basa sui principi di equivalenza dei sistemi lineari. Un sistema lineare resta equivalente se si eseguono le seguenti operazioni elementari:

- moltiplicazione di una equazione per uno scalare;
- sostituzione di una equazione con la somma della stessa con un'altra;
- scambio di equazione.

Attraverso queste operazioni, l'algoritmo di Gauss trasforma il sistema lineare in uno triangolare superiore. Il costo computazionale è pari a $o(\frac{n^3}{3})$, ed n^3 è l'ordine di grandezza massimo accettato per la risoluzione di sistemi lineari. Il metodo così presentato funziona solo per matrici diagonalmente dominanti per righe(purché non singolari) e per colonne, cioè per matrici con gli elementi diagonali non nulli e per matrici simmetriche definite positive. Può capitare che l'elemento diagonale sia molto piccolo, in tal caso si possono generare degli overflow. Di fatto, con questo metodo il condizionamento del sistema finale può aumentare in maniera significativa. Quest'ultimo problema può essere ovviato tramite il pivoting parziale. L'algoritmo di Gauss opera la fattorizzazione $A = LU$, cioè trasforma la matrice d'origine nel prodotto di due matrici triangolari. Quindi il sistema diventa:

$$LUx = b$$

ponendo $Ux = y$, si ottengono due sistemi triangolari facilmente risolvibili in cascata:

$$\begin{cases} Ly = b \\ Ux = y \end{cases}$$

Partendo da questa forma si arriva alla fattorizzazione A=LU=LDR, dove la U è uguale al prodotto di una matrice diagonale D per una matrice R triangolare superiore caratterizzata dall'avere in diagonale dei coefficienti pari a 1.

2.5 Algoritmo di Gauss con pivoting parziale

Con il pivoting parziale si elimina i rischio di overflow e si riduce la propagazione dell'errore praticamente a costo zero perché non si hanno operazioni in virgola mobile. Operando col pivoting, ad ogni passo l'algoritmo ricerca il maggiore elemento diagonale a_{kk} presente sotto diagonale. L'algoritmo di Gauss opera la fattorizzazione $PA = LU$ dove P è la matrice di permutazione, una matrice ortogonale che ha la funzione di permutare le colonne di A.

2.6 Fattorizzazione di Cholesky

La fattorizzazione di Cholesky, che deriva sempre dalla LU, è applicabile a matrici simmetriche definite positive e trasforma la matrice dei dati i questo modo:

$$A = R^T R$$

Il vantaggio consiste nell'avere un solo fattore, ed inoltre si dimezza la complessità computazionale di un mezzo rispetto all'algoritmo di Gauss $o(\frac{n^3}{6})$. Il sistema finale si presenta nella forma:

$$R^T Rx = b \Rightarrow \begin{cases} R^T y = b \\ Rx = y \end{cases}$$

2.7 Fattorizzazione QR

Una matrice, non necessariamente quadrata, può essere decomposta nel prodotto:

$$A = QR$$

dove Q è una matrice ortogonale ed R una matrice triangolare superiore delle stesse dimensioni di A. Il vantaggio principale è nel buon condizionamento del sistema finale:

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

In pratica permette di risolvere un sistema lineare senza peggiorare il condizionamento. Per i calcolo di QR è necessario calcolare la matrice di Householder, una matrice elementare definita come:

$$H = I - 2ww^T = I - \frac{1}{\beta} vv^T$$

La seconda espressione è preferibile perché non contiene operazioni in virgola mobile, ed in essa i termini sono così definiti:

- I = *matrice identità*;
- $\beta = \sigma(\sigma + |x_1|)$
- $\sigma = \|x\|$
- x= *primo vettore colonna della matrice A*
- $v = x - ke_1$
- $k = -sign(x_1)\sigma$
- e_1= *primo vettore della base canonica.*

La complessità computazionale è pari a $o(\frac{2}{3}n^3)$, il doppio dell'algoritmo di Gauss con pivoting, ma con il vantaggio di avere un condizionamento che cresce di meno.

Capitolo 3 Implementazione degli algoritmi

3.1 Script principale

Gli algoritmi precedentemente descritti sono stati implementati su matlab versione 6.5. Le fattorizzazioni sono richiamate come *functions* da 3 script principali che generano, tramite un loop in *i*, un numero di matrici variabile con *n* crescente:

1. *alg_random.m*: genera 80 matrici casuali con *n =5, 10,…,95, 200;*
2. *alg_hilb.m*: genera 9 matrici con $n = 3 \div 12$;
3. *alg_pascal.m*: genera 37 matrici con $n = 3 \div 40$.

Il primo algoritmo genera matrici composte da numeri casuali. Una matrice *random* moltiplicata per se stessa diventa simmetrica, caratteristica fondamentale per la riuscita degli algoritmi di fattorizzazione.

```
A=rand(i);              % definizione di una matrice %
A=A'*A;                 % positiva e simmetrica      %
```

Il secondo e terzo script generano delle matrici simmetriche che di base sono mal condizionate: il loop è più corto perché, anche per n piccolo, l'algoritmo si blocca in quanto la matrice diventa numericamente singolare.

```
A=hilb(i);              %  definizione matrice di Hilbert%

A=pascal(i);             %  definizione  matrice di Pascal %
```

Ad ogni matrice è imposto un vettore soluzione i cui elementi sono dei coefficienti tutti pari a 1, e si calcola il termine noto *b* del sistema lineare *Ax=b*. In tal modo è possibile avere un confronto tra la soluzione vera e quella trovata dopo la fattorizzazione della matrice dei coefficienti.

```
e=ones(i,1);            % il vettore soluzione%
b= A*e;                 % termine noto
```

Infine, all'interno del loop è calcolato anche il numero di condizionamento della i-esima matrice generata, poi memorizzato in un vettore:

```
K(k)=cond(A);                % CONDIZIONAMENTO DELLA MATRICE
```

All'interno del loop, sono presenti le chiamate alle diverse fattorizzazioni, che hanno come variabili di input A e b, ed in uscita il vettore soluzione x e i termini della fattorizzazione. Subito dopo la chiamata ci sono le istruzioni di memorizzazione della dimensione dell'errore sulla soluzione e sulla fattorizzazione. Ad esempio, per la fattorizzazione QR sarà:

```
[x,Q,R]=fattqr(A,b);        %chiamata alla fattorizzazione QR
eqr(k)=norm(A-Q*R);         % memorizzazione errore sulla fattorizzazione
axqr(k)=norm(x-e);          % memorizzazione errore sulla  soluzione
```

3.2 Function "fattlu"

Questa funzione esegue la triangolarizzazione della matrice secondo il metodo di Gauss, risolve il sistema triangolare superiore $Ux=b$ e restituisce x, L, U.

```
%%%%%%%%%%%%%%%triangolarizzazione%%%%%%%%%%%%%%%
for k=1:(n-1)
    for i=(k+1):n
        L(i,k)=U(i,k)/U(k,k);
        for j=(k+1):n
            U(i,j)=U(i,j)-L(i,k)*U(k,j);
        end
        b(i)=b(i)-L(i,k)*b(k);
        U(i,k)=0;
    end

end
```

3.3 Function "fattpalu"

Questa è molto simile alla precedente, contiene il loop del pivoting di colonna seguito da quello per la triangolarizzazione. Rispetto alla funzione precedente, questa restituisce anche la matrice di permutazione P.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%pivoting%%%%%%%%%%%%%%%%%
for w=1:n-1
    D=eye(n);
    t=abs(U(w,w));
    r=w;
    for q=w+1:n
        if abs(U(q,w))>t
            t=abs(U(q,w));
            r=q;
        end
    end

    if t<eps
        stop
    end

    for e=w:n
        t=U(w,e);
        U(w,e)=U(r,e);
        U(r,e)=t;
    end

       U(r,e)=t;
   end

   for e=1:n
       c=D(w,e);
       D(w,e)=D(r,e);
       D(r,e)=c;
   end
   P=D*P;

   t=b(w);
   b(w)=b(r);
   b(r)=t;
   %%%%%%%%%%%%%%triangolarizzazione%%%%%%%%%%%%%%%%%%%%

   for i=(w+1):n
       L(i,w)=U(i,w)/U(w,w);
       for j=(w+1):n
           U(i,j)=U(i,j)-L(i,w)*U(w,j);
       end
       b(i)=b(i)-L(i,w)*b(w);
       U(i,w)=0;
   end

end
```

3.4 Function chole1

Questa funzione esegue la fattorizzazione di Cholesky, quindi restituisce oltre che il vettore x, la matrice R.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%cholesky %%%%%%%%%%%%%%%%%%%%%%%%%

R(1,1)=sqrt(A(1,1));
for j=2:n
    for i=1:j-1
        s=0;
        for k=1:i-1
            s=s+(R(k,i)*R(k,j));
        end
        R(i,j)=(A(i,j)-s)/R(i,i);
    end
    s=0;
    for k=1:j-1
        s=s+(R(k,j)*R(k,j));
    end
    R(j,j)=sqrt(A(j,j)-s);

end
```

