Metodi diretti per la risoluzione di sistemi lineari

Marco Simone
Alessandro Schirra

6 aprile 2011

Indice

1 Introduzione ai sistemi lineari

Un sistema di n equazioni lineari in n incognite assume la forma

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1\mathrm{n}}x_\mathrm{n} = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2\mathrm{n}}x_\mathrm{n} = b_2 \\ \cdots \\ a_{\mathrm{n}1}x_1 + a_{\mathrm{n}2}x_2 + \cdots + a_{\mathrm{nn}}x_\mathrm{n} = b_\mathrm{n} \end{cases}$$

Esiste una rappresentazione equivalente di tipo matriciale

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

Un sistema lineare viene spesso rappresentato in forma compatta come

$$\mathbf{Ax} = \mathbf{b}$$

con

- $\mathbf{A} \in \mathbb{R}^{\mathrm{nxn}}$ matrice dei coefficenti;
- $\mathbf{x} \in \mathbb{R}^\mathrm{n}$ soluzione;
- $\mathbf{b} \in \mathbb{R}^\mathrm{n}$ vettore dei termini noti;

Un sistema lineare ammette una e una sola soluzione (problema ben posto) se e solo se è verificata una delle seguenti proprietà equivalenti:

1. il determinante della matrice A è diverso da zero (matrice non singolare);
2. il rango di A è uguale a n (matrice a rango pieno);
3. il sistema omogeneo
$$A\mathbf{x} = 0$$
ammette la sola soluzione banale ($x_\mathrm{i} = 0, i = 1, \ldots, n$)

Un aspetto importante di cui tener conto nello studio di un metodo di risoluzione per un sistema lineare è il **condizionamento**. Sia δd una perturbazione dei dati d di un problema e sia δx la corrispondente perturbazione sulla sua soluzione x. Sia inoltre $||.||$ una qualsiasi norma vettoriale. Il numero di condizionamento assoluto $K = K(d)$ è definito dalla relazione

$$||\delta x|| \leq K||\delta d||$$

mentre il numero di condizionamento relativo (o più semplicemente **numero di condizionamento**) $k = k(d)$ verifica la diseguaglianza

$$\frac{||\delta x||}{||x||} \leq k\frac{||\delta d||}{||d||}$$

Il condizionamento è una misura di quanto un errore sui dati è amplificato nei risultati. Nell'applicazione a sistemi lineari ha particolare interesse il condizionamento di una matrice A. Si definisce numero di condizionamento di una matrice, relativamente alla risoluzione di un sistema lineare, la quantità

$$k(A) = ||A||||A||^{-1}$$

In generale un algoritmo propaga gli errori. Si definisce **stabile** un algoritmo la cui successione di operazioni non amplifica eccessivamente gli errori. Un algoritmo stabile è possibile solo in presenza di problemi ben condizionati.

Un altro aspetto di cui tenere conto nella valutazione delle prestazioni di un algoritmo di risoluzione è la **complessità computazionale**. La complessità computazionale di un algoritmo è il numero di operazioni in virgola mobile necessarie per risolvere un problema mediante l'algoritmo stesso.

Per la risoluzione di sistemi di equazioni lineari sono disponibili diverse tipologie di algoritmi, tra questi i cosiddetti **metodi diretti**. Un metodo diretto consiste nel trasformare attraverso un numero finito di iterazioni un sistema lineare generico in un sistema lineare equivalente dotato di una struttura particolare che ne semplifichi la risoluzione. Si considera come esempio di fattorizzazione LU la fattorizzazione di Cholesky, e come caso di fattorizzazione QR la fattorizzazione di Householder.

2 Fattorizzazione LU

Considerata una matrice $A \in^{n \times n}$, una fattorizzazione LU fattorizza la matrice A nella forma

$$A = LU$$

con
L matrice $n \times n$ triangolare inferiore
U matrice $n \times n$ triangolare superiore
Nota la fattorizzazione, la risoluzione del sistema lineare $Ax = b$ può essere ricondotta alla risoluzione in cascata di due sistemi triangolari

$$Ax = B \qquad \rightarrow \qquad \begin{cases} Ly = b \\ Ux = y \end{cases}$$

2.1 Fattorizzazione LU di Cholesky

La decomposizione di Cholesky è la fattorizzazione di una matrice hermitiana e definita positiva in una matrice triangolare inferiore e nella sua trasposta coniugata.

Sia $A = LU$. Se si pone

$$D = diag(u_{11}, \ldots, u_{nn}) \qquad , \qquad R = D^{-1}U$$

la matrice A risulta decomposta nella forma

$$A = LDR$$

con L triangolare inferiore, R triangolare superiore, con elementi diagonali

$$l_{ii} = r_{ii} = 1 \qquad i = 1, \ldots, n$$

Se A è simmetrica risulta che $R = L^{\mathrm{T}}$ e quindi

$$A = LDL^{\mathrm{T}}$$

Si definisce **inerzia** di una matrice A hermitiana la terna di numeri naturali

$$Inerzia(A) := \{\Pi, \nu, \delta\}$$

che indicano rispettivamente il numero di autovalori positivi, negativi, e nulli della matrice A.
Vale il teorema seguente

Legge di Inerzia di Sylvester
Siano A e B due matrici Hermitiane. Esiste una matrice C non singolare tale che $B = C^*AC$ se e solo se

$$Inerzia(A) = Inerzia(B)$$

Se la matrice A è simmetrica definita positiva, applicando il teorema suddetto ad una fattorizzazione di tipo $A = LDL^{\mathrm{T}}$ si può concludere che tutti gli autovalori della matrice D sono positivi, ed è quindi possibile definire una matrice

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

e una matrice R

$$R = D^{1/2}L^{\mathrm{T}} \quad , \quad r_{\mathrm{ii}} > 0$$

tale che

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

La fattorizzazione $A = R^{\mathrm{T}}R$ della matrice simmetrica definita positiva A viene detta **fattorizzazione di Cholesky**.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} r_{11} & & & & \\ \vdots & \ddots & & & \\ r_{1i} & \cdots & r_{ii} & & \\ \vdots & & & \ddots & \\ r_{1n} & \cdots & r_{1n} & \cdots & r_{nn} \end{bmatrix} \begin{bmatrix} r_{11} & \cdots & r_{1j} & \cdots & r_{1n} \\ & \ddots & & & \vdots \\ & & r_{jj} & \cdots & r_{jn} \\ & & & \ddots & \vdots \\ & & & & r_{nn} \end{bmatrix}$$

E' possibile esprimere gli elementi del triangolo superiore di A in funzione degli elementi di R

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

Viceversa, eplicitando l'ultimo termine della sommatoria, e distinguendo i casi $i < j$ e $i = j$, si possono ricavare gli elementi di R in funzione di quelli di A

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

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

La proprietà di matrice definita positiva garantisce un radicando sempre positivo.

2.2 Algoritmo di fattorizzazione di Cholesky

L'algoritmo *chol.m* presente nelle librerie di MATLAB è implementato per colonne, la matrice R viene costruita colonna per colonna nel modo seguente

1. for $j = 1, \ldots, n$
   (a) for $i = 1, \ldots, j-1$
      i. $r_{ij} = \frac{1}{r_{ii}} \left(a_{ij} - \sum_{k=1}^{j-1} r_{ki} r_{kj} \right) \quad , \quad i < j$
   (b) $r_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2 \right)^{\frac{1}{2}} \quad , \quad i < j$

Ciò è dovuto al fatto che in MATLAB (così come in FORTRAN, linguaggio in cui è scritta la funzione chol.m) le matrici sono memorizzate per colonne. Pertanto un algoritmo con questa struttura presenta dei vantaggi in termini di allocazione di memoria e di swap, soprattutto in caso di matrici di grande dimensione.

La nostra implementazione dell'algoritmo in codice MATLAB segue lo stesso approccio, così come mostrato nella funzione *mychol.m*:

```
function R = mychol(A)
  [n,n] = size(A);
  R = zeros(n);
  R(1,1) = sqrt(A(1,1));
  for j=2:n
    for i=1:j-1
      R(i,j) = ( A(i,j) - R([1:i-1],i)' * R([1:i-1],j) )./ R(i,i);
    end
    R(j,j) = sqrt( A(j,j) - ( (R([1:j-1],j))' * R([1:j-1],j) ) );
end
```

E' stata sfruttata la sintassi vettoriale di MATLAB per esprimere la sommatoria in k ed eliminare un ciclo *for*. E' possibile ottimizzare ulteriormente il codice eliminando anche il secondo ciclo for, ma solo costruendo la matrice per righe e non per colonne. Infatti è facile vedere come l'elemento

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

dipende dagli elementi

$$r_{kj} \qquad\qquad k = 1, \ldots, i-1$$

ad esso soprastanti lungo la colonna j-esima. Quindi non è possibile, fissata la colonna j, determinare tutti assieme gli elementi di tale colonna con un'unica istruzione vettoriale MATLAB, poichè devono essere calcolati in successione uno dopo l'altro.

