UNIVERSITÀ DEGLI STUDI DI ROMA “TOR VERGATA”

FACOLTÀ DI SCIENZE M.F.N.

Corso di Laurea in Matematica

ALGORITMI VELOCI PER LA RISOLUZIONE NUMERICA DI SISTEMI LINEARI STRUTTURATI

Relatore
Prof. Giuseppe Rodriguez

Candidato
Emiliano Reali
(SM960)

ANNO ACCADEMICO 1998–1999

Ai mie genitori
sostegno della mia
brama di sapere

Indice

1 Introduzione **4**
- 1.1 Definizioni preliminari 5
- 1.2 Sistemi lineari 7
- 1.3 Matrici strutturate 9
- 1.4 Note Tecniche 10

2 Matrici strutturate classiche **11**
- 2.1 Matrici di Toeplitz 11
  - 2.1.1 Risultati numerici 17
- 2.2 Matrici di Vandermonde 22
  - 2.2.1 Risultati numerici 27
- 2.3 Altre classi di matrici strutturate 30
  - 2.3.1 Matrici di Cauchy 30
  - 2.3.2 Matrici di Hankel 31

3 Problemi che conducono a sistemi lineari strutturati **33**
- 3.1 Risoluzione di equazioni differenziali 33
- 3.2 Approssimazione polinomiale ai minimi quadrati in $L^2[a, b]$ 35

4 Struttura di spostamento **38**
- 4.1 Rango di spostamento 38
- 4.2 Struttura di spostamento 44
- 4.3 L'algoritmo di Schur 48
- 4.4 Fattorizzazione veloce e stabile di matrici di Cauchy 52
  - 4.4.1 Risultati numerici 57
- 4.5 Conversione di struttura 62
  - 4.5.1 Conversione da Toeplitz-like a Cauchy-like 63
  - 4.5.2 Conversione da Vandermonde-like a Cauchy-like 66
  - 4.5.3 Risultati numerici 67

Capitolo 1

Introduzione

Questa tesi ha come oggetto lo studio delle proprietà di un'ampia classe di matrici strutturate.

La nozione di "struttura di spostamento" (*displacement structure*), introdotta da Kailath [12], consente infatti di definire classi molto generali di matrici strutturate, che comprendono, come casi particolari, molte matrici strutturate classiche, quali le matrici di Vandermonde, Toeplitz, Cauchy, Hankel, etc.

All'interno di tali classi è possibile individuare importanti quantità invarianti, e costruire algoritmi di fattorizzazione, utilizzabili quindi per la risoluzione numerica di sistemi lineari, aventi complessità computazionale dell'ordine di n^2 [6].

Alcune ricerche più recenti, inoltre, sono state incentrate sulla costruzione di algoritmi caratterizzati da una maggiore stabilità numerica [3] e sullo studio dell'influenza del contenuto informativo derivante dalla struttura sul numero di condizionamento associato ad una matrice [4].

Questi studi hanno dato origine ad un importante filone di ricerca che ha coinvolto ricercatori di prim'ordine operanti nel settore dell'Algebra Lineare e dell'Analisi Numerica.

Il piano della tesi è il seguente: nel presente capitolo verranno introdotti i temi trattati nella tesi e verranno date alcune definizioni preliminari.

Nel Capitolo 2 fisseremo l'attenzione sulle matrici strutturate più note e presenteremo alcuni algoritmi *veloci* per la risoluzione di sistemi lineari caratterizzati da alcune di queste classi di matrici.

Nel Capitolo 3 saranno presentati alcuni problemi in cui intervengono matrici strutturate.

Infine nel Capitolo 4 parleremo della struttura di spostamento, che è materia di studi piuttosto recente e che riveste grande importanza sia teorica che applicativa. Questa teoria, attraverso la definizione di un opportuno operatore, permette di generalizzare il concetto di struttura e di individuare alcune importanti quan-

tità invarianti. Sfruttando le proprietà dell'operatore di spostamento, verrà quindi studiata una versione dell'algoritmo di fattorizzazione LU di Schur con *pivoting* di colonna, ottimizzato per matrici *Cauchy-like*. Infine verranno illustrati degli algoritmi che consentono di trasformare alcuni tipi di matrici strutturate in matrici *Cauchy-like*.

Tutti gli algoritmi studiati sono stati implementati e sono stati confrontati sia tra loro che col classico metodo di triangolarizzazione di Gauss, mediante una sperimentazione numerica tesa a verificare la loro *performance*.

1.1 Definizioni preliminari

Definizione 1.1 *Fissata una norma matriciale, si definisce numero di condizionamento di un sistema lineare la quantità*

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|,$$

essendo A la matrice dei coefficienti del sistema.

Il numero di condizionamento di un problema quantifica la sensibilità del risultato rispetto agli errori sui dati. Specificatamente si parlerà di problema malcondizionato se per piccole perturbazioni sui dati si possono avere grandi perturbazioni sui risultati. Viceversa un problema è ben condizionato se per piccole perturbazioni sui dati si hanno piccole perturbazioni sui risultati.

Un esempio chiarificatore di questo concetto è dato dal seguente sistema

$$\begin{cases} x - y = 1 \\ x - \alpha y = 0 \end{cases} \tag{1.1}$$

con $\alpha \in \mathbb{R}$. Questo sistema dal punto di vista matriciale si può scrivere nella forma $A\mathbf{x} = \mathbf{b}$ con

$$A = \begin{bmatrix} 1 & -1 \\ 1 & -\alpha \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Consideriamo una piccola perturbazione sui dati ed in particolare

$$\alpha_1 = 1,00001 \quad e \quad \alpha_2 = 0,99999$$

e osserviamo cosa succede ai risultati. Nel caso di $\alpha = \alpha_1$ abbiamo

$$\begin{cases} x = 10^5 + 1 \\ y = 10^5 \end{cases}$$

e con $\alpha = \alpha_2$ abbiamo

$$\begin{cases} x = -10^5 + 1 \\ y = -10^5 \end{cases},$$

si ha cioè un errore dell'ordine di 10^5 sulla soluzione a fronte di una variazione di 10^{-5} sui dati. Infatti in entrambi i casi per il condizionamento si ha il valore

$$\kappa(A) = 4 \cdot 10^5$$

relativamente alla norma 2.

Le informazioni sul condizionamento di un sistema lineare sono cruciali, in quanto operando su un calcolatore digitale tutti i numeri reali utilizzati sono affetti da un errore relativo dell'ordine della precisione di macchina (circa 10^{-16}, se si utilizzano variabili in doppia precisione). Inoltre, nelle applicazioni i dati sono spesso perturbati in modo molto più consistente a causa di errori sperimentali e/o di misura.

La successione delle operazioni che porta alla soluzione di un problema è detta *algoritmo di risoluzione*. Spesso per uno stesso problema esistono diversi algoritmi e quindi è cruciale sapere quale degli eventuali candidati risulta il migliore. Per questo ultimo concetto si devono fare una serie di distinzioni, infatti un algoritmo può risultare più veloce dal punto di vista del tempo impiegato da un elaboratore per eseguirlo, ma non essere sufficientemente accurato.

Definizione 1.2 *Definiremo stabilità di un algoritmo la sensibiltà dell'algoritmo stesso rispetto agli errori sulle operazioni.*

La stabilità è una proprietà intrinseca di un algoritmo e algoritmi differenti risulteranno più o meno stabili.

Definizione 1.3 *Definiremo complessità computazionale di un algoritmo il numero di operazioni aritmetiche che sono necessarie per il calcolo effettivo della soluzione.*

Il termine complessità avrà prevalentemente un significato aritmetico come l'intuizione della parola lascia intendere. Nella sua valutazione la complessità di un algoritmo è sempre legata alla dimensione del problema in esame. Se quest'ultima è n, la funzione di n che rappresenta la sua complessità sarà spesso considerata per il suo comportamento asintotico, cioè al crescere indefinito di n. Per chiarezza con $g(n) = O(f(n))$ $(f(n), g(n) \geq 0)$ si intenderà il fatto che la funzione $g(n)$ "non cresce più rapidamente di $f(n)$", cioè che esiste una costante c tale che $g(n) \leq cf(n)$ con l'eccezione di un insieme (eventualmente vuoto) di

valori non negativi di n. Il simbolo $O(1)$ denoterà una funzione delimitata, al crescere di n, da una costante. Per unità di misura della complessità di un algoritmo si intenderà una singola operazione aritmetica $(+, -, *, \div)$, spesso indicata con *flop* (*floating point operation*).

La complessità di calcolo può essere riguardata principalmente da due punti di vista: la *sintesi* e l'*analisi* di algoritmi. La sintesi consiste nella costruzione di algoritmi sempre più efficienti e veloci; l'analisi consiste invece nello studio di limiti inferiori della complessità di un problema e riguarda più direttamente il grado di difficoltà intrinseca della sua risoluzione.

1.2 Sistemi lineari

La risoluzione dei sistemi lineari è il problema più diffuso nelle applicazioni della matematica e completamente noto dal punto di vista teorico. L'Algebra Lineare fornisce condizioni per l'esistenza e l'unicità della soluzione e algoritmi per il suo calcolo effettivo. Il problema di alcuni algoritmi classici è dato dall'elevato numero di operazioni aritmetiche richiesto per la loro applicazione. È facile verificare ad esempio che la formula di Cramer richiede un numero di operazioni dell'ordine di $(n+1)!$ se n è la dimensione del sistema. Tale complessità computazionale rende di fatto non risolubili sistemi di dimensioni moderatamente elevate anche sui computer più veloci attualmente disponibili.

Un sistema di equazioni lineari può essere scritto nella forma

$$\sum_{j=1}^{n} a_{ij} x_j = b_j \quad i = 1, \ldots, n$$

o, in notazione compatta,

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

dove $A = (a_{ij})$ è una matrice $n \times n$ e $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$. Gli algoritmi numerici per la risoluzione dei sistemi lineari sono basati sulla fattorizzazione della matrice A, contenente i coefficienti del sistema, nel prodotto di 2 o più matrici aventi una forma più semplice rispetto alla matrice di partenza. È necessario inoltre che la fattorizzazione e la risoluzione dei sistemi vengano eseguiti con algoritmi numericamente stabili. L'algoritmo di Gauss, ad esempio, fornisce una fattorizzazione del tipo

$$A = LU$$

dove L è una matrice triangolare inferiore con elementi diagonali $l_{ii} = 1$, e U una matrice triangolare superiore. I sistemi lineari risultanti sono quindi

$$\begin{cases} L\mathbf{y} = \mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{cases}$$

che si possono risolvere rispettivamente mediante gli algoritmi di *forward* e *backward substitution*. Questo procedimento è efficace se il costo computazionale della fattorizzazione è contenuto e se i due sistemi lineari risultanti, come in questo caso, sono in una forma più semplice di quello di partenza. Per la fattorizzazione LU la complessità è $O(\frac{1}{3}n^3)$ e per la soluzione dei due sistemi triangolari è $O(n^2)$.

La fattorizzazione consente anche di effettuare il calcolo del determinante e della matrice inversa. Infatti poiché la matrice L ha tutti 1 sulla diagonale principale, il determinante di A si riduce al prodotto degli elementi della diagonale principale di U. Invece, riguardo alla matrice inversa, se pensiamo ad A^{-1} composta di vettori colonna

$$A^{-1} = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$$

si deve avere

$$A\,[\mathbf{v}_1, \ldots, \mathbf{v}_n] = [\mathbf{e}_1, \ldots, \mathbf{e}_n]$$

dove gli $\mathbf{e}_k$ sono i vettori colonna della base canonica di $\mathbb{R}^n$ e quindi

$$A\mathbf{v}_k = \mathbf{e}_k, \quad k = 1, \ldots, n.$$

Questi n sistemi lineari possono essere risolti con complessità $O(n^3)$, effettuando la fattorizzazione LU della matrice A e risolvendo $2n$ sistemi triangolari.

L'algoritmo di Gauss applicato ad una qualsiasi matrice non singolare potrebbe bloccarsi nel caso in cui trovi un *pivot*, cioè un elemento diagonale, uguale a zero. Questo inconveniente è superabile con l'introduzione del pivoting parziale o di colonna. In pratica questa modifica dell'algoritmo di Gauss prevede che al passo k si cerchi l'elemento di massimo modulo nella colonna k-esima e si scambi di posto la relativa riga con la riga k. Questa variante migliora l'algoritmo di Gauss rendendolo applicabile a qualsiasi matrice non singolare e più stabile numericamente.

Dal punto di vista matriciale la fattorizzazione con pivoting si scrive

$$PA = LU,$$

e conduce alla risoluzione dei due sistemi triangolari

$$\begin{cases} L\mathbf{y} = P\mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{cases}.$$

Quando l'algoritmo di Gauss viene applicato col pivoting di colonna, si può dare un limite superiore per la crescita del numero di condizionamento

$$\kappa_\infty(U) \leq 4^{n-1}\kappa_\infty(A) \tag{1.2}$$

dove con κ_∞ indichiamo il numero di condizionamento calcolato con la norma infinito. Anche se questo limite non viene raggiunto nella maggioranza dei casi, esso rende comunque sconsigliabile l'applicazione dell'algoritmo di Gauss a matrici malcondizionate.

1.3 Matrici strutturate

In molti problemi si presentano sistemi lineari la cui matrice dei coefficienti è dotata di una particolare struttura, cioè dipende da un numero di parametri inferiore ad n^2.

Definizione 1.4 *Diremo che una matrice è strutturata se dipende da αn parametri con α indipendente da n.*

La struttura di una matrice consente quindi un enorme guadagno nell'occupazione di memoria, soprattutto per matrici di grandi dimensioni.

Un semplice esempio di matrici strutturate sono le matrici a banda, i cui elementi a_{ij} sono nulli per $i - j > m_1$ e $i - j < m_2$, essendo m_1 e m_2 gli interi che indicano le *ampiezze di banda.*

Definizione 1.5 *Diremo che un algoritmo per la risoluzione di un sistema lineare, o la fattorizzazione di una matrice, è "veloce" se ha una complessità $O(n^p)$, con $p < 3$.*

Alcuni algoritmi veloci classici non risultano numericamente stabili, in quanto non prevedono l'uso del pivoting. L'applicazione del pivoting risulta spesso impossibile in quanto può distruggere la struttura di una matrice.

In taluni casi la struttura di una matrice risulta nascosta, pur essendo comunque presente. Ad esempio, in una matrice a banda le cui righe e colonne abbiano subito una permutazione risulta visibile ad un primo esame solo la sparsità, ma non la struttura. Vediamo anche un'altro esempio non banale. Sappiamo che una matrice di Toeplitz non singolare $n \times n$, con i minori principali diversi da zero, si inverte con un numero di operazioni aritmetiche dell'ordine di n^2 che, comparate all'ordine di n^3 necessarie per una matrice generica, rappresentano un enorme vantaggio soprattutto per n abbastanza grande. Ma se sappiamo in qualche modo, che una matrice non di Toeplitz è l'inversa di qualche matrice di Toeplitz, allora si

mostra che considerando questo aspetto è possibile invertire la matrice non Toeplitz con $O(n^2)$ moltiplicazioni, anche se la struttura non è di fatto presente. Ad esempio se T è la matrice di Toeplitz

$$T = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix},$$

la sua inversa è

$$T^{-1} = \frac{1}{10} \begin{bmatrix} 6 & -5 & 0 & 1 \\ -5 & 10 & -5 & 0 \\ 0 & -5 & 10 & -5 \\ 1 & 0 & -5 & 6 \end{bmatrix}$$

che non è di Toeplitz. Risulterebbe utile, in casi come questo, un metodo per rilevare la presenza della struttura, e degli algoritmi che traggano vantaggio da essa.