3.5 Function fattqr

L'ultima funzione chiamata, esegue una fattorizzazione di tipo QR, e restituisce Q, R ed x. Il primo passaggio di fattorizzazione, fuori da ogni loop calcola la prima matrice di Householder.

```
c=eye(n,1);
xx=R(1:n,1);
s=norm(xx,2);
kk=-sign(xx(1))*s;
B=s*(s+abs(xx(1)));
v=(xx-kk*c);
sH= Q-1/B*v*v';
R=sH*R;
Q=Q*sH;
```

Nel loop, per k che parte da 2, avvengono tutti i successivi passaggi.

```
for k=2:n-1
        G=eye(n+1-k);
        c=eye(n+1-k,1);
        xx=R(k:n,k);
        s=norm(xx,2);
        kk=-sign(xx(1))*s;
        B=s*(s+abs(xx(1)));
        v=(xx-kk*c);
        sH= G-1/B*v*v';

        a1=eye(k-1);
        a2=zeros(k-1,n-k+1);
        a3=zeros(n-k+1,k-1);
        H = [a1,a2;a3,sH];
        R=H*R ;
        Q=Q*H;

        clear c;
        clear xx;
end
```

Capitolo 4 Analisi dell'errore sulla fattorizzazione

Il confronto delle diverse fattorizzazione è stato fatto calcolando la norma dell'errore sulla fattorizzazione, mentre l'errore sul calcolo del sistema lineare è stato calcolato con la norma della differenza tra il vettore soluzione calcolato e quello dato:

- Fattorizzazione A=LU

```
ealu(k)= norm(A-L*U);
exlu(k)=norm(x-e);
```

- Fattorizzazione PA=LU

```
epalu(k)=norm(P*A-L*U);
expalu(k)=norm(x-e);
```

- Fattorizzazione di Cholesky

```
arr(k)=norm(A-R'*R);
exarr(k)=norm(x-e);
```

- Fattorizzazione QR

```
eqr(k)=norm(A-Q*R);
axqr(k)=norm(x-e);
```

4.1 confronto con gli algoritmi di Matlab

Una volta implementati gli algoritmi di fattorizzazione, sono stati confrontati con quelli di Matlab misurando ancora la norma sull'errore di fattorizzazione:

- Fattorizzazione A=LU

```
[LL,UU] = lu(A);
ealum(k)=norm(A-LL*UU);
```

- Fattorizzazione di Cholesky

```
T=chol(A);
aarrm(k)=norm(A-T'*T);
```

- Fattorizzazione QR

```
[Q,R]=qr(A) ;
eqrm(k)=norm(A-Q*R);
```

Dai grafici che seguono, si nota come la funzione *fattlu* genera un errore che sostanzialmente coincide con quello di Matlab, mentre la fattorizzazione *fattpalu* ha un andamento più variabile, con la curva che mediamente sta sopra quella relativa alla fattorizzazione LU senza pivoting (Figura 1). La funzione *chole1* ha addirittura un errore inferiore rispetto a quello relativo all'algortimo di Matlab, soprattutto per matrici con n>60, mostrando come esso sia potenzialmente migliorabile. La discontinuità nella curva è dovuta alla presenza in quel punto di un errore nullo (Figura 2). Per quanto riguarda la fattorizzazione QR, l'algoritmo implementato *fattqr* genera un errore che oscilla rispetto a quello causato dall'algoritmo di Matlab(Figura 3).