E' tuttavia possibile determinare tutti gli elementi della riga i-esima con un'unica operazione vettoriale, purchè si contruisca la matrice riga per riga a partire dalla prima (cioè fissando i). Considerando ad esempio la prima riga della matrice R, gli elementi $r_{12}, r_{13} \ldots r_{1\mathrm{n}}$ (la prima riga a eccezione dell'elemento diagonale) sono pari a

$$r_{1\mathrm{j}} = \frac{1}{r_{11}} a_{1\mathrm{j}} \qquad\qquad j = 2, \ldots, n$$

dove r_{11} (determinato in precedenza) e gli a_{ij} sono noti. E' quindi possibile determinare questa riga con l'istruzione MATLAB

$$R(1, 2:n) = A(1, 2:n)/R(1,1)$$

Per quanto riguarda invece la seconda riga, i suoi elementi alla destra della diagonale sono pari a

$$r_{2\mathrm{j}} = \frac{1}{r_{22}}(a_{2\mathrm{j}} - r_{12} r_{1\mathrm{j}}) \qquad\qquad j = 3, \ldots, n$$

e dipendono dagli elementi $r_{1\mathrm{j}}$ della prima riga, determinati tutti al passo precedente. L'istruzione vettoriale in MATLAB che descrive questa operazione è

$$R(2, 3:n) = (A(2, 3:n) - R(1,2) * R(1, 3:n))/R(i,i)$$

Applicando questo ragionamento a tutta la matrice si ottiene il seguente algoritmo ottimizzato:

```
function R=optchol(A)
  [n,n] = size(A);
  R=zeros(n);
  for i=1:n
    R(i,i)=sqrt(A(i,i)-((R([1:i-1],i))'*R([1:i-1],i)));
    R(i,i+1:n)=(A(i,i+1:n)-(R(1:i-1,i))'*R(1:i-1,i+1:n )/R(i,i);
  end
```

Lo svantaggio in termini di ottimizzazione nell'allocazione della memoria è, come mostreremo tra poco, ampiamente compensato da un notevole aumento delle prestazioni nell'esecuzione, almeno per matrici relativamente piccole. Allo scopo di testare la velocità dei due algoritmi e confrontarli con la versione compilata in MATLAB, è stato scritto lo script che segue.

```
%test velocità Cholesky

N=20; %numero iterazioni
s=50; %step
T=zeros(N,3);

for i = 1:N
  n=s*i
  A=rand(n);
  A=A'*A;
```

```
R=zeros(n,n); %preallocazione e inizializzazione matrice R

%Cholesky non ottimizzato
tic
R=mychol(A);
T(i,1)=toc;

%Cholesky ottimizzato
tic
R=optchol(A);
T(i,2)=toc;

%Cholesky built-in
tic
R=chol(A);
T(i,3)=toc;
end

%grafico tempi
n=linspace(s,s*N,N);
plot(n,T(:,1),'r-o',n,T(:,2),'b-o',n, T(:,3),'g-o')
legend('Cholesky non ottimizzato', 'Cholesky ottimizzato', 'Cholesky built-in')
```

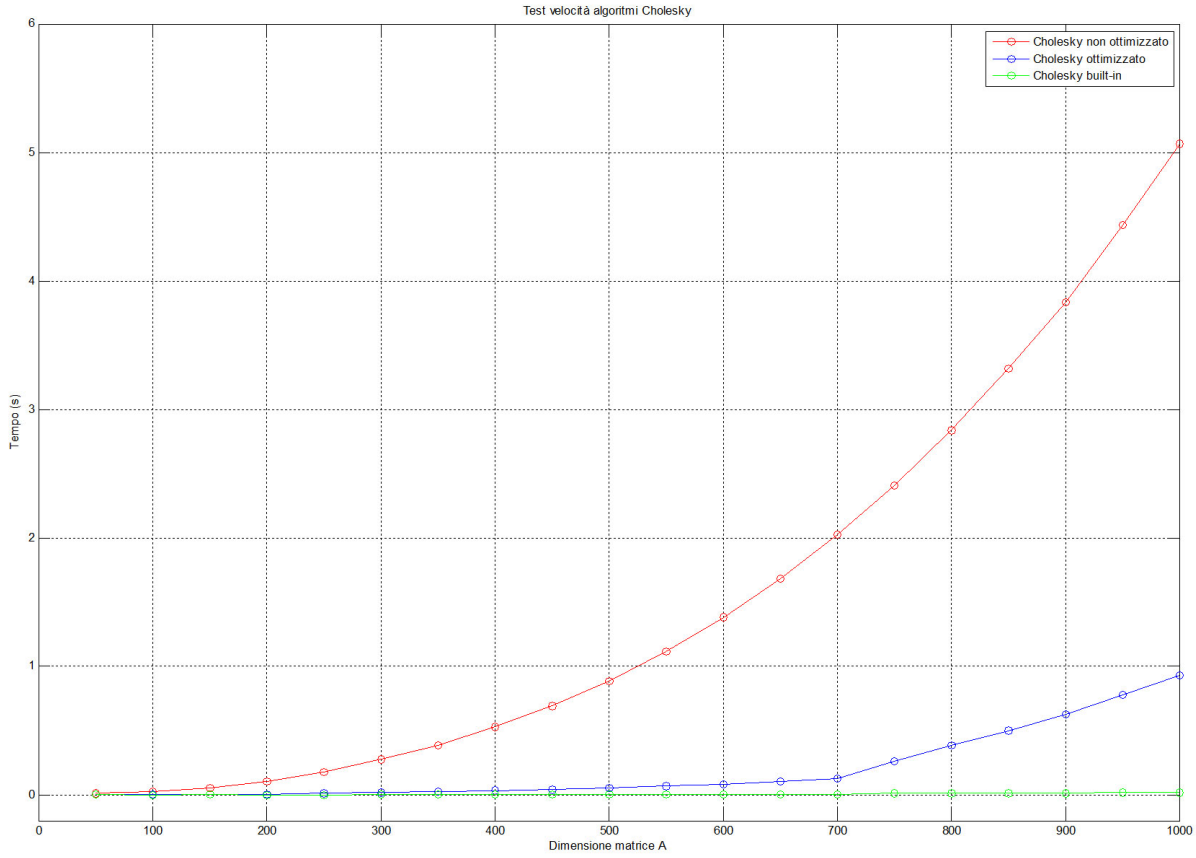

Figura 1: Comparativa dei tempi di esecuzione tra i diversi algoritmi di Cholesky.

Come si osserva in figura 1, l'algoritmo compilato in MATLAB *chol.m* rimane il più veloce, seguito da *optchol.m* e infine da *mychol.m*, di gran lunga il più lento.

Per quanto riguarda l'analisi della stabilità degli algoritmi, è stato implementato il seguente script:

```
%test velocità Cholesky

N=20; %numero iterazioni
s=10; %step matrice random
Err=zeros(N,3); %preallocazione vettore errore (matrice casuale)
ErrH=zeros(11,3); %preallocazione vettore errore (matrice Hilbert)

%matrice casuale simmetrica definita positiva
for i = 1:N
  n=s*i
  %creazione matrice e vettore dei termini noti
  A=rand(n);
  A=A'*A;
  sol=ones(n,1);
  b=A*sol;

  %Cholesky non ottimizzato
  R=mychol(A);
  y=R'\b;
  x=R\y;
  Err(i,1)=norm(x-sol);

  %Cholesky ottimizzato
  R=optchol(A);
  y=R'\b;
  x=R\y;
  Err(i,2)=norm(x-sol);

  %Cholesky built-in
  R=chol(A);
  y=R'\b;
  x=R\y;
  Err(i,3)=norm(x-sol);
end

%matrice di Hilbert
for i = 1:11
  m=i+2
  %creazione matrice e vettore dei termini noti
  H=hilb(m);
  sol=ones(m,1);
  b=H*sol;

  %Cholesky non ottimizzato
  R=mychol(H);
  y=R'\b;
  x=R\y;
  ErrH(i,1)=norm(x-sol);

  %Cholesky ottimizzato
  R=optchol(H);
  y=R'\b;
  x=R\y;
  ErrH(i,2)=norm(x-sol);

  %Cholesky built-in
  R=chol(H);
```

```
  y=R'\b;
  x=R\y;
  ErrH(i,3)=norm(x-sol);
 end

%grafici
n=linspace(s,s*N,N);
m=[3:1:13];

subplot(2,1,1); semilogy(n,Err(:,1),'r-o',n,Err(:,2),'g-o',n,Err(:,3),'b-o')
title('Matrice Casuale Definita Positiva')
legend('Cholesky non ottimizzato','Cholesky ottimizzato', 'Cholesky built-in');

subplot(2,1,2); semilogy(m,ErrH(:,1),'r-o',m,ErrH(:,2),'g-o',m,ErrH(:,3),'b-o')
title('Matrice di Hilbert')
legend('Cholesky non ottimizzato','Cholesky ottimizzato', 'Cholesky built-in');
```

Come si osserva in figura 2 tutti e tre gli algoritmi risultano ugualmente stabili, sia nel caso di una matrice casuale che nel caso di una matrice a condizionamento elevato come la matrice di Hilbert. L'implementazione dell'algoritmo di Cholesky presentata in *optchol.m* risulta quindi molto più veloce di quella in *mychol.m* senza alcuna perdita di stabilità.