1.4 Note Tecniche

Gli algoritmi studiati in questa tesi sono stati implementati usando l'ambiente di calcolo e visualizzazione scientifica Matlab [14]. I calcoli sono stati effettuati su un elaboratore Pentium 166 Mhz. Alcune elaborazioni su matrici di grandi dimensione sono state effettuate, sempre in Matlab, sulla workstation Alpha-server 1000 5/266 Mhz (axdid.mat.uniroma2.it) dell'Università di Roma "Tor Vergata". Per la scrittura della presente tesi è stato usato il linguaggio di composizione testi LaTeX. I grafici presenti nella tesi sono stati prodotti con Matlab a partire dai dati sperimentali e sono stati esportati in formato *Encapsulated Postscript* (EPS) per essere incorporati nel testo LaTeX.

Capitolo 2

Matrici strutturate classiche

2.1 Matrici di Toeplitz

Definizione 2.1 *Una matrice $T \in \mathbb{R}^{n\times n}$ si dice di Toeplitz se esistono $2n-1$ scalari $t_{-n+1}, \ldots, t_0, \ldots, t_{n-1}$ tali che T = $(t_{ij})_{i,j=1,\ldots,n}$ con $t_{ij} = t_{i-j}$*

$$T = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & \ldots & t_{-n+1} \\ t_1 & t_0 & t_{-1} & \ldots & t_{-n+2} \\ t_2 & t_1 & t_0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & t_{-1} \\ t_{n-1} & t_{n-2} & \ldots & t_1 & t_0 \end{bmatrix}. \tag{2.1}$$

Se la matrice T è simmetrica, allora dipende da n scalari $t_0, \ldots, t_{n-1}$ tali che t_{ij} = $t_{|i-j|}$

$$T = \begin{bmatrix} t_0 & t_1 & t_2 & \ldots & t_{n-1} \\ t_1 & t_0 & t_1 & \ldots & t_{n-2} \\ t_2 & t_1 & t_0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & t_1 \\ t_{n-1} & t_{n-2} & \ldots & t_1 & t_0 \end{bmatrix}. \tag{2.2}$$

Le matrici di Toeplitz sono un importante esempio di matrici strutturate. Va subito evidenziato come per queste matrici sia limitata l'occupazione di memoria. Infatti, se per una matrice qualsiasi di ordine n le celle di memoria da occupare sono n^2, per una Toeplitz esse sono appunto $2n-1$ e n nel caso di una matrice

di Toeplitz simmetrica. Queste semplici osservazioni, unite alla presenza di queste matrici in numerosi problemi applicativi, giustificano lo studio sistematico di algoritmi per matrici di Toeplitz.

Esistono degli algoritmi ormai classici per questa classe di matrici. Nel caso delle matrici di Toeplitz simmetriche definite positive i più noti sono l'algoritmo di Durbin (1960) e quello di Levinson (1947).

L'algoritmo di Durbin, dati i numeri reali $r_0 = 1, r_1, \ldots, r_n$ tali che $T = (r_{|i-j|})_{i,j=1,\ldots,n}$ è definita positiva, calcola $\mathbf{y} \in \mathbb{R}^n$ tale che

$$T\mathbf{y} = -(r_1, \ldots, r_n)^T.$$

Questa equazione è detta di Yule-Walker ed è legata a problemi di predizione lineare. Supporre $r_0 = 1$ non è restrittivo in quanto un sistema lineare caratterizzato da una generica matrice di Toeplitz definita positiva del tipo (2.2) può essere ricondotto in questa forma dividendo la matrice dei coefficienti ed il termine noto per l'elemento diagonale t_0.

L'algoritmo risolve ricorsivamente, per $k = 1, \ldots, n$, il sistema di Yule-Walker di ordine k

$$T_k\mathbf{y} = -\mathbf{r}_k = -(r_1, \ldots, r_k)^T,$$

dove

$$T_k = \begin{bmatrix} 1 & r_1 & r_2 & \cdots & r_{k-1} \\ r_1 & 1 & \ddots & & r_{k-2} \\ r_2 & \ddots & 1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & r_1 \\ r_{k-1} & r_{k-2} & \cdots & r_1 & 1 \end{bmatrix}.$$

Osserviamo che una matrice di Toeplitz è persimmetrica [1].

Definizione 2.2 *Una matrice $A \in \mathbb{R}^{n\times n}$ si dice persimmetrica se è simmetrica rispetto all'antidiagonale, cioè rispetto alla diagonale che unisce gli elementi dal posto $(1, n)$ al posto $(n, 1)$.*

In sostanza gli elementi di una matrice persimmetrica verificano la relazione

$$a_{ij} = a_{n-j+1,n-i+1}, \quad i, j = 1, \ldots, n.$$

Introduciamo la matrice *di scambio* E_n di ordine n, che si ottiene dalla matrice identità invertendo l'ordine delle righe. Questa matrice è idempotente in quanto

$E_n E_n = I$. È immediato verificare che una matrice A è persimmetrica se e solo se

$$E_n A E_n = A^T.$$

Una conseguenza di questa relazione è che l'inversa di una matrice persimmetrica è a sua volta persimmetrica. Inoltre se una matrice è sia simmetrica che persimmetrica (ossia *centrosimmetrica*), come nel nostro caso, si ha

$$AE_n = E_n A.$$

Supponiamo di aver risolto il sistema di Yule-Walker di ordine k. Mostriamo che il sistema di Yule-Walker di ordine $k+1$

$$\begin{bmatrix} T_k & E_k \mathbf{r}_k \\ \mathbf{r}_k^T E_k & 1 \end{bmatrix} \quad \begin{bmatrix} \mathbf{z} \\ \alpha \end{bmatrix} = - \begin{bmatrix} \mathbf{r}_k \\ r_{k+1} \end{bmatrix}$$

si risolve con $O(k)$ operazioni. Osserviamo innanzitutto che

$$\mathbf{z} = T_k^{-1}(-\mathbf{r}_k - \alpha E_k \mathbf{r}_k) = \mathbf{y} - \alpha T_k^{-1} E_k \mathbf{r}_k$$

e

$$\alpha = -r_{k+1} - \mathbf{r}_k^T E_k \mathbf{z}.$$

Poiché T_k^{-1} è persimmetrica, abbiamo $T_k^{-1} E_k = E_k T_k^{-1}$ e di conseguenza

$$\mathbf{z} = \mathbf{y} - \alpha E_k T_k^{-1} \mathbf{r}_k = \mathbf{y} + \alpha E_k \mathbf{y}.$$

Ora, se inseriamo nell'espressione di α il valore della $\mathbf{z}$ trovato, otteniamo

$$\alpha = -r_{k+1} - \mathbf{r}_k^T E_k (\mathbf{y} + \alpha E_k \mathbf{y}) = -\frac{r_{k+1} + \mathbf{r}_k^T E_k \mathbf{y}}{1 + \mathbf{r}_k^T \mathbf{y}}.$$

Di questa espressione bisogna notare che il denominatore è positivo. Infatti, ponendo

$$M = \begin{bmatrix} I & E_k \mathbf{y} \\ 0 & 1 \end{bmatrix}$$

risulta

$$M^T T_{k+1} M = \begin{bmatrix} T_k & 0 \\ 0 & 1 + \mathbf{r}_k{}^T \mathbf{y} \end{bmatrix}.$$

Da questa relazione, essendo T_{k+1} definita positiva e M non singolare, segue la positività del numero $1 + \mathbf{r}_k^T \mathbf{y}$.

In questo modo abbiamo illustrato il passo k-esimo dell'algoritmo proposto da Durbin. Questo procedimento risolve ricorsivamente il sistema di Yule-Walker

$$T_k \mathbf{y}^{(k)} = -\mathbf{r}_k$$

per $k = 1, \ldots, n$ come segue

$$\begin{array}{l} \mathbf{y}^{(1)} = -\mathbf{r}_1 \\ \text{for } k = 1, \ldots, n-1 \\ \qquad \beta_k = 1 + \mathbf{r}_k^T \mathbf{y}^{(k)} \\ \qquad \alpha_k = -\dfrac{r_{k+1} + \mathbf{r}_k^T E_k \mathbf{y}^{(k)}}{\beta_k} \\ \qquad \mathbf{z}^{(k)} = \mathbf{y}^{(k)} + \alpha_k E_k \mathbf{y}^{(k)} \\ \qquad \mathbf{y}^{(k+1)} = \begin{bmatrix} \mathbf{z}^{(k)} \\ \alpha_k \end{bmatrix} \\ \text{end} \end{array}$$

che conduce a determinare $\mathbf{y} = \mathbf{y}^{(n)}$. Procedendo così si richiedono $3n^2$ *flops.* È possibile tuttavia ridurre la complessità usando un'altra espressione per β_k

$$\begin{aligned} \beta_k &= 1 + \mathbf{r}_k^T \mathbf{y}^{(k)} \\ &= 1 + \begin{bmatrix} \mathbf{r}_{k-1}^T & r_k \end{bmatrix} \begin{bmatrix} \mathbf{y}^{(k-1)} + \alpha_{k-1} E_{k-1} \mathbf{y}^{(k-1)} \\ \alpha_{k-1} \end{bmatrix} \\ &= (1 + \mathbf{r}_{k-1}^T \mathbf{y}^{(k-1)}) + \alpha_{k-1}(\mathbf{r}_{k-1}^T E_{k-1} \mathbf{y}^{(k-1)} + r_k) \\ &= \beta_{k-1} + \alpha_{k-1}(-\beta_{k-1}\alpha_{k-1}) \\ &= (1 - \alpha_{k-1}^2)\beta_{k-1}. \end{aligned}$$

Questo algoritmo ha una complessità di $O(n^2)$ e se usiamo la seguente implementazione in ambiente Matlab

```
function y=dur(r)
n=size(r,1);
y=zeros(n,1);
a=zeros(n,1);
y(1)=-r(1);
b=1;
a=-r(1);
for  k=1:n-1
```

```
    b=(1-a^2)*b;
    a=-(r(k+1)+(r(k:-1:1)'*y(1:k)))/b;
    z(1:k)=y(1:k)+a*y(k:-1:1);
    y(1:k+1)=[z(1:k);a];
end
```

eseguiamo $2n^2$ operazioni o *flops.*

L'algoritmo di Levinson consente la risoluzione di un sistema lineare simmetrico definito positivo dotato di un termine noto arbitrario. Infatti, dati $\mathbf{b} \in \mathbb{R}^n$ e i numeri reali $r_0 = 1, r_1, \ldots, r_{n-1}$ tali che $T = (r_{|i-j|}) \in \mathbb{R}^{n\times n}$ è definita positiva, l'algoritmo di Levinson calcola $\mathbf{x} \in \mathbb{R}^n$ tale che $T\mathbf{x} = \mathbf{b}$. Supponiamo di aver risolto il sistema di ordine k

$$T_k\mathbf{x} = \mathbf{b}_k = (b_1, \ldots, b_k)^T, \tag{2.3}$$

dove T_k è la matrice introdotta precedentemente, e di voler risolvere il sistema

$$\begin{bmatrix} T_k & E_k\mathbf{r}_k \\ \mathbf{r}_k^T E_k & 1 \end{bmatrix} \quad \begin{bmatrix} \mathbf{v} \\ \mu \end{bmatrix} = \begin{bmatrix} \mathbf{b}_k \\ b_{k+1} \end{bmatrix}. \tag{2.4}$$

Qui $\mathbf{r}_k = (r_1, \ldots, r_k)^T$ come prima. Assumiamo anche che la soluzione del sistema di Yule-Walker di ordine k

$$T_k\mathbf{y} = -\mathbf{r}_k$$

sia disponibile. Dalla relazione

$$T_k\mathbf{v} + \mu E_k\mathbf{r}_k = \mathbf{b}_k,$$

corrispondente alle prime k righe del sistema 2.4, sfruttando la persimmetria della matrice T_k, si ottiene che

$$\mathbf{v} = T_k^{-1}(\mathbf{b}_k - \mu E_k\mathbf{r}_k) = \mathbf{x} - \mu T_k^{-1}E_k\mathbf{r}_k = \mathbf{x} + \mu E_k\mathbf{y}$$

e quindi

$$\begin{aligned} \mu &= b_{k+1} - \mathbf{r}_k^T E_k\mathbf{v} \\ &= b_{k+1} - \mathbf{r}_k^T E_k\mathbf{x} - \mu\mathbf{r}_k^T\mathbf{y} \\ &= \frac{b_{k+1} - \mathbf{r}_k^T E_k\mathbf{x}}{1 + \mathbf{r}_k^T\mathbf{y}}. \end{aligned}$$

Resta così mostrato che per passare da (2.3) a (2.4) è sufficiente eseguire $O(k)$ operazioni. In conclusione un metodo efficiente di risoluzione per il sistema $T\mathbf{x} = \mathbf{b}$ si ottiene risolvendo *in parallelo* i due sistemi

$$T_k\mathbf{x}^{(k)} = \mathbf{b}_k \quad \text{e} \quad T_k\mathbf{y}^{(k)} = -\mathbf{r}_k$$

per $1 \leq k \leq n$. Questi sono i passi dell'algoritmo di Levinson.

Come per l'algoritmo di Durbin, la complessità asintotica è $O(n^2)$. Utilizzando la seguente implementazione Matlab

```
function x=lev(t,b)
if length(t)==1
   x = b/t;
   return
end
r=t(2:end)/t(1);
b=b/t(1);
n=size(t,1);
x=zeros(n,1);
y=zeros(n,1);
u=zeros(n,1);
a=zeros(n,1);
y(1)=-r(1);
x(1)=b(1);
beta=1;
a=-r(1);
for k=1:n-1
   beta=(1-a^2)*beta;
   u=(b(k+1)-(r(1:k)'*x(k:-1:1)))/beta;
   x(1:k+1)=[x(1:k)+u*y(k:-1:1);u];
   if k<n-1
      a=-(r(k+1)+(r(1:k)'*y(k:-1:1)))/beta;
      z(1:k)=y(1:k)+a*y(k:-1:1);
      y(1:k+1)=[z(1:k);a];
   end
end
```

vengono eseguite esattamente $4n^2$ *flops*. Tale implementazione è applicabile a matrici di Toeplitz simmetriche definite positive con elemento diagonale qualsiasi.

Si può utilizzare una ricorsione simile anche per il caso non simmetrico. Supponiamo che, dati gli scalari $r_1, \ldots, r_{n-1}$, $p_1, \ldots, p_{n-1}$ e $b_1, \ldots, b_n$, sia necessa-

rio risolvere il sistema lineare $T\mathbf{x} = \mathbf{b}$ della forma

$$\begin{bmatrix} 1 & r_1 & r_2 & \cdots & r_{n-1} \\ p_1 & 1 & & & r_{n-2} \\ p_2 & & \ddots & & \vdots \\ \vdots & & & 1 & r_1 \\ p_{n-1} & p_{n-2} & \cdots & p_1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \\ \vdots \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \\ \vdots \\ \\ b_n \end{bmatrix}.$$

Nel procedimento che segue si richiede che le sottomatrici principali $T_k, k = 1, \ldots, n$, siano non singolari. Usando la notazione introdotta precedentemente, si mostra che note le soluzioni dei sistemi di ordine k

$$\begin{aligned} T_k^T \mathbf{y} &= -\mathbf{r}_k = -[r_1, r_2, \ldots, r_k]^T \\ T_k \mathbf{w} &= -\mathbf{p}_k = -[p_1, p_2, \ldots, p_k]^T \\ T_k \mathbf{x} &= \mathbf{b}_k = [b_1, b_2, \ldots, b_k]^T, \end{aligned}$$

si può ottenere la soluzione dei sistemi

$$\begin{bmatrix} T_k & E_k\mathbf{r}_k \\ \mathbf{p}_k^T E_k & 1 \end{bmatrix}^T \begin{bmatrix} \mathbf{z} \\ \alpha \end{bmatrix} = - \begin{bmatrix} \mathbf{r}_k \\ r_{k+1} \end{bmatrix}$$

$$\begin{bmatrix} T_k & E_k\mathbf{r}_k \\ \mathbf{p}_k^T E_k & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \nu \end{bmatrix} = - \begin{bmatrix} \mathbf{p}_k \\ p_{k+1} \end{bmatrix} \tag{2.5}$$

$$\begin{bmatrix} T_k & E_k\mathbf{r}_k \\ \mathbf{p}_k^T E_k & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mu \end{bmatrix} = \begin{bmatrix} \mathbf{b}_k \\ b_{k+1} \end{bmatrix}$$

con $O(n)$ operazioni. In questo modo è possibile risolvere un sistema di Toeplitz non simmetrico con $O(n^2)$ operazioni. Tuttavia la stabiltà del processo è assicurata solo se le matrici T_k sono sufficientemente ben condizionate.

