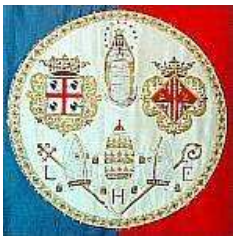


UNIVERSITA' DEGLI STUDI DI CAGLIARI
FACOLTA' DI INGEGNERIA

Corso di Laurea in Ingegneria Elettronica

Calcolo numerico 2

Prof. Giuseppe Rodriguez

Analisi matriciale: le Fattorizzazioni

A cura di:

Laura Marcias	37924
Rita Perria	38796

Anno accademico 2008/2009

Indice

Risoluzione di sistemi lineari

In matematica, e più precisamente in algebra lineare, una equazione lineare è una equazione di primo grado con un certo numero di incognite. 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 $x_1 \dots x_n$ che, sostituiti alle incognite, rende le equazioni delle identità.
Un sistema di n equazioni lineari in n incognite può essere scritto nel modo seguente:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

dove $x_1, \ldots, x_n$ sono le **incognite**, i numeri a_{ij}, sono i **coefficienti** e b_i sono i **termini noti**.
Una n-upla $(x_1, \ldots, x_n)$ è una **soluzione** se soddisfa tutte le n 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:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

In modo molto più stringato, si scrive:

$$A\mathbf{x} = \mathbf{b}$$

dove A è la matrice $n \times n$ dei coefficienti, $\mathbf{x}$ è il vettore delle n incognite e $\mathbf{b}$ è il vettore degli n termini noti.
Un sistema lineare ammette una e una sola soluzione se e solo se è verificata una delle seguenti proprietà equivalenti:

- il determinante di A è diverso da zero;
- il rango di A è uguale a n;
- il sistema omogeneo $A\mathbf{x} = 0$ ammette la sola soluzione banale, ovvero $x_i = 0$ con $i = 1,2,\ldots,n$.

Sistemi lineari semplici

Definiamo sistemi lineari semplici quelli dotati di una struttura tale da renderne immediata la risoluzione. Rientrano in tale classe i sistemi diagonali, ortogonali e triangolari. Vediamoli in dettaglio:

- un sistema diagonale ha la seguente forma:

$$D\mathbf{x} = \mathbf{b}$$

dove D = diag(d_1,…,d_n) con $d_i \neq 0$, $i = 1,\ldots,n$.
La soluzione è data da:

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

- un sistema ortogonale ha la seguente forma:

$$Q\mathbf{x} = \mathbf{b}$$

dove Q è una matrice ortogonale tale che $Q^TQ = QQ^T = I$.
La soluzione è data da:

$$\mathbf{x} = Q^T\boldsymbol{b} \qquad \text{ossia} \qquad x_i = \sum_{j=1}^{n} q_{ji} b_j \qquad \text{con } i = 1,\ldots,n$$

- in un sistema triangolare, la matrice dei coefficienti è triangolare inferiore ($a_{ij} = 0$ per $i < j$) o superiore ($a_{ij} = 0$ per $i > j$).

Nel caso di matrice triangolare inferiore si ha:

$$L\mathbf{x} = \mathbf{b} \qquad\qquad l_{ij} = 0 \text{ per } i < j$$

Il primo passo è quello di risolvere la prima equazione, che è in una sola variabile, il valore ottenuto viene sostituito nella seconda equazione per ricavare la seconda variabili, e così via. Si tratta di un algoritmo di sostituzione in avanti, detto di ***Forward Substitution***.

Nel caso di matrice triangolare superiore si ha:

$$U\mathbf{x} = \mathbf{b} \qquad\qquad u_{ij} = 0 \text{ per } i > j$$

Si risolve dapprima l'ultima equazione e poi si procede a ritroso verso la prima equazione, in questo caso si parla di algoritmo di sostituzione all'indietro, detto di ***Backward Substitution***.

Il metodo di Gauss

Abbiamo visto come i sistemi triangolari sono di facile risoluzione, il metodo di Gauss consente proprio di trasformare un generico sistema lineare in un sistema triangolare superiore ad esso equivalente.
Le operazioni consentite per ottenere un sistema equivalente a quello dato sono:

- scambiare tra loro due equazioni
- moltiplicare una equazione per una costante diversa da zero
- sommare a una equazione un'altra equazione

Descriviamo l'algoritmo utilizzando *k* come indice di iterazione, così da caratterizzare la matrice dei coefficienti ed il termine noto al passo *k.* All'inizio avremo:

$$A^{(1)} = A \quad \text{e} \quad \mathbf{b}^{(1)} = \mathbf{b}$$

quindi il sistema lineare si rappresenta nel seguente modo

$$\begin{cases} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ a_{21}^{(1)}x_1 + a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n = b_2^{(1)} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}^{(1)}x_1 + a_{n2}^{(1)}x_2 + \cdots + a_{nn}^{(1)}x_n = b_n^{(1)} \end{cases}$$

il primo passo dell'algoritmo consiste nell'eliminare la prima variabile x_1 dalla seconda fino all'ultima equazione, così da avere il sistema nella forma equivalente:

$$\begin{cases} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ \qquad\quad a_{22}^{(2)}x_2 + \cdots + a_{2n}^{(2)}x_n = b_2^{(2)} \\ \qquad\quad \cdots\cdots\cdots\cdots\cdots\cdots \\ \qquad\quad a_{n2}^{(2)}x_2 + \cdots + a_{nn}^{(2)}x_n = b_n^{(2)} \end{cases}$$

questo primo passo può essere fatto nell'ipotesi che $a_{11}^{(1)} \neq 0$, sottraendo la prima equazione dalle successive dopo averla moltiplicata per opportuni scalari m_{i1}, dove : $m_{i1} = \dfrac{a_{i1}^{(1)}}{a_{11}^{(1)}}$ garantisce l'annullarsi dei coefficienti della variabile x_1 nelle equazioni ottenute.
Questo passo viene eseguito fino a quando si riesce a portare il sistema in forma triangolare superiore:

$$\begin{cases} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ \qquad\quad a_{22}^{(2)}x_2 + \cdots + a_{2n}^{(2)}x_n = b_2^{(2)} \\ \qquad\quad \cdots\cdots\cdots\cdots\cdots\cdots \\ \qquad\qquad\qquad\qquad\quad a_{nn}^{(n)}x_n = b_n^{(n)} \end{cases}$$

Il metodo genererà una successione di matrici $\{A^{(k)}\}_{k=1,\ldots,n}$ e di termini noti $\{\mathbf{b}^{(k)}\}_{k=1,\ldots,n}$ con la richiesta che ogni sistema $A^{(k)}\mathbf{x} = \mathbf{b}^{(k)}$ sia equivalente al sistema di partenza e $A^{(n)}$ sia triangolare superiore.
Analizziamo la struttura della matrice del sistema e del termine noto al generico passo *k*

$$\left[A^{(k)}\middle|b^{(k)}\right]=\left[\begin{array}{ccccccc|c} a_{11}^{(1)} & \cdots & \cdots & \cdots & \cdots & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & \ddots & & & & & \vdots & \vdots \\ \vdots & \ddots & a_{k-1,k-1}^{(k-1)} & \cdots & \cdots & \cdots & a_{k-1,n}^{(k-1)} & b_{k-1}^{(k-1)} \\ \vdots & & 0 & a_{kk}^{(k)} & a_{k,k+1}^{(k)} & \cdots & a_{kn}^{(k)} & b_k^{(k)} \\ \vdots & & \vdots & a_{k+1,k}^{(k)} & a_{k+1,k+1}^{(k)} & \cdots & a_{k+1,n}^{(k)} & b_{k+1}^{(k)} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & \cdots & 0 & a_{nk}^{(k)} & a_{n,k+1}^{(k)} & \cdots & a_{nn}^{(k)} & b_n^{(k)} \end{array}\right]$$

La matrice è stata parzialmente triangolarizzata nei passi precedenti, il passo *k* deve annullare gli elementi dalla riga *k*+1-esima fino alla *n*-esima posti nella colonna *k*-esima, sottraendo la *k*-esima riga, moltiplicata per opportuni scalari m_{ik}, dalle righe con indice compreso tra *k*+1 e *n*. Resteranno immutate le prime *k* righe e si modificheranno soltanto gli elementi dalla riga *k*+1-esima fino alla *n*-esima e dalla colonna *k*+1-esima fino alla *n*-esima.
Al termine del passo *k* la matrice ed il termine noto avranno la seguente struttura:

$$\left[A^{(k+1)}\middle|b^{(k+1)}\right]=\left[\begin{array}{ccccccc|c} a_{11}^{(1)} & \cdots & \cdots & \cdots & \cdots & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & \ddots & & & & & \vdots & \vdots \\ \vdots & \ddots & a_{k-1,k-1}^{(k-1)} & \cdots & \cdots & \cdots & a_{k-1,n}^{(k-1)} & b_{k-1}^{(k-1)} \\ \vdots & & 0 & a_{kk}^{(k)} & a_{k,k+1}^{(k)} & \cdots & a_{kn}^{(k)} & b_k^{(k)} \\ \vdots & & \vdots & 0 & a_{k+1,k+1}^{(k+1)} & \cdots & a_{k+1,n}^{(k+1)} & b_{k+1}^{(k+1)} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & a_{n,k+1}^{(k+1)} & \cdots & a_{nn}^{(k+1)} & b_n^{(k+1)} \end{array}\right]$$