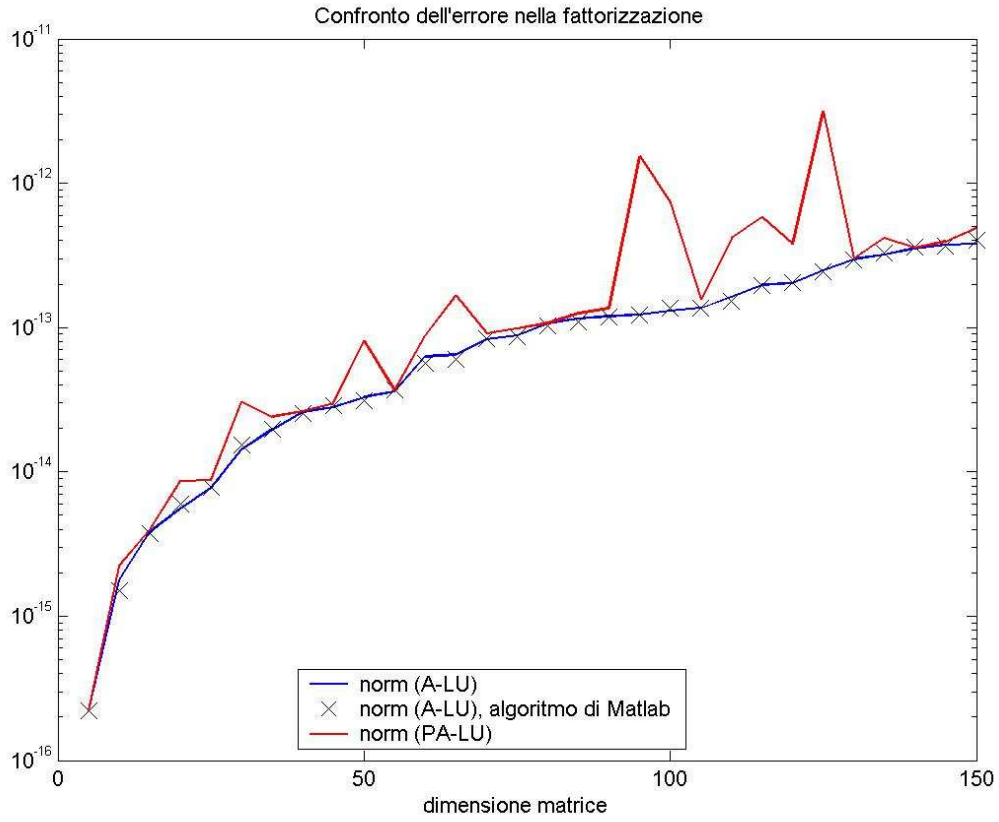

Figura 1

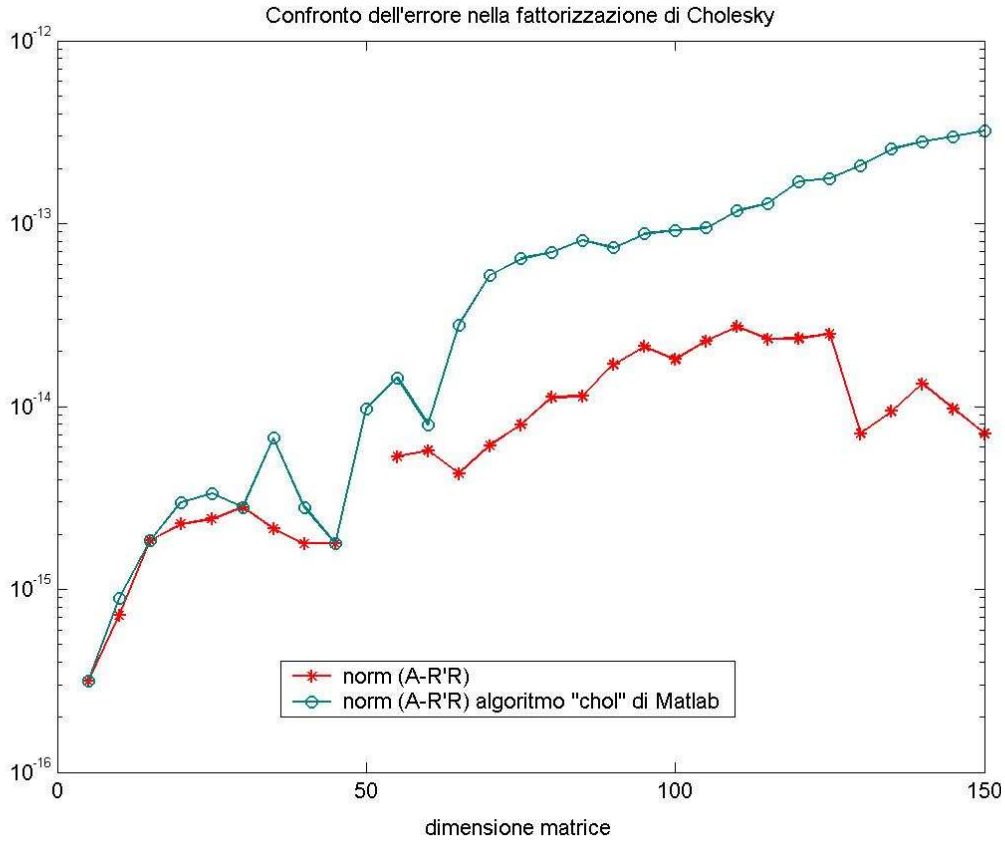

Figura 2

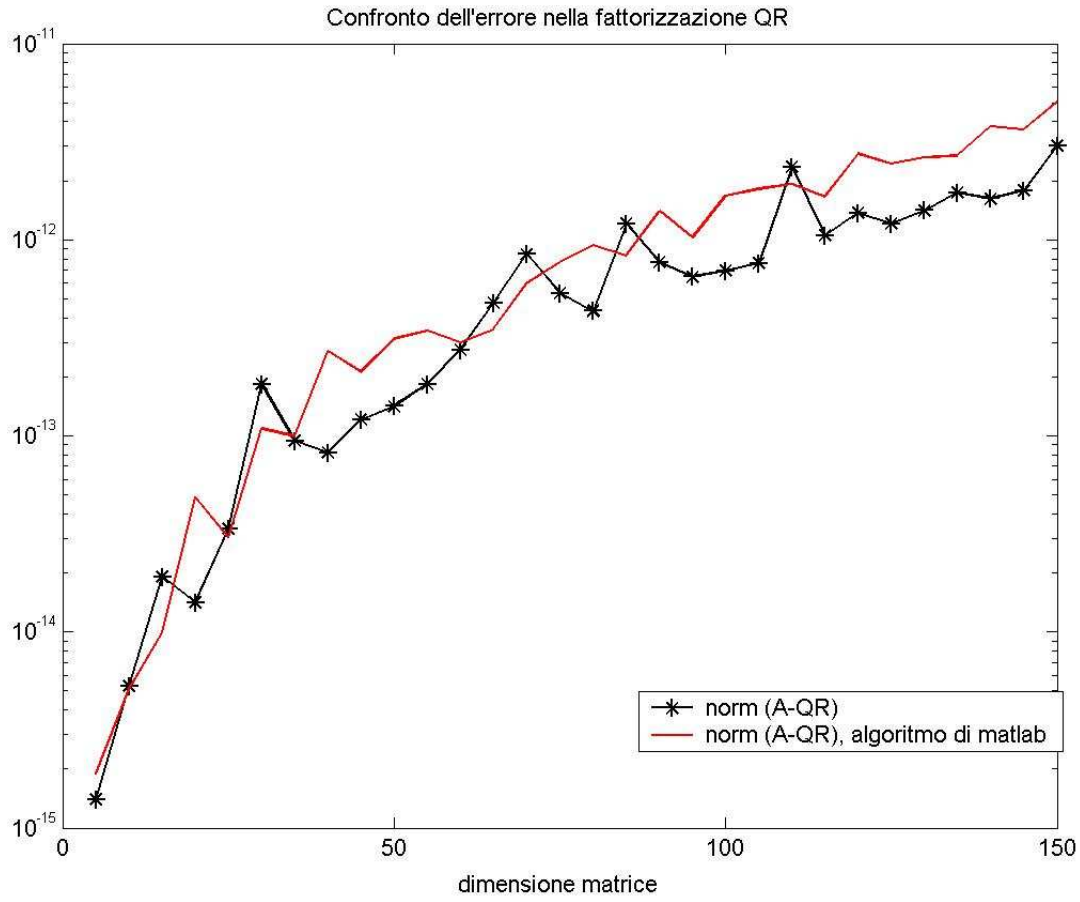

Figura 3

Di seguito sono stati utilizzati gli algoritmi implementati.

4.2 Fattorizzazione delle matrici casuali

Confrontando in uno stesso grafico gli errori dovuti alla fattorizzazione, si nota che i valori iniziali si attestano ad un'ordine di grandezza di 10^{-16} per poi crescere all'aumentare delle dimensioni della matrice. Alla fattorizzazione di Cholesky corrisponde la dimensione dell'errore più piccola: per *n* grande si attesta attorno ad un ordine di grandezza pari a 10^{-14}. La fattorizzazione che amplifica maggiormente l'errore è la QR (Figura 4).

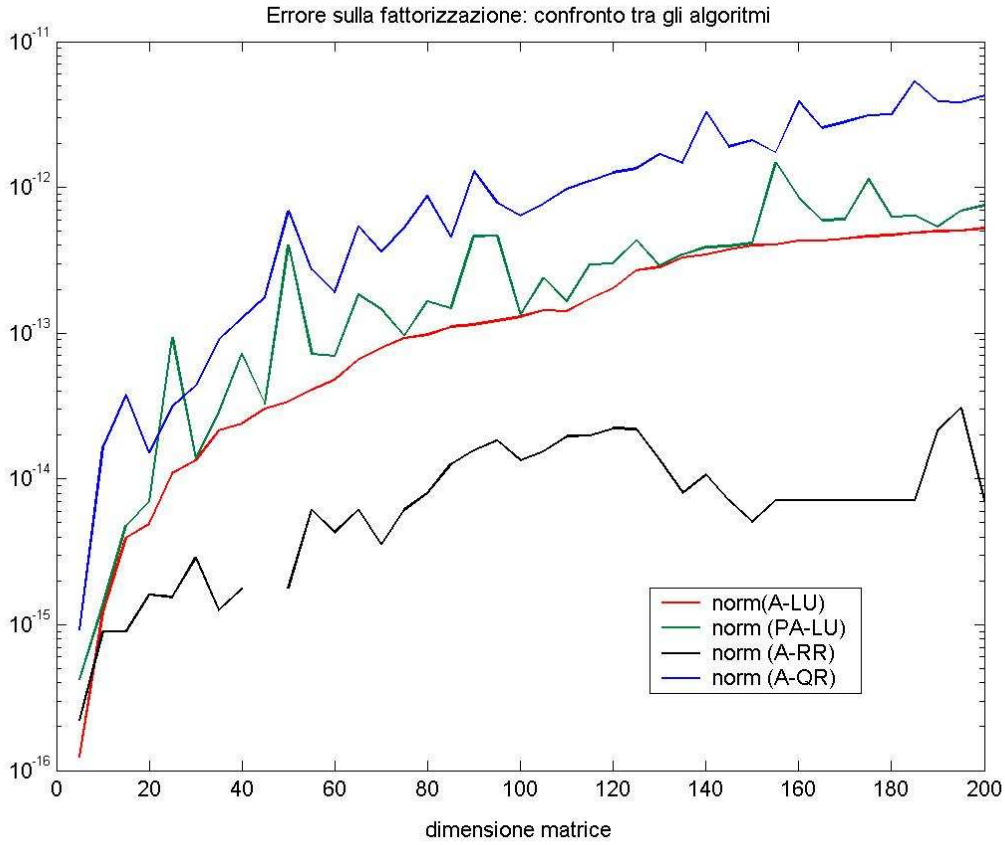

Figura 4

Nel grafico successivo si può vedere l'errore in relazione al condizionamento della matrice di partenza (Figura 5).