2.1.1 Risultati numerici

Per verificare numericamente le prestazioni dell'algoritmo di Levinson è stato considerato un sistema lineare del tipo

$$T\mathbf{x} = \mathbf{b},$$

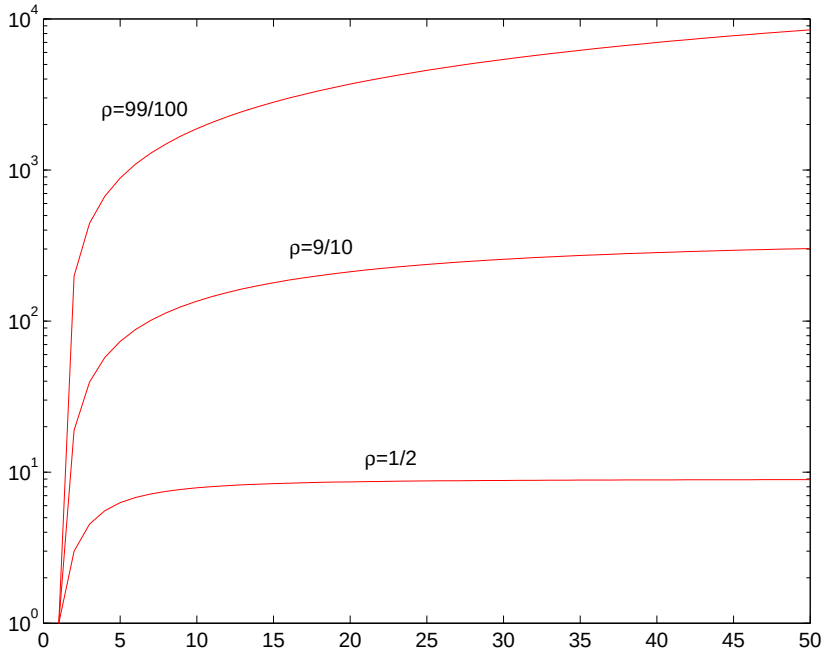

Figura 2.1: condizionamento delle matrici *KMS*

dove la matrice T è stata scelta in alcune famiglie di matrici test presenti in letteratura [11] e il termine noto è stato costruito mediante il prodotto $\mathbf{b} = T\mathbf{e}$, fissando arbitrariamente la soluzione $\mathbf{e} = (1, 1, \ldots, 1)^T$.

Tale sistema lineare è stato risolto, al crescere della dimensione, mediante la nostra implementazione dell'algoritmo di Levinson e mediante l'algoritmo di Gauss con pivoting di colonna disponibile nella libreria di Matlab [14]. Nel corso del calcolo sono stati memorizzati il numero di operazioni eseguite dai due algoritmi e gli errori riscontrati nella soluzione numerica calcolati in norma-∞

$$\|\mathbf{x} - \mathbf{e}\|_\infty = \max_{i=1,\ldots,n} |x_i - 1|.$$

Un primo test è stato effettuato utilizzando matrici *KMS* (Kac-Murdock-Szego). Queste matrici dipendono da un parametro ρ, e sono definitive positive per $0 < \rho < 1$. Il suo elemento generico è dato da

$$T(i, j) = \rho^{|i-j|}$$

e per i nostri test abbiamo fissato $\rho = 1/2, 9/10, 99/100$. Come evidenziato in Figura 2.1, il condizionamento di queste matrici è molto sensibile alla variazione del parametro ρ.

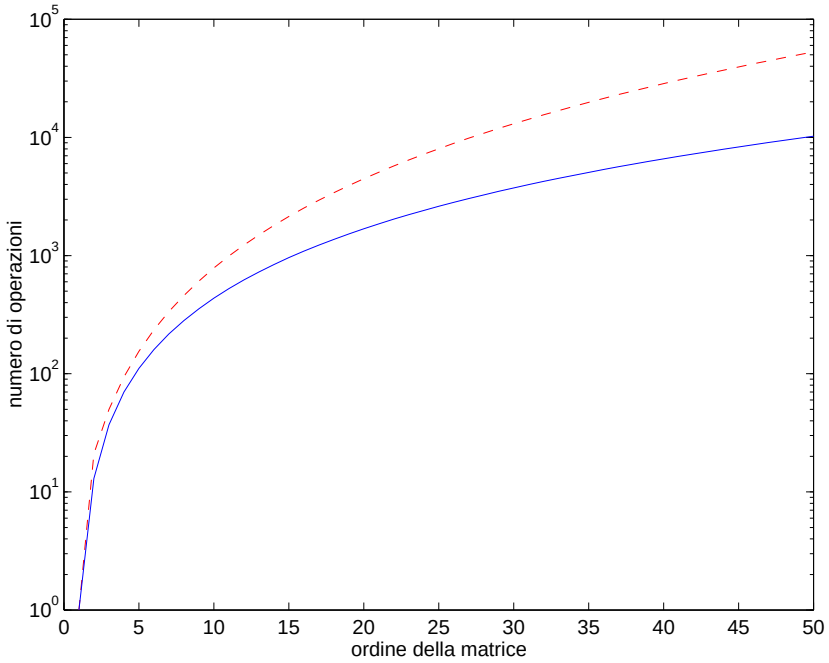

Figura 2.2: complessità computazionale, Levinson vs. Gauss

La Figura 2.2, che riporta la complessità computazionale dei due algoritmi utilizzati, evidenzia il vantaggio computazionale derivante dall'utilizzazione dell'algoritmo veloce: come già detto la complessità computazionale per Gauss (linea tratteggiata) segue l'ordine di $O(n^3)$ e per Levinson (linea continua) segue l'ordine di $O(n^2)$. Nelle Figure 2.3, 2.4 e 2.5 sono invece riportati gli errori riportati dai due algoritmi per i vari valori di ρ.

Come si può osservare, l'algoritmo di Gauss risulta più accurato al crescere del condizionamento, grazie alla maggiore stabilità assicurata dal pivoting di colonna. Quando il condizionamento è basso, invece, i due algoritmi hanno la stessa accuratezza. In particolare per $\rho = 1/2$ l'algoritmo di Levinson fornisce un errore nullo, mentre l'errore prodotto dall'algoritmo di Gauss è dell'ordine della precisione di macchina.

Il nostro secondo esempio e costituito dalle matrici *PROLATE* [17] definite da

$$T = (t_{i-j}) \quad \text{con} \quad t_k = \begin{cases} 2w & \text{se } k = 0, \\ \frac{\sin(2\pi w k)}{\pi k} & \text{altrimenti.} \end{cases}$$

Esse dipendono da un parametro w, da noi fissato a 0.25, e risultano definite positive per $0 < w < 1/2$. Sono matrici molto mal condizionate, come evidenziato dalla Figura 2.6, e per questo motivo abbiamo limitato la dimensione a $n \leq 25$. La Figura 2.7 illustra i risultati numerici ottenuti in questo caso, e conferma le

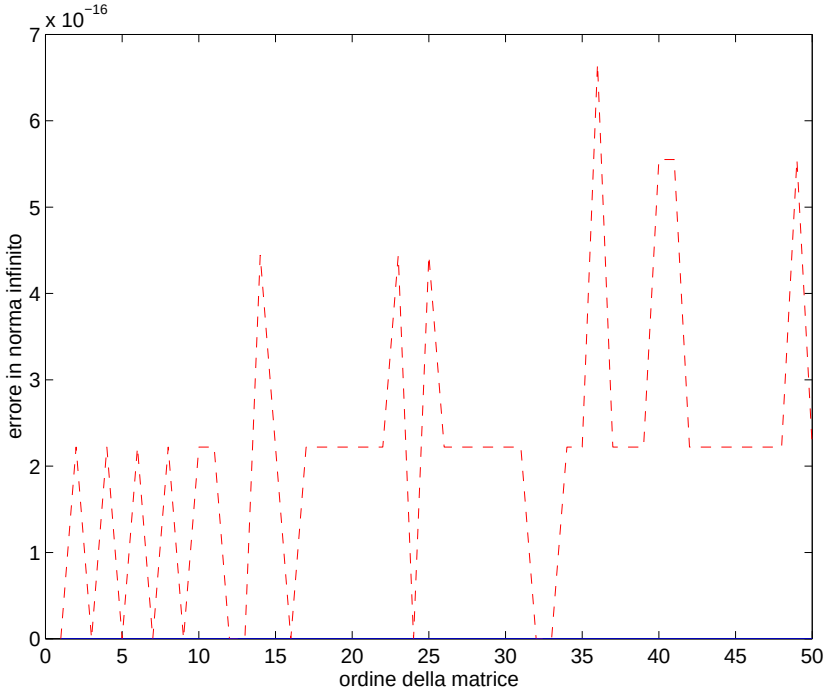

Figura 2.3: errori, *KMS*, $\rho = 1/2$

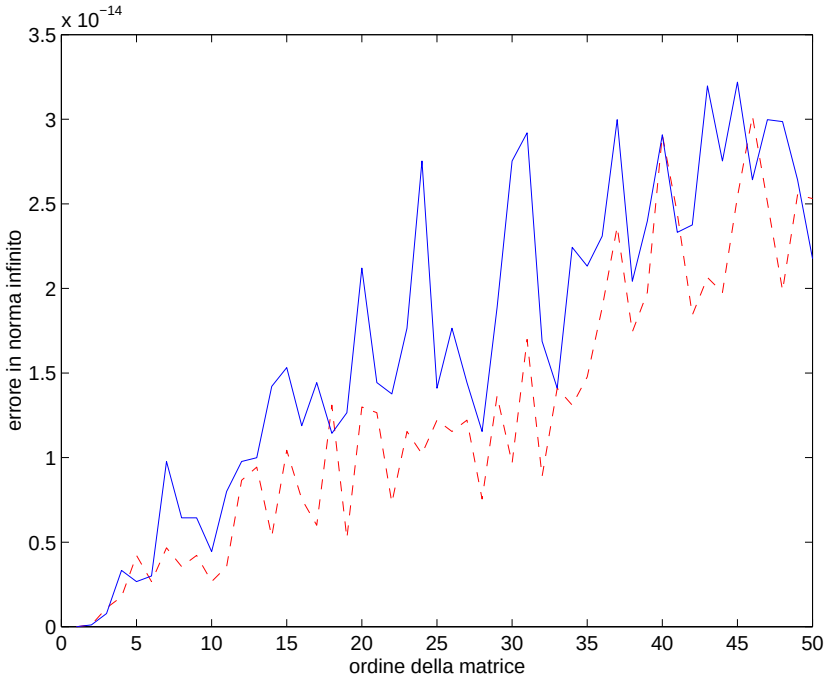

Figura 2.4: errori, *KMS*, $\rho = 9/10$

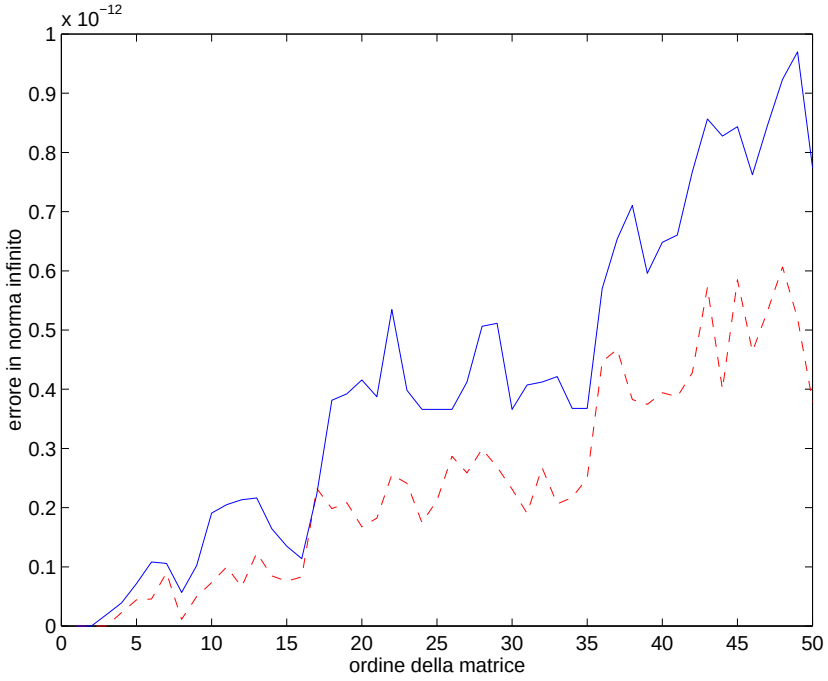

Figura 2.5: errori, *KMS*, $\rho = 99/100$

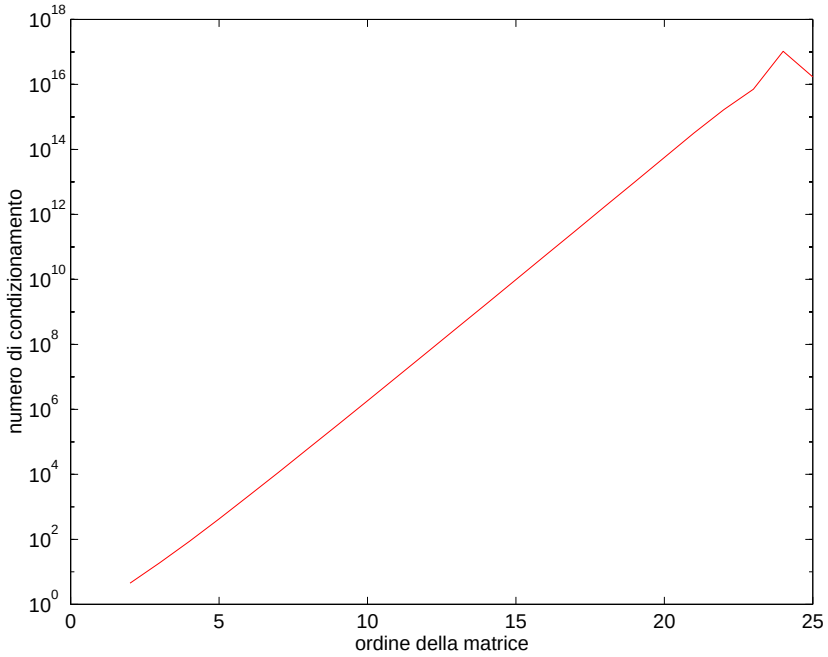

Figura 2.6: condizionamento delle matrici *PROLATE*

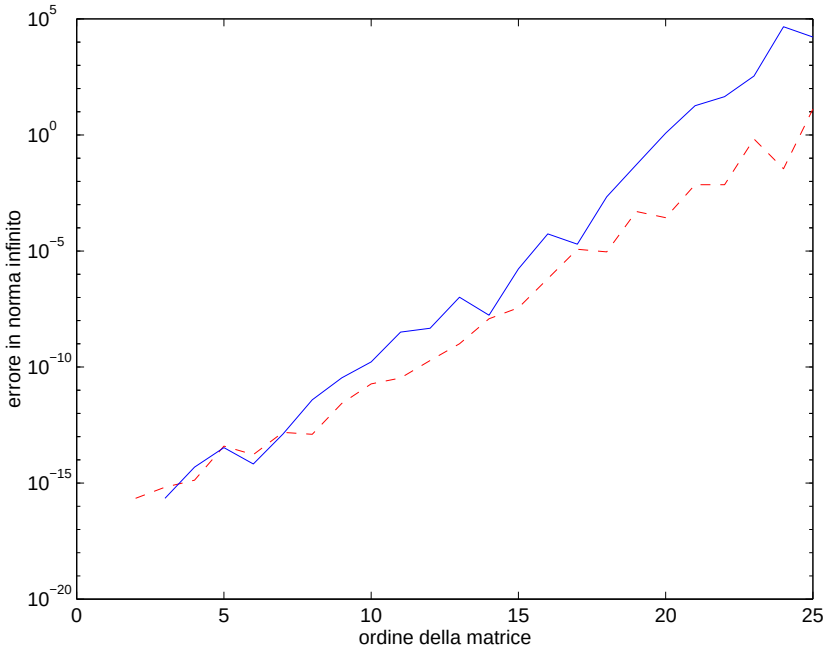

Figura 2.7: errori, *PROLATE*, $w = 0.25$

osservazioni fatte sul primo esempio riguardo alla superiorità dell'algoritmo di Gauss sull'algoritmo di Levinson in presenza di matrici molto mal condizionate.