Gli elementi della matrice $A^{(k+1)}$ sono definiti dalla regola:

$$a_{ij}^{(k+1)}=\begin{cases} a_{ij}^{(k)}, & i=1,\cdots,k, opp.\, j=1,\cdots,k-1, \\ 0, & j=k, i=k+1,\cdots,n, \\ a_{ij}^{(k)}-m_{ik}a_{kj}^{(k)}, & i,j=k+1,\cdots,n \end{cases}$$

e il nuovo termine noto è dato da:

$$b_i^{(k+1)} = \begin{cases} b_i^{(k)}, & i = 1, \cdots, k, \\ b_i^{(k)} - m_{ik} b_k^{(k)}, & i = k+1, \cdots, n \end{cases}$$

dove $m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$. L'elemento $a_{kk}^{(k)}$ è detto **elemento pivot** al passo *k*, e le iterazioni possono proseguire solo se tutti gli elementi pivot risultano essere non nulli. L'algoritmo termina dopo $n - 1$ passi con una matrice $U = A^{(n)}$ triangolare superiore.
Questo algoritmo dovrà subire delle modifiche per poter essere applicata a tutte le matrici non singolari, in quanto la non singolarità di una matrice non garantisce che tutti gli elementi pivot siano non nulli. Il metodo di Gauss è applicabile nella forma sopra citata nel caso di matrici simmetriche definite positive, matrici diagonalmente dominanti per righe i cui elementi verificano la seguente condizione:

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \qquad i = 1, \ldots, n$$

e matrici diagonalmente dominanti per colonne, definite in modo analogo.

IMPLEMENTIAMO L'ALGORITMO DI GAUSS IN MATLAB:

```
n = 8;                                  % dimensione della matrice
A = rand(n)                             % generiamo una matrice A casuale di
                                        % dimensione n
sol = ones(n,1);                        % imponiamo che la soluzione sia un
                                        % vettore unitario
b = A*sol                               % il vettore dei termini noti è dato
                                        % dal prodotto tra A e il vettore
                                        % "sol"
for k = 1:n-1                           % implementazione dell'algoritmo
    for i = k+1:n
        mik = A(i,k)/A(k,k);
        for j = k+1:n
            A(i,j) = A(i,j) - mik*A(k,j);
        end
        A(i,k) = 0;
        b(i) = b(i) - mik*b(k);
    end
end
```

La matrice *A* casuale generata e il vettore dei termini noti b iniziali sono:

```
A =                                     b =

    0.6551    0.4984    0.5853              1.7387
    0.1626    0.9597    0.2238              1.3462
    0.1190    0.3404    0.7513              1.2107
```

l'obiettivo dell'algoritmo di Gauss è quello di triangolarizzare la matrice dei coefficienti del sistema lineare $A\mathbf{x} = \mathbf{b}$, l'algoritmo modifica inoltre il vettore dei termini noti ottenendo un sistema triangolare equivalente a quello di partenza, la matrice triangolare superiore e il nuovo vettore dei termini noti sono:

```
A =                                         b =

    0.6551    0.4984    0.5853                1.7387
         0    0.8360    0.0785                0.9146
         0         0    0.6215                0.6215
```

come si può notare dalla matrice e dal vettore che abbiamo ottenuto, la prima riga di entrambi risulta essere uguale a quella della matrice A e del vettore dei termini noti **b** di partenza.

Il metodo di Gauss con pivoting

L'algoritmo di Gauss precedentemente descritto può essere applicato solo se tutti glie elementi pivot a_{kk} , $k = 1,\ldots, n\text{-}1$ sono diversi da zero. Questo risulta essere vero per matrici simmetriche definite positive o diagonalmente dominanti non singolari, ma non per tutte le matrici invertibili. Si dimostra che se A è non singolare allora esiste, al passo k, almeno un indice l compreso tra k ed n tale che $a_{lk} \neq 0$. Sarà quindi sufficiente scambiare le righe k ed l per evitare che l'algoritmo si blocchi.
Infatti se al passo k risultasse $a_{ik} = 0$ con $i = k,\ldots,n$ l'algoritmo di triangolarizzazione potrebbe andare avanti semplicemente saltando il passo k come conseguenza però la matrice $U = A^{(n)}$ presenterebbe un elemento diagonale nullo e quindi sarebbe singolare. Questa procedura consente di impedire il blocco dell'algoritmo in presenza di una matrice non singolare, motivi di stabilità numerica suggeriscono però una scelta più accurata. Infatti, dal momento che l'elemento pivot si trova a denominatore nell'espressione di m_{ik} uno scambio delle righe che porti a massimizzare questo elemento può ridurre la propagazione degli errori dovuti all'aritmetica di macchina ed evitare, in casi estremi il verificarsi di overflow. Questi ed altri motivi conducono alla strategia del **pivoting parziale** o di colonna.

1. **al passo** k
    1. trova l tale che $\left|a_{lk}^{(k)}\right| = \max_{i=k,\ldots,n}\left|a_{ik}^{(k)}\right|$
    2. scambia la riga k con la riga l
    3. scambia le componenti k ed l del termine noto.

Tale procedura garantisce il valore più basso possibile per i moltiplicatori m_{ik} usati nell'algoritmo, in particolare si ha $|m_{ik}| \leq 1$.

IMPLEMENTIAMO L'ALGORITMO DI GAUSS CON PIVOTING PARZIALE

```
n = 4;
A = rand(n)
sol = ones(n,1);
b = A*sol
for k = 1:n-1
    % Pivoting
    [piv, l] = max(abs(A(k:n,k)));
```

```
    if (l ~= k)
        % Scambio righe
        temp = A(l+k-1,:);
        A(l+k-1,:) = A(k,:);
        A(k,:) = temp;
        tempor = b(l+k-1);
        b(l+k-1) = b(k);
        b(k) = tempor;
    end
    % Algoritmo di Gauss
    for i = k+1:1:n
        mik = A(i,k)/A(k,k);
        A(i,k) = 0;
        for j = k+1:1:n
            A(i,j) = A(i,j) - mik*A(k,j);
            b(i) = b(i) - mik*b(k);
        end
    end
end
```

La matrice *A* di partenza e il vettore dei termini noti **b** generati sono i seguenti:

```
A =

    0.5822    0.3181    0.4795    0.5439
    0.5407    0.1192    0.6393    0.7210
    0.8699    0.9398    0.5447    0.5225
    0.2648    0.6456    0.6473    0.9937

b =

    1.9237
    2.0203
    2.8770
    2.5513
```

La matrice triangolare superiore e il nuovo vettore dei termini noti ottenuti sono:

```
A =

    0.8699    0.9398    0.5447    0.5225
         0   -0.4650    0.3007    0.3963
         0         0    0.7140    1.1411
         0         0         0    0.0670

b =

    2.8770
   -3.3445
   -5.2474
   -0.0133
```

Fattorizzazione LU

Lo scopo della fattorizzazione è quello di scomporre una matrice in un prodotto di matrici aventi una struttura più semplice.

Il metodo di Gauss oltre a trasformare un sistema lineare in un sistema triangolare fornisce anche la fattorizzazione *LU* della matrice *A* del sistema.
La matrice *L* è una matrice triangolare inferiore i cui elementi soddisfano la seguente relazione:

$$l_{ij} = \begin{cases} m_{ij}, i > j \\ 1, i = j \\ 0, i < j \end{cases}$$

dove m_{ij} sono i moltiplicatori calcolati durante l'esecuzione dell'algoritmo di Gauss.
La matrice U è una matrice triangolare superiore i cui elementi soddisfano la seguente relazione:

$$u_{ij} = \begin{cases} a_{ij}^{(n)}, i \leq j \\ 0, i > j \end{cases}$$

dove $a_{ij}^{(n)}$ sono i coefficienti del sistema triangolare superiore finale. Si dimostra che tali matrici soddisfano l'uguaglianza:

$$A = LU$$

Nota la fattorizzazione della matrice *A* la risoluzione del sistema lineare $A\mathbf{x} = \mathbf{b}$ viene effettuata risolvendo due sistemi triangolari in cascata, ossia:

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

Matlab calcola la fattorizzazione LU di una matrice con il comando:

[L,U] = lu(A)

e si può procedere alla risoluzione del sistema tramite i comandi:

y = L\b
x = U\y

IMPLEMENTAZIONE IN MATLAB DELLA FATTORIZZAZIONE LU

```
n = 3;          % dimensione della matrice
A = rand(n);    % generiamo una matrice casuale di dimensione n
A = A'*A;       % condizione per cui la matrice A sia simmetrica definita
                % positiva
e = ones(n,1);  % generiamo un vettore colonna con n elementi tutti unitari
```

```
b = A*e;        % vettore dei termini noti
[L, U] = lu(A); % comando per la fattorizzazione LU della matrice A
y = L\b;        % soluzione del primo dei due sistemi triangolari in cascata
x = U\y;        % soluzione del sistema
```

la matrice *A* generata è:

```
A =

    1.9782    1.4555    1.7520
    1.4555    1.5946    1.3617
    1.7520    1.3617    1.5871
```

le matrici *L* ed *U* sono:

```
L =                                   U =

    1.0000         0         0              1.9782    1.4555    1.7520
    0.7358    1.0000         0                   0    0.5238    0.0727
    0.8857    0.1387    1.0000                   0         0    0.0253
```

come possiamo vedere il prodotto tra la matrice *L* e la matrice *U* ci restituisce la matrice *A*:

```
L*U

ans =

    1.9782    1.4555    1.7520
    1.4555    1.5946    1.3617
    1.7520    1.3617    1.5871
```