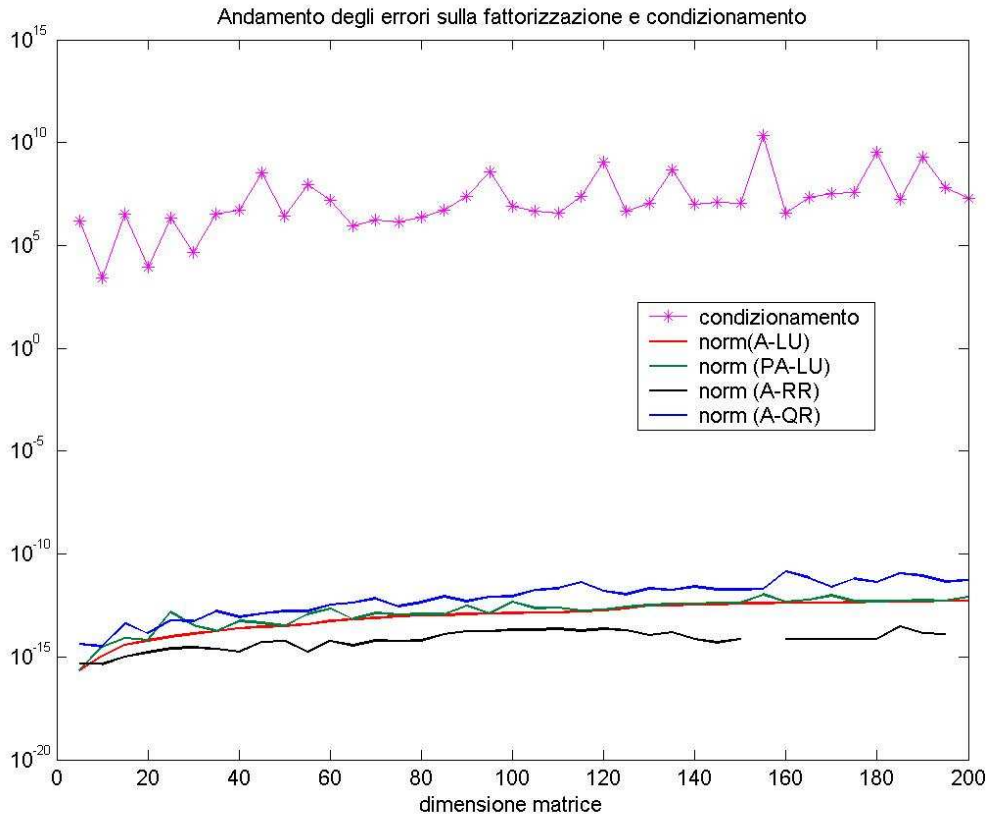

Figura 5

4.3. Fattorizzazione delle matrici di Hilbert

Analizzando il comportamento degli algoritmi sulle matrici mal condizionate di Hilbert per *n* fino a 12, si può notare che la fattorizzazione di Cholesky genera un errore nullo per $n = 3 \div 10$, mentre l'errore della fattorizzazione QR ha un andamento altalenante a partire da un'ordine di grandezza sopra le altre curve ma che poi segue un andamento costante attorno al valore di 10^{-15}. Anche l'algoritmo *fattlu* si dimostra abbastanza stabile, mentre *fattpalu* mostra una crescita esponenziale. Entrambi gli algoritmi di Gauss comunque sono poco adatti per le matrici di Hilbert, perché causano, per n >12, il blocco dello script perché la matrice diventa numericamente singolare (Figura 6).

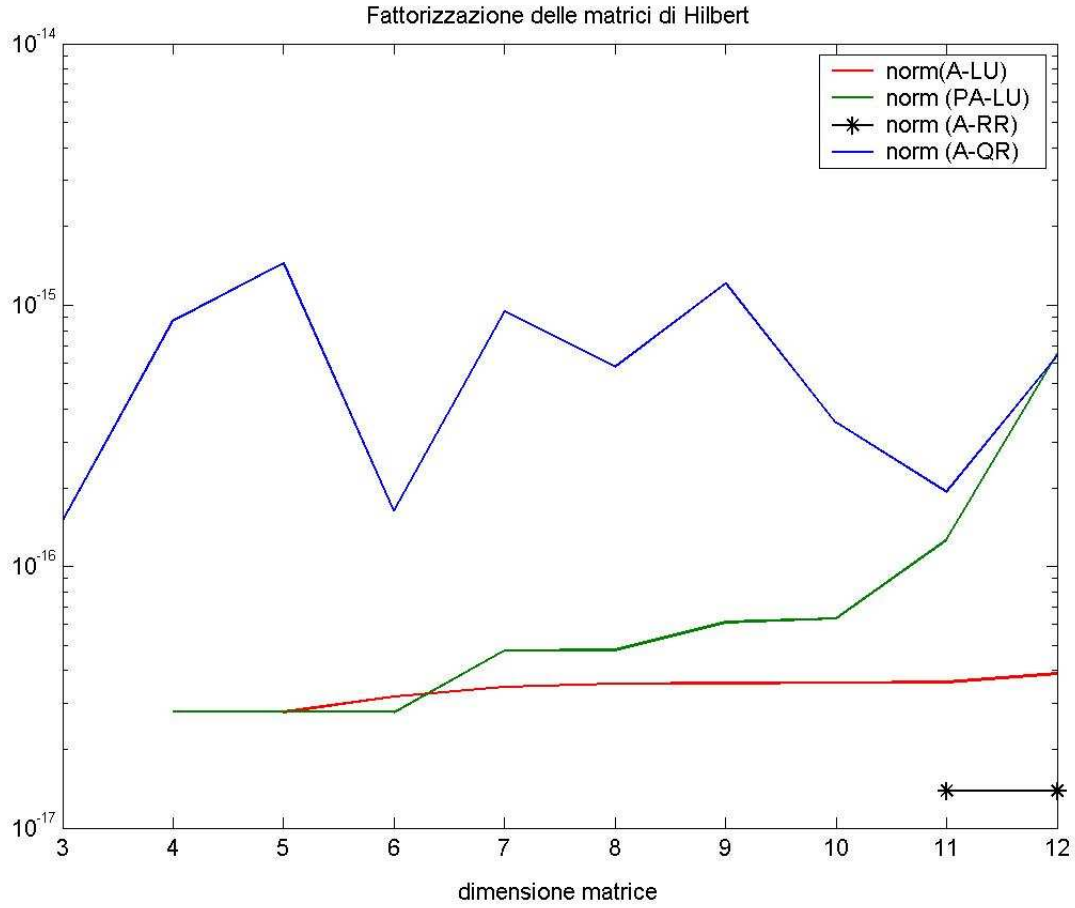

Figura 6

Si può vedere questo grafico anche in relazione al numero di condizionamento, che cresce inizialmente poi si stabilizza per $n \cong 15$ attorno a valori dell'ordine di grandezza pari a 10^{-19} (figura 7 e 8).

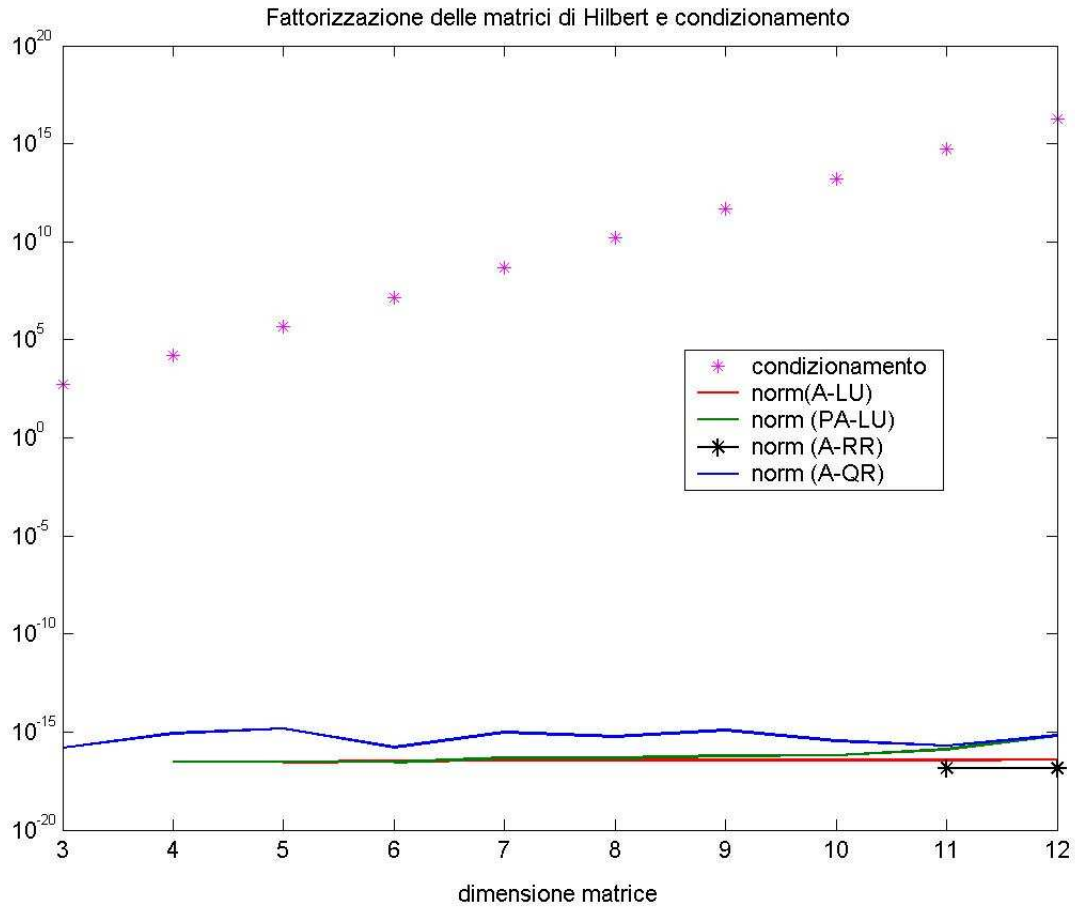

Figura 7

In realtà, l'algoritmo di fattorizzazione QR è quello più adatto per matrici mal condizionate come quella di Hilbert: ciò si evince portando le dimensioni della matrice di Hilbert fino a 200, ed applicando *fattqr* e *chole1*, gli unici tra gli algoritmi utilizzati che non si bloccano per n>12. Osservando il grafico relativo, si nota la stabilità dell'errore indipendentemente dal numero di condizionamento crescente, mentre la fattorizzazione di Cholesky mostra un andamento proporzionale al numero di condizionamento (Figura 8).