2.2 Matrici di Vandermonde

Definizione 2.3 *Una matrice $V \in \mathbb{R}^{(n+1)\times(n+1)}$ si dice di Vandermonde se esiste un vettore $\mathbf{v} \in \mathbb{R}^n$, tale che $V = V(v_0, \ldots, v_n) = (v_{ij})_{i,j=0,\ldots,n}$ con $v_{ij} = v_i^j$ e quindi*

$$V = \begin{bmatrix} 1 & v_0 & \ldots & v_0^n \\ 1 & v_1 & \ldots & v_1^n \\ 1 & v_2 & \ldots & v_2^n \\ \vdots & \vdots & & \vdots \\ 1 & v_n & \ldots & v_n^n \end{bmatrix}.$$

Come già visto per le Toeplitz, anche per queste matrici valgono le considerazioni sui vantaggi di lavorare su uno spazio di memoria molto minore di quello richiesto per matrici generiche.

I sistemi e quindi le matrici di Vandermonde sorgono spesso in problemi di interpolazione e approssimazione. Infatti risolvere il sistema $V\mathbf{a} = \mathbf{f}$ è equivalente a calcolare un polinomio interpolante.

Fissati i valori (x_i, f_i) per $i = 0, \ldots, n$, il problema dell'interpolazione polinomiale consiste nel trovare il polinomio $p \in \Pi_n$ tale che $p(x_i) = f_i$ per $i = 0, \ldots, n$. Tale polinomio esiste ed è unico se le ascisse di interpolazione sono distinte. Esprimendo p nella base canonica

$$p(x) = \sum_{j=0}^{n} a_j x^j,$$

le condizioni di interpolazione

$$\sum_{j=0}^{n} a_j x_i^j = f_i \quad i = 0, \ldots, n,$$

possono essere riscritte in forma matriciale

$$\begin{bmatrix} 1 & x_0 & \ldots & x_0^n \\ 1 & x_1 & \ldots & x_1^n \\ 1 & x_2 & \ldots & x_2^n \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \ldots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ \vdots \\ f_n \end{bmatrix}.$$

La matrice $V = (x_i^j)_{i,j=0,\ldots,n}$ è di Vandermonde ed è non singolare se e solo se $x_i \neq x_j$ per $i \neq j$. Un algoritmo veloce per la risoluzione di questo sistema può essere ottenuto esprimendo il polinomio interpolante nella forma di Newton

$$\begin{aligned} p(x) \;=\; & c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots \\ & \cdots + c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \end{aligned}$$

e cioè

$$p(x) = c_0 + \sum_{k=1}^{n} c_k \Big(\prod_{i=0}^{k-1} (x - x_i) \Big) \tag{2.6}$$

dove i coefficienti c_k del polinomio coincidono con le differenze divise

$$c_k = f[x_0, x_1, \ldots, x_k].$$

I coefficienti c_k possono quindi essere calcolati applicando la definizione ricorsiva delle differenze divise

$$
\begin{aligned}
&f[x_i] = f_i, \quad i = 0, 1, \ldots, n \\
&f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}.
\end{aligned}
$$

Definiamo ora i polinomi

$$
\begin{aligned}
&p_n(x) = c_n \\
&p_k(x) = c_k + (x - x_k) p_{k+1}(x), \quad k = n-1, n-2, \ldots, 0
\end{aligned}
\tag{2.7}
$$

e osserviamo che

$$p_0(x) = p(x).$$

L'algoritmo

$$
\begin{aligned}
&(c_0, \ldots, c_n) = (f_0, \ldots, f_n) \\
&\text{for } k = 0, 1, \ldots, n-1 \\
&\qquad \text{for } i = n, n-1, \ldots, k+1 \\
&\qquad\qquad c_i = \frac{c_i - c_{i-1}}{x_i - x_{i-k-1}} \\
&\qquad \text{end} \\
&\text{end}
\end{aligned}
\tag{2.8}
$$

calcola appunto i coefficienti c_k del polinomio utilizzando la definizione 2.7. Se esprimiamo il polinomio $p_k(x)$ in forma canonica

$$p_k(x) = \sum_{j=0}^{n-k} a_{k+j}^{(k)} x^j \tag{2.9}$$

possiamo ricavare i coefficienti $a_i^{(k)}$ uguagliando i termini di ugual grado nelle due espressioni di $p_k(x)$. Questo porta alla ricorsione

$$
\begin{aligned}
&a_n^{(n)} = c_n \\
&\text{for } k = n-1, n-2, \ldots, 0 \\
&\qquad a_k^{(k)} = c_k - x_k a_{k+1}^{(k+1)} \\
&\qquad \text{for } i = k+1, k+2, \ldots, n-1 \\
&\qquad\qquad a_i^{(k)} = a_i^{(k+1)} - x_k a_{i+1}^{(k+1)} \\
&\qquad \text{end} \\
&\qquad a_n^{(k)} = a_n^{(k+1)} \\
&\text{end}
\end{aligned}
\tag{2.10}
$$

che fornisce le componenti del vettore incognito $\mathbf{a}$ in quanto

$$a_i = a_i^{(0)}, \quad i = 0, \ldots, n.$$

Fissati il vettore $\mathbf{x} = (x_0, \ldots, x_n)^T$, avente elementi distinti, ed il vettore $\mathbf{f} = (f_0, \ldots, f_n)^T$, la seguente implementazione Matlab dell'algoritmo sovrascrive $\mathbf{f}$ con la soluzione $\mathbf{a} = (a_0, \ldots, a_n)^T$ del sistema di Vandermonde $V(x_0, \ldots, x_n)\mathbf{a} = \mathbf{f}$

```
function f=vsolve(x,f)
n=size(x,1);
for k=1:n
   for i=n:-1:k+1
      f(i)=(f(i)-f(i-1))/(x(i)-x(i-k));
   end
end
for k=n-1:-1:1
   for i=k:n-1
      f(i)=f(i)-f(i+1)*x(k);
   end
end
```

L'algoritmo richiede $5n^2/2$ *flops.*

È interessante dare una formulazione matriciale dell'algoritmo appena esposto. Definiamo la matrice bidiagonale inferiore $L_k(\alpha) \in \mathbb{R}^{(n+1)\times(n+1)}$ con

$$L_k(\alpha) = \left[\begin{array}{c|cccc} I_k & & \mathbf{0}^T & & \\ \hline & 1 & \cdots & & 0 \\ & -\alpha & 1 & & \\ \mathbf{0} & \vdots & \ddots & \ddots & \vdots \\ & & & \ddots & \ddots & \\ & 0 & \cdots & -\alpha & 1 \end{array}\right]$$

e la matrice diagonale

$$D_k = \mathrm{diag}(\underbrace{1, \ldots, 1}_{k+1}, x_{k+1} - x_0, \ldots, x_n - x_{n-k-1}).$$

Definiamo inoltre le matrici

$$L = D_{n-1}^{-1} L_{n-1}(1) \cdots D_0^{-1} L_0(1)$$
$$U = L_0(x_0)^T \cdots L_{n-1}(x_{n-1})^T$$

rispettivamente triangolare inferiore e superiore. È immediato verificare che il vettore $\mathbf{c} = (c_0, \ldots, c_n)^T$ delle differenze divise può essere ottenuto dal vettore $\mathbf{f} = (f_0, \ldots, f_n)^T$ mediante la relazione

$$\mathbf{c} = L\mathbf{f}.$$

Inoltre l'algoritmo 2.10 è equivalente al prodotto matriciale

$$\mathbf{a} = U\mathbf{c}.$$

Questo ci permette di concludere che l'algoritmo di risoluzione del sistema $V\mathbf{a} = \mathbf{f}$ è equivalente alla fattorizzazione UL della matrice V^{-1}, e quindi alla fattorizzazione LU di V, in quanto

$$\mathbf{a} = V^{-1}\mathbf{f} = UL\mathbf{f}$$

implica le fattorizzazioni

$$V^{-1} = UL \quad \text{e} \quad V = L^{-1}U^{-1}.$$

Queste considerazioni consentono di risolvere con un algoritmo veloce anche il sistema *trasposto*

$$V^T\mathbf{z} = \mathbf{b},$$

utilizzando la fattorizzazione appena ottenuta:

$$\begin{aligned} z &= (V^{-1})^T\mathbf{b} = L^TU^T\mathbf{b} \\ &= \big(L_0(1)^TD_0^{-1}\cdots L_{n-1}(1)^TD_{n-1}^{-1}\big)\big(L_{n-1}(x_{n-1})\cdots L_0(x_0)\big)\mathbf{b} \end{aligned}$$

Dati i vettori $\mathbf{x} = (x_0, \ldots, x_n)^T$ con elementi distinti e $\mathbf{b} = (b_0, \ldots, b_n)^T$ il seguente algoritmo sovrascrive $\mathbf{b}$ con la soluzione $\mathbf{z} = (z_0, \ldots, z_n)^T$ nel sistema di Vandermonde

$$V(x_0, \ldots, x_n)^T\mathbf{z} = \mathbf{b}.$$

Una sua implementazione in ambiente Matlab è

```
function b=vtsolve(x,b)
n=size(x,1);
for k=1:n
   for i=n:-1:k+1
      b(i)=b(i)-x(k)*b(i-1);
   end
end
for k=n:-1:1
   for i=k+1:n
```

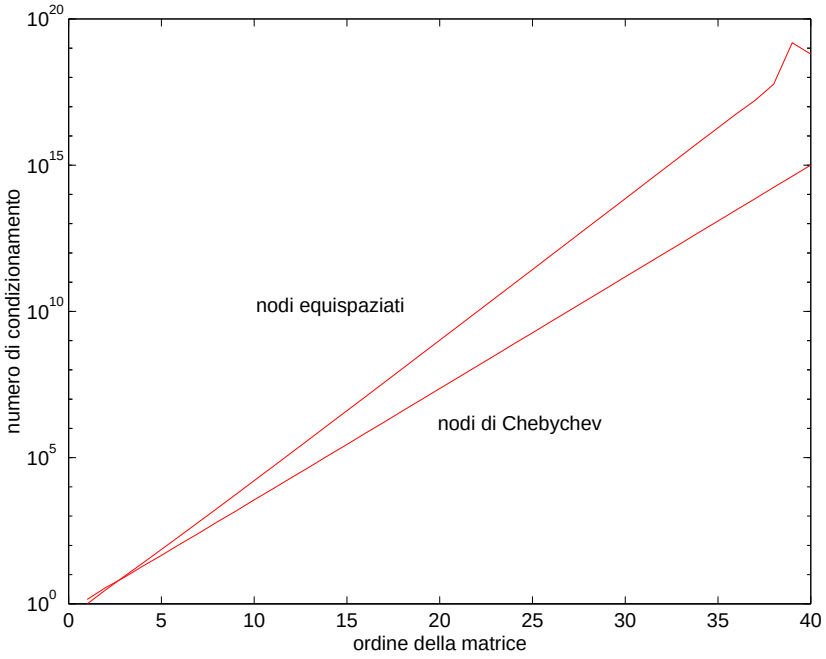

Figura 2.8: condizionamento delle matrici di Vandermonde

```
        b(i)=b(i)/(x(i)-x(i-k));
    end
    for i=k+1:n
        b(i-1)=b(i-1)-b(i);
    end
end
```

che richiede $5n^2$ operazioni. La prima parte del programma calcola i coefficienti del polinomio ed esegue il calcolo di $U^T\mathbf{b}$ e la seconda parte calcola la soluzione eseguendo il prodotto $L^T(U^T\mathbf{b})$.

2.2.1 Risultati numerici

In modo analogo a quello fatto per le matrici di Toeplitz, sono stati fatti due test per verificare l'efficienza degli algoritmi `vsolve` e `vtsolve` comparati con quello di Gauss. Sono state costruite due matrici di Vandermonde, partendo da un campionamento uniforme dell'intervallo $(-1, 1)$ e dai nodi di Chebychev [15]. L'elevato numero di condizionamento delle matrici ottenute è illustrato in Figura 2.8. I sistemi lineari caratterizzati da tali matrici e dalle loro trasposte sono stati poi risolti con gli algoritmi `vsolve`, `vtsolve` e con l'algoritmo di Gauss.