IMPLEMENTAZIONE IN MATLAB DELLA FATTORIZZAZIONE LU CON L'ALGORITMO REALIZZATO DA NOI

```
n = 3;
A = rand(n)
sol = ones(n,1);
b = A*sol ;
L = eye(n);
for k = 1:n-1
    for i = k+1:n
        mik = A(i,k)/A(k,k);
        if i > k
            L(i,k) = mik;
        end
        for j = k+1:n
            A(i,j) = A(i,j) - mik*A(k,j);
        end
        A(i,k) = 0;
    b(i) = b(i) - mik*b(k);
    end
end
U = triu(A);
for i = 1:n
    for j = 1:n
        if i < j
                L(i,j) = 0;
```

```
            end
    end
end
y = L\b;
x = U\y;
```

La matrice casuale *A* di partenza è:

```
A =

    0.3922    0.7060    0.0462
    0.6555    0.0318    0.0971
    0.1712    0.2769    0.8235
```

Le matrici *L* ed *U*, rispettivamente triangolare inferiore e superiore, sono:

```
L =                                   U =

    1.0000         0         0            0.3922    0.7060    0.0462
    1.6712    1.0000         0                 0   -1.1481    0.0200
    0.4364    0.0272    1.0000                 0         0    0.8028
```

come si può vedere il prodotto tra la matrice *L* e la matrice *U* restituisce proprio la matrice *A* di partenza:

```
L*U

ans =

    0.3922    0.7060    0.0462
    0.6555    0.0318    0.0971
    0.1712    0.2769    0.8235
```

Fattorizzazione PA=LU

Definiamo matrice di scambio quella ottenuta scambiando la riga *k* e la riga *s* in una matrice identità che può essere rappresentata nel seguente modo:

$$P^{(k,s)} = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & 0 & \cdots & 1 & \\ & & & \vdots & & \vdots & \\ & & & 1 & \cdots & 0 & \\ & & & & & & 1 \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \text{(riga } k\text{)} \\ \\ \text{(riga } s\text{)} \\ \\ \\ \end{matrix}$$

La matrice $P^{(k,s)}$ è ortogonale simmetrica, e il suo determinante vale -1.

Applicare tale matrice ad un vettore **x** ha l'effetto di scambiare le componenti x_k e x_s, quindi premoltiplicare A per una matrice di scambio ($P^{(k,s)}A$) equivale a scambiare tra loro due righe, viceversa moltiplicare A per una matrice di scambio ($AP^{(k,s)}$) equivale a scambiare due colonne.
Si definisca matrice di permutazione il prodotto di più matrici di scambio:

$$P = P^{(k_m, s_m)} P^{(k_{m-1}, s_{m-1})} \dots P^{(k_1, s_1)}$$

P è una matrice ortogonale avente un solo elemento pari a 1 in ogni riga o colonna e gli altri elementi nulli. Il suo determinante vale $(-1)^{\#(P)}$, dove $\#(P)$ indica il numero degli scambi che compongono la permutazione P.
L'algoritmo di Gauss con pivoting di colonna si può rappresentare nella forma:

$$PA = LU$$

Si tratta della forma più generale della fattorizzazione LU ed è applicabile ad ogni matrice non singolare.
La risoluzione di un sistema lineare $A\mathbf{x} = \mathbf{b}$, tramite la fattorizzazione sopra citata, si riconduce alla risoluzione di due sistemi triangolari:

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

Matlab calcola la fattorizzazione $PA = LU$ di una matrice con il comando:

[L, U, P] = lu(A);

e si può procedere alla risoluzione del sistema:

y = L\(P*b);
x = U\y;

IMPLEMENTAZIONE IN MATLAB DELLA FATTORIZZAZIONE PA = LU

```
n = 4;              % dimensione della matrice
A = rand(n);        % il comando rand genera una matrice casuale di dimensione n
e = ones(n,1);      % il comando ones(n,m) genera una matrice di dimensione n*m
b = A*e;            % b è il vettore dei termini noti
[L, U, P] = lu(A);  % comando per la fattorizzazione
y = L\(P*b);        % soluzione del primo sistema triangolare
x = U\y;            % soluzione del sistema Ax = b
```

la matrice casuale generata è la seguente:

```
A =

    0.6761    0.0680    0.8444    0.0067
    0.2891    0.2548    0.3445    0.6022
    0.6718    0.2240    0.7805    0.3868
    0.6951    0.6678    0.6753    0.9160
```

Le matrici *L* e *U* sono:

```
L =

    1.0000         0         0         0
    0.9726    1.0000         0         0
    0.4158    0.0394    1.0000         0
    0.9664    0.7246   -0.1427    1.0000

U =

    0.6951    0.6678    0.6753    0.9160
         0   -0.5816    0.1875   -0.8842
         0         0    0.0562    0.2561
         0         0         0    0.1787
```

La matrice *PA* è:

```
P*A

ans =

    0.6951    0.6678    0.6753    0.9160
    0.6761    0.0680    0.8444    0.0067
    0.2891    0.2548    0.3445    0.6022
    0.6718    0.2240    0.7805    0.3868
```

che come possiamo notare è uguale al prodotto tra la matrice *L* e la matrice *U*:

```
L*U

ans =

    0.6951    0.6678    0.6753    0.9160
    0.6761    0.0680    0.8444    0.0067
    0.2891    0.2548    0.3445    0.6022
    0.6718    0.2240    0.7805    0.3868
```

IMPLEMENTAZIONE IN MATLAB DELL'ALGORITMO PA = LU IMPLEMENTATO DA NOI

```
n = 3;
A = rand(n)
sol = ones(n,1);
b = A*sol ;
P = 1:n;
for k = 1:n-1
    % Pivoting
    [piv, l] = max(abs(A(k:n,k)));
    if (l ~= k)
        % Scambio righe
        aux = P(l);
```

```
        P(l) = P(k);
        P(k) = aux;
        temp = A(l+k-1,:);
        A(l+k-1,:) = A(k,:);
        A(k,:) = temp;
        tempor = b(l+k-1);
        b(l+k-1) = b(k);
        b(k) = tempor;
    end
    % Algoritmo di Gauss
    for i = k+1:1:n
        mik = A(i,k)/A(k,k);
        if i > k
            L(i,k) = mik;
        end
        A(i,k) = 0;
        for j = k+1:1:n
            A(i,j) = A(i,j) - mik*A(k,j);
            b(i) = b(i) - mik*b(k);
        end
    end
end
for i = 1:n
    for j = 1:n
        if i == j
            L(i,j) = 1;
        else if i < j
                L(i,j) = 0;
            end
        end
        if i <= j
            U(i,j) = A(i,j);
        else U(i,j) = 0;
        end
    end
end
```

P non è la matrice di permutazione ma solo una sua rappresentazione nel senso che è un vettore che indica quali sono gli scambi delle righe della matrice identità da fare.
La matrice **A** generata di partenza è:

A =

```
    0.2819    0.4991    0.1239
    0.5386    0.5358    0.4904
    0.6952    0.4452    0.8530
```

La matrice *A* triangolarizzata è:

```
A =

    0.6952    0.4452    0.8530
         0    0.1909   -0.1705
         0         0    0.0627
```

Il vettore risulta essere:

```
P =
      3     2     1
```

Questo significa che la matrice di permutazione **P** con cui moltiplicare la matrice **A** di partenza sarà:

```
P =
     0     0     1
     0     1     0
     1     0     0
```

Verifichiamo come **P** moltiplicata per la matrice *A* di partenza è pari ad *L*U*:

```
P*A =                                    L*U =

    0.6952    0.4452    0.8530              0.6952    0.4452    0.8530
    0.5386    0.5358    0.4904              0.5386    0.5358    0.4904
    0.2819    0.4991    0.1239              0.2819    0.4991    0.1239
```

Altre fattorizzazioni LU

La fattorizzazione $A = LU$ esiste se e solo se la matrice A è non singolare e tutti i suoi minori principali sono non nulli. Da questo discende l'esistenza ed unicità della cosidetta fattorizzazione LDR (o LDU).
Sia $A = LU$, se poniamo

$$D = \text{diag}(u_{11},\ldots,u_{nn})$$

e $R = D^{-1}U$, la matrice può essere decomposta nella forma

$$A = LDR$$

dove L è una matrice triangolare inferiore, R una matrice triangolare superiore, $l_{ii} = r_{ii} = 1$, $i = 1,\ldots,n$. Se la matrice A è simmetrica è facile osservare che risulta $R = L^T$ e di conseguenza

$$A = LDL^T$$

L'algoritmo di Gauss può essere modificato in modo opportuno per costruire queste fattorizzazioni.
Si definisce ***inerzia*** di una matrice hermitiana A la terna di numeri naturali

$$\text{Inerzia}(A):\{\pi,\upsilon,\delta\}$$

che indicano rispettivamente il numero di autovalori positivi, negativi e nulli di A.