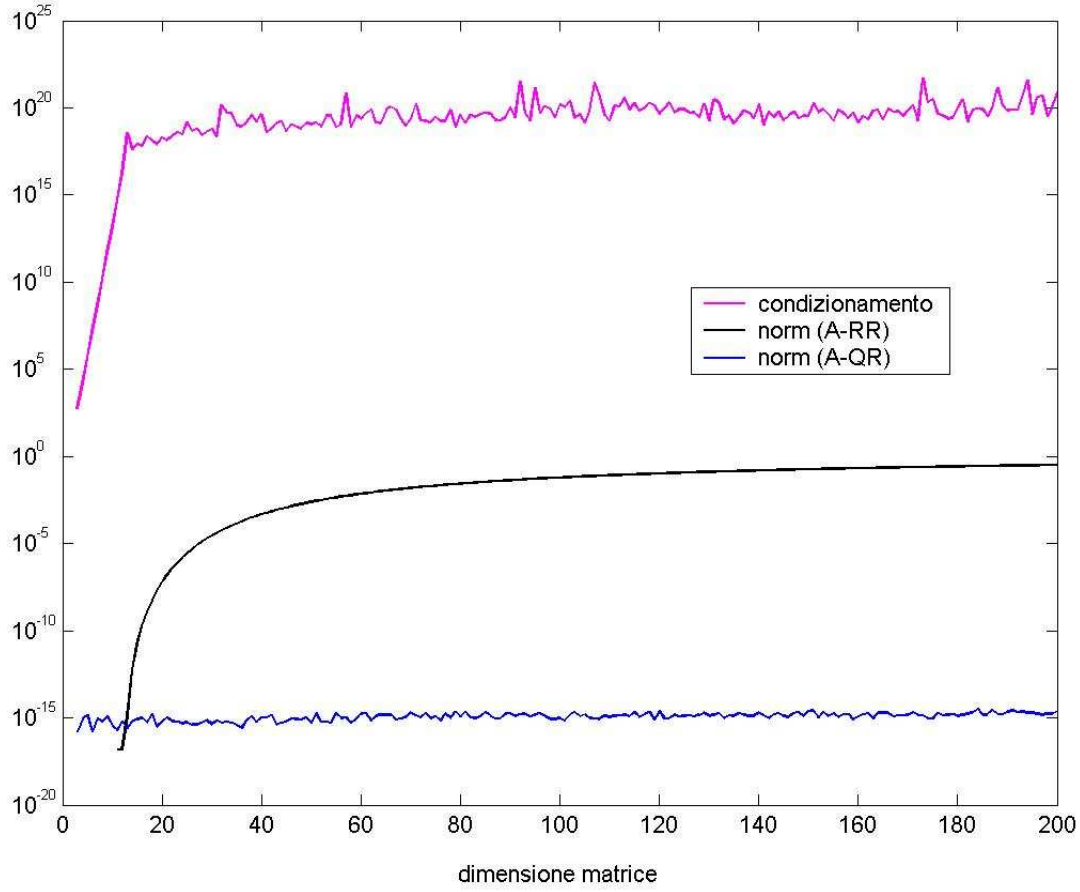

Figura 8

4.4. Fattorizzazione delle matrici di Pascal

A differenza delle matrici di Hilbert, il condizionamento delle matrici di Pascal, aumenta al crescere di *n* senza mai stabilizzarsi (Figura 9). Ciò implica che anche con l'algoritmo per la fattorizzazione QR, ben presto si raggiungono dimensioni dell'errore che divengono inaccettabili. Da notare che per *n<31*, la fattorizzazione di Cholesky e quella di Gauss senza pivoting, producono un errore nullo, mentre quella di Gauss con pivoting risulta inaffidabile (Figura 10). In questo caso tuttavia nessuno degli algoritmi si blocca.