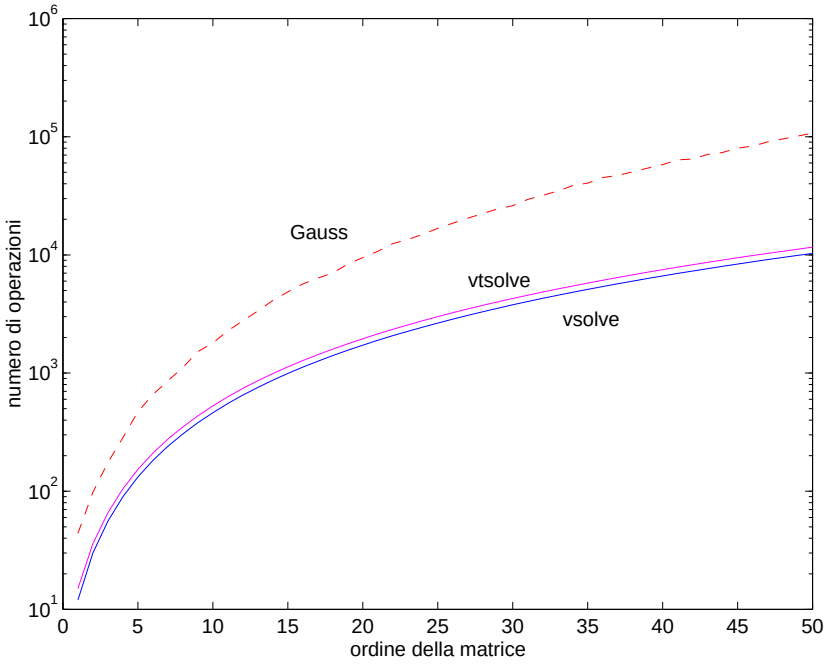

Figura 2.9: complessità computazionale

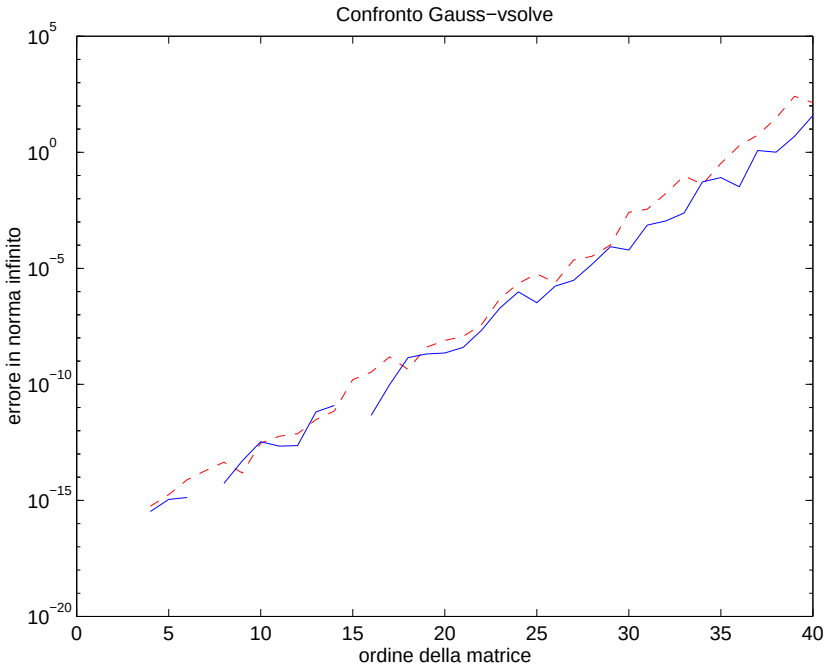

Figura 2.10: errori, `vsolve` vs. Gauss, I caso

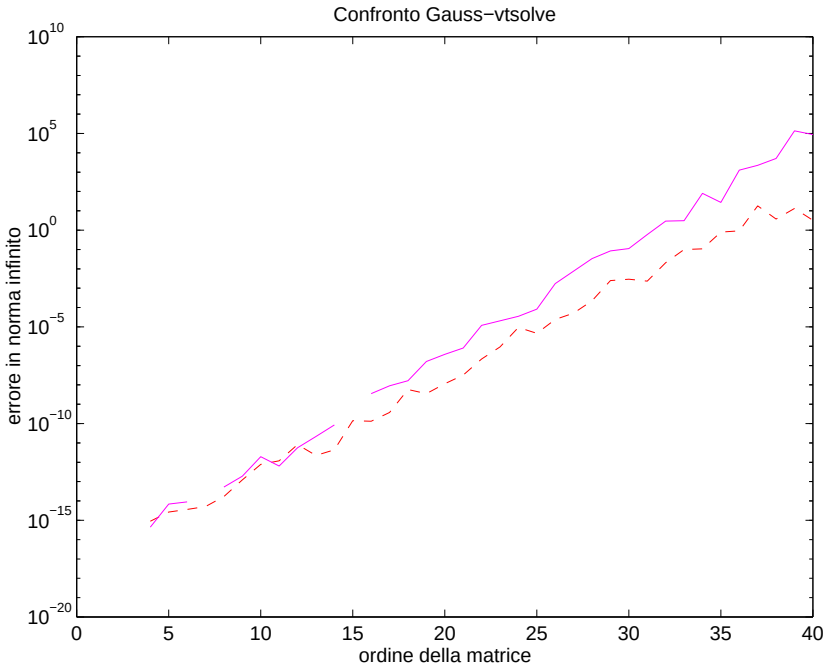

Figura 2.11: errori, `vtsolve` vs. Gauss, I caso

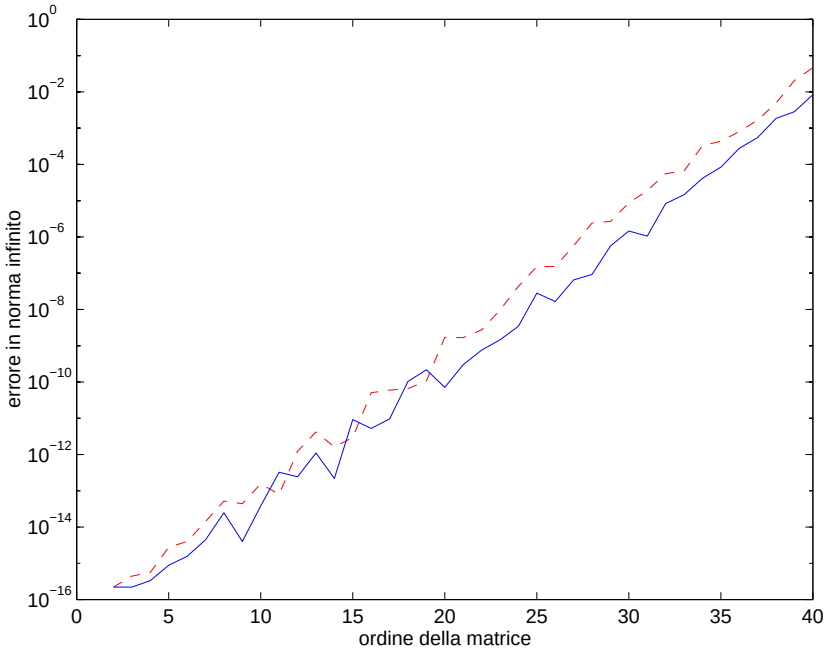

Figura 2.12: errori, `vsolve` vs. Gauss, II caso

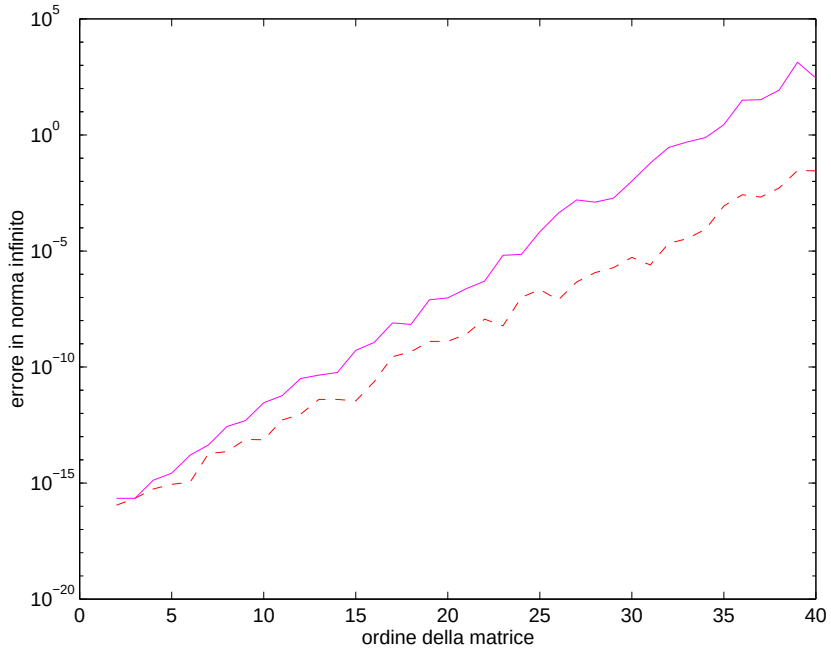

Figura 2.13: errori, `vtsolve` vs. Gauss, II caso

Il vantaggio computazionale degli algoritmi veloci è confermato dai grafici riportati in Figura 2.9. Le Figure 2.10 e 2.11 riportano invece gli errori misurati in presenza della prima matrice di Vandermonde, mentre le Figure 2.12 e 2.13 mostrano gli errori riferiti alla matrice costruita a partire da nodi di Chebychev. Questi grafici mostrano una sostanziale equivalenza dei vari metodi, con un leggero svantaggio in termini di accuratezza per gli algoritmi veloci nei sistemi trasposti.

2.3 Altre classi di matrici strutturate

2.3.1 Matrici di Cauchy

Definizione 2.4 *Una matrice $C \in \mathbb{C}^{n\times n}$ si dice di Cauchy se esistono dei vettori $\mathbf{t}, \mathbf{s} \in \mathbb{C}^n$ tali che $C = (c_{ij})_{i,j=1,\dots,n}$ con $c_{ij} = \frac{1}{t_i - s_j}$ e quindi*

$$C = \begin{bmatrix} \frac{1}{t_1-s_1} & \frac{1}{t_1-s_2} & \cdots & \frac{1}{t_1-s_n} \\ \frac{1}{t_2-s_1} & \frac{1}{t_2-s_2} & \cdots & \frac{1}{t_2-s_n} \\ \vdots & \vdots & & \vdots \\ \frac{1}{t_n-s_1} & \frac{1}{t_n-s_2} & \cdots & \frac{1}{t_n-s_n} \end{bmatrix}.$$

Questa definizione classica è stata estesa per comprendere una classe più ampia di matrici.

Definizione 2.5 *Una matrice $C \in \mathbb{C}^{n\times n}$ si dice Cauchy-like se esistono dei vettori $\mathbf{t}, \mathbf{s} \in \mathbb{C}^n$ e dei vettori $\boldsymbol{\phi}_i, \boldsymbol{\psi}_i \in \mathbb{C}^\alpha$, $i = 1, \ldots, n$, dove α è un numero generalmente piccolo rispetto ad n, tali che $C = (c_{ij})_{i,j=1,\ldots,n}$ con $c_{ij} = \frac{\boldsymbol{\phi}_i^* \boldsymbol{\psi}_j}{t_i - s_j}$ e quindi*

$$C = \begin{bmatrix} \frac{\boldsymbol{\phi}_1^* \boldsymbol{\psi}_1}{t_1 - s_1} & \frac{\boldsymbol{\phi}_1^* \boldsymbol{\psi}_2}{t_1 - s_2} & \cdots & \frac{\boldsymbol{\phi}_1^* \boldsymbol{\psi}_n}{t_1 - s_n} \\ \frac{\boldsymbol{\phi}_2^* \boldsymbol{\psi}_1}{t_2 - s_1} & \frac{\boldsymbol{\phi}_2^* \boldsymbol{\psi}_2}{t_2 - s_2} & \cdots & \frac{\boldsymbol{\phi}_2^* \boldsymbol{\psi}_n}{t_2 - s_n} \\ \vdots & \vdots & & \vdots \\ \frac{\boldsymbol{\phi}_n^* \boldsymbol{\psi}_1}{t_n - s_1} & \frac{\boldsymbol{\phi}_n^* \boldsymbol{\psi}_2}{t_n - s_2} & \cdots & \frac{\boldsymbol{\phi}_n^* \boldsymbol{\psi}_n}{t_n - s_n} \end{bmatrix}$$

È immediato osservare che la prima definizione è ricavabile dalla seconda ponendo $\alpha = 1$ e $\boldsymbol{\phi}_i = \boldsymbol{\psi}_i = 1$, $i = 1, \ldots, n$.

Le matrici *Cauchy-like* non hanno una grande rilevanza applicativa, ma hanno una notevole importanza per il nostro studio in quando, come verrà illustrato nel Capitolo 4, per questa classe di matrici è possibile utilizzare un algoritmo di fattorizzazione veloce e numericamente stabile, per la possibilità di implementare il pivoting di colonna senza distruggere la struttura della matrice.

2.3.2 Matrici di Hankel

Definizione 2.6 *Una matrice $H \in \mathbb{R}^{n\times n}$ si dice di Hankel se esistono $2n-1$ scalari $h_0, \ldots, h_{2n-2}$ tali che $H = (h_{ij})_{i,j=1,\ldots,n}$ con $h_{ij} = h_{i+j-2}$ e quindi*

$$H = \begin{bmatrix} h_0 & h_1 & \cdots & h_{n-2} & h_{n-1} \\ h_1 & h_2 & \cdots & h_{n-1} & h_n \\ \vdots & \vdots & & \vdots & \vdots \\ h_{n-2} & h_{n-1} & \cdots & h_{2n-4} & h_{2n-3} \\ h_{n-1} & h_n & \cdots & h_{2n-3} & h_{2n-2} \end{bmatrix}.$$

Spesso le matrici di Hankel si trovano inquadrate nella classe più generale delle matrici *Toeplitz-plus-Hankel.*

Definizione 2.7 *Una matrice A si dice Toplitz-plus-Hankel, o più brevemente matrice T+H, se è esprimibile nella forma $A = T + H$, dove T è una matrice di Toeplitz e H è una matrice di Hankel.*

Le *matrici T+H* sono estremamente diffuse nelle applicazioni, come ad esempio nei processi di ortogonalizzazione [7], nel campo delle equazioni differenziali a derivate parziali, nel *signal processing*, etc.

Per questo motivo c'è attualmente un grosso interesse per gli algoritmi destinati al trattamento di queste matrici, si veda ad es. [9].

Capitolo 3

Problemi che conducono a sistemi lineari strutturati

3.1 Risoluzione di equazioni differenziali

Consideriamo l'equazione differenziale con condizioni al contorno (problema di Dirichlet) del second'ordine

$$\begin{cases} -y'' + q(x)y = f(x) \\ y(a) = \alpha, \quad y(b) = \beta \end{cases} \tag{3.1}$$

dove assumiamo che $q, f \in C[a,b]$, cioè sono funzioni continue in $[a,b]$, e $q(x) \geq 0$ per $x \in [a,b]$. Si tratta della celebre equazione di Sturm-Liouville, per la quale è noto che esiste una soluzione unica $y : [a,b] \to \mathbb{R}$.

Per discretizzare l'equazione (3.1) dividiamo l'intervallo $[a,b]$ in $n+1$ sottointervalli uguali

$$a = x_0 < x_1 < \cdots < x_n < x_{n+1} = b$$

con

$$x_j = a + jh, \quad h = \frac{b-a}{n+1}$$

e denotiamo con u_i l'approssimazione della soluzione sull'i-esimo punto della discretizzazione

$$u_i \simeq y(x_i), \quad i = 0, \ldots, n.$$

Sostituiamo ora l'operatore differenziale $D^2y(x) = y''(x)$ con l'operatore alle differenze centrali

$$\Delta^2 y(x) = \frac{y(x+h) - 2y(x) + y(x-h)}{h^2}, \tag{3.2}$$

che approssima la derivata seconda in quanto si può dimostrare [16] che

$$\Delta^2 y(x) = y''(x) + O(h^2)$$

se $y \in C^4[a, b]$.

Operando questa sostituzione nell'equazione (3.1) valutata sul generico punto x_i della discretizzazione, si ottiene

$$\frac{2u_i - u_{i-1} - u_{i+1}}{h^2} + q(x_i)u_i = f(x_i, u_i) = f_i.$$

Ora, imponendo $y(a) = \alpha$, $y(b) = \beta$ e introducendo il simbolo $q_i = q(x_i)$ si arriva al sistema di n equazioni lineari

$$\begin{cases} (2 + h^2 q_1)u_1 - u_2 = h^2 f_1 + \alpha \\ -u_{i-1} + (2 + h^2 q_i)u_i - u_{i+1} = h^2 f_i, \quad i = 2, \ldots, n-1 \\ -u_{n-1} + (2 + h^2 q_n)u_n = h^2 f_n + \beta \end{cases} .$$

Questo sistema può essere scritto nella forma matriciale

$$A\mathbf{u} = \mathbf{g}$$

con

$$\mathbf{u} = \begin{pmatrix} u_1 \\ \vdots \\ \vdots \\ \vdots \\ u_n \end{pmatrix}, \quad \mathbf{g} = \begin{pmatrix} f_1 h^2 + \alpha \\ f_2 h^2 \\ \vdots \\ f_{n-1} h^2 \\ f_n h^2 + \beta \end{pmatrix}$$

e dove

$$A = \begin{bmatrix} 2 + q_1 h^2 & -1 & & & 0 \\ -1 & 2 + q_2 h^2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ 0 & & & -1 & 2 + q_n h^2 \end{bmatrix}$$

è una matrice tridiagonale di Toeplitz.