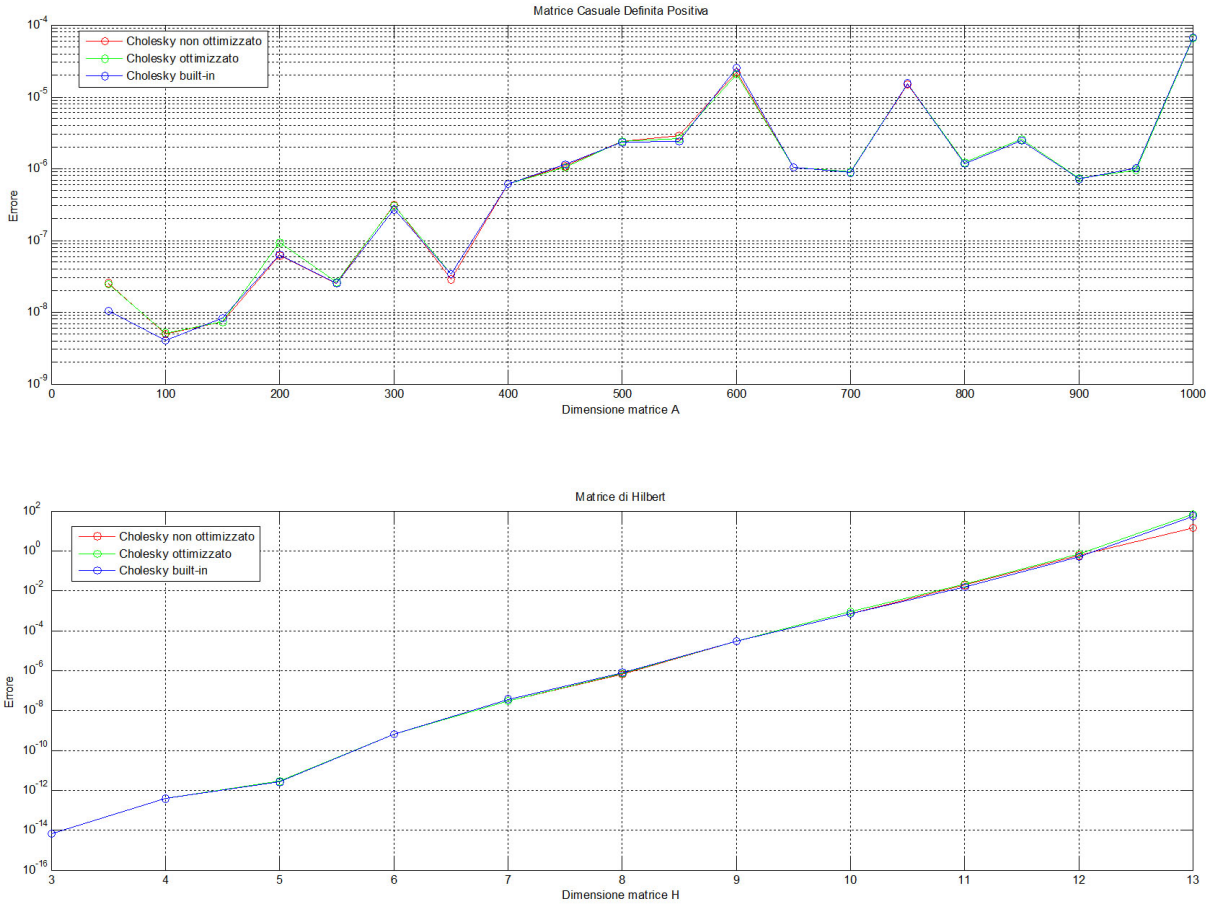

Figura 2: Errore in norma-2 dei diversi algoritmi di Cholesky

3 Fattorizzazione QR

Una fattorizzazione QR fattorizza una matrice A di dimensione $m \times n$ nella forma

$$A = QR$$

con Q matrice $m \times m$ ortogonale e R matrice $m \times n$ triangolare superiore.
Se A è una matrice quadrata non singolare, è possibile utilizzare la fattorizzazione QR per la risoluzione del sistema lineare

$$Ax = b$$

Sostituendo alla matrice A la sua forma fattorizzata, si trasforma il problema in una coppia di sistemi più semplici

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

in cui dal primo si trova facilmente

$$y = Q^{-1}b = Q^{\mathrm{T}}b$$

e sostituendo nel secondo il valore di y

$$Rx = y$$

si risolve per sostituzione all'indietro.

Un vantaggio della fattorizzazione QR rispetto alla LU riguarda la crescita del condizionamento.

Teorema Se $A = QR$, oppure $A = RQ$, allora $||A||_2 = ||R||_2$.

Teorema La matrice R ha lo stesso numero di condizionamento della matrice A rispetto alla norma-2.

3.1 Fattorizzazione QR di Householder

La fattorizzazione di Householder è' un algoritmo di triangolazione: a partire da una matrice quadrata A si genera una successione di matrici $A^{(\mathrm{i})} \quad i = 1, 2, \ldots, n$ tale che $A^{(\mathrm{n})}$ sia una matrice tiangolare superiore.

Matrice elementare di Householder Una matrice elementare H di Householer è del tipo

$$H = I - 2\mathbf{w}\mathbf{w}^{\mathrm{T}}, \qquad \mathbf{w} \in \mathbb{R}^{\mathrm{n}} \quad , \quad ||\mathbf{w}|| = \mathbf{1}$$

dove la norma utilizzata è solitamente quella euclidea. $2\mathbf{w}\mathbf{w}^{\mathrm{T}}$ è una matrice di dimensione $n \times n$ di rango unitario. Una matrice elementare di Householder è simmetrica e ortogonale, ossia

$$H = H^{\mathrm{T}} \quad , \quad H^*H = H^{\mathrm{T}}H = I$$

Assegnato il generico vettore $\mathbf{x} \in \mathbb{R}^{\mathrm{n}}$ si vuole determinare il vettore $\mathbf{w}$ tale che

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

dove $\mathbf{e_1}$ è il primo vettore della base canonica di $\mathbb{R}^{\mathrm{n}}$ e $k \in \mathbb{R}$.
L'algoritmo di costruzione della matrice elementare di Householder è il seguente

1. $\sigma = ||x||$
2. $k = -sign(x_1)\sigma$

3. $\lambda = \sqrt{2\sigma(\sigma + |x_1|)}$
4. $\mathbf{w} = (\mathbf{x} - k\mathbf{e_1})/\lambda$
5. $H = I - 2\mathbf{w}\mathbf{w}^{\mathrm{T}}$

ed è stato implementato in MATLAB come segue:

```
function [w,k]=house(x)
  n=size(x,1);
  s=norm(x,2);
  e1=zeros(n,1);
  e1(1)=1;
  if x(1)==0
    k=s;
  else
    k=-sign(x(1))*s;
l=sqrt( 2*s*(s+abs(x(1))) );
w=(x-k*e1)/l;
end
```

Fattorizzazione di Householder L'algoritmo prevede $n-1$ passi in caso di matrice quadrata (dove n è la dimensione della matrice) o n passi nel caso di matrice rettangolare $m \times n \quad (m > n)$.

