Fattorizzazioni per la risoluzione di sistemi lineari: implementazione e benchmark comparativo con MATLAB

Simone Secchi

13 giugno 2007

1 Introduzione

In questo elaborato è descritta la risoluzione di un sistema lineare

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

mediante l'uso di alcune fattorizzazioni della matrice dei coefficienti A.

L'obbiettivo del lavoro è stato l'implementazione di due algoritmi per la fattorizzazione ortogonale-triangolare QR (basati sulla costruzione delle matrici di Householder e delle matrici di Givens) e di un algoritmo per la fattorizzazione di Cholesky della matrice A, utilizzabile in tutti i casi in cui tale matrice risulti essere simmetrica e definita positiva. In seguito sono stati eseguiti dei test di confronto tra gli algoritmi *personalizzati* e le relative versioni pre-implementate e funzionanti in ambiente di calcolo MATLAB (Release 14).

2 Gli algoritmi

La fattorizzazione QR è un importante strumento per la risoluzione di sistemi lineari: ogni matrice dei coefficienti può essere fattorizzata nella forma

$$\mathrm{A} = \mathrm{QR}$$

che viene detta anche forma *ortogonale-triangolare* in quanto la matrice Q è una matrice ortogonale e la matrice R una matrice triangolare superiore (nel caso di matrice A quadrata) o composta da un blocco triangolare superiore e un blocco di zeri (nel caso di matrice A rettangolare con più righe che colonne).

È importante notare che *qualunque* matrice A può essere fattorizzata in questo modo: in particolare tale scomposizione esiste anche per matrici singolari (nel qual caso, essendo per una matrice ortogonale $det(Q) \neq 0$, sarà singolare la matrice R).

Esistono diversi modi per ottenere una fattorizzazione $A = QR$, tra i quali si è scelto di implementare il metodo delle matrici ortogonali di Householder e il metodo delle matrici di rotazione di Givens.

2.1 La fattorizzazione QR di Householder

La fattorizzazione QR di Householder è incentrata sulla determinazione delle matrici elementari di Householder.
Una matrice elementare di Householder è una matrice del tipo:

$$H = I - 2\mathbf{w}\mathbf{w}^{\mathbf{T}}, \quad \mathbf{w} \in \Re^n, \quad \|w\| = 1$$

Si può determinare un valore di $\mathbf{w}$ tale da fare in modo che la moltiplicazione a sinistra di questa matrice per un qualsiasi vettore di $\Re^n$ lo trasformi in un vettore parallelo al primo versore $\mathbf{e_1}$ della base canonica, ossia:

$$H\mathbf{x} = k\mathbf{e_1}$$

La determinazione analitica del vettore $\mathbf{w}$ e quindi della matrice di Householder H conduce alla definizione del seguente algoritmo:

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

La fattorizzazione QR di Householder si basa sulla costruzione, ad ogni passo dell'algoritmo, della matrice di Householder che trasforma la parte sottodiagonale della i-sima colonna della matrice A in un vettore parallelo al primo versore $\mathbf{e_1}$ della base canonica dello spazio $\Re^{n-i+1}$; in questo modo si arriva ad avere, dopo $n-1$ passi (nel caso di matrice A quadrata), una matrice ortogonale Q data dal prodotto delle $n-1$ matrici ortogonali di Householder:

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

e una matrice triangolare R tali che:

$$A = QR$$

Si noti che nel caso di matrice A rettangolare con m righe e n colonne (con $m > n$) sono necessari n passi per portare a termine la fattorizzazione. L'algoritmo risultante per la fattorizzazione è il seguente:

1. **input** A,n
2. $Q = I$
3. $R = A$
4. **for** $i = 1, 2, \dots, n-1$
   (a) Costruisci H_i

(b) $Q = QH_i$

(c) $R = H_iR$

5. **output** Q,R

I due algoritmi appena presentati sono stati implementati in funzioni MATLAB apposite (`construct_H.m` e `My_hous_QR.m`) e in seguito saranno presentati i risultati del loro confronto con le rispettive versioni *built-in* di MATLAB.

2.2 La fattorizzazione QR di Givens

La fattorizzazione QR di Givens rappresenta un modo alternativo per ottenere la scomposizione ortogonale-triangolare $A = QR$ particolarmente conveniente da un punto di vista computazionale per matrici A sparse.