Schemi di discretizzazione differenti da (3.2) producono differenti matrici di Toeplitz, mentre la versione bi-dimensionale del problema ai limiti (3.1) conduce ad un sistema lineare caratterizzato da una matrice con struttura di Toeplitz a blocchi.

3.2 Approssimazione polinomiale ai minimi quadrati in $L^2[a,b]$

Consideriamo lo spazio di Hilbert $L^2[a,b]$ con prodotto interno

$$\langle f, g\rangle = \int_a^b f(x)g(x)dx$$

e con la norma

$$\|f\| = \langle f, f\rangle^{1/2}$$

indotta dal prodotto scalare. Con Π_n denotiamo lo spazio dei polinomi di grado al più n. Vogliamo trovare il polinomio $p_n \in \Pi_n$ che approssima meglio la f nel senso dei minimi quadrati, cioè

$$\min_{p\in\Pi_n} \|f-p\|_2^2 = \min_{p\in\Pi_n} \int_a^b [f(x)-p(x)]^2\, dx. \tag{3.3}$$

È possibile caratterizzare la migliore approssimazione nel senso dei minimi quadrati mediante il seguente teorema [15].

Teorema 3.1 *Sia $f(x) \in L^2[a,b]$, allora p_n è l'approssimazione di f in Π_n nel senso dei minimi quadrati se e solo se*

$$\langle f - p_n, p\rangle = 0$$

per ogni $p \in \Pi_n$.

Se dotiamo Π_n della base canonica $\{1, x, x^2, \ldots, x^n\}$, il precedente teorema conduce al sistema lineare

$$\langle f - p_n, x^i\rangle = 0 \quad i = 0, 1, \ldots, n$$

che è detto sistema delle *equazioni normali.* Se esprimiamo la migliore approssimazione nella base canonica

$$p_n(x) = \sum_{j=0}^{n} \alpha_j x^j$$

otteniamo

$$\langle p_n, x^i\rangle = \langle f, x^i\rangle, \quad i = 0, \ldots, n$$

$$\Big\langle \sum_{j=0}^{n} \alpha_j x^j, x^i \Big\rangle = \langle f, x^i \rangle, \quad i = 0, \dots, n$$

$$\sum_{j=0}^{n} \langle x^i, x^j \rangle \alpha_j = \langle f, x^i \rangle, \quad i = 0, \dots, n.$$

Il sistema delle equazioni normali può essere scritto in forma matriciale

$$H\boldsymbol{\alpha} = \mathbf{f}$$

dove si ha

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} \quad \mathbf{f} = \begin{pmatrix} \langle f, 1 \rangle \\ \langle f, x \rangle \\ \vdots \\ \langle f, x^n \rangle \end{pmatrix}$$

e dove

$$\langle f, x^i \rangle = \int_a^b x^i f(x) dx$$

sono detti i momenti della funzione f. La matrice H è data da

$$H_{ij} = \langle x^i, x^j \rangle = \int_a^b x^{i+j} dx = \left. \frac{x^{i+j+1}}{i+j+1} \right|_a^b$$

ed è detta matrice di Gram. In particolare se $[a, b] = [0, 1]$

$$H_{ij} = \int_0^1 x^{i+j} dx = \frac{1}{i+j+1} \cdot \tag{3.4}$$

La matrice così ottenuta

$$H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \vdots & & \frac{1}{n+1} \\ \frac{1}{3} & \vdots & \vdots & & \frac{1}{n+2} \\ \vdots & \vdots & \vdots & & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{bmatrix}$$

è detta matrice di Hilbert. Questa matrice ha la proprietà di essere simultaneamente di Cauchy e di Hankel. Essa risulta fortemente malcondizionata e si può dimostrare che l'andamento asintotico del suo condizionamento rispetto alla norma 2 è dato da

$$\kappa_2(H) \approx \mathrm{e}^{3.5n}.$$

Capitolo 4

Struttura di spostamento

4.1 Rango di spostamento

Definizione 4.1 *Il rango di spostamento superiore di una matrice R $n \times n$ è il più piccolo intero $\alpha_+(R)$ tale che si può scrivere:*

$$R = \sum_{i=1}^{\alpha_+(R)} L_i U_i$$

dove $\{L_i\}_{i=1}^{\alpha_+(R)}$ sono matrici di Toeplitz triangolari inferiori e $\{U_i\}_{i=1}^{\alpha_+(R)}$ sono matrici di Toeplitz triangolari superiori.

Definizione 4.2 *Il rango di spostamento inferiore di una matrice R $n \times n$ è il più piccolo intero $\alpha_-(R)$ tale che si può scrivere:*

$$R = \sum_{i=1}^{\alpha_-(R)} \mathcal{U}_i \mathcal{L}_i$$

dove $\{\mathcal{L}_i\}_{i=1}^{\alpha_+(R)}$ sono matrici di Toeplitz triangolari inferiori e $\{\mathcal{U}_i\}_{i=1}^{\alpha_+(R)}$ sono matrici di Toeplitz triangolari superiori.

Introduciamo la matrice di spostamento in avanti (*forward-shift*)

$$Z = \begin{bmatrix} 0 & & & 0 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & 1 & 0 \end{bmatrix}.$$

L'*operatore di spostamento* per una matrice hermitiana di dimensione n è stato definito per la prima volta in [12] come

$$\Delta(R) = R - ZRZ^* \tag{4.1}$$

dove Z^* rappresenta l'aggiunta della matrice Z, ossia la matrice trasposta coniugata. L'espressione ZRZ^* corrisponde allo spostamento (*shift* o *displacement* in inglese) della matrice R verso il basso lungo la diagonale principale di una posizione, il che giustifica il nome attribuito all'operatore Δ. Se la matrice $\Delta(R)$ ha rango r, indipendente da n, allora R è detta *strutturata* rispetto allo spostamento definito in (4.1) e il numero r coincide con il rango di spostamento superiore $\alpha_+(R)$, come verrà mostrato nel Teorema 4.1. La definizione di operatore di spostamento si può estendere ad una matrice non hermitiana e anche a matrici non quadrate.

Lemma 4.1 *Dati i vettori colonna* $\mathbf{x} = \{x_i\}_{i=1}^{\alpha}, \mathbf{y} = \{y_i\}_{i=1}^{\alpha}$*, l'equazione matriciale*

$$R - ZRZ^* = \sum_{i=1}^{\alpha} \mathbf{x}_i \mathbf{y}_i^* \tag{4.2}$$

ha l'unica soluzione

$$R = \sum_{i=1}^{\alpha} L(\mathbf{x}_i) U(\mathbf{y}_i^*), \tag{4.3}$$

dove $L(\mathbf{x})$ *denota una matrice triangolare inferiore di Toeplitz la cui prima colonna è* $\mathbf{x}$*, e* $U(\mathbf{y}^*)$ *denota una matrice triangolare superiore di Toeplitz la cui prima riga è* $\mathbf{y}^*$*. Allo stesso modo l'equazione matriciale*

$$R - Z^*RZ = \sum_{i=1}^{\alpha} \mathbf{x}_i \mathbf{y}_i^* \tag{4.4}$$

ha l'unica soluzione

$$R = \sum_{i=1}^{\alpha} U(\tilde{\mathbf{x}}_i^*) L(\tilde{\mathbf{y}}_i), \tag{4.5}$$

dove

$$\tilde{\mathbf{x}}^* = [x_n, \ldots, x_1] \quad \textit{quando} \quad \mathbf{x}^* = [x_1, \ldots, x_n]\,.$$

Dimostrazione. Per l'unicità notiamo che

$$R_1 - ZR_1Z^* = R_2 - ZR_2Z^*$$

implica

$$R_1 - R_2 = Z(R_1 - R_2)Z^*,$$

equazione che ammette l'unica soluzione $R_1 - R_2 = 0$.

Per dimostrare l'esistenza, osserviamo innanzitutto che

$$L(\mathbf{x})U(\mathbf{y}^*) - Z\left[L(\mathbf{x})U(\mathbf{y}^*)\right]Z^* = \mathbf{x}\mathbf{y}^*. \tag{4.6}$$

Tale proprietà si può verificare per il caso di matrici 3×3 ed estendere poi facilmente al caso generale. Sostituendo ora la (4.3) nell'equazione (4.2) si ottiene

$$\begin{aligned} \sum_{i=1}^{\alpha} L_iU_i - Z\Big(\sum_{i=1}^{\alpha} L_iU_i\Big)Z^* &= \\ = \sum_{i=1}^{\alpha} (L_iU_i - ZL_iU_iZ^*) &= \sum_{i=1}^{\alpha} \mathbf{x}_i\mathbf{y}_i^* \end{aligned}$$

per la (4.6). Per semplicità abbiamo posto $L(\mathbf{x}_i) = L_i$ e $U(\mathbf{y}_i^*) = U_i$.

La seconda parte del teorema si dimostra in modo analogo. □

Esiste una caratterizzazione del rango di spostamento che fornisce anche un algoritmo per la sua determinazione.

Teorema 4.1 *I ranghi di spostamento α_+ e α_- si possono ottenere mediante le seguenti relazioni*

$$\begin{aligned} \alpha_+(R) &= \operatorname{rank}(R - ZRZ^*), \\ \alpha_-(R) &= \operatorname{rank}(R - Z^*RZ), \end{aligned}$$

dove $\operatorname{rank}(A)$ *indica il rango della matrice* A*.*

Dimostrazione. La dimostrazione segue immediatamente dal Lemma 4.1. Infatti se R ha una rappresentazione minima

$$R = \sum_{i=1}^{\alpha_+} L(\mathbf{x}_i)U(\mathbf{y}_i^*),$$

allora

$$R - ZRZ^* = \sum_{i=1}^{\alpha_+} \mathbf{x}_i\mathbf{y}_i^*,$$

che ha rango α_+ in quanto il *rango* di una matrice A di dimensione n, coincide con il minimo intero t che permette di scrivere

$$A = \sum_{i=1}^{t} \mathbf{x}_i \mathbf{y}_i^*.$$

Viceversa se $R - ZRZ^*$ ha rango α_+, allora devono esistere dei vettori $\mathbf{x}_i = \{x_i\}_{i=1}^{\alpha_+}$, $\mathbf{y}_i = \{y_i\}_{i=1}^{\alpha_+}$. tali che

$$R - ZRZ^* = \sum_{i=1}^{\alpha_+} \mathbf{x}_i \mathbf{y}_i^*.$$

Per il Lemma 4.1 abbiamo allora

$$R = \sum_{i=1}^{\alpha_+} L(\mathbf{x}_i) U(\mathbf{y}_i^*).$$

Si può usare una dimostrazione simile per la rappresentazione di α_-. □

È immediato osservare che

$$R - ZRZ^* = \begin{bmatrix} & & * \\ & \overline{R} & \vdots \\ * & \cdots & * \end{bmatrix} - \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \overline{R} & \\ 0 & & \end{bmatrix}, \tag{4.7}$$

dove $\overline{R}$ è la sottomatrice principale di R di ordine $n-1$. Perciò per una matrice di Toeplitz T si ha

$$T - ZTZ^* = \left[\begin{array}{c|ccc} t_1 & t_2 & \cdots & t_n \\ \hline t_2 & & & \\ \vdots & & 0 & \\ t_n & & & \end{array}\right],$$

che ha rango 2, e questo prova che $\alpha_+(T) = 2$. NelI'altra notazione si ha similarmente

$$R - Z^*RZ = \begin{bmatrix} * & \cdots & * \\ \vdots & \underline{R} & \\ * & & \end{bmatrix} - \begin{bmatrix} & & 0 \\ & \underline{R} & \vdots \\ 0 & \cdots & 0 \end{bmatrix}. \tag{4.8}$$

Le espressioni (4.7) e (4.8) spiegano, anche graficamente, le ragioni che hanno condotto all'introduzione del termine *rango di spostamento*. Infatti, come già accennato precedentemente, ZRZ^* sposta la matrice di partenza di una posizione verso il basso lungo la diagonale principale, inserendo nella prima riga e colonna un vettore di zeri. Se invece consideriamo Z^*RZ, lo spostamento di R avviene allo stesso modo, ma di una posizione verso l'alto e i vettori di zeri si inseriscono all'ultima riga e colonna. Nel caso delle matrici di Toeplitz, la differenza $R - ZRZ^*$ risulta essere una matrice che ha rango $r = 2$ indipendente da n. La proprietà interessante, oltre che nuova, è che questo fatto vale per una classe di matrici più ampia di quella delle matrici di Toeplitz.

Poiché sappiamo che la struttura di Toeplitz non è invariante per svariate operazioni molto frequenti quali la moltiplicazione, l'inversione, la fattorizzazione LU, etc., vorremmo cercare di capire per quali operazioni risulta invariante il rango di spostamento. Un risultato particolarmente significativo è fornito dal seguente teorema.

Teorema 4.2 *Il rango di spostamento superiore ed inferiore di una matrice non singolare è uguale al rango di spostamento inferiore e superiore, rispettivamente, della sua inversa*

$$\begin{aligned} \alpha_+(R) &= \alpha_-(R^{-1}) \\ \alpha_-(R) &= \alpha_+(R^{-1}). \end{aligned}$$

Dimostrazione. Notiamo che

$$\begin{aligned} \alpha_-(R^{-1}) &= \operatorname{rank}(R^{-1} - Z^*R^{-1}Z) \\ &= \operatorname{rank}((R^{-1} - Z^*R^{-1}Z)R) \\ &= \operatorname{rank}(I - Z^*R^{-1}ZR) \end{aligned}$$

poiché il rango non è influenzato dalla moltiplicazione per una matrice non singolare. Applicando la proprietà

$$\operatorname{rank}(I - AB) = \operatorname{rank}(I - BA)$$

possiamo continuare la catena di uguaglianze

$$\begin{aligned} \alpha_-(R^{-1}) &= \operatorname{rank}(I - ZRZ^*R^{-1}) \\ &= \operatorname{rank}((I - ZRZ^*R^{-1})R) \\ &= \operatorname{rank}(R - ZRZ^*) = \alpha_+(R). \end{aligned}$$

Nello stesso modo si può dimostrare l'altra uguaglianza del teorema

$$\alpha_+(R^{-1}) = \alpha_-(R).$$

□

Un'altra proprietà per cui è invariante il rango di spostamento è legata alla cosiddetta *inerzia di spostamento* di una matrice R.

Definizione 4.3 *L'inerzia di una matrice hermitana R è la terna di numeri interi non negativi (p, z, q) che indicano, rispettivamente, il numero di autovalori positivi, nulli e negativi di R.*

È immediato osservare che la somma $p + q$ coincide con il rango di R.

Un risultato classico relativo all'inerzia di una matrice è dato dal seguente teorema [6].

Teorema 4.3 (Legge di inerzia di Sylvester) *Se R è una matrice simmetrica e X è una matrice non singolare, allora R e X^*RX hanno la stessa inerzia.*

Se R è una matrice hermitiana, sappiamo che $\Delta(R)$ è anch'essa una matrice hermitiana.

Definizione 4.4 *Si definisce inerzia di spostamento di una matrice hermitana R la terna di numeri interi non negativi (p, z, q) che indicano, rispettivamente, il numero di autovalori positivi, nulli e negativi di $\Delta(R)$. Inoltre la somma $p + q$ coincide col rango di spostamento (superiore) di R.*

Useremo questa definizione per evidenziare un'altra proprietà della struttura di spostamento.