Si descrive per semplicità il solo passo i-esimo per una matrice quadrata. La matrice generata dall'iterazione precedente è

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

dove gli asterischi indicano termini in generale non nulli che nn verranno modificati nelle iterazioni successive. La sottomatrice viene descritta in termini delle sue colonne

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

Si crea la matrice elementare di Householder $\hat{H}_{\mathrm{i}}$ di dimensione $n - i + 1$ tale che

$$\hat{H}_{\mathrm{i}} \hat{\mathbf{a}}_{\mathrm{i}}^{(\mathrm{i})} = k_{\mathrm{i}} \mathbf{e}_{\mathrm{i}}$$

Si ottiene la matrice H_{i} orlando la $\hat{H}_{\mathrm{i}}$ con $i-1$ righe e colonne della matrice identità e si moltiplica a sinistra questa matrice per $A^{(\mathrm{i})}$, ottenendo

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

ossia

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

Nel caso di una matrice quadrata $n \times n$ l'algoritmo termina al passo $n-1$ con la matrice triangolare superiore

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

L'algoritmo espresso in forma matriciale assume la forma

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

La matrice

$$Q = H_1 H_2 \cdots H_{n-1}$$

è ortogonale in quanto prodotto di matrici ortogonali, pertanto la precedente relazione implica che

$$A = QR$$

che costituisce la **fattorizzazione** QR della matrice A.

3.2 Algoritmo di fattorizzazione QR di Householder

L'algoritmo che costruisce le matrici Q, R ha la forma seguente

1. input A,n
2. $Q = I$
3. for $i = 1, 2, \ldots, n-1$
   1. costruisci H_{i}
   2. $Q = Q * H_{\mathrm{i}}$
   3. $A = H_{\mathrm{i}} A$
4. output Q, A

in cui al passo 3.1 viene richiamata la subroutine che crea la matrice elementare di Householder. Una prima semplice implementazione in codice MATLAB è contenuta nella funzione *myqr.m*:

```
function [Q,R] = myqr(A)
  n=size(A);
  Q=eye(n);
  R=A;
  for i=1:n-1
    [w,k]=house( R(i:n,i) );  %calcolo parametri w k Householder
    H=eye(n);
    H(i:n,i:n)=H(i:n,i:n)-2*w*w';  %creazione matrice Householder con orlatura
    Q=Q*H;  %aggiornamento matrice Q
     R=H*R;  %aggiornamento matrice R
  end
end
```

La routine *house.m* è quella vista in precedenza. Si noti che questa funzione calcola la fattorizzazione corretta solo per matrici quadrate, e che esegue due prodotti matrice $\times$ matrice a ogni passo, risultando computazionalmente molto onerosa. E' possibile ottimizzare questo codice esprimendo tali prodotti in altra forma. Infatti:

$$Q^{(i+1)} = Q^{(i)} H_i = Q^{(i+1)} (I - 2ww') = Q^{(i)} - 2(Q^{(i)} w) w'$$

$$R^{(i+1)} = H_i R^{(i)} = (I - 2ww')R^{(i)} = R^{(i)} - 2w(w'R^{(i)})$$

Aggiornando le matrici Q ed R con due prodotti matrice $\times$ vettore invece che con un prodotto matrice $\times$ matrice la complessità computazionale è ridotta di quasi un ordine. Questa modifica, insieme al supporto per matrici rettangolari, è stata inserita nella versione ottimizzata dell'algoritmo, riportata di seguito.

```
function [ Q,R ] = optqr( A )
  [m,n]=size(A);
  if m==n
    t=n-1;
  else
    t=n;
  end
  Q=eye(m);
  R=A;
  for i=1:t
    [w,k]=house( R(i:m,i) );  %calcolo parametri w,k
    H=eye(m);
    H(i:m,i:m)=H(i:m,i:m)-2*w*w';   %creazione matrice Householder con orlatura
    Q(:,i:m)=Q(:,i:m) - (Q(:,i:m)*w)*2*w';   %aggiornamento matrice Q
    R(i:m,i:n)=R(i:m,i:n) - 2*w*(w'*R(i:m,i:n));   %aggiornamento matrice R
  end
end
```

E' facile verificare, con uno script come quello riportato sotto, che la velocità di esecuzione del secondo algoritmo è assai superiore.

```
%test velocità algoritmi fattorizzazione QR (Householder)

N=20; %numero iterazioni
s=10; %step
T=zeros(N,4);

for i = 1:N
  n=s*i
  A=rand(n);
  Q=eye(n);
  R=zeros(n,n);

  %Householder non ottimizzato
  tic
  [Q,R] = myqr(A);
  T(i,1)=toc;

  %Householder ottimizzato
  tic
  [Q,R] = optqr(A);
  T(i,2)= toc;

  %Householder built-in
  tic
  [Q,R] = qr(A);
  T(i,3)= toc;

  %Gauss built-in
  tic
  [L,U] = lu(A);
```

```
  T(i,4)= toc;

end

%grafico tempi
n=linspace(s,s*N,N);
plot(n,T(:,1),'r-o',n,T(:,2),'b-o',n, T(:,3),'g-o',n,T(:,4),'k-o')
legend('Householder non ottimizzato','Householder ottimizzato',
            'Householder built-in','Gauss built-in');
```

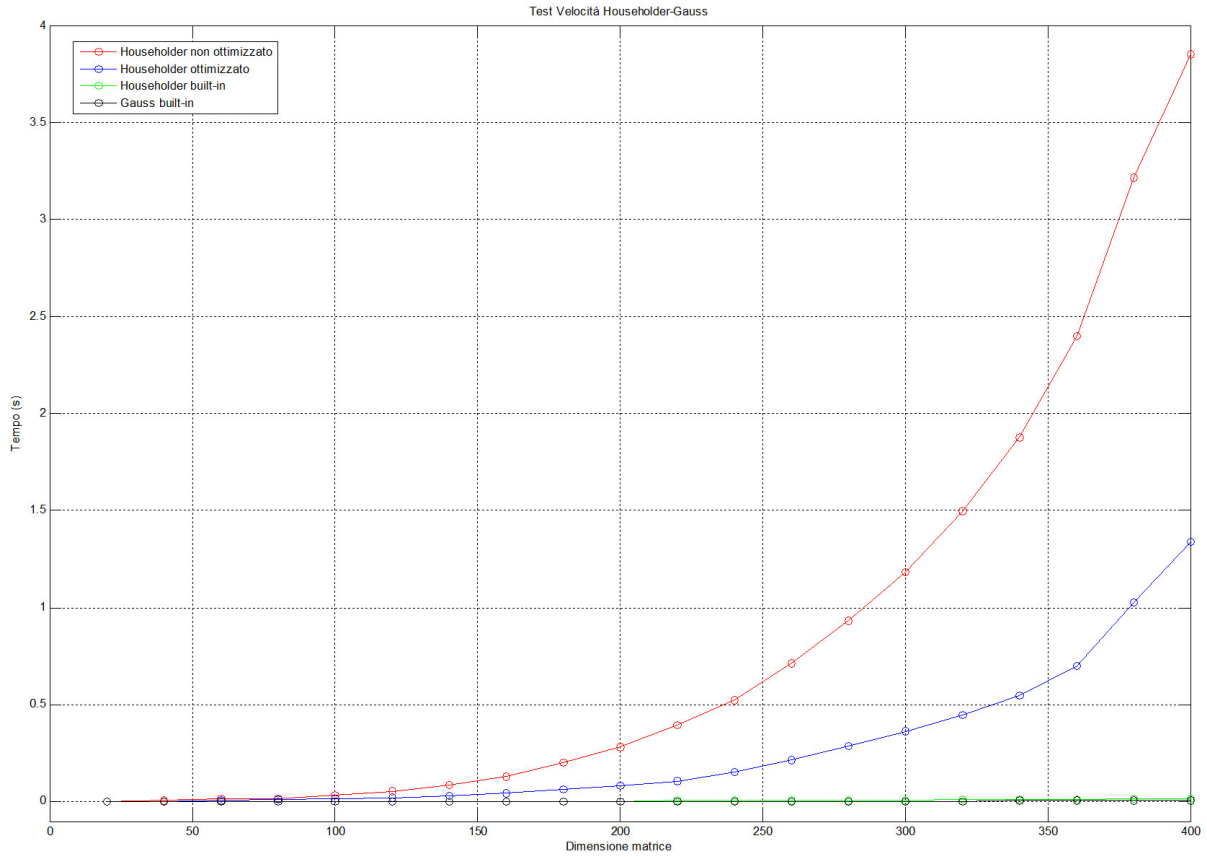

Figura 3: Comparativa dei tempi di esecuzione tra i diversi algoritmi di Householder