Una matrice di Givens G_{ij} di dimensione n rappresenta una rotazione in $\Re^n$ ed è strutturata come segue:

$$\mathcal{A} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & \cdots & s & & \\ & & \vdots & & \vdots & & \\ & & -s & \cdots & c & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

dove le righe (e le colonne) differenti dalla matrice identità sono la i-sima e la j-sima; inoltre per i fattori c,s deve valere la relazione $c^2 + s^2 = 1$.

L'applicazione di questa matrice ad un vettore $\mathbf{x}$ genera un vettore $\mathbf{y}$ le cui componenti sono:

$$y_k = \begin{cases} cx_i + sx_j & k = i \\ -sx_i + cx_j & k = j \\ x_k & k \neq i, j \end{cases}$$

È immediato notare che è possibile calcolare i valori di c e s in modo da annullare una componente del vettore $\mathbf{y}$ risultante dalla moltiplicazione. In questo modo è possibile, dato un vettore $\mathbf{x} \in \Re^n$, costruire $n - 1$ matrici di Givens che, moltiplicate a sinistra per il vettore, forniscono un modo per *parallelizzare* tale vettore rispetto al primo versore $\mathbf{e_1}$ della base canonica.

Si ha quindi, analogamente a quanto avviene con le matrici elementari di Householder:

$$G_{1,n-1} \dots G_{13} G_{12} \mathbf{x} = k\mathbf{e_1}$$

È evidente quindi la possibilità di ottenere una fattorizzazione QR sfruttando le matrici elementari di Givens: tale tecnica risulta inoltre più conveniente della tecnica di Householder nel caso in cui la matrice dei coefficienti A sia una matrice sparsa, in quanto l'algoritmo deve operare la costruzione della relativa matrice di Givens solo per il generico elemento $a_{ij} \neq 0$.

La matrice ortogonale (si ricordi che le matrici di rotazione di Givens sono matrici ortogonali) Q sarà in questo caso data da:

$$Q = \prod_{i=1}^{n-1} \left(\prod_{j=i+1}^{n} G_{ij}^T \right)$$

Per questo tipo di fattorizzazione sono stati implementati l'algoritmo per il calcolo dei parametri c e s della matrice G_{ij} e l'algoritmo per la fattorizzazione QR ottimizzato nel caso di matrici sparse; i file relativi sono `construct_G.m` e `My_giv_QR.m`, in seguito saranno presentati i risultati del loro confronto con le rispettive versioni *built-in* di MATLAB.

Uno schema concettuale dell'algoritmo per il calcolo degli elementi della matrice G_{ij} il seguente:

1. **input** x_i, x_j
2. **if** $x_i = 0$
   (a) c=0
   (b) s=1
3. **if** $x_j = 0$
   (a) c=1
   (b) s=0
4. **elseif** $|x_j| \geq |x_i|$
   (a) $t = x_i/x_j$
   (b) $z = \sqrt{1+t^2}$
   (c) $s = 1/z$
   (d) c=ts

5. **else**

   (a) $t = x_j/x_i$

   (b) $z = \sqrt{1+t^2}$

   (c) $c = 1/z$

   (d) s=ts

6. **output** *c,s*

L'algoritmo per la fattorizzazione QR è analogo al caso della tecnica di Householder, con la sola differenza che in questo caso la routine di costruzione della matrice di Givens relativa all'elemento a_{ij} sarà attivata solo nel caso in cui questo non sia nullo.

2.3 La fattorizzazione di Cholesky

Se la matrice dei coefficienti A del sistema lineare considerato è simmetrica e definita positiva è possibile, partendo dalla fattorizzazione $A = LU$, dimostrare che esiste una fattorizzazione alternativa, denominata *fattorizzazione di Cholesky*, del tipo $A = R^T R$, dove R è una matrice triangolare superiore della stessa dimensione di A. Esprimendo quindi gli elementi di A in funzione di quelli di R e invertendo la relazione è possibile calcolare i singoli elementi di R in funzione dei singoli elementi a_{ij}. In questo modo si ottiene l'algoritmo per il calcolo della fattorizzazione di Cholesky, che è riportato di seguito:

1. **input** *A,n*

2. **for** $j = 1, 2, \ldots, n$

   (a) **for** $i = 1, 2, \ldots, j-1$

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

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

4. **output** *Q,R*

Tale algoritmo è stato implementato in MATLAB attraverso il file `My_chol.m` ed in seguito sarà presentato un test di confronto con la versione *built-in* della fattorizzazione di Cholesky.