LEGGE DI INERZIA DI SYLVESTER.
Siano A e B due matrici hermitiane. Esiste una matrice non singolare C tale che $B = C^*AC$ se e solo se

$$\text{Inerzia}(A) = \text{Inerzia}(B)$$

Se la matrice A è simmetrica definita positiva, applicando questo teorema alla fattorizzazione $A = LDL^T$ possiamo concludere che tutti gli elementi diagonali d_i della matrice D sono positivi e possiamo definire la matrice

$$D^{1/2} = diag\left(\sqrt{d_1}, \cdots, \sqrt{d_n}\right)$$

e la matrice $R = D^{1/2}L^T$ con $r_{ii} > 0$ in modo che si abbia

$$A = LD^{1/2}D^{1/2}L^T = R^TR$$

La fattorizzazione $A = R^TR$ della matrice definita positiva A prende il nome di ***fattorizzazione di Cholesky***.

Il sistema $A\mathbf{x} = \mathbf{b}$ è ricondotto alla risoluzione di due sistemi triangolari in cascata:

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

L'analisi del prodotto matriciale:

$$A = R^TR = \begin{bmatrix} r_{11} & & & & \\ \vdots & \ddots & & & \\ r_{1i} & \cdots & r_{ii} & & \\ \vdots & & & \ddots & \\ r_{1n} & \cdots & \cdots & \cdots & r_{nn} \end{bmatrix} \begin{bmatrix} r_{11} & \cdots & r_{1j} & \cdots & r_{1n} \\ & \ddots & \vdots & & \vdots \\ & & r_{jj} & & \vdots \\ & & & \ddots & \vdots \\ & & & & r_{nn} \end{bmatrix}$$

permette di esprimere gli elementi del triangolo superiore di A in funzione degli elementi di R:

$$a_{ij} = \sum_{k=1}^{i} r_{ki}r_{kj} \qquad i \leq j$$

Distinguiamo i due casi $i < j$ e $i = j$ per l'ultimo termine della sommatoria:

$$r_{ij} = \frac{1}{r_{ii}}\left(a_{ij} - \sum_{k=1}^{i-1} r_{ki}r_{kj}\right) \qquad i < j$$

$$r_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2 \right)^{1/2} \qquad i = j$$

La funzione utilizzata in Matlab per la fattorizzazione di Cholesky è la funzione “chol”

```
n = 4;                % dimensione della matrice A
A = rand(n);          % rand genera una matrice casuale di dimensione n
A = A'*A;             % A deve essere hermitiana
sol = ones(n,1);      % ones(n,m) genera una matrice di 1 di dimensione n*m
b = A*sol;            % vettore dei termini noti
R = chol(A);          % funzione che restituisce la fattorizzazione di Cholesky
y = R'\b;             % soluzione del primo sistema triangolare
x = R\y;              % vettore soluzione
z = norm(x-sol);      % errore
```

La matrice *A* generata è:

```
A =

    0.7994    0.7019    0.9492    0.3718
    0.7019    1.0673    1.2569    0.6819
    0.9492    1.2569    1.5590    0.7752
    0.3718    0.6819    0.7752    0.4586
```

La matrice *R* ottenuta con la fattorizzazione di Cholesky è:

```
R =

    0.8941    0.7851    1.0616    0.4159
         0    0.6715    0.6306    0.5293
         0         0    0.1853   -0.0005
         0         0         0    0.0739
```

La fattorizzazione mi dice che $A = R^T R$ verifichiamolo:

```
R'*R

ans =

    0.7994    0.7019    0.9492    0.3718
    0.7019    1.0673    1.2569    0.6819
    0.9492    1.2569    1.5590    0.7752
    0.3718    0.6819    0.7752    0.4586
```

Un’implementazione per colonne dell’algoritmo di Cholesky, ossia che costruisce la matrice *R* colonna per colonna è descritta nel seguente algoritmo:

1. for $j = 1,\ldots,n$
    1. for $i = 1,\ldots, j\text{-}1$

1. $$r_{ij} = \frac{1}{r_{ii}}\left(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}\right)$$

2. $$r_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2\right)^{1/2}$$

IMPLEMENTAZIONE IN MATLAB DELL'ALGORITMO DI CHOLESKY PER COLONNE

```
function [R] = funzchol(A)                  % definiamo una nuova funzione
                                            % chiamata funzchol che ha come
                                            % input la matrice A e genera
                                            % la matrice R
[n,n] = size(A);
R = zeros(n,n);
for j = 1:n
    if (A(j,j)-sum(R(1:j-1,j).^2)) < 0      % verifichiamo che la matrice A sia
                                            % simmetrica e definita positiva
        error ('La matrice A non è definita positiva')
    else
        for i = 1:j-1
            R(i,j)= (A(i,j)- sum(R(1:i-1,i).*R(1:i-1,j)))/R(i,i);
        end
    R(j,j) = sqrt(A(j,j)- sum(R(1:j-1,j).^2));
    end
end
```

Fattorizzazione QR

La fattorizzazione $A = QR$ è possibile per qualsiasi matrice A di dimensione $m \times n$, dove Q è una matrice ortogonale $m \times m$, tale per cui $Q^TQ = QQ^T = I$, e R è una matrice triangolare superiore con dimensioni pari a quelle di A.
Considerando una matrice quadrata non singolare, allora la fattorizzazione QR può essere utilizzata per risolvere il sistema lineare $A\mathbf{x} = \mathbf{b}$. Quindi sostituendo alla matrice A la sua fattorizzazione QR si otterranno i due sistemi:

$$\begin{cases} Qc = b \\ Rx = c \end{cases}$$

La soluzione del primo sistema è data da $\mathbf{c} = Q^T\mathbf{b}$, che fornisce il termine noto del secondo sistema che andrà risolto per sostituzione all'indietro.
Per il calcolo di questa tipologia di fattorizzazione sono presenti diversi algoritmi con complessità differente.
Per introdurre questi algoritmi è necessario definire due tipi di matrici elementari:

- Matrici elementari di Householder
- Matrici elementari di Givens

Matrici elementari di Householder

Definiamo la matrice elementare reale di Householder nel seguente modo:

$$H = I - 2\mathbf{w}\mathbf{w}^T$$

dove $\mathbf{w} \in R$ e $\|\mathbf{w}\|_2 = 1$, inoltre $2\mathbf{w}\mathbf{w}^T$ è una matrice $n \times n$ di rango 1.
Si dimostra che la matrice H è una matrice simmetrica e ortogonale, infatti per quanto riguarda la simmetria, facendo la trasposta della matrice H otteniamo la stessa matrice H:

$$H^T = (I - 2\mathbf{w}\mathbf{w}^T)^T = I - 2\mathbf{w}\mathbf{w}^T = H$$

Per quanto riguarda invece l'ortogonalità, dobbiamo dimostrare che: $H^T H = I$

$$\begin{aligned} H^T H &= (I - 2\mathbf{w}\mathbf{w}^T)(I - 2\mathbf{w}\mathbf{w}^T) = \\ &= I - 2\mathbf{w}\mathbf{w}^T - 2\mathbf{w}\mathbf{w}^T + 4\mathbf{w}\mathbf{w}^T\mathbf{w}\mathbf{w}^T = \\ &= I - 4\mathbf{w}\mathbf{w}^T + 4\mathbf{w}(\mathbf{w}^T\mathbf{w})\mathbf{w}^T \end{aligned}$$

sapendo che $\mathbf{w}^T\mathbf{w} = \|\mathbf{w}\|_2$ allora:

$$\begin{aligned} H^T H &= I - 4\mathbf{w}\mathbf{w}^T + 4\mathbf{w}\|\mathbf{w}\|_2\mathbf{w}^T = \\ &= I - 4\mathbf{w}\mathbf{w}^T + 4\mathbf{w}\mathbf{w}^T = I \end{aligned}$$

Prendendo un vettore $\mathbf{x}$ vogliamo determinare $\mathbf{w}$ in modo tale che: $H\mathbf{x} = k\mathbf{e}_1$ dove $\mathbf{e}_1$ è il primo versore della base canonica di R^n e $k \in R$. La costruzione di H consiste in tre fasi:

1. Poiché H è ortogonale segue che:

   $$\sigma = \|\mathbf{x}\| = \|H\mathbf{x}\| = \|k\mathbf{e}_1\| = |k|$$

   perciò $k = \pm \sigma$

2. Dalla definizione di H si ha:

   $$H\mathbf{x} = (I - 2\mathbf{w}\mathbf{w}^T)\mathbf{x} = \mathbf{x} - 2\mathbf{w}\mathbf{w}^T\mathbf{x} = \mathbf{x} - 2\mathbf{w}(\mathbf{w}^T\mathbf{x})$$

   essendo $(\mathbf{w}^T\mathbf{x})$ uno scalare, $H\mathbf{x} = \mathbf{x} - 2(\mathbf{w}^T\mathbf{x})\mathbf{w} = k\mathbf{e}_1$

   allora: $$w = \frac{x - ke_1}{2w^T x} = \frac{x - ke_1}{\|x - ke_1\|}$$

   in quanto $\mathbf{w}$ norma unitaria.