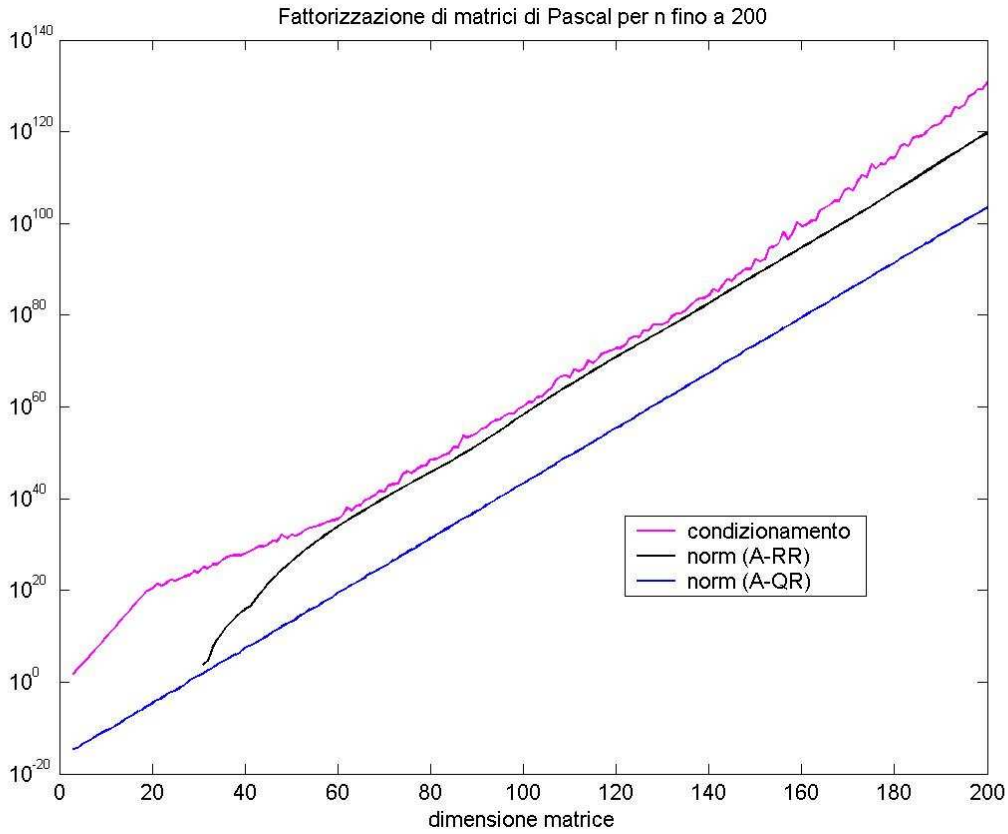

Figura 9

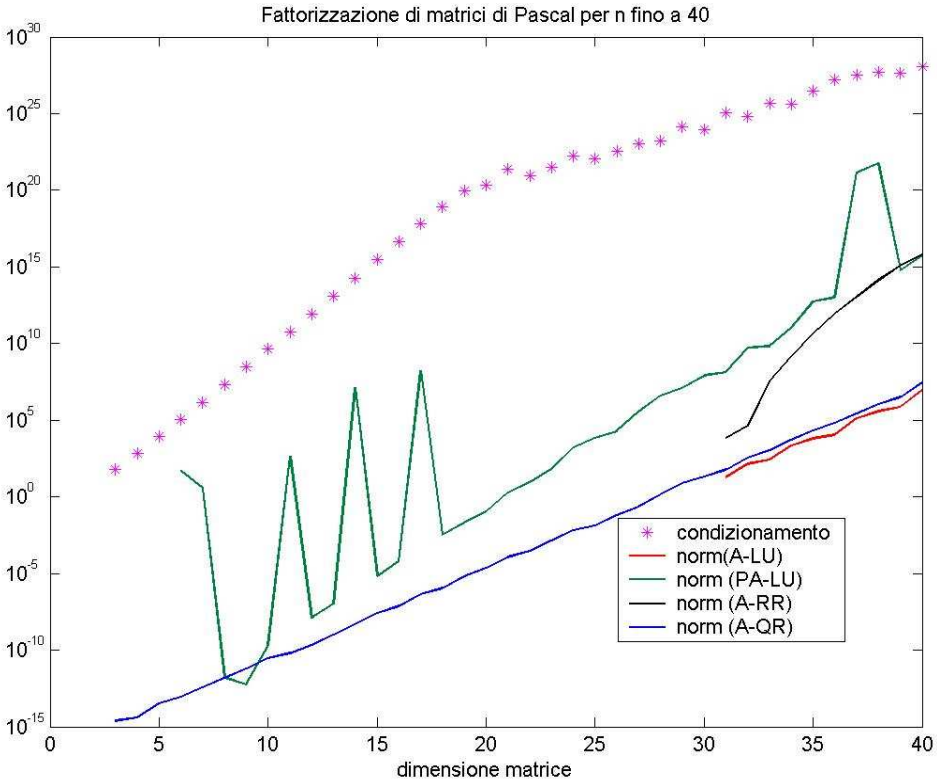

Figura 10

Capitolo 5 Analisi dell’errore sulla soluzione

5.1 Matrici random

L’errore sulla soluzione ha un andamento oscillante e crescente a prescindere dall’algoritmo di fattorizzazione utilizzato (figura 11).

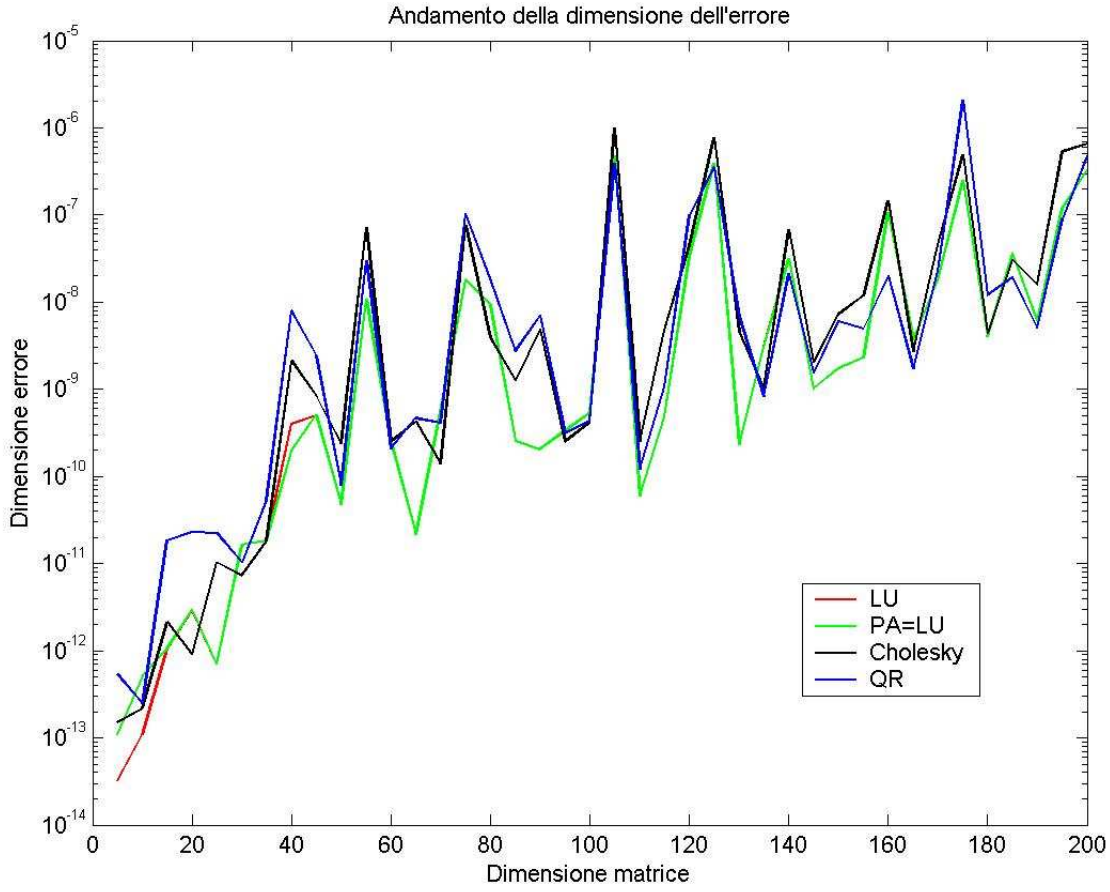

Figura 11

Sembra che tale andamento oscillatorio segua il condizionamento della matrice dei coefficienti (figura 12). Infatti relazionando il numero di condizionamento con l’errore si può notare una proporzionalità. La figura 13, relativa alla fattorizzazione LU è rappresentativa anche delle altre.

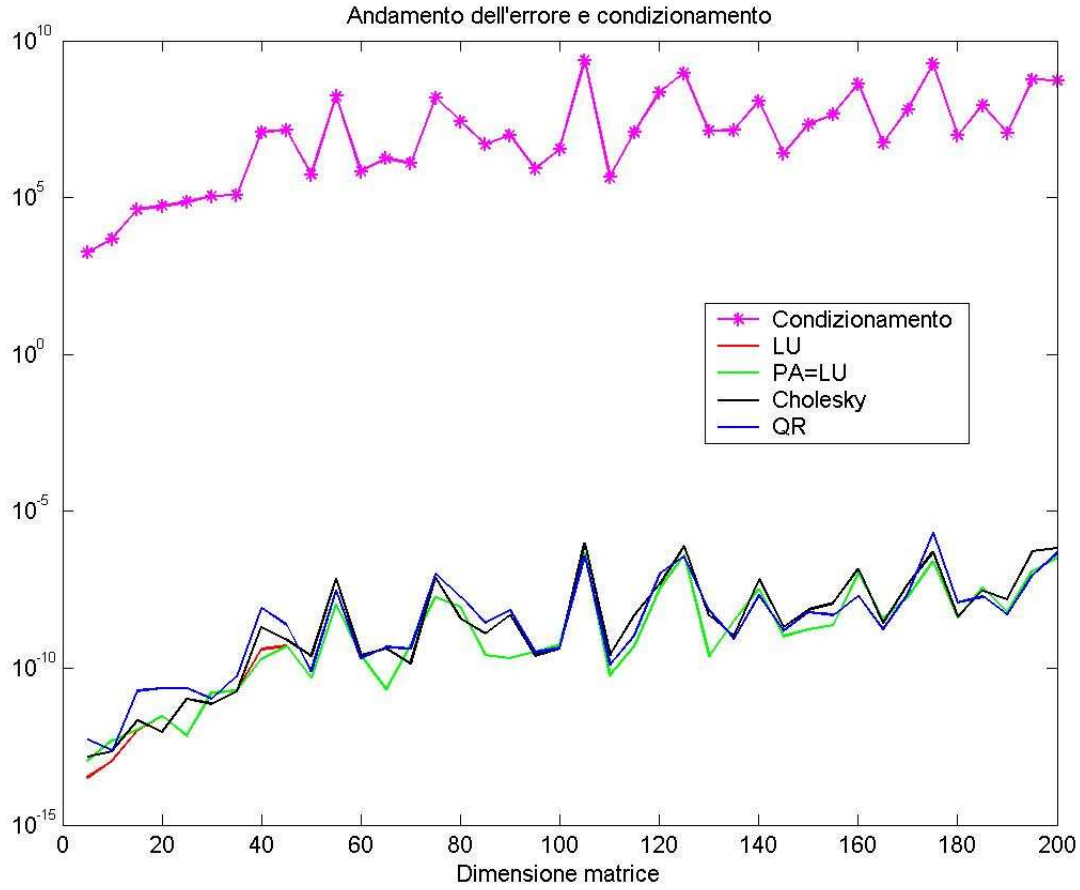
Andamento dell'errore e condizionamento
10^10
10^5
10^0
10^-5
10^-10
10^-15
0
20
40
60
80
100
120
140
160
180
200
Dimensione matrice
Condizionamento
LU
PA=LU
Cholesky
QR