3 Test e risultati

In questa sezione sono presentati i test di confronto effettuati tra gli algoritmi implementati e le rispettive versioni presenti in ambiente MATLAB. I risultati sono riportati in forma grafica per una più semplice e veloce interpretazione.

3.1 Fattorizzazione QR di Housholder

Il test della fattorizzazione QR di Householder è basato sul calcolo dei tempi di esecuzione, dell'errore (dove si è definito errore la matrice $E = A - QR$) in norma-∞ e in norma di Frobenius e sul confronto di queste quantità con le rispettive quantità calcolate relativamente alla funzione *built-in* di MATLAB

$$[Q, R] = qr(A)$$

Per quanto concerne i tempi di esecuzione si è sfruttata la coppia di comandi `tic-toc` che permettono di misurare il tempo utilizzato dall'insieme di istruzioni comprese tra essi. Tale analisi (sia per i tempi di esecuzione che per l'errore) è stata effettuata al variare della dimensione della matrice A, da un minimo di 2 fino ad un massimo di 502.
Per quanto riguarda il confronto dei tempi di esecuzione si ottenuto quanto riportato nelle figure 1 e 2. Come si vede facilmente la routine di MATLAB è di di gran lunga più efficiente della routine *personalizzata*.

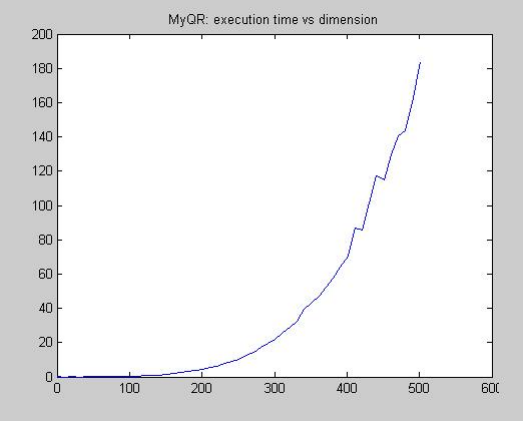

Figura 1: Tempo di esecuzione dell'algoritmo personalizzato

Per quanto invece riguarda gli errori in norma-∞ e in norma di Frobenius si ottengono prestazioni simili tra i due algoritmi; tali prestazioni sono accettabili in quanto, sebbene l'errore esatto dovrebbe essere nullo (infatti

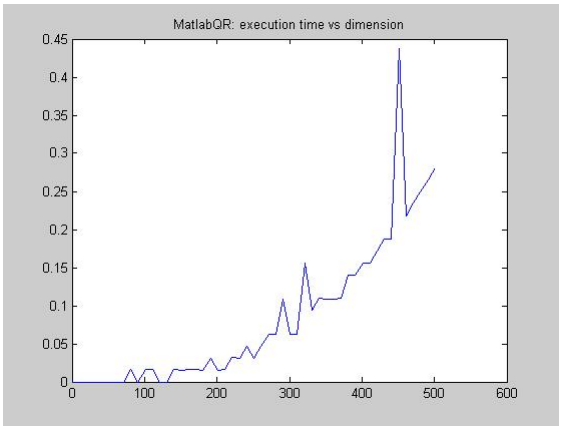

Figura 2: Tempo di esecuzione dell'algoritmo di MATLAB

vale la relazione esatta $A = QR$) si deve necessariamente tenere conto della propagazione degli errori di aritmetica finita. Per un riscontro visivo si vedano le figure 3, 4, 5 e 6:

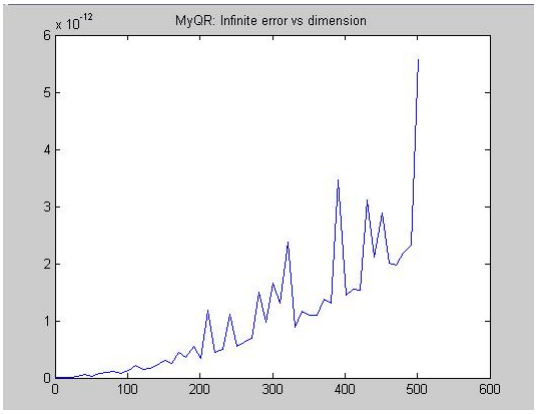

Figura 3: Errore in norma-∞ dell'algoritmo personalizzato

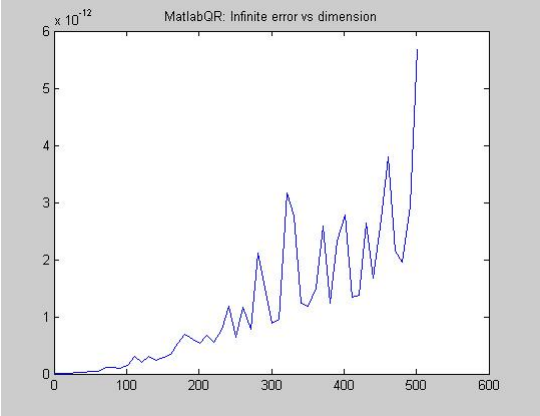

Figura 4: Errore in norma-∞ dell'algoritmo di MATLAB

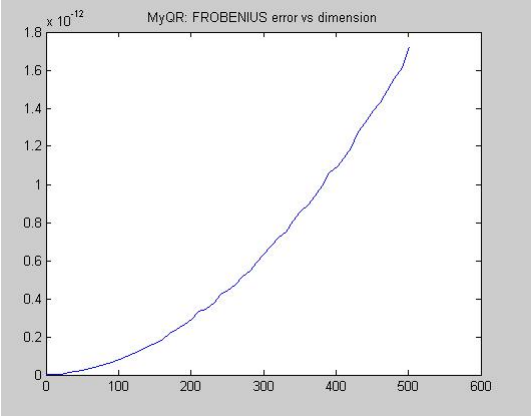

Figura 5: Errore in norma di Frobenius dell'algoritmo personalizzato

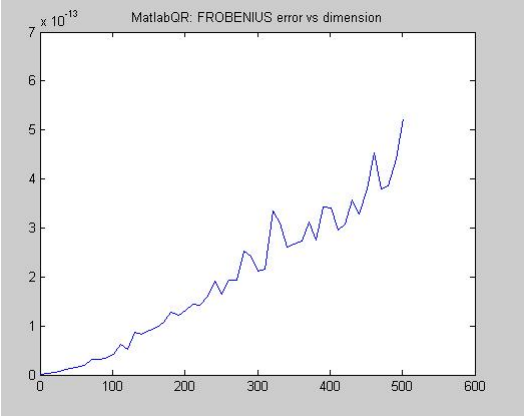

Figura 6: Errore in norma di Frobenius dell'algoritmo di MATLAB

3.2 Fattorizzazione QR di Givens

Analogamente a quanto ottenuto con la fattorizzazione di Householder, anche in questo caso il confronto ha designato come più efficiente la routine di MATLAB in termini di tempi di esecuzione, mentre in termini di errore i due algoritmi ottengono risultati dello stesso ordine di grandezza:

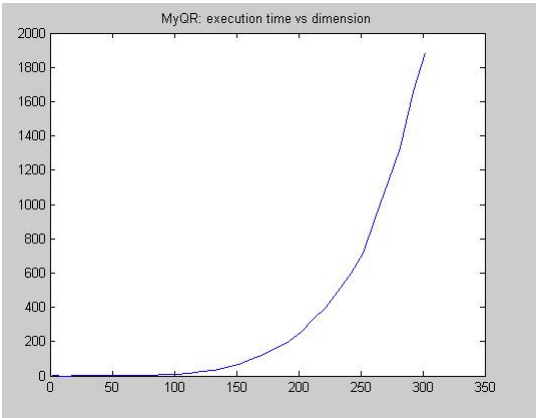

Figura 7: Tempo di esecuzione dell'algoritmo personalizzato

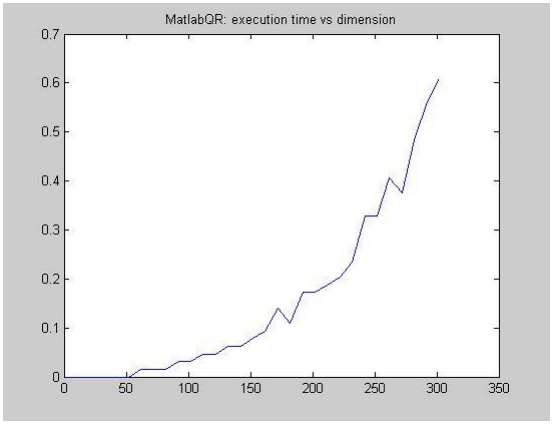

Figura 8: Tempo di esecuzione dell'algoritmo di MATLAB

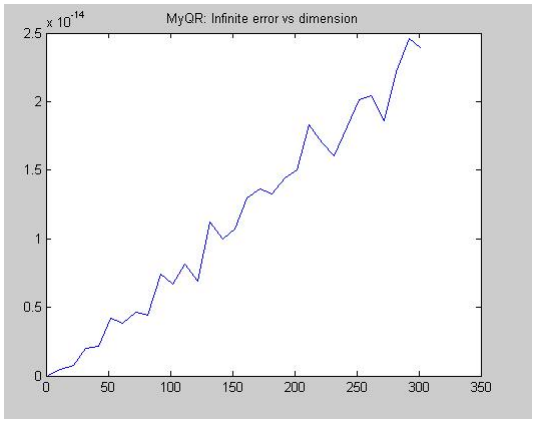

Figura 9: Errore in norma-∞ dell'algoritmo personalizzato

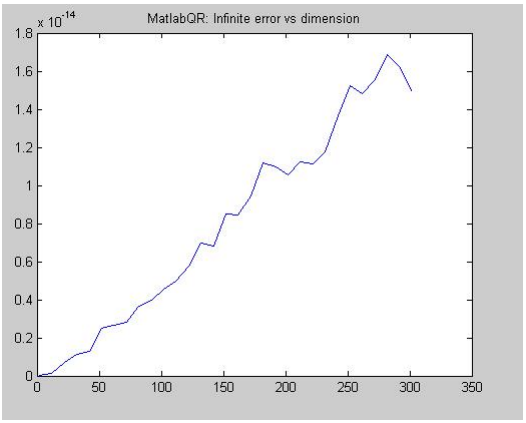

Figura 10: Errore in norma-∞ dell'algoritmo di MATLAB

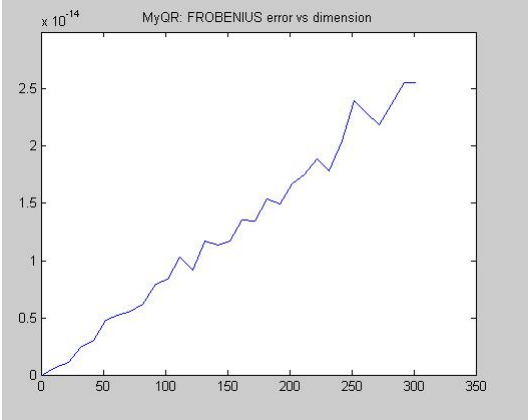

Figura 11: Errore in norma di Frobenius dell'algoritmo personalizzato

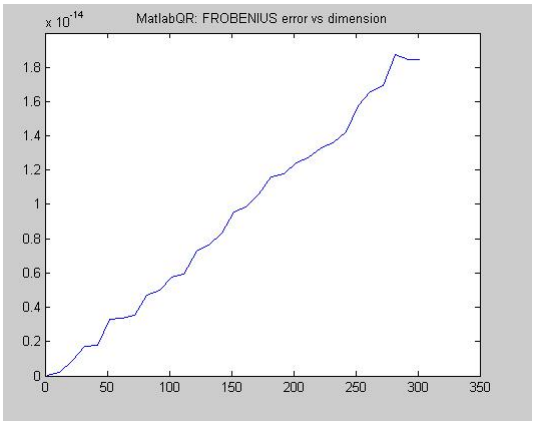

Figura 12: Errore in norma di Frobenius dell'algoritmo di MATLAB

3.3 Fattorizzazione di Cholesky

I test degli algoritmi per la fattorizzazione di Cholesky sono stati impostati in modo simile a quanto visto finora: si è trattato di confrontare i tempi di esecuzione dei due algoritmi e gli errori valutati in norma-∞ e in norma di Frobenius. L'errore è ora definito come la matrice $E = A - R^T R$.