3. Dalla relazione

   $$\mathbf{x}^T H\mathbf{x} = \mathbf{x}^T(\mathbf{x} - 2\mathbf{w}\mathbf{w}^T\mathbf{x}) = \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{w}\mathbf{w}^T\mathbf{x} =$$

$$= \|\mathbf{x}\|_2 - 2(\mathbf{x}^T\mathbf{w})(\mathbf{w}^T\mathbf{x}) = \sigma^2 - 2(\mathbf{w}^T\mathbf{x})^2$$

quindi

$$\sigma^2 - 2(\mathbf{w}^T\mathbf{x})^2 = \mathbf{x}^T(k\mathbf{e}_1)$$
$$\sigma^2 - 2(\mathbf{w}^T\mathbf{x})^2 = k(\mathbf{x}^T\mathbf{e}_1)$$
$$\sigma^2 - 2(\mathbf{w}^T\mathbf{x})^2 = kx_1$$ dove x_1 è la prima componente del vettore **x**

dall'ultima relazione l'unica incognita è: $\mathbf{w}^T\mathbf{x}$ perciò:

$$\left(w^T x\right) = \sqrt{\frac{\sigma^2 - kx_1}{2}}$$

$$2\left(w^T x\right) = 2\sqrt{\frac{\sigma^2 - kx_1}{2}} \quad \rightarrow \quad 2\left(w^T x\right) = \sqrt{2\left(\sigma^2 - kx_1\right)} = \left\|x - ke_1\right\|$$

l'uso di questa espressione permette di ridurre il carico computazionale richiesto per la normalizzazione di **w**.
Per evitare pericoli di cancellazione è bene scegliere il segno di k, lasciato fino a ora arbitrario, in modo tale che $-kx_1$ sia positivo, questo porta alle due espressioni:

$$k = -\text{sign}(x_1)\sigma$$

$$\|\mathbf{x} - k\mathbf{e}_1\| = [2\sigma(\sigma + |x_1|)]^{1/2}$$

L'algoritmo che ne risulta è il seguente:

1. $\sigma = \|\mathbf{x}\|$
2. $k = -\text{sign}(x_1)\sigma$
3. $\lambda = [2\sigma(\sigma + |x_1|)]^{1/2}$
4. $\mathbf{w} = (\mathbf{x} - k\mathbf{e}_1)/\lambda$
5. $H = I - 2\mathbf{w}\mathbf{w}^T$

In genere si preferisce scrivere la matrice *H* in un'altra forma, in modo da eliminare la radice quadrata e la normalizzazione del vettore **w**, e di conseguenza si ridurranno anche gli errori introdotti dall'algoritmo, vediamo come:

$$2ww^T = \frac{\left(x - ke_1\right)}{\lambda} \cdot \frac{\left(x - ke_1\right)^T}{\lambda} = \frac{2}{\lambda^2} vv^T$$

Quindi la matrice *H*:

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

essendo: $\mathbf{v} = \mathbf{x} - k\mathbf{e}_1$ e $\beta = \sigma(\sigma + |x_1|) = \frac{2}{\lambda^2}$

L'algoritmo nella sua forma equivalente risulta:

1. $\sigma = \|\mathbf{x}\|$
2. $k = -\text{sign}(x_1)\sigma$
3. $\lambda = [2\sigma(\sigma + |x_1|)]^{1/2}$
4. $\mathbf{w} = (\mathbf{x} - k\mathbf{e}_1)/\lambda$
5. $H = I - \frac{1}{\beta} vv^T$

Inoltre, non è indispensabile costruire esplicitamente *H*, dato che la conoscenza di **w** o di **v**, la determina univocamente. Può essere utile applicare la matrice *H* ad un vettore **y**, questo può essere fatto nel seguente modo:

$$H\mathbf{y} = (I - 2\mathbf{w}\mathbf{w}^T)\mathbf{y} = \mathbf{y} - \mathbf{w}\mathbf{w}^T\mathbf{y} = \mathbf{y} - \gamma\mathbf{w}$$

dove $\gamma = 2\mathbf{w}^T\mathbf{y}$.

Utilizzando la seconda rappresentazione di *H*:

$$Hy = \left(I - \frac{1}{\beta} vv^T \right) y = y - \delta v$$

con $\delta = \frac{1}{\beta} v^T y$.

IMPLEMENTAZIONE IN MATLAB DELL'ALGORITMO PER LA COSTRUZIONE DELLA MATRICE DI HOUSEHOLDER H

```
function [H,k] = housemat(x)
n = size(x,1)
sigma = norm(x);
if x(1)>=0
    k = -sigma;
else
    k = sigma;
end
beta = sigma*(sigma+abs(x(1)));
e = [1;zeros((n - 1),1)];
v = x- k*e;
H = eye(n)-(1/beta)*v*v';
```

Fattorizzazione QR di Householder

Le matrici elementari di Householder permettono di costruire la fattorizzazione QR di una matrice A. Consideriamo che A sia una matrice quadrata, genereremo una successione di matrici $A^{(i)}$, $i = 1,\ldots,n$ in modo tale che $A^{(n)}$ sia una matrice triangolare superiore. Analizziamo passo per passo l'algoritmo.

- passo 1: scriviamo la matrice A in termini delle sue colonne

$$A = A^{(1)} = [\mathbf{a}_1^{(1)} \quad \mathbf{a}_2^{(1)} \quad \ldots \quad \mathbf{a}_n^{(1)}]$$

Applicando la tecnica vista precedentemente è possibile costruire la matrice elementare di Householder H_1 in modo che

$$H_1 \, \mathbf{a}_1^{(1)} = k_1 \mathbf{e}_1$$

Moltiplichiamo questa matrice per la matrice $A^{(1)}$ per costruire la prossima matrice della successione

$$A^{(3)} = H_2 A^{(2)} = \begin{bmatrix} 1 & 0^T \\ 0 & \hat{H}_2 \end{bmatrix} \begin{bmatrix} k_1 & v_1^T \\ 0 & \hat{H}_2 \hat{A}^{(2)} \end{bmatrix}$$

$$A^{(2)} = H_1 A^{(1)} = [\mathbf{a}_1^{(2)} \quad \mathbf{a}_2^{(2)} \quad \ldots \quad \mathbf{a}_n^{(2)}]$$

con $\mathbf{a}_1^{(2)} = k_1 \mathbf{e}_1$, $\mathbf{a}_j^{(2)} = H_1 \, \mathbf{a}_j^{(1)}$, $j = 2,\ldots,n$.

La matrice $A^{(2)}$ ha la struttura seguente:

$$A^{(2)} = \begin{bmatrix} k_1 & a_{12}^{(2)} & \cdots & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix} = \begin{bmatrix} k_1 & v_1^T \\ 0 & \hat{A}^{(2)} \end{bmatrix}$$

dove **0** è un vettore colonna di dimensioni appropriate che ha tutte le componenti nulle e $\hat{A}^{(2)}$ è una sottomatrice di dimensione $n - 1$;

- passo 2: consideriamo inizialmente la sottomatrice

$$\hat{A}^{(2)} = \begin{bmatrix} \hat{a}_2^{(2)} & \hat{a}_3^{(2)} & \cdots & \hat{a}_n^{(2)} \end{bmatrix}$$

Proseguiamo con la costruzione della matrice di Householder $\hat{H}_2$ di dimensione *n* - 1 tale che

$$\hat{H}_2 \hat{a}_2^{(2)} = k_2 e_1$$

dove il vettore $\mathbf{e}_1$ ha lunghezza *n* – 1. A questo punto, aggiungiamo alla matrice $\hat{H}_2$ una riga ed una colonna della matrice identità in modo da che la matrice raggiunga la dimensione *n* si otterrà perciò la seguente matrice:

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

questa matrice verrà moltiplicata a sinistra per $A^{(2)}$ ottenendo:

$$A^{(3)} = H_2 A^{(2)} = \begin{bmatrix} 1 & 0^T \\ 0 & \hat{H}_2 \end{bmatrix} \begin{bmatrix} k_1 & v_1^T \\ 0 & \hat{A}^{(2)} \end{bmatrix} = \begin{bmatrix} k_1 & v_1^T \\ 0 & \hat{H}_2 \hat{A}^{(2)} \end{bmatrix}$$

ovvero

$$A^{(3)} = \begin{bmatrix} k_1 & * & \cdots & \cdots & * \\ 0 & k_2 & * & \cdots & * \\ \vdots & 0 & & & \\ \vdots & \vdots & & \hat{A}^{(3)} & \\ 0 & 0 & & & \end{bmatrix}$$

Il passo successivo opererà sulla sottomatrice $\hat{A}^{(3)}$ di dimensione *n* – 2.