Gli algoritmi compilati in MATLAB *qr.m* (Householder built-in) e *lu.m* (Gauss built-in) risutano i più veloci; inoltre la differenza tra i tempi di esecuzione di *myqr.m* e *optqr.m* cresce al crescere della dimensione della matrice, come era lecito aspettarsi.
Per testare la stabilità degli algoritmi è stato scritto il seguente codice:

```
%test stabilità Householder

N=20; %numero iterazioni
s=10; %step matrice random
Err=zeros(N,4); %preallocazione vettore errore (matrice casuale)
ErrH=zeros(11,4); %preallocazione vettore errore (matrice Hilbert)

for i = 1:N
  n=s*i

  %creazione matrice e vettore dei termini noti
```

```
  A=rand(n);
  sol=ones(n,1);
  b=A*sol;

  %Householder non ottimizzato
  [Q,R]=myqr(A);
  y=Q\b;
  x=R\y;
  Err(i,1)=norm(x-sol);

  %Householder ottimizzato
  [Q,R]=optqr(A);
  y=Q\b;
  x=R\y;
  Err(i,2)=norm(x-sol);

  %Householder Built-in
  [Q,R]=qr(A);
  y=Q\b;
  x=R\y;
  Err(i,3)=norm(x-sol);

  %Gauss Built-in
  x=A\b;
  Err(i,4)=norm(x-sol);

end

%matrice di Hilbert
for i = 1:11
  m=i+2

  %creazione matrice e vettore dei termini noti
  H=hilb(m);
  sol=ones(m,1);
  b=H*sol;

  %Householder non ottimizzato
  [Q,R]=myqr(H);
  y=Q\b;
  x=R\y;
  ErrH(i,1)=norm(x-sol);

  %Householder ottimizzato
  [Q,R]=optqr(H);
  y=Q\b;
  x=R\y;
  ErrH(i,2)=norm(x-sol);

  %Householder Built-in
  [Q,R]=qr(H);
  y=Q\b;
  x=R\y;
  ErrH(i,3)=norm(x-sol);

  %Gauss Built-in
  x=H\b;
  ErrH(i,4)=norm(x-sol);
```

```
end

%grafici
n=linspace(s,s*N,N);
m=[3:1:13];

subplot(2,1,1);
semilogy(n,Err(:,1),'r-o',n,Err(:,2),'g-o',n,Err(:,3),'b-o',n,Err(:,4),'k-o')
title('Matrice Casuale')
legend('Householder non ottimizzato','Householder ottimizzato', 'Householder built-in',
      'Gauss built-in');
subplot(2,1,2); semilogy(m,ErrH(:,1),'r-o',m,ErrH(:,2),'g-o',m,ErrH(:,3),'b-o',m,
      ErrH(:,4),'k-o')
title('Matrice di Hilbert')
legend('Householder non ottimizzato','Householder ottimizzato','Householder built-in',
      'Gauss built-in');
```

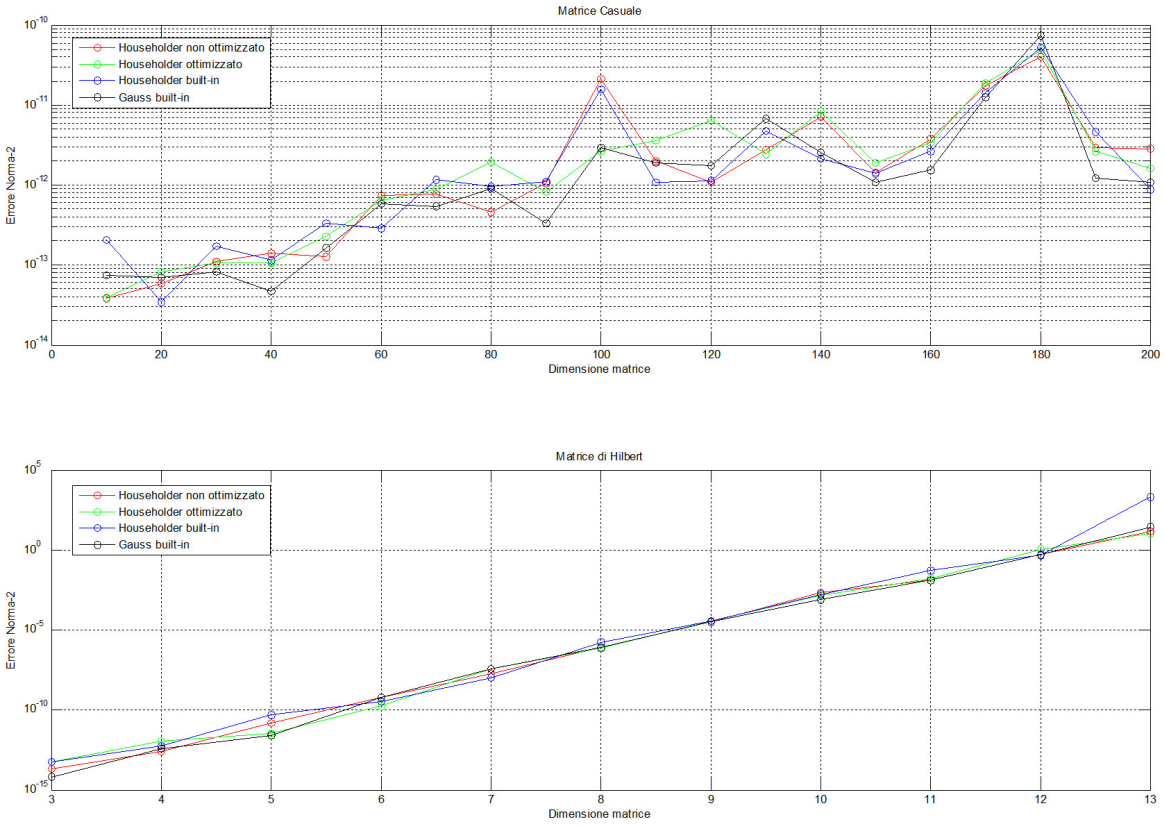

Figura 4: Errore in norma-2 per i diversi algoritmi di Householder

L'errore non varia significativamente tra gli algoritmi testati, sia per matrici casuali che per matrici a condizionamento elevato (Hilbert). L'algoritmo *optqr.m* risulta pertanto più efficiente di *myqr.m* senza alcuna perdita di precisione e sarà utilizzato nel seguito.

4 Problemi sovradeterminati

Un problema è ben posto se esso possiede, in un fissato campo di definizione, una e una sola soluzione, e questa dipende con continuità dai dati. In caso contrario, il problema viene detto mal posto.
Si consideri la forma generale di un sistema lineare

$$Ax = b$$

con

- $\mathbf{A} \in \mathbb{R}^{\mathrm{mxn}}$ matrice dei coefficenti;
- $\mathbf{x} \in \mathbb{R}^{\mathrm{n}}$ soluzione;
- $\mathbf{b} \in \mathbb{R}^{\mathrm{m}}$ vettore dei termini noti;

Se $m = n$ la matrice A è quadrata e (se non singolare) invertibile. Il sistema è risolvibile attraverso diversi metodi, tra cui quelli trattati in questa tesina. Nelle applicazioni reali è però possibile avere a che fare con fenomeni descritti da sistemi lineari aventi un numero di equazioni diverso dal numero di incognite. Ipotizzando che la matrice A sia a rango pieno, si distinguono due casi. Se $m > n$ il problema è sovraderminato, il problema risulta malposto in quanto si hanno più equazioni che incognite, potrebbe non esistere una soluzione. Se $m < n$ il problema è sottodeterminato, il problema risulta malposto in quanto si hanno meno equazioni che incognite, la soluzione esiste ma non è unica. Il caso a rango non pieno può essere ricondotto a uno di questi due casi.

Per ottenere una soluzione in senso classico per un problema mal posto è necessario trasformarlo in un problema ben posto. Si consideri il caso di sistema sovradeterminato: non essendo possibile verificare contemporaneamente tutte le equazioni del sistema si richiede che la varianza tra i membri del sistema sia minima.

$$\min_{x \in \mathbb{R}^n} ||Ax - b||_2$$

Se il minimo è nullo, il sistema originario ammette una soluzione in senso classico, in caso contrario si ottiene la soluzione nel senso dei minimi quadrati del sistema lineate sovradeterminato.