Figura 12

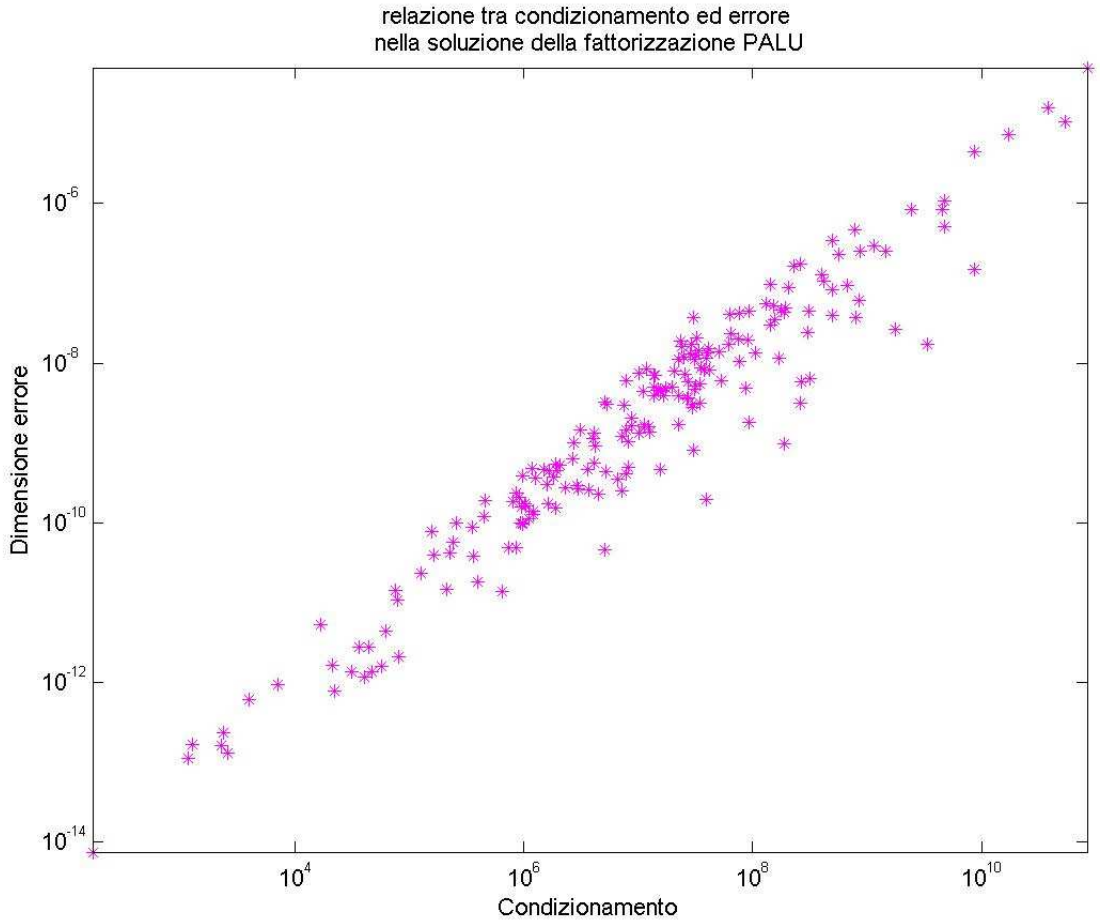

Figura 13

5.2 Matrici di Hilbert

Su queste matrici mal condizionate, per n fino a 14, la risoluzione del sistema genera un errore che si tiene basso per la fattorizzazione di Gauss con e senza pivoting, mentre cresce velocemente se la matrice dei coefficienti è stata trattata con la fattorizzazione di Cholesky o QR (figura 14). Aumentando n fino a 200 per i soli algoritmi di fattorizzazione QR e Cholesky, si conferma una rapida crescita iniziale dell'errore sulla soluzione ed una successiva stabilizzazione dello stesso (figura 15).