- passo i : la matrice prodotta dall'iterazione precedente sarà nella forma

$$A^{(i)} = \begin{bmatrix} k_1 & * & \cdots & \cdots & \cdots & * \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & k_{i-1} & * & \cdots & * \\ \vdots & & 0 & & & \\ \vdots & & \vdots & & \hat{A}^{(i)} & \\ 0 & \cdots & 0 & & & \end{bmatrix} = \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ 0 & \hat{A}^{(i)} \end{bmatrix}$$

dove gli asterischi indicano elementi della matrice che non verranno più modificati mentre la sottomatrice $\hat{A}^{(i)}$ è del tipo:

$$\hat{A}^{(i)} = \begin{bmatrix} \hat{a}_i^{(i)} & \hat{a}_{i+1}^{(i)} & \cdots & \hat{a}_n^{(i)} \end{bmatrix}$$

creiamo quindi una matrice $\hat{H}_i$ di dimensione $n - i + 1$ tale che

$$\hat{H}_i \hat{a}_i^{(i)} = k_i e_1$$

dopo aver aggiunto alla matrice $\hat{H}_i$ i -1 righe e colonne della matrice identità si avrà:

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

ossia

$$A^{(i)} = \begin{bmatrix} k_1 & * & \cdots & \cdots & \cdots & * \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & k_i & * & \cdots & * \\ \vdots & & 0 & & & \\ \vdots & & \vdots & & \hat{A}^{(i+1)} & \\ 0 & \cdots & 0 & & & \end{bmatrix}$$

Quando la matrice A è quadrata, l'algoritmo termina al passo $n - 1$ con la matrice triangolare superiore:

$$A^{(n)} = H_{n-1} A^{(n-1)} = \begin{bmatrix} k_1 & * & \cdots & * \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & * \\ 0 & \cdots & 0 & k_n \end{bmatrix} = R$$

L'intero algoritmo in formato matriciale si può riassumere nel seguente modo:

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

essendo la matrice $Q = H_1 H_2 \ldots H_{n-1}$ ortogonale, in quanto prodotto di matrici ortogonali, la precedente relazione implica:

$$A = QR$$

ovvero la fattorizzazione QR della matrice A.

Nel caso in cui la matrice A sia rettangolare di dimensione $m \times n$, con $m > n$, l'unica differenza rispetto al caso precedente è che è necessario un ulteriore passo che azzeri gli elementi della n-esima colonna di A. Dopo n passi otterremo la matrice:

$$R = A^{(n+1)} = \begin{bmatrix} k_1 & * & \cdots & * \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & * \\ \vdots & & \ddots & k_n \\ \vdots & & & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

con R_1 matrice $n \times n$ triangolare superiore, non singolare se la matrice A è a rango pieno.
Il legame tra R e A è:

$$R = H_n H_{n-1} \ldots H_1 A$$

da cui:

$$A = QR \quad \text{con } Q = H_1 H_2 .. H_n$$

La fattorizzazione QR di Householder nel caso $m = n$ è definita dal seguente algoritmo:

1. input A , n
2. $Q = I$
3. for $i = 1,\ldots,n\text{-}1$
    1. costruisci H_i
    2. $Q = QH_i$
    3. $A = H_i A$
4. output Q,A (A contiene il fattore triangolare R)

IMPLEMENTAZIONE IN MATLAB DELLA FATTORIZZAZIONE QR DI HOUSEHOLDER (MATRICE QUADRATA)

```
n = 5;                                  % dimensione della matrice A
A = rand(n);                            % genero una matrice casuale A di
                                        % dimensione n
Q = eye(n);                             % inizializzo la matrice Q
for i = 1:n-1                           % il ciclo for è costituito da n – 1
                                        % passi
    sottomatr = housemat(A(i:n,i));     % sottomatr è l'output della funzione
                                        % housemat
```

```
                                        % che prende in ingresso la
                                        % colonna i-esima della matrice A
                                        % e le righe dalla i alla n

    for j = i:n                         % aggiorno la matrice A
        A(i:n,j) = sottomatr*A(i:n,j);
    end

    H = eye(n,n);                       % prepariamo la matrice H nel caso in
                                        % cui sia necessaria
                                        % un' "orlatura"

    [m,m] = size(sottomatr);            % dimensione della matrice sottomatr
    if m ~= n
       H((n-m+1):n,(n-m+1):n)= sottomatr; % dobbiamo verificare se la dim.della
                                        % matrice sottomatr
                                        % è pari a n x n, in caso
                                        % contrario dobbiamo "orlare" la
                                        % matrice sottomatr inserendo
                                        % tante righe e colonne della
                                        % matrice identità in modo da
                                        % ottenere la dim. n x n
    else
        H = sottomatr;                  % se m = n non occorre nessuna
                                        % orlatura
    end
    Q = Q*H;                            % Q è data dal prodotto delle i
                                        % matrici H ottenute
                                        % R è uguale alla matrice A
end
```

La matrice *A* ottenuta dalla fattorizzazione è:

```
A =

   -1.6536   -1.1405   -1.2569   -1.1757   -1.1833
    0.0000    0.9661    0.6341    1.0098    0.7639
    0.0000   -0.0000   -0.8816   -0.3885    0.0277
   -0.0000   -0.0000    0.0000    0.1784    0.7007
   -0.0000   -0.0000   -0.0000   -0.0000   -0.0996
```

Come si può notare *A* è una matrice triangolare superiore e contiene la matrice *R*, essendo in questo caso la matrice *A* quadrata, *R* coincide proprio con *A*. Se la matrice *A* fosse stata rettangolare avrebbe avuto la forma:

$$R = A^{(n+1)} = \begin{bmatrix} k_1 & * & \cdots & * \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & * \\ \vdots & & \ddots & k_n \\ \vdots & & & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

Verifichiamo anche che *Q* è una matrice ortogonale:

```
Q*Q'

ans =

    1.0000    0.0000   -0.0000    0.0000   -0.0000
    0.0000    1.0000    0.0000   -0.0000   -0.0000
   -0.0000    0.0000    1.0000    0.0000    0.0000
    0.0000   -0.0000    0.0000    1.0000         0
   -0.0000   -0.0000    0.0000         0    1.0000
```

Matrice elementare di Givens

Nello spazio R^2 una rotazione piana è rappresentata dalla matrice:

$$G(\theta)=\begin{bmatrix}\cos(\theta) & \sin(\theta)\\ -\sin(\theta) & \cos(\theta)\end{bmatrix}$$

se questa matrice viene moltiplicata per un vettore **x**, questa matrice ruota di un angolo θ in senso orario.
La matrice elementare di Givens G_{ij} ($i < j$) opera in R^m una rotazione che lascia inalterato il sottospazio ortogonale al piano definito dai vettori $\mathbf{e}_i$ e $\mathbf{e}_j$ della base canonica. La matrice è definita come segue:

$$G_{ij}=\begin{bmatrix}1 & & & & & & \\ & \ddots & & & & & \\ & & c & \cdots & s & & \text{riga } i\\ & & \vdots & & \vdots & & \\ & & -s & \cdots & c & & \text{riga } j\\ & & & & & \ddots & \\ & & & & & & 1\end{bmatrix}$$

con $c^2 + s^2 = 1$.

Per applicare questa matrice ad un vettore non è necessario eseguire esplicitamente il prodotto matriciale poiché:

$$\mathbf{y} = G_{ij}\mathbf{x} \quad \rightarrow \quad y_k=\begin{cases} cx_i + sx_j, & k = i\\ -sx_i + cx_j, & k = j\\ x_k, & k \neq i, j\end{cases}$$

Si possono determinare i parametri c e s in modo tale da annullare la componente j-esima del vettore **x**, modificando solo la componente i-esima e lasciando inalterate tutte le altre. Infatti, imponendo $y_j = 0$ si ottiene:

$$\begin{cases} -sx_i + cx_j = 0 \\ c^2 + s^2 = 1 \end{cases} \rightarrow \begin{cases} sx_i = cx_j \\ c^2 + s^2 = 1 \end{cases} \rightarrow \begin{cases} s = \dfrac{x_j}{x_i} c \\ c^2 + \dfrac{x_j^2}{x_i^2} c^2 = 1 \end{cases}$$

$$\begin{cases} s = \dfrac{x_j}{x_i} c \\ c^2 \left(1 + \dfrac{x_j^2}{x_i^2} \right) = 1 \end{cases} \rightarrow \begin{cases} s = \dfrac{x_j}{x_i} c \\ c^2 \left(\dfrac{x_i^2 + x_j^2}{x_i^2} \right) = 1 \end{cases}$$

da cui risulta:

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

indicando con: $\sigma = \sqrt{x_i^2 + x_j^2}$ si ha $c = \dfrac{x_i}{\sigma}$ e $s = \dfrac{x_j}{\sigma}$

CALCOLO DEI PARAMETRI DI UNA ROTAZIONE DI GIVENS

function [c,s] = GIVROT(x_i,x_j)

1. if $x_j = 0$
   1. $c = 1$, $s = 0$
2. else if $|x_j| > |x_i|$
   1. $t = x_i / x_j$, $z = (1 + t^2)^{1/2}$
   2. $s = 1 / z$, $c = ts$
3. else
   1. $t = x_j / x_i$, $z = (1 + t^2)^{1/2}$
   2. $c = 1$, $s = tc$

L'implementazione in Matlab della funzione che calcola i parametri della rotazione di Givens è la seguente:

```
function [c,s] = GIVROT(x1, x2)
if x2 == 0
    c = 1;
    s = 0;
```

```
else
    if abs(x2) > abs(x1)
        t = x1/x2;
        z = sqrt(1 + t^2);
        s = 1/z;
        c = t*s;
    else
        t = x2/x1;
        z = sqrt(1 + t^2);
        c = 1/z;
        s = c*t;
    end
end
```

Tramite un'opportuna successione di matrici elementari di Givens si può trasformare un vettore di R^m analogamente all'azione di una matrice elementare di Householder.