Lemma 4.2 *L'inerzia di spostamento di una matrice hermitiana non singolare R rispetto a $R - ZRZ^*$ è uguale all'inerzia di spostamento rispetto a $R^{-1} - Z^*R^{-1}Z$, e cioè*

$$\text{Inerzia}(R - ZRZ^*) = \text{Inerzia}(R^{-1} - Z^*R^{-1}Z). \tag{4.9}$$

Dimostrazione. Consideriamo la seguente relazione

$$\begin{aligned}
\begin{bmatrix} R & Z \\ Z^* & R^{-1} \end{bmatrix} &= \\
&= \begin{bmatrix} I & 0 \\ Z^*R^{-1} & I \end{bmatrix} \cdot \begin{bmatrix} R & 0 \\ 0 & R^{-1} - Z^*R^{-1}Z \end{bmatrix} \cdot \begin{bmatrix} I & 0 \\ Z^*R^{-1} & I \end{bmatrix}^* \\
&= \begin{bmatrix} I & ZR \\ 0 & I \end{bmatrix} \cdot \begin{bmatrix} R - ZRZ^* & 0 \\ 0 & R^{-1} \end{bmatrix} \cdot \begin{bmatrix} I & ZR \\ 0 & I \end{bmatrix}^*,
\end{aligned}$$

dove abbiamo utilizzato l'identità $R(R^*)^{-1} = I = R^{-1}R^*$.

Ora, per il Teorema 4.3 le matrici

$$\begin{bmatrix} R & 0 \\ 0 & R^{-1} - Z^*R^{-1}Z \end{bmatrix} \quad \text{e} \quad \begin{bmatrix} R - ZRZ^* & 0 \\ 0 & R^{-1} \end{bmatrix}$$

hanno la stessa inerzia, avendo entrambe la stessa inerzia della matrice

$$\begin{bmatrix} R & Z \\ Z^* & R^{-1} \end{bmatrix}.$$

Ma poiché l'inerzia di una matrice R è uguale all'inerzia della sua inversa, visto che gli autovalori della inversa sono gli inversi degli autovalori di R e quindi hanno lo stesso segno, la tesi che $\text{Inerzia}(R - ZRZ^*) = \text{Inerzia}(R^{-1} - Z^*R^{-1}Z)$ rimane provata. □

4.2 Struttura di spostamento

L'operatore di spostamento (4.1), apparso per la prima volta in connessione con le matrici di Toeplitz in [2] e [12] e successivamente formalizzato in una teoria estesa in [10], è stato generalizzato nella forma

$$\Delta_{\{F.A\}}(R) = R - F \cdot R \cdot A, \tag{4.10}$$

dove F e A sono due matrici di ordine n, allo scopo di estendere le proprietà dimostrate per le matrici di Toeplitz ad altre classi di matrici strutturate. L'operatore $\Delta_{\{F,A\}}$ è detto *operatore di spostamento di Stein* relativo alle matrici $\{F, A\}$.

Nel nostro studio utilizzeremo lo stesso operatore nella cosiddetta *forma di Sylvester*, introdotta per la prima volta in [10] ed utilizzata in numerosi lavori successivi, come ad esempio in [3]. Siano F e A due matrici di ordine n e sia $R \in \mathbb{C}^{n\times n}$ una matrice che soddisfi l'equazione

$$\nabla_{\{F.A\}}(R) = F \cdot R - R \cdot A = G \cdot B \tag{4.11}$$

per determinate matrici rettangolari $G \in \mathbb{C}^{n\times\alpha}$ e $B \in \mathbb{C}^{\alpha\times n}$, dove il numero α è piccolo rispetto a n e comunque indipendente da esso.

Definizione 4.5 *Le matrici G e B sono dette $\{F, A\}$-generatori di R, e il più piccolo intero*

$$\alpha = \alpha_{\{F,A\}}(R) = \text{rank}(FR - RA)$$

tra tutti gli $\{F, A\}$-generatori è chiamato rango di $\{F, A\}$-spostamento della matrice R.

Se F e A sono matrici non singolari, è immediato osservare che questa definizione di rango di spostamento è equivalente a quella basata sull'operatore (4.10). La forma di Sylvester (4.11) dell'operatore di spostamento è stata in seguito ulteriormente generalizzata in [13] nella forma

$$\nabla_{\{\Omega,\Delta,F,A\}} = \Omega \cdot R \cdot \Delta - F \cdot R \cdot A$$

che chiaramente include entrambe le definizioni precedenti.

Il risultato cruciale, che consente di esprimere in maniera molto più generale il concetto di struttura è che *per ognuno dei modelli di struttura introdotti nel Capitolo 2 è possibile scegliere una coppia di matrici* $\{F, A\}$ *per definire un operatore di spostamento* $\nabla_{\{F,A\}}(\cdot) : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ *della forma*

$$\nabla_{\{F,A\}}(R) = FR - RA \tag{4.12}$$

in modo tale che il rango di $\{F, A\}$*-spostamento delle matrici della classe in esame risulti costante.*

Questo fatto consente, ad esempio, di memorizzare una matrice strutturata semplicemente mediante i suoi generatori e soprattutto, come vedremo, di sviluppare algoritmi veloci di fattorizzazione che necessitino dell'aggiornamento dei soli generatori.

Inoltre, si verifica che l'insieme delle matrici che, per una data scelta di una coppia $\{F, A\}$, mantengano un rango di $\{F, A\}$-spostamento costante risulta essere molto più vasto della classe di matrici di strutturate che ha portato a quella scelta, includendo, ad esempio, anche le inverse delle matrici strutturate di partenza. Per questo motivo ciascun insieme di matrici viene denotato aggiungendo il suffisso *-like* al nome della classe strutturata di partenza.

Per chiarire la situazione, analizzeremo ora le generalizzazioni delle classi di matrici strutturate presentate nel Capitolo 2, presentando le scelte usuali per le coppie $\{F, A\}$ ed illustrando la forma che assumono i generatori in corrispondenza delle matrici strutturate classiche. A questo scopo è utile introdurre la matrice

$$Z_\phi = \begin{bmatrix} 0 & 0 & \dots & 0 & \phi \\ 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & 0 \end{bmatrix}.$$

Se $\phi = 1$, la matrice Z_1 è chiamata *matrice ciclica di spostamento in avanti.*

1. **Matrici Toeplitz-like**.

$$F = Z_1,$$
$$A = Z_{-1}.$$

Per una matrice di Toeplitz T della forma (2.1) si ha

$$\nabla_{\{Z_1, Z_{-1}\}}(T) = Z_1 T - T Z_{-1} = G \cdot B$$

dove due possibili generatori assumono la forma

$$G = \begin{bmatrix} t_0 & 1 \\ t_1 + t_{-(n-1)} & 0 \\ t_2 + t_{-(n-2)} & 0 \\ \vdots & \vdots \\ t_{n-1} + t_{-1} & 0 \end{bmatrix}$$

e

$$B = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ t_{n-1} - t_{-1} & t_{n-2} - t_{-2} & \cdots & t_1 - t_{-(n-1)} & t_0 \end{bmatrix}.$$

Questo ci mostra che una matrice di Toeplitz ha rango di $\{Z_1, Z_{-1}\}$-spostamento uguale a due.

2. **Matrici Vandermonde-like**.

$$F = D_{\mathbf{x}} = \mathrm{diag}(x_1, \ldots, x_n),$$
$$A = Z_1.$$

Per una matrice di Vandermonde classica $V(\mathbf{x})$ si ha

$$\nabla_{\{D_{\mathbf{x}}, Z_1\}}(V(\mathbf{x})) = D_{\mathbf{x}} V(\mathbf{x}) - V(\mathbf{x}) Z_1 = G \cdot B$$

dove i generatori assumono la forma

$$G = \begin{bmatrix} x_1^n - 1 \\ x_2^n - 1 \\ \vdots \\ x_n^n - 1 \end{bmatrix}$$

e

$$B = \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix}.$$

Si vede, dunque, che una matrice di Vandermonde ha rango di $\{D_{\mathbf{x}}, Z_1\}$-spostamento uguale ad uno.

3. **Matrici Cauchy-like**.

$$F = D_{\mathbf{t}} = \mathrm{diag}(t_1, \ldots, t_n),$$
$$A = D_{\mathbf{s}} = \mathrm{diag}(s_1, \ldots, s_n).$$

Questo insieme di matrici strutturate coincide esattamente con la classe di matrici Cauchy-like precedentemente definite nel Paragrafo 2.3.1. In questo caso, se C è una matrice Cauchy-like, abbiamo

$$\nabla_{\{D_{\mathbf{t}}, D_{\mathbf{s}}\}}(C) = D_{\mathbf{t}}C - CD_{\mathbf{s}} = G \cdot B$$

dove i generatori sono della forma

$$G^* = \begin{bmatrix} \boldsymbol{\phi}_1 & \boldsymbol{\phi}_2 & \cdots & \boldsymbol{\phi}_n \end{bmatrix}$$
$$B = \begin{bmatrix} \boldsymbol{\psi}_1 & \boldsymbol{\psi}_2 & \cdots & \boldsymbol{\psi}_n \end{bmatrix}$$

con $\boldsymbol{\phi}_i, \boldsymbol{\psi}_i \in \mathbb{C}^{\alpha}$ per $i = 1, \ldots, n$. Si vede, dunque, che una matrice Cauchy-like ha rango di $\{D_{\mathbf{t}}, D_{\mathbf{s}}\}$-spostamento uguale ad α.

4. **Matrici Toeplitz-plus-Hankel-like**.

$$F = Y_{00},$$
$$A = Y_{11},$$

dove $Y_{\gamma,\delta}$ è definita da

$$Y_{\gamma,\delta} = \begin{bmatrix} \gamma & 1 & 0 & \ldots & 0 \\ 1 & 0 & 1 & \ddots & \vdots \\ 0 & 1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 & 1 \\ 0 & \ldots & 0 & 1 & \delta \end{bmatrix}.$$

4.3 L'algoritmo di Schur

L'algoritmo di Schur per la fattorizzazione LU di una matrice A non singolare consiste essenzialmente in una riformulazione del classico algoritmo di triangolarizzazione di Gauss.

Data una matrice $A = A^{(1)}$ per cui esista la fattorizzazione LU, consideriamo il seguente prodotto matriciale, equivalente al primo passo dell'algoritmo di Gauss

$$L_1 A^{(1)} = \begin{bmatrix} 1 & 0 \\ -\mathbf{l}_1 & I_{n-1} \end{bmatrix} \cdot \begin{bmatrix} d_1 & \mathbf{c}_1^* \\ \mathbf{b}_1 & R_1 \end{bmatrix} = \begin{bmatrix} d_1 & \mathbf{c}_1^* \\ \mathbf{b}_1 - d_1\mathbf{l}_1 & R_1 - \mathbf{l}_1\mathbf{c}_1^* \end{bmatrix}. \quad (4.13)$$

Per annullare gli elementi della prima colonna situati al di sotto della diagonale principale è necessario assegnare

$$\mathbf{l}_1 = d_1^{-1}\mathbf{b}_1.$$

Si definisce allora *complemento di Schur* dell'elemento d_1 di posto $(1,1)$ nella matrice $A^{(1)}$ la matrice

$$S_1 = R_1 - d_1^{-1}\mathbf{b}_1\mathbf{c}_1^*.$$

Questa matrice rappresenta la sottomatrice principale formata dalle ultime $n-1$ righe e colonne della matrice $A^{(2)}$ risultante dalla applicazione del primo passo dell'algoritmo di Gauss alla matrice A.

Il passo k dell'algoritmo di fattorizzazione di Gauss consiste nel prodotto matriciale

$$A^{(k+1)} = L_k A^{(k)} = \begin{bmatrix} I_{k-1} & 0 \\ 0 & \tilde{L}_k \end{bmatrix} \cdot \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ 0 & A_{22}^{(k)} \end{bmatrix} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ 0 & \tilde{L}_k A_{22}^{(k)} \end{bmatrix}$$

il cui blocco di posto $(2,2)$ si può rappresentare nella forma

$$\tilde{L}_k A_{22}^{(k)} = \begin{bmatrix} 1 & 0 \\ -\mathbf{l}_k & I_{n-k} \end{bmatrix} \cdot \begin{bmatrix} d_k & \mathbf{c}_k^* \\ \mathbf{b}_k & R_k \end{bmatrix} = \begin{bmatrix} d_k & \mathbf{c}_k^* \\ 0 & S_k \end{bmatrix}$$

dove

$$\mathbf{l}_k = d_k^{-1}\mathbf{b}_k$$

e la matrice

$$S_k = R_k - d_k^{-1}\mathbf{b}_k\mathbf{c}_k^*$$

è il complemento di Schur dell'elemento d_k nella matrice $A_{2,2}^{(k)}$. Si vede dunque che l'algoritmo di Gauss corrisponde alla complementazione di Schur ricorsiva di una sequenza di sottomatrici principali di A.

I fattori L e U della matrice A si possono ottenere memorizzando i vettori $\mathbf{l}_k$ e $\mathbf{c}_k$ ottenuti nel corso dell'algoritmo nelle matrici

$$L = \begin{bmatrix} 1 & & & & & & \\ \vdots & 1 & & & & & \\ \vdots & \vdots & \ddots & & & & \\ \mathbf{l}_1 & \vdots & & 1 & & & \\ \vdots & \mathbf{l}_2 & & \vdots & \ddots & & \\ \vdots & \vdots & & \mathbf{l}_k & & \ddots & \\ \vdots & \vdots & & \vdots & & & 1 \end{bmatrix}$$

e

$$U = \begin{bmatrix} d_1 & \cdots & \cdots & \mathbf{c}_1^* & \cdots & \cdots & \cdots \\ & d_2 & \cdots & \cdots & \mathbf{c}_2^* & \cdots & \cdots \\ & & \ddots & & & & \\ & & & d_k & \cdots & \mathbf{c}_k^* & \cdots \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & & d_n \end{bmatrix}.$$

L'applicazione dell'algoritmo di Schur a una generica matrice A non comporta nessun vantaggio in termini di complessità rispetto all'algoritmo di Gauss. La situazione è differente per matrici strutturate che soddisfino particolari tipi di equazioni di spostamento, come evidenziato dal seguente teorema.

Teorema 4.4 *Sia data la matrice*

$$R_1 = \begin{bmatrix} d_1 & \mathbf{u}_1^* \\ \mathbf{l}_1 & \tilde{R}_1 \end{bmatrix}$$

che soddisfa l'equazione di spostamento di Sylvester

$$\nabla_{\{F_1,A_1\}}(R_1) = \begin{bmatrix} f_1 & 0 \\ * & F_2 \end{bmatrix} \cdot R_1 - R_1 \cdot \begin{bmatrix} a_1 & * \\ 0 & A_2 \end{bmatrix} = G_1 \cdot B_1, \qquad (4.14)$$

per determinate matrici triangolari F_1 e A_1, con $G_1 \in \mathbb{C}^{n\times\alpha}$ e $B_1 \in \mathbb{C}^{\alpha\times n}$. Il simbolo $*$ indica una sottomatrice il cui contenuto è ininfluente ai fini del ragionamento.