4.1 Risoluzione di un problema ai minimi quadrati mediante fattorizzazione LU

Poichè il problema di minimizzazione coinvolge una norma, è possibile considerare il quadrato della norma del residuo:

$$||A\mathbf{x} - \mathbf{b}||^2 = (A\mathbf{x} - \mathbf{b})^T(A\mathbf{x} - \mathbf{b}) = x^T A^T Ax - 2x^T A^T b + b^T b$$

per minizzare la norma euclidea del residuo si impone l'annullamento del gradiente

$$\frac{1}{2}\nabla(||A\mathbf{x} - \mathbf{b}||^2) = A^T A\mathbf{x} - A^T\mathbf{b} = 0$$

da cui si ottiene il **sistema normale**

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

Nel caso in cui la matrice A sia a rango pieno la matrice $A^T A$ è invertibile e la soluzione del sistema lineare è unica

$$\mathbf{x}_{LS} = (A^T A)^{-1} A^T\mathbf{b}$$

Se il rango di A è inferiore a n, la matrice $A^T A$ è singolare, ma il sistema è ancora consistente in quanto il vettore $A^T\mathbf{b}$ appartiene all'immagine di A^T, che coincide con l'immagine di $A^T A$. Solitamente si assume come soluzione $\mathbf{x_{LS}}$ quella che minimizza la norma euclidea (soluzione normale). La matrice

$$A^\dagger = (A^T A)^{-1} A^T$$

è detta **pseudo-inversa** di A, ed è un'inversa sinistra. Nel caso in cui A sia a rango pieno la matrice $A^T A$ è simmetrica definita positiva ed è quindi possibile risolvere il sistema normale tramite

la fattorizzazione di Cholesky. Calcolata la fattorizzazione $A^TA = R^TR$ con R matrice triangolare inferiore di dimensione $n \times n$, il vettore $\mathbf{x_{LS}}$ può essere calcolato risolvendo i due sistemi trangolari

$$\begin{cases} R^Ty = A^Tb \\ Rx = y \end{cases}$$

Il costo computazionale del metodo dipende dal calcolo del prodotto matriciale A^TA e dalla fattorizzazione di Cholesky. Un possibile svantaggio di un simile approccio riguarda la stabilità, in quanto la matrice A^TA ha un condizionamento pari al quadrato di quello di$A$.

4.2 Risoluzione di un problema ai minimi quadrati mediante fattorizzazione QR

Il problema

$$\min_{x \in \mathbb{R}^n} ||Ax - b||_2$$

può essere risolto mediante la fattorizzazione QR. Sostituendo all'interno di suddetta formulazione la fattorizzazione $A = QR$ si ottiene

$$||Ax - b||^2 = ||QRx - b||^2 = ||Q(Rx - Q^{\mathrm{T}}b)||^2 = ||Rx - c||^2$$

dove si è posto $\mathbf{c} = Q^{\mathrm{T}}\mathbf{b}$. La matrice R ha dimensione $m \times n$ ed è strutturata come segue

$$R = \begin{bmatrix} k_1 & * & \cdots & * \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & * \\ \vdots & & \ddots & k_n \\ \vdots & & & 0 \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

con R_1 matrice triangolare superiore e, se A è a rango pieno, non singolare. Si partiziona in maniera coerente il vettore $\mathbf{c}$

$$c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \qquad c_1 \in \mathbb{R}^n \quad , \quad c_2 \in \mathbb{R}^{m-n}$$

ottenendo

$$||Ax - b||^2 = ||Rx - c||^2 = \left|\left| \begin{bmatrix} R_1 \\ 0 \end{bmatrix} - \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \right|\right|^2 = ||R_1\mathbf{x} - \mathbf{c_1}||^2 + ||\mathbf{c_2}||^2$$

Se $det(R_1) \neq 0$ il sistema $R_1\mathbf{x} - \mathbf{c}_1$ ammette una e una sola soluzione, tale che $||R_1\mathbf{x} - \mathbf{c_1}||^2 = 0$, in corrispondenza della quale si ha

$$\min_{x \in \mathbb{R}^n} ||Ax - b|| = ||c_2||$$

Se il vettore $\mathbf{c_2}$ fosse nullo $\mathbf{x}$ sarebbe la soluzione classica del sistema lineare $Ax = b$, in caso contrario essa è la soluzione nel senso dei minimi quadrati, e la norma di $\mathbf{c_2}$ fornisce la misura del residuo.

4.3 Algoritmi e loro applicazione al sistema normale

4.3.1 La routine CNDMX

Per operare i test desiderati sui sistemi sovradeterminati è stato necessario scrivere una funzione che crei una matrice di dimensioni qualsiasi $m \times n$ e con il condizionamento richiesto. Per determinare tale matrice utilizziamo la decomposizione

$$A = Q_{(m\times m)} \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_n \\ 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}_{m\times n} U_{(n\times n)}$$

con $Q^TQ = I_m$ e $U^TU = I_n$.
Il condizionamento in norma-2 di questa matrice è definito come

$$K_2 = \frac{\sigma_{\max}}{\sigma_{\min}}$$

pertanto basta scegliere i parametri in modo opportuno per ottenere la matrice desiderata. Nel nostro caso, dovendo ottenere una matrice casuale, Q e U saranno matrici casuali ortogonali. Per ottenere i σ_i si è invece scelto di costruire una successione esponenziale crescente

$$\{\sigma_1,\ \sigma_2, \cdots, \sigma_n\} = \{10^{s_1}, 10^{s_2}, \cdots, 10^{s_n}\}$$

e di determinare gli esponenti in modo che risulti

$$\sigma_1 = 10^{s_1} = \sigma_{\max} \quad \text{e} \quad \sigma_n = 10^{s_n} = \sigma_{\min}$$

e tali che formino uno spazio lineare centrato intorno allo zero, quindi ($s_1 + s_n = 0$) Si ottiene pertanto il seguente codice MATLAB per la funzione *cndmx.m*:

```
function [A] = cndmx( m,n,k )

  Q=orth(rand(m));
  U=orth(rand(n));
  S=zeros(m,n);

  smin=-0.5*log10(k);
  smax=0.5*log10(k);

  sigma=logspace(smin,smax,n);
  sigma=sigma';

  S(1:n,1:n)=diag(sigma);
  A=Q*S*U;

end
```

4.3.2 Risoluzione di sistemi sovradeterminati

Allo scopo di fare un confronto tra la risoluzione del sistema normale con Cholesky e la risoluzione mediante fattorizzazione QR è stato scritto il seguente script MATLAB:

```
%Test Sistema Sovradeterminato
N=20; %numero iterazioni
s=10; %step matrice random
Err=zeros(N,4); %preallocazione vettore errore (matrice casuale)

for i = 1:N
m=1.5*s*i
n=s*i

sol=ones(n,1);       %soluzione vettore unitario
```

```
A=rand(m,n);         %creazione matrice random e vettore dei termini noti
a=A*sol;

B=cndmx(m,n,10^8);   %creazione matrice condizionamento arbitrario e
b=B*sol;                                  vettore dei termini noti

A1=A'*A;              %Sistema Normale A'Ax=A'a mediante Cholesky
R=optchol(A1);
y=R'\(A'*a);
x=R\y;
Err(i,1)=norm(x-sol);

B1=B'*B;               %Sistema Normale B'Bx=B'b mediante Cholesky
R=optchol(B1);
y=R'\(B'*b);
x=R\y;
Err(i,3)=norm(x-sol);

[Q,R]=optqr(A);         %Sistema Ax=a mediante fattorizzazione QR
R1=R(1:n,1:n);
c=Q'*a;
c1=c(1:n);
x=R1\c1;
Err(i,2)=norm(x-sol);

[Q,R]=optqr(B);          %Sistema Bx=b mediante fattorizzazione QR
R1=R(1:n,1:n);
c=Q'*b;
c1=c(1:n);
x=R1\c1;
Err(i,4)=norm(x-sol);

end

%grafici
n=linspace(s,s*N,N);

subplot(2,1,1); semilogy(n,Err(:,1),'r-o',n,Err(:,2),'b-o')
title('Matrice Casuale m>n')
legend('Sistema normale mediante Cholesky','Fattorizzazione QR');

subplot(2,1,2); semilogy(n,Err(:,3),'r-o',n,Err(:,4),'b-o')
title('Matrice Condizionamento Arbitrario m>n')
legend('Sistema normale mediante Cholesky','Fattorizzazione QR');
```

dove si è usata la funzione *cndmx.m* vista in precedenza per creare una matrice rettangolare con condizionamento 10^8.

Come era lecito aspettarsi, si può osservare in figura che l'errore è significativamente maggiore per il metodo del sistema normale, tanto che nel caso della matrice ad alto condizionamento è del tutto inutilizzabile, a differenza del metodo con fattorizzazione QR che fornisce degli errori tutto sommato accettabili (figura 5).