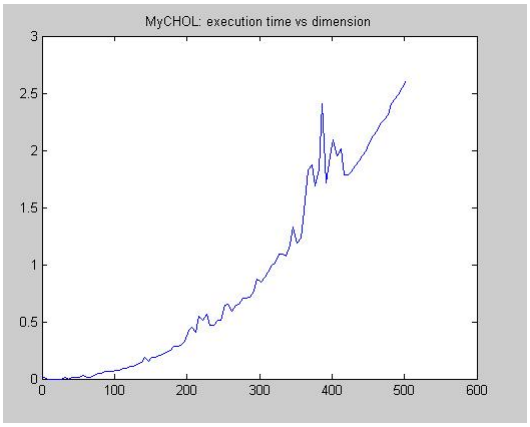

Figura 13: Tempo di esecuzione dell'algoritmo personalizzato

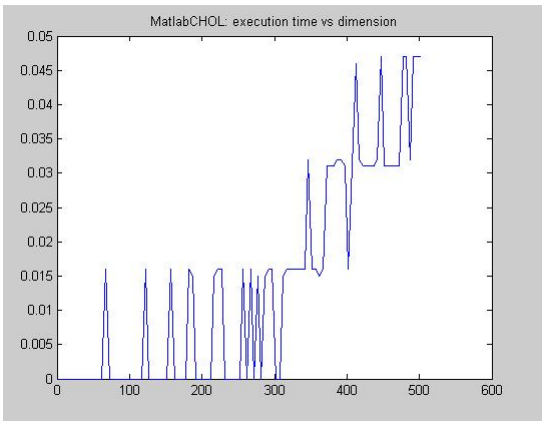

Figura 14: Tempo di esecuzione dell'algoritmo di MATLAB

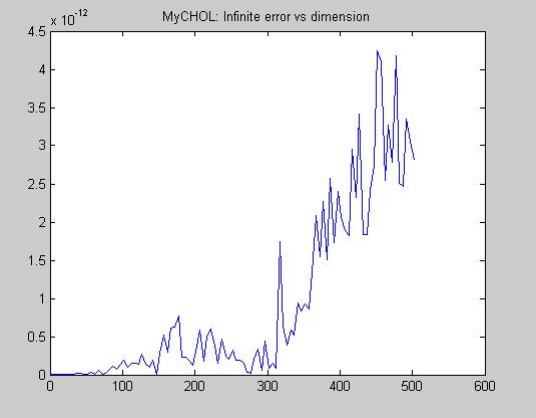

Figura 15: Errore in norma-∞ dell'algoritmo personalizzato

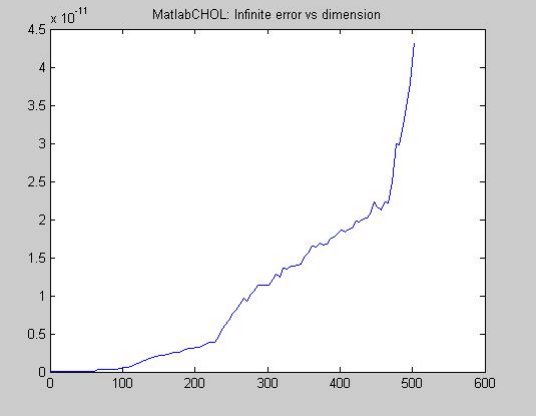

Figura 16: Errore in norma-∞ dell'algoritmo di MATLAB

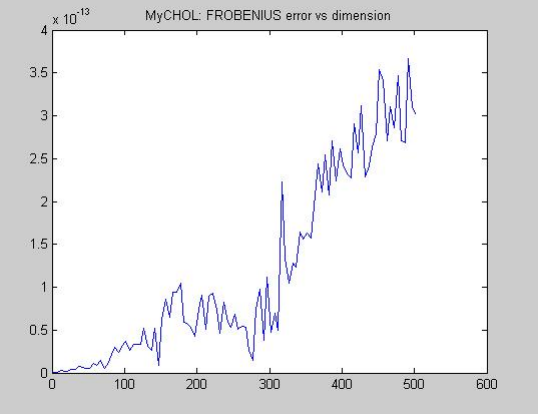

Figura 17: Errore in norma di Frobenius dell'algoritmo personalizzato

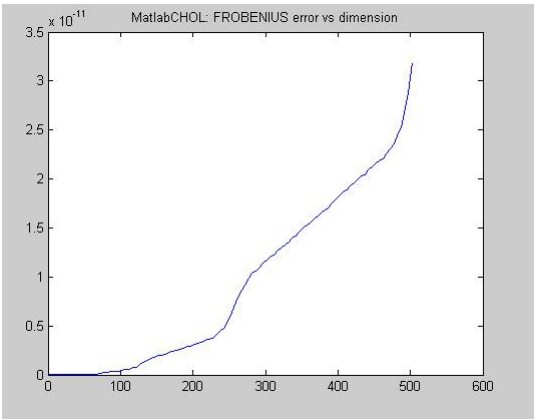

Figura 18: Errore in norma di Frobenius dell'algoritmo di MATLAB