Fattorizzazione QR di Givens

L'algoritmo di fattorizzazione QR di Givens consiste nell'azzerare tutti gli elementi del triangolo inferiore della matrice A a partire dal secondo elemento della prima colonna e proseguendo dapprima verso il basso con gli altri elementi della prima colonna, e poi con le successive colonne.
Se la matrice A è quadrata ci si ferma con l'ultimo elemento della penultima colonna, e il procedimento si rappresenta nella seguente forma matriciale:

$$G_{n-1,n}G_{n-2,n}G_{n-2,n-1} \dots G_{2n} \dots G_{23}G_{1n} \dots G_{13}G_{12}A = R$$

Se la matrice A è rettangolare con m righe e n colonne ($m > n$) occorre azzerare anche gli elementi subdiagonali dell'ultima colonna, e l'algoritmo diventa:

$$G_{n,m} \dots G_{n,n+1} \dots G_{2,m} \dots G_{23}G_{1m} \dots G_{13}G_{12}A = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

dove R_1 è una matrice triangolare superiore $n \times n$.
In entrambi i casi il prodotto delle matrici di Givens G_{ij} fornisce il fattore Q^T della fattorizzazione QR di A. Tale fattore può essere costruito esplicitamente, oppure possono essere memorizzati i parametri c_{ij} e s_{ij} che definiscono ciascuna matrice di rotazione G_{ij}, e che consentono quindi, all'occorrenza, di calcolare il prodotto $Q^T\mathbf{b}$ riapplicando al vettore b tutte le rotazioni già applicate alla matrice A.

FATTORIZZAZIONE QR DI GIVENS ($m > n$)

1. input A, m, n
2. for $k = 1, \dots, n$
    1. for $i = k + 1, \dots, m$
        1. $[c,s]$=GIVROT(a_{kk},a_{ik})
        2. for $j = k, \dots, n$

1. $t = ca_{kj} + sa_{ij}$
2. $a_{ij} = -sa_{kj} + ca_{ij}$
3. $a_{kj} = t$

3. $c_{ki} = c$, $s_{ki} = s$

3. output A (contiene il fattore triangolare R)
   C, S (contengono i parametri delle rotazioni)

IMPLEMENTAZIONE IN MATLAB DELLA FATTORIZZAZIONE QR DI GIVENS

```
m = 6;                                              % numero righe di A
n = 3;                                              % numero colonne di A
A =rand(m,n);                                       % generiamo una matrice A
                                                    % casuale
for k = 1:n
    for i = k+1: m
        [c, s] = givrot(A(k,k), A(i,k));            % generiamo i parametri di
                                                    % rotazione di Givens
        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;
            Q = eye(m);
            G = eye(m);
            G(i,i) = c;
            G(j,j) = c;
            G(i,j) = s;
            G(j,i) = -s;
            Q = G*Q;
        end
    end
end
```

Eseguendo l'algoritmo la matrice A fattorizzata risulta:

```
A =

    1.4959    0.6305    1.2389
         0    0.9233    0.4141
         0         0   -0.4715
    0.0000   -0.0000    0.0000
         0    0.0000         0
   -0.0000         0   -0.0000
```

Verifichiamo ancora che Q è ortogonale:

```
Q*Q'

ans =

    1.0000         0         0         0         0         0
         0    1.0000         0         0         0         0
         0         0    1.0000         0         0         0
         0         0         0    1.0000         0         0
         0         0         0         0    1.0000         0
         0         0         0         0         0    1.0000
```

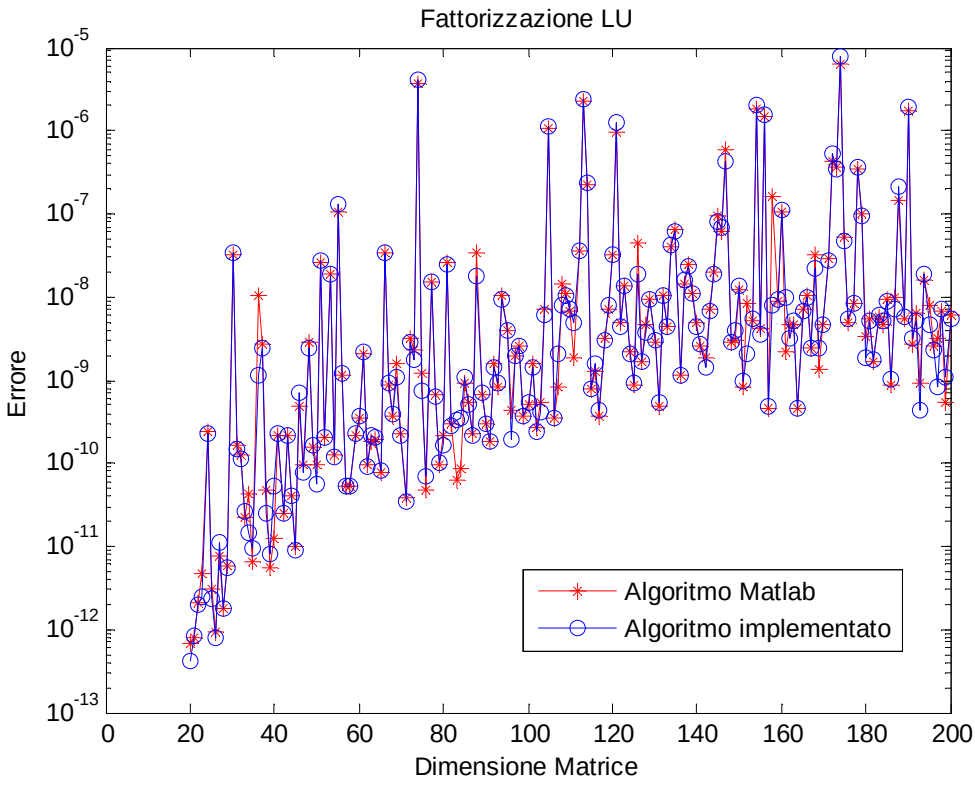
Confronto tra alcuni algoritmi di Matlab e i corrispondenti da noi implementati

FATTORIZZAZIONE LU

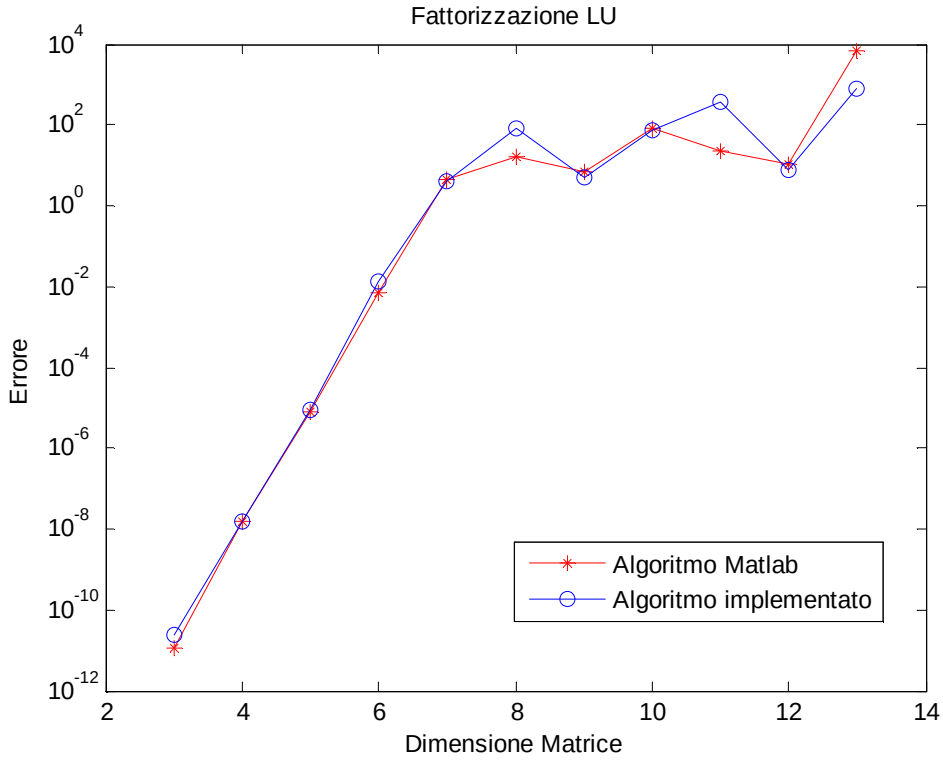
```
for n = 20:200;
A = rand(n);
A = A'*A;
sol = ones(n,1);
b = A*sol;

% Risoluzione con la funzione "lu" di Matlab
[L, U] = lu(A);
y = L\b;
x = U\y;
% Errore usando usando la funzione "lu" di Matlab
errore1(n) = norm(x - sol);

% Risoluzione con l'algoritmo implementato
L = eye(n);
for k = 1:n-1
    for i = k+1:n
        mik = A(i,k)/A(k,k);
        if i > k
            L(i,k) = mik;
        end
        for j = k+1:n
            A(i,j) = A(i,j) - mik*A(k,j);
        end

    end
end
U = triu(A);
for i = 1:n
    for j = 1:n
        if i < j
           L(i,j) = 0;
        end
    end
end
y = L\b;
x = U\y;
% Errore usando usando l'algoritmo da noi implementato
errore2(n) = norm(x - sol);
end

%Grafico errore
t = linspace(1, n, n);
semilogy(t,errore1,'r-*',t,errore2,'b-o');
legend('Algoritmo Matlab','Algoritmo implementato');
xlabel('Dimensione Matrice');
ylabel('Errore');
title('Fattorizzazione LU');
```