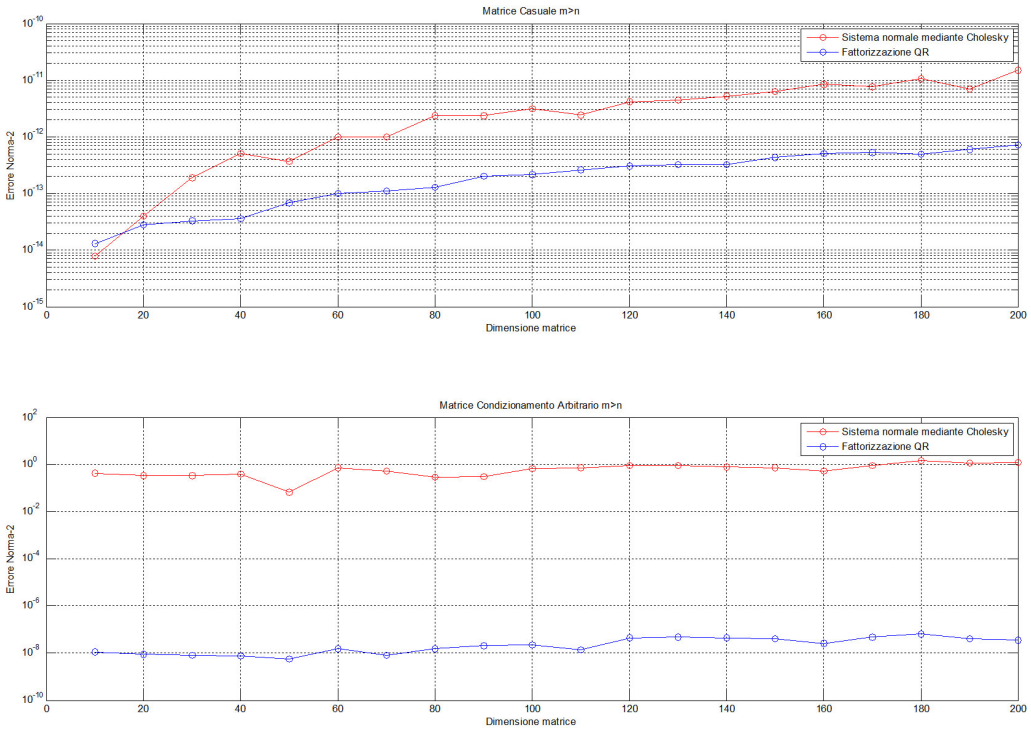

Figura 5: Confronto dell'errore in norma-2 tra il sistema normale e la fattorizzazione QR

4.4 Interpolazione ed approssimazione ai minimi quadrati

Un'applicazione significativa della risoluzione di sistemi sovradeterminati è l'approssimazione ai minimi quadrati di funzioni tramite l'interpolazione. Calcolare il polinomio di migliore approssimazione per una funzione f significa determinare il polinomio di grado n che ne minimizzi una norma dell'errore

$$\min_{p_n \in \Pi_n} ||p_n - f||$$

Trattando il problema termini di norma infinito

$$||p_n - f||_\infty = \max_{x \in [a,b]} |p_n(x) - f(x)|$$

si parla di **migliore approssimazione uniforme** mentre il polinomio che minimizza la norma di L^2[a,b]

$$||p_n - f||_2 = \left(\int_a^b [p_n(x) - f(x)]^2 dx \right)^{1/2}$$

è detto **migliore approssimazione nel senso dei minimi quadrati**. Il secondo problema è facilmente trattabile dal punto di vista computazionale.

Se la funzione da approssimare è nota solo attraverso un certo numero di valori affetti da errore è preferibile effettuare un'approssimazione ai minimi quadrati attraverso una discretizzazione della norma-2 piuttosto che effettuare la semplice interpolazione dei punti. Siano $\{x_0, x_1, \ldots, x_n\}$ le ascisse degli $m+1$ punti per i quali sono noti i valori $\{y_0, y_1, \ldots, y_n\}$ della funzione $f(x)$. Fissato un $n \leq m$ è possibile considerare per ogni $p_n(x) \in \Pi_n$ la norma discreta

$$||p_n - f||_2 = \left(\sum_{i=0}^{m} [p_n(x_i) - y_i]^2 \right)^{1/2}$$

L'obbiettivo è determinare il polinomio $p_n^*(x)$ di grado n che risolve il problema di minimizzazione

$$\min_{p_n \in \Pi_n} ||p_n - f||_2$$

per semplicità, e per equivalenze dai due problemi, si minimizzerà il quadrato nella norma

$$\min_{p_n \in \Pi_n} ||p_n - f||_2^2$$

Per $m = n$ la soluzione coincide con il polinomio interpolante, se invece $m > n$ la soluzione fornisce la migliore approssimazione nel senso dei minimi quadrati rispetto alla norma discreta.
Utilizzando la base canonica si ottiene

$$p_n(x_i) = \sum_{j=0}^{n} a_j x_i^j = (Xa)_i \qquad i = 0, \ldots, m$$

dove $\mathbf{a} = (a_0, \ldots, a_n)^T \in \mathbb{R}^{n+1}$ è il vettore dei coefficenti del polinomio e X è la matrice di Vandermonde di dimensione (m+1)x(n+1)

$$X = \begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \ldots & x_m^n \end{bmatrix}$$

si ha

$$||p_n - f||_2^2 = \sum_{i=0}^{m} [(X\mathbf{a})_i - y_i]^2 = ||X\mathbf{a} - \mathbf{y}||_2^2$$

ed il problema di minimo risulta essere equivalente alla soluzione nel senso dei minimi quadrati del sistema lineare sovradeterminato

$$X\mathbf{a} = \mathbf{y}$$

La soluzione di tale sistema lineare può essere calcolata attraverso la fattorizzazione di Cholesky del sistema normale

$$X^T X\mathbf{a} = X^T\mathbf{y}$$

oppure utilizzando la fattorizzazione QR della matrice X per il problema ai minimi quadrati. Quest'ultimo metodo è il più conveniente dal punto di vista della stabilità numerica.

4.5 Implementazione con gli algoritmi di Cholesky e Householder

Al fine di testare gli algoritmi di Cholesky e di Householder visti precedentemente, sono state considerate le approssimazioni di una funzione sinusoidale e di una funzione gaussiana.

Gli script che seguono costruiscono la matrice di Vandermonde (dal condizionamento piuttosto alto) e calcolano i coefficienti del polinomio approssimante risolvendo un sistema sovradeterminato sia con Cholesky che mediante fattorizzazione QR.

```
Function_least_square.m
m=50; %numero punti -1
n=10; %grado polinomio interpolante

x=linspace(-1,1,m+1); %ascisse degli m+1 punti noti
x=x';

%costruzione matrice Vandermonde
```

```
X=zeros(m+1,n+1);
for j=0:n
    X(:,j+1)=x(:).^j;
end

y=sin(pi*x)+0.1*randn(m+1,1); %ordinate degli m+1 punti noti

%costruzione funzione seno senza rumore gaussiano
t=linspace(-1,1,201);
f=sin(pi*t);

%risoluzione del sistema normale X'Xa=X'y mediante Cholesky
X1=(X')*X;
R=optchol(X1);
z=R'\((X')*y);
a=R\z;
a=a(n+1:-1:1);
p1=polyval(a,t);

%risoluzione del sistema Xa=y mediante fattorizzazione QR
[Q,R]=optqr(X);
R1=R(1:n+1,1:n+1);
c=Q'*y;
c1=c(1:n+1);
a=R1\c1;
a=a(n+1:-1:1);
p2=polyval(a,t);

%grafici
plot(x,y,'ro',t,f,'k--',t,p1,'b-',t,p2,'g-')
legend('Punti funzione seno con errore Gaussiano','Funzione seno',
      'Polinomio interpolante Cholesky', 'Polinomio interpolante Householder');

Function_least_square_2.m
m=50; %numero punti -1
n=10; %grado polinomio interpolante

x=linspace(-1,1,m+1); %ascisse degli m+1 punti noti
x=x';

%costruzione matrice Vandermonde
X=zeros(m+1,n+1);
for j=0:n
    X(:,j+1)=x(:).^j;
end

y=exp((-0.5*x.^2)./.02)+0.1*randn(m+1,1); %ordinate degli m+1 punti noti

%costruzione funzione senza rumore
t=linspace(-1,1,201);
f=exp((-0.5*t.^2)./.02);

%risoluzione del sistema normale X'Xa=X'y mediante Cholesky
X1=(X')*X;
R=optchol(X1);
z=R'\((X')*y);
a=R\z;
a=a(n+1:-1:1);
p1=polyval(a,t);
```

```
%risoluzione del sistema Xa=y mediante fattorizzazione QR
[Q,R]=optqr(X);
R1=R(1:n+1,1:n+1);
c=Q'*y;
c1=c(1:n+1);
a=R1\c1;
a=a(n+1:-1:1);
p2=polyval(a,t);

%grafici
plot(x,y,'ro',t,f,'k--',t,p1,'b-',t,p2,'g-')
legend('Punti funzione Gaussiana con rumore','Funzione Gaussiana',
'Polinomio interpolante Cholesky','Polinomio interpolante Householder')
```

Nel grafico generato da questo codice il numero m dei punti è pari a 51, mentre il grado del polinomio è pari a 10. Si vede come i polinomi ottenuti coi due metodi siano del tutto indistinguibili. Per ottenere delle curve che si discostino in maniera significativa è necessario aumentare il grado del polinomio interpolante (e aumentare così il condizionamento della matrice di Vandermonde). Le figure seguenti sono state ottenute con un valore di n pari a 35. Come si può notare le curve sono diverse, ma è evidente dalle numerose oscillazioni che l'approssimazione ottenuta non è quella cercata ed è peggiore di quella ottenuta con n=10.