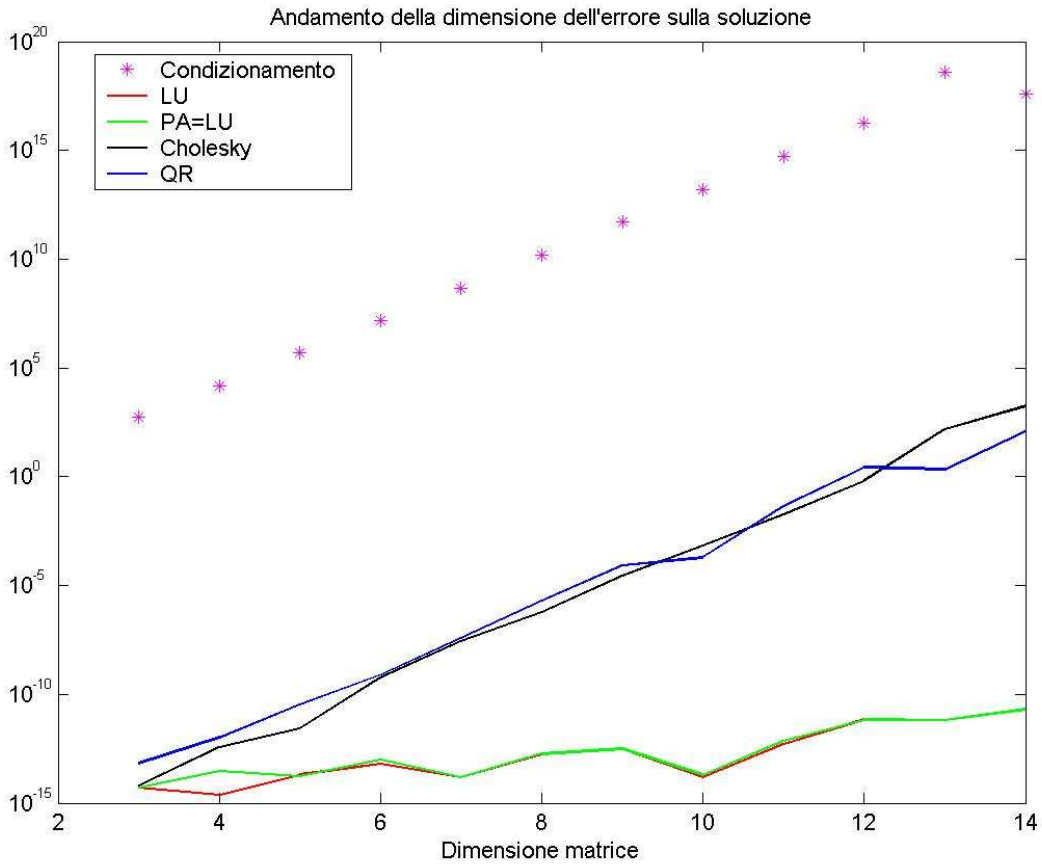

Figura 14

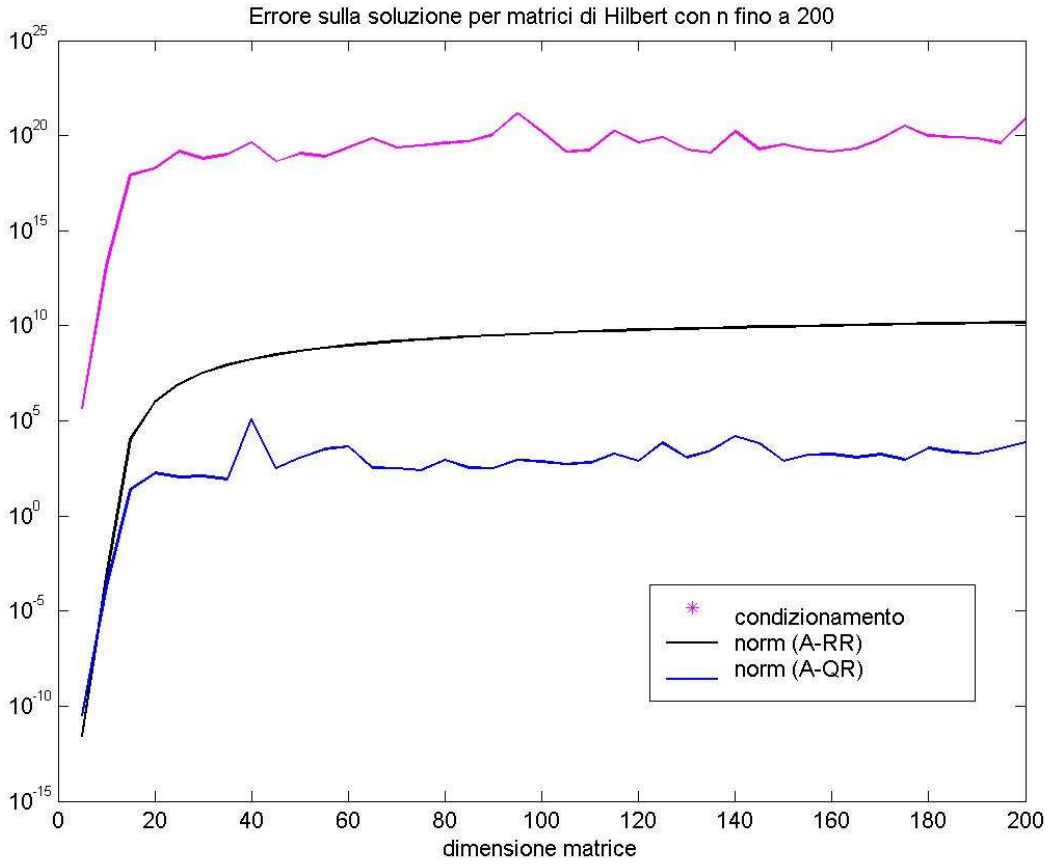

Figura 15

5.3 Matrici di Pascal

Per questo tipo di matrici mal condizionate, c'è una divergenza di comportamenti a seconda dell'algoritmo usato. La fattorizzazione QR e quella di Gauss con pivoting arrivano rapidamente a valori non tollerabili dell'errore, anche per n piccolo. L'algoritmo di Gauss senza pivoting e la fattorizzazione di Cholesky, portano ad avere un errore sulla soluzione che è nullo per $n = 3 \div 30$, mentre successivamente segue l'andamento degli altri errori (figura 16).

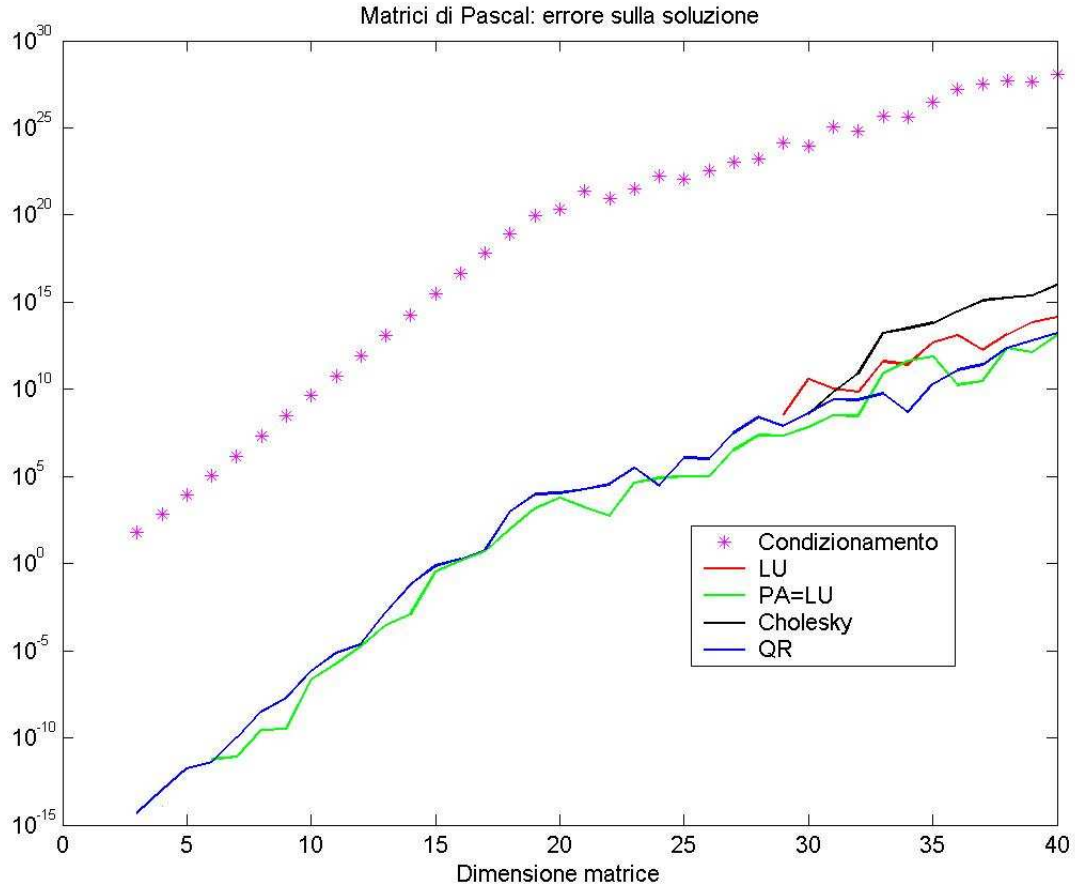

Figura 16

Per curiosità è possibile osservare l'andamento dell'errore sulla soluzione per gli algoritmi di Cholesky e QR per n fino a 200. Analogamente all'errore sulla fattorizzazione, continua a crescere in funzione del condizionamento della matrice dei coefficienti (figura 17).

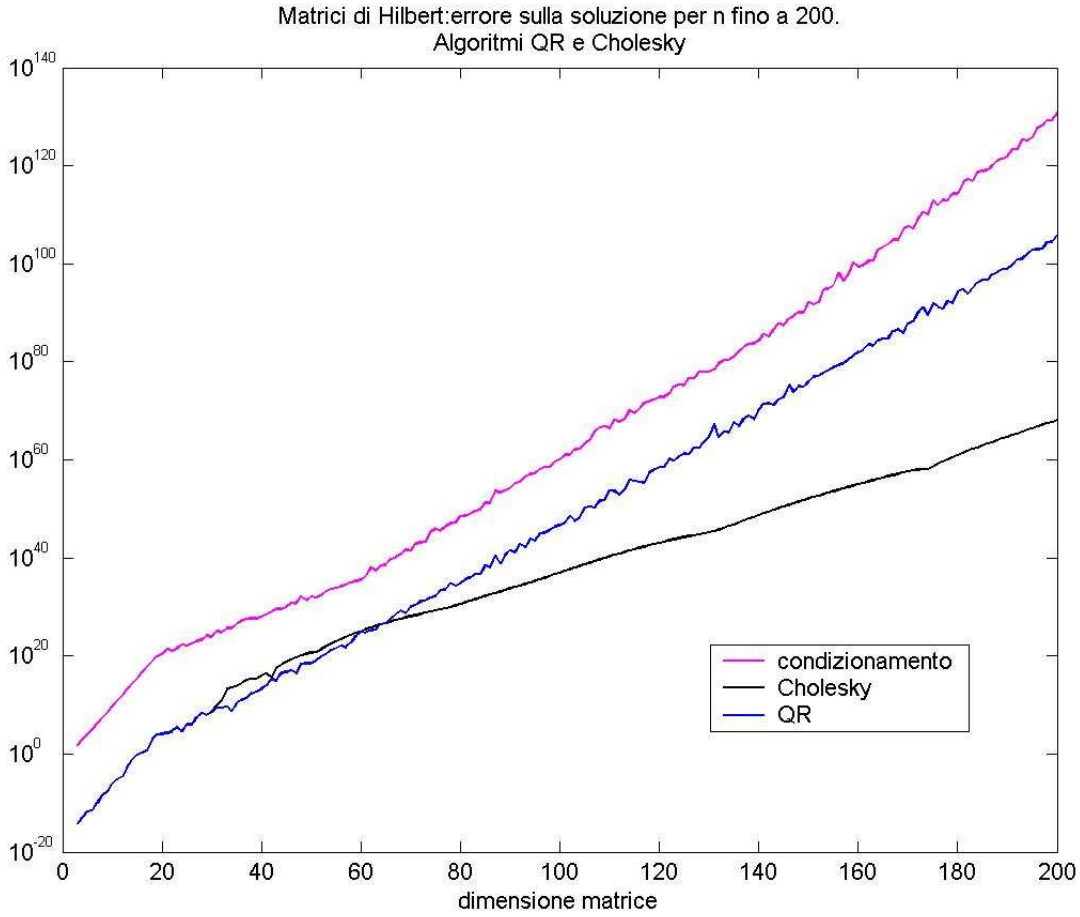
Matrici di Hilbert:errore sulla soluzione per n fino a 200.
Algoritmi QR e Cholesky
condizionamento
Cholesky
QR
dimensione matrice

Figura 17