Il grafico mette a confronto l'errore tra la soluzione "e" imposta da noi e la soluzione "x" ottenuta eseguendo gli algoritmi considerando sia l'algoritmo esistente già in Matlab sia

quello implementato da noi. Come si può veder i due andamenti sono pressoché uguali e l'errore è molto basso oscilla in un range tra 10^{-13} e 10^{-5}.



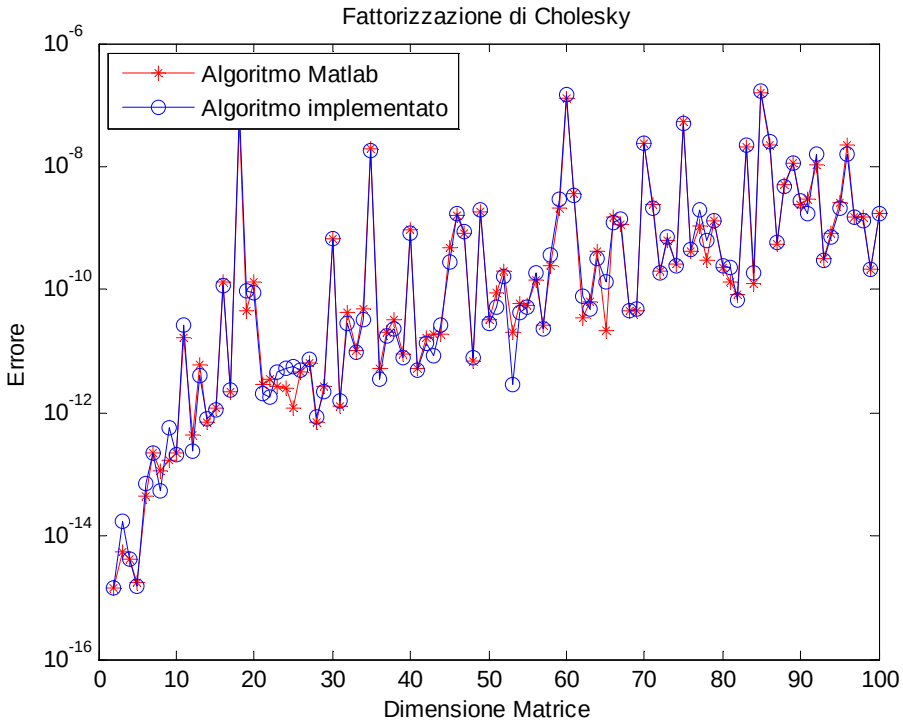
Utilizziamo adesso una matrice di Hilbert, che per definizione è mal condizionata, il comando matlab che genera la matrice di Hilbert è: A = hilb(n), consideriamo come esempio una matrice con dimensione n che varia tra 3 e 13:



FATTORIZZAZIONE DI CHOLESKY

```
for n = 1:100
A = rand(n); % poiché con il comando “rand” si ottiene una matrice simmetrica
             % definita positiva, sarà sicuramente rispettata la condizione
             % esplicitata quando abbiamo definito la funzione: “funzchol”
A = A'*A;
e = ones(n,1);
b = A*e;

% Risoluzione con la funzione "chol" di Matlab
R = chol(A);
y = R'\b;
x = R\y;
% Errore usando usando la funzione "chol" di Matlab
errore1(n) = norm(x - e);

% Risoluzione con l'algoritmo implementato
R = funzchol(A);
y = R'\b;
x = R\y;
% Errore usando usando l'algoritmo da noi implementato
errore2(n) = norm(x - e);
end

%Grafico errore
t = linspace(1, n, n);
semilogy(t,errore1,'r-*',t,errore2,'b-o');
legend('Algoritmo Matlab','Algoritmo implementato');
xlabel('Dimensione Matrice');
ylabel('Errore');
title('Fattorizzazione di Cholesky');
```

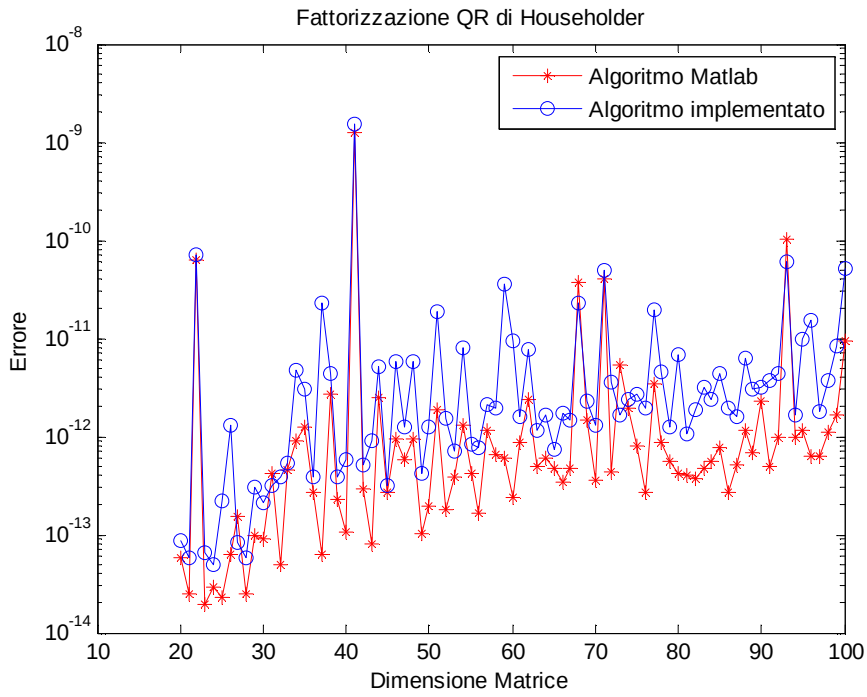
Matlab ci restituisce il seguente grafico, come si può osservare lo scostamento tra i due algoritmi è minimo, con un errore che oscilla tra 10^{-16} e 10^{-6}.



FATTORIZZAZIONE QR DI HOUSEHOLDER

```
for n = 20:100;
A = rand(n);
sol = ones(n,1);
b = A*sol;

% Risoluzione con la funzione "qr" di Matlab
[Q, R] = qr(A);
c = Q'*b;
x = R\c;
% Errore usando usando la funzione "qr" di Matlab
errore1(n) = norm(x - sol);

% Risoluzione con l'algoritmo implementato
Q = eye(n);
for i = 1:n-1
    sottomatr = housemat(A(i:n,i));
    for j = i:n
        A(i:n,j) = sottomatr*A(i:n,j);
    end
    H = eye(n,n);
    [m,m] = size(sottomatr);
    if m ~= n
       H((n-m+1):n,(n-m+1):n)= sottomatr;
    else
        H = sottomatr;
    end
    Q = Q*H;
end
R = A;
c = Q'*b;
x = R\c;
% Errore usando usando l'algoritmo da noi implementato
errore2(n) = norm(x - sol);
end

%Grafico errore
t = linspace(1, n, n);
semilogy(t,errore1,'r-*',t,errore2,'b-o');
legend('Algoritmo Matlab','Algoritmo implementato');
xlabel('Dimensione Matrice');
ylabel('Errore');
title('Fattorizzazione QR di Householder');
```

come si può vedere dal grafico che segue l'andamento dell'errore prodotto dai due algoritmi è simile, anche se l'algoritmo relativo alla funzione di Matlab produce un errore leggermente più piccolo, questo è dovuto anche al fatto che abbiamo utilizzato una matrice *A* random che risulta essere una matrice ben condizionata, ciò vuol dire che un errore in ingresso non viene amplificato eccessivamente sul risultato.



Se invece di utilizzare una matrice random utilizziamo una matrice di Hilbert, che per definizione è mal condizionata otteniamo un errore in uscita molto elevato, come si può notare dal seguente grafico, in cui l'errore oscilla tra 10^{-15} e 10^{2}.