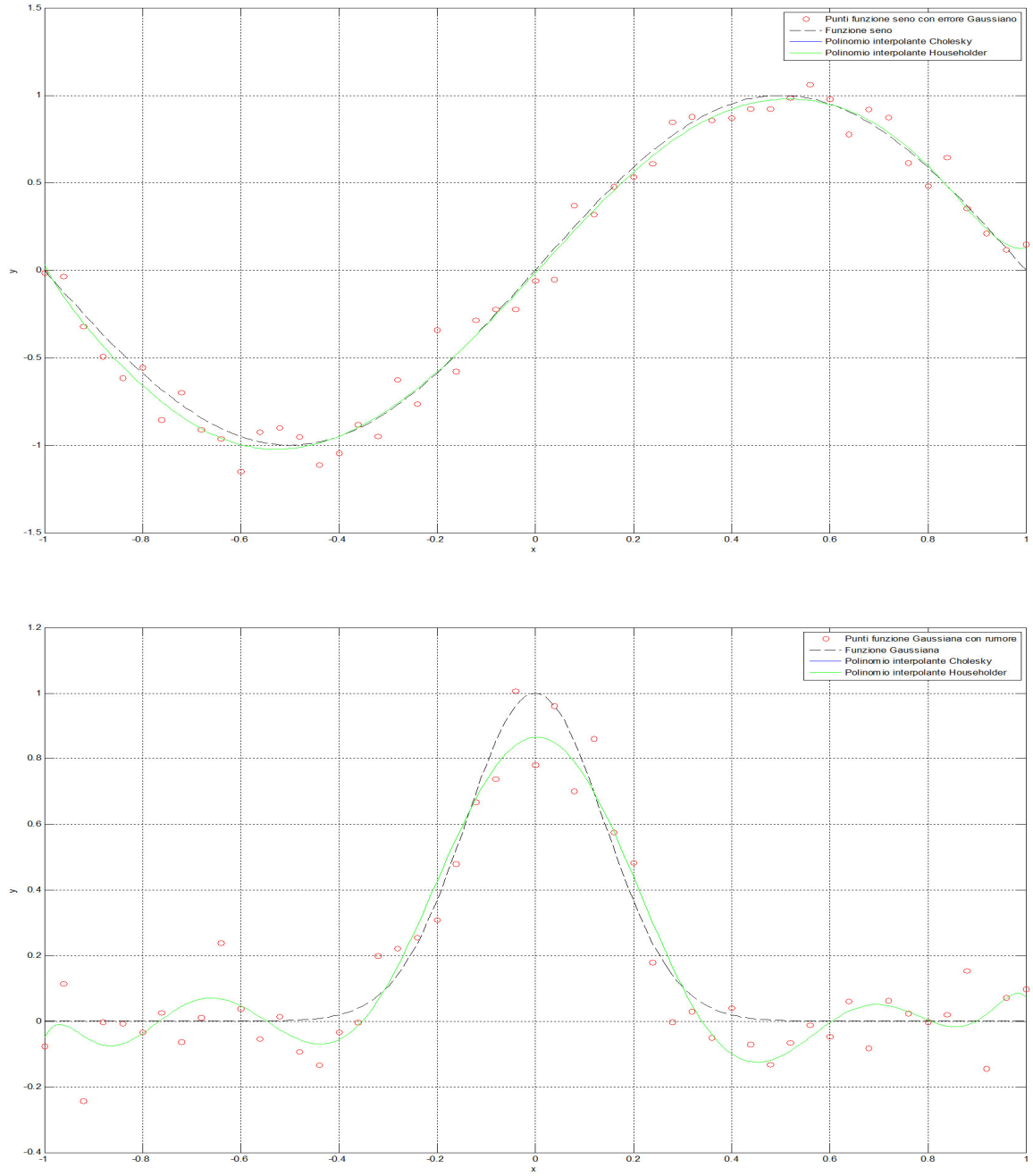

Figura 6: Approssimazioni ai minimi quadrati di una funzione sinusoidale e una Gaussiana (n=10)

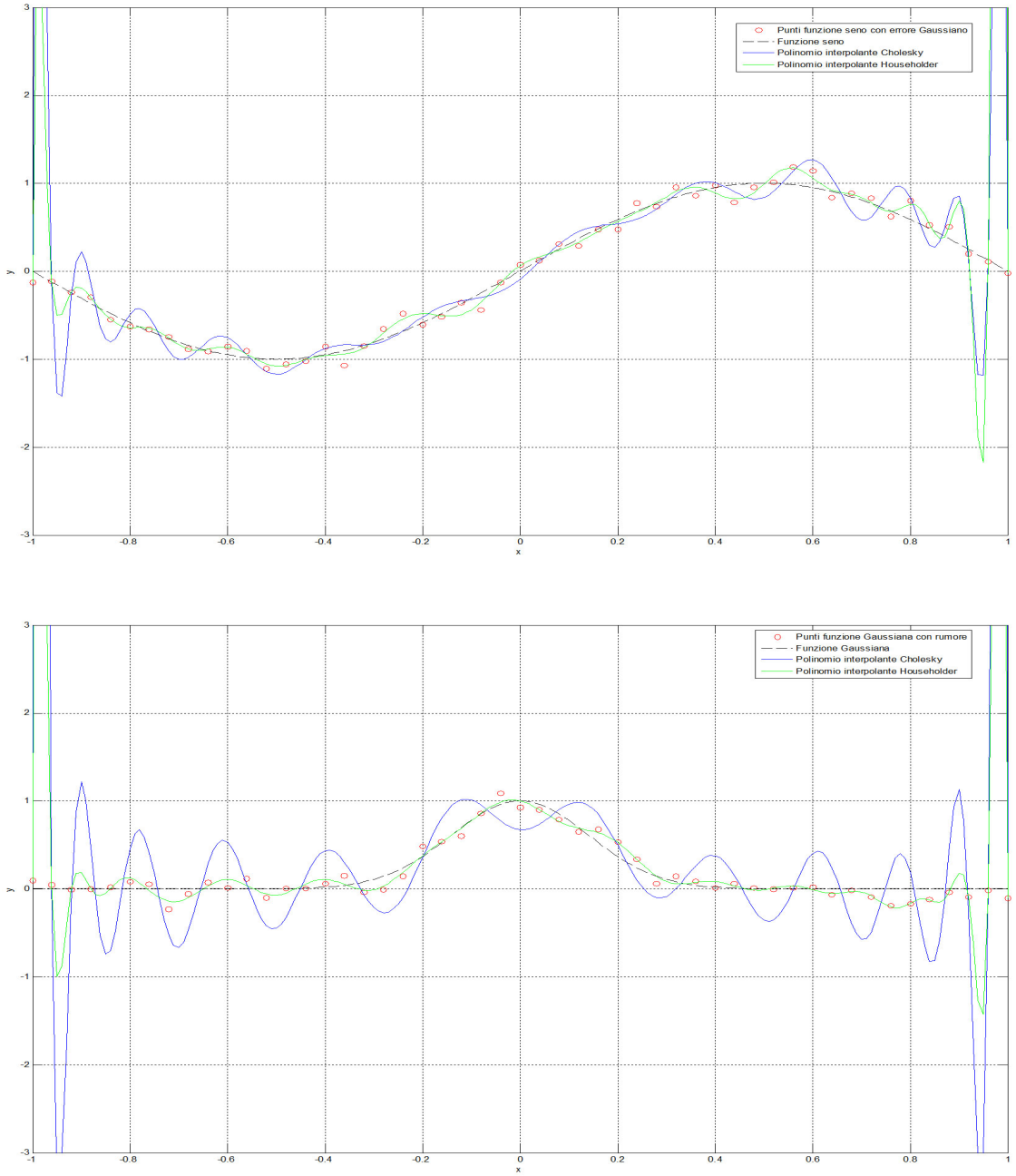

Figura 7: Approssimazioni ai minimi quadrati di una funzione sinusoidale e una Gaussiana (n=35)