Se l'elemento d_1 di posto $(1,1)$ è diverso da zero, allora il complemento di Schur $R_2 = \tilde{R}_1 - d_1^{-1}\mathbf{l}_1\mathbf{u}_1^*$ soddisfa l'equazione di spostamento di Sylvester

$$F_2 \cdot R_2 - R_2 \cdot A_2 = G_2 \cdot B_2,$$

con

$$\begin{aligned} \begin{bmatrix} 0 \\ G_2 \end{bmatrix} &= G_1 - \begin{bmatrix} 1 \\ d_1^{-1}\mathbf{l}_1 \end{bmatrix} \cdot \mathbf{g}_1^*, \\ \begin{bmatrix} 0 & B_2 \end{bmatrix} &= B_1 - \mathbf{b}_1 \cdot \begin{bmatrix} 1 & d_1^{-1}\mathbf{u}_1^* \end{bmatrix}, \end{aligned} \qquad (4.15)$$

dove $\mathbf{g}_1^*$ e $\mathbf{b}_1$ sono la prima riga di G_1 e la prima colonna di B_1, rispettivamente.

Dimostrazione. Dalla formula di complementazione di Schur (4.13) è possibile ricavare la fattorizzazione

$$R_1 = \begin{bmatrix} 1 & 0 \\ d_1^{-1}\mathbf{l}_1 & I \end{bmatrix} \cdot \begin{bmatrix} d_1 & 0 \\ 0 & R_2 \end{bmatrix} \cdot \begin{bmatrix} 1 & d_1^{-1}\mathbf{u}_1^* \\ 0 & I \end{bmatrix}.$$

Sostituendo questa espressione di R_1 nella equazione di spostamento (4.14) ed associando alcuni termini, si ottiene

$$\begin{aligned} &\begin{bmatrix} f_1 & 0 \\ *' & F_2 \end{bmatrix} \cdot \begin{bmatrix} d_1 & 0 \\ 0 & R_2 \end{bmatrix} \cdot \begin{bmatrix} 1 & d_1^{-1}\mathbf{u}_1^* \\ 0 & I \end{bmatrix} \\ &\qquad - \begin{bmatrix} 1 & 0 \\ d_1^{-1}\mathbf{l}_1 & I \end{bmatrix} \cdot \begin{bmatrix} d_1 & 0 \\ 0 & R_2 \end{bmatrix} \cdot \begin{bmatrix} a_1 & *' \\ 0 & A_2 \end{bmatrix} = G_1 \cdot B_1, \end{aligned}$$

da cui, utilizzando l'espressione dell'inversa di una matrice elementare di Gauss, si ottiene

$$\begin{bmatrix} 1 & 0 \\ -d_1^{-1}\mathbf{l}_1 & I \end{bmatrix} \cdot \begin{bmatrix} f_1 & 0 \\ *' & F_2 \end{bmatrix} \cdot \begin{bmatrix} d_1 & 0 \\ 0 & R_2 \end{bmatrix} - \begin{bmatrix} d_1 & 0 \\ 0 & R_2 \end{bmatrix} \cdot \begin{bmatrix} a_1 & *' \\ 0 & A_2 \end{bmatrix} \cdot \begin{bmatrix} 1 & -d_1^{-1}\mathbf{u}_1^* \\ 0 & I \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -d_1^{-1}\mathbf{l}_1 & I \end{bmatrix} \cdot G_1 \cdot B_1 \cdot \begin{bmatrix} 1 & -d_1^{-1}\mathbf{u}_1^* \\ 0 & I \end{bmatrix}$$

e

$$\begin{bmatrix} f_1 & 0 \\ *'' & F_2 \end{bmatrix} \cdot \begin{bmatrix} d_1 & 0 \\ 0 & R_2 \end{bmatrix} - \begin{bmatrix} d_1 & 0 \\ 0 & R_2 \end{bmatrix} \cdot \begin{bmatrix} a_1 & *'' \\ 0 & A_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -d_1^{-1}\mathbf{l}_1 & I \end{bmatrix} \cdot G_1 \cdot B_1 \cdot \begin{bmatrix} 1 & -d_1^{-1}\mathbf{u}_1^* \\ 0 & I \end{bmatrix}. \quad (4.16)$$

Scrivendo i generatori in modo da evidenziare la prima riga di G_1 e la prima colonna di B_1, otteniamo

$$\begin{bmatrix} 1 & 0 \\ -d_1^{-1}\mathbf{l}_1 & I \end{bmatrix} \cdot \begin{bmatrix} \mathbf{g}_1^* \\ \tilde{G}_1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{b}_1 & \tilde{B}_1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -d_1^{-1}\mathbf{u}_1^* \\ 0 & I \end{bmatrix} = \begin{bmatrix} \mathbf{g}_1^* \\ \tilde{G}_1 - d_1^{-1}\mathbf{l}_1\mathbf{g}_1^* \end{bmatrix} \cdot \begin{bmatrix} \mathbf{b}_1 & \tilde{B}_1 - d_1^{-1}\mathbf{b}_1\mathbf{u}_1^* \end{bmatrix}.$$

Definendo

$$G_2 = \tilde{G}_1 - d_1^{-1}\mathbf{l}_1\mathbf{g}_1^*$$
$$B_2 = \tilde{B}_1 - d_1^{-1}\mathbf{b}_1\mathbf{u}_1^*,$$

il blocco di posto $(2,2)$ dell'equazione (4.16) può essere scritto nella forma

$$F_2 \cdot R_2 - R_2 \cdot A_2 = G_2 \cdot B_2$$

e inoltre si ha

$$\begin{bmatrix} 0 \\ G_2 \end{bmatrix} = G_1 - \begin{bmatrix} 1 \\ d_1^{-1}\mathbf{l}_1 \end{bmatrix} \mathbf{g}_1^*$$
$$\begin{bmatrix} 0 & B_2 \end{bmatrix} = B_1 - \mathbf{b}_1 \cdot \begin{bmatrix} 1 & d_1^{-1}\mathbf{u}_1^* \end{bmatrix}.$$

Resta così dimostrata la tesi del teorema. □

Si vede dunque che il complemento di Schur mantiene la stessa struttura di spostamento delle matrici che l'hanno generato, e che è possibile ottenere i generatori del complemento di Schur a partire dai generatori delle matrici di partenza con un basso costo computazionale.

Questo fatto suggerisce la possibilità di applicare l'algoritmo di Schur ad una matrice strutturata in modo da operare solo sui generatori dei complementi di Schur, senza costruire esplicitamente le matrici intermedie. Un'accorta implementazione di tale algoritmo porterebbe ad un risparmio nell'occupazione di memoria e ad un riduzione della complessità computazionale.

4.4 Fattorizzazione veloce e stabile di matrici di Cauchy

L'implementazione dell'algoritmo di Schur per la fattorizzazione LU di una matrice R_1 che soddisfi un equazione di spostamento

$$\nabla_{\{F.A\}}(R_1) = F \cdot R_1 - R_1 \cdot A = G_1 \cdot B_1 \tag{4.17}$$

può essere riassunto con i seguenti passi:

1. Ricavare dai generatori G_1 e B_1 la prima riga e la prima colonna della matrice R_1. Queste forniscono, rispettivamente, la prima colonna di L e la prima riga di U.

2. Calcolare, mediante le relazioni (4.15), i generatori del complemento di Schur R_2.

3. Procedere ricorsivamente ripetendo i passi 1. e 2. sulla matrice R_2.

Per una matrice Cauchy-like, questa procedura consiste nel seguente algoritmo

for $k = 1, \ldots, n$
  $l_{kk} = 1$
  for $j = k+1, \ldots, n$
    $l_{jk} = \frac{\boldsymbol{\phi}_j^* \cdot \boldsymbol{\psi}_k}{t_j - s_k}$
  end
  for $j = k, \ldots, n$
    $u_{kj} = \frac{\boldsymbol{\phi}_k^* \cdot \boldsymbol{\psi}_j}{t_k - s_j}$
  end
  for j=$k+1, \ldots, n$
    $l_{jk} = \frac{l_{jk}}{u_{kk}}$
    $\boldsymbol{\phi}_j = \boldsymbol{\phi}_j - \overline{l_{jk}} \boldsymbol{\phi}_k$
    $\boldsymbol{\psi}_j = \boldsymbol{\psi}_j - \frac{u_{kj}}{u_{kk}} \boldsymbol{\psi}_k$
  end
end

di cui diamo anche una implementazione Matlab, scritta in modo da essere applicabile sia a matrici di Cauchy che a matrici Cauchy-like

```
function [L,U]=lu_c(t,s,phi,psi)
n=size(t,1);
if ~any(nargin==[2 4])
   error( 'errato numero di parametri')
end
if nargin==2
   phi=ones(1,n);
   psi=phi;
end
L=eye(n);
U=zeros(n);
for k = 1:n
   for j = k+1:n
      L(j,k)=phi(:,j)'*psi(:,k)/(t(j)-s(k));
   end
   for j=k:n
      U(k,j)=phi(:,k)'*psi(:,j)/(t(k)-s(j));
   end
   for j=k+1:n
      L(j,k)=L(j,k)/U(k,k);
      phi(:,j)=phi(:,j)-phi(:,k)*L(j,k)';
```

```
            psi(:,j)=psi(:,j)-psi(:,k)*(U(k,j)/U(k,k));
        end
    end
```

La complessità computazionale di questo algoritmo è $4\alpha n^2$, dove α, come già ricordato, è un intero indipendente da n che rappresenta la dimensione dei vettori $\boldsymbol{\phi}_j$ e $\boldsymbol{\psi}_j$.

Il vantaggio derivante dall'applicazione di questo algoritmo a matrici Cauchy-like consiste nella possibilità di implementare in esso la strategia del pivoting parziale, pur conservando la struttura. Questo, infatti, non è possibile per altre classe di matrici strutturate, come ad esempio per le matrici di Toeplitz o di Hankel.

Consideriamo una matrice Cauchy-like R_1 che soddisfi l'equazione (4.17) per determinate matrici diagonali $F = D_{\mathbf{t}}$ e $A = D_{\mathbf{s}}$.

Nell'applicare il pivoting parziale si richiede un movimento dell'elemento di più grande modulo della prima colonna di R_1 usando uno scambio di riga o, equivalentemente, una moltiplicazione a sinistra per la corrispodente matrice di permutazione P_1. Dopo lo scambio di riga la matrice $\hat{R_1} = P_1 R_1$ soddisfa ancora un'equazione di spostamento del tipo (4.17), in cui la matrice F è rimpiazzata dalla matrice $\hat{F} = P_1 F_1 P_1^*$ e il generatore G_1 viene sostituito da $\hat{G}_1 = P_1 G_1$. Questi movimenti di righe non distruggono la struttura di spostamento per le matrici Cauchy-like.

Il seguente algoritmo esegue la fattorizzazione $PR = LU$ per una matrice Cauchy-like R in $O(n^2)$ operazioni. Successivamente, con $O(n^2)$ operazioni è possibile risolvere i sistemi triangolari caratterizzati dalle matrici L e U mediante *forward* e *backward substitution*. Nell'algoritmo il simbolo I_n rappresenta la

matrice identità di dimensione n.

$P = I_n$
for $k = 1, \ldots, n$
    for $j = k, \ldots, n$
        $l_{jk} = \frac{\boldsymbol{\phi}_j^* \cdot \boldsymbol{\psi}_k}{t_j - s_k}$
    end
    trova $k \leq q \leq n : |l_{q,k}| = \max_{k \leq j \leq n} |l_{jk}|$
    $u_{kk} = l_{qk}$
    scambia t_k e t_q
    scambia $\boldsymbol{\phi}_k$ e $\boldsymbol{\phi}_q$
    scambia le righe k e q in L
    scambia le colonne k e q in P
    for $j = k+1, \ldots, n$
        $u_{kj} = \frac{\boldsymbol{\phi}_k^* \cdot \boldsymbol{\psi}_j}{t_k - s_j}$
    end
    $l_{kk} = 1$
    for j=$k+1, \ldots, n$
        $l_{jk} = \frac{l_{jk}}{u_{kk}}$
        $\boldsymbol{\phi}_j = \boldsymbol{\phi}_j - \overline{l_{jk}} \boldsymbol{\phi}_k$
        $\boldsymbol{\psi}_j = \boldsymbol{\psi}_j - \frac{u_{kj}}{u_{kk}} \boldsymbol{\psi}_k$
    end
end

Segue una implementazione in Matlab che restituisce i fattori P, L e U di una matrice Cauchy-like o Cauchy, a seconda dei parametri che vengono inseriti.

```
function [L,U,P]=lupp_c(t,s,phi,psi)
n=size(t,1);
if ~any(nargin==[2 4])
   error( 'errato numero di parametri')
end
if nargin==2
   phi=ones(1,n);
   psi=phi;
end
L=eye(n);
U=zeros(n);
P=eye(n);
for k = 1:n
   for j = k:n
```

```
      L(j,k)=phi(:,j)'*psi(:,k)/(t(j)-s(k));
   end
   [m q] = max(abs(L(k:end,k)));
   q = q+k-1;
   U(k,k) = L(q,k);
   t([k q]) = t([q k]);
   phi(:,[k q]) = phi(:,[q k]);
   L([k q],1:k) = L([q k],1:k);
   P([k q],:) = P([q k],:);
   L(k,k) = 1;
   for j = k+1:n
      U(k,j)=phi(:,k)'*psi(:,j)/(t(k)-s(j));
   end
   for j=k+1:n
      L(j,k)=L(j,k)/U(k,k);
      phi(:,j)=phi(:,j)-phi(:,k)*L(j,k)';
      psi(:,j)=psi(:,j)-psi(:,k)*(U(k,j)/U(k,k));
   end
end
```

Per avere uniformità con le implementazione degli altri algoritmi *veloci* visti sinora, abbiamo inoltre sviluppato due programmi per la risoluzione di un sistema lineare

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

caratterizzato da una matrice Cauchy-like C. Il primo utilizza l'algoritmo di fattorizzazione di Schur senza pivoting

```
function x = ge_c(t,s,phi,psi,b)
if nargin == 3
   b = phi;
   [L U] = lu_c(t,s);
elseif nargin == 5
   [L U] = lu_c(t,s,phi,psi);
else
   error( 'errato numero di parametri')
end
x = U \ (L \ b);
```

mentre il secondo fa uso del pivoting di colonna

```
function x = gepp_c(t,s,phi,psi,b)
n=length(t);
```

```
if nargin == 3
   b = phi;
   [L U P] = lupp_c(t,s);
elseif nargin == 5
   [L U P] = lupp_c(t,s,phi,psi);
else
   error( 'errato numero di parametri')
end
x = U \ (L \ (P*b));
```

4.4.1 Risultati numerici

Per verificare la *performance* di questi algoritmi abbiamo considerato un sistema lineare del tipo

$$C\mathbf{x} = \mathbf{b},$$

caratterizzato da una matrice C di Cauchy. Come nelle altre prove numeriche la soluzione è stata arbitrariamente fissata a $\mathbf{e} = (1, 1, \ldots, 1)^T$ e il termine noto è stato ricavato di conseguenza mediante il prodotto $\mathbf{b} = C\mathbf{e}$. Il sistema, è stato risolto al crescere della dimensione con la nostra implementazione dell'algoritmo di Schur con e senza pivoting, cioè con i programmi `gepp_c` e `ge_c`, e con l'algoritmo di Gauss generico con pivoting di colonna disponibile in Matlab. Durante il calcolo sono stati memorizzati il numero di operazioni eseguite dai tre algoritmi e gli errori sulla soluzione in norma-∞.

La prima matrice test è stata costruita in modo da mantenere un condizionamento contenuto anche per grandi dimensioni fissando i vettori $\mathbf{t}$ e $\mathbf{s}$ come segue

$$\begin{aligned}\mathbf{t} &= (2, 4, \ldots, 2n+1)^T \\ \mathbf{s} &= (2n-1, 2n-3, \ldots, 3, 1)^T.\end{aligned}$$

La Figura 4.1 mostra l'andamento del numero di condizionamento di questa matrice.

Come risulta dalla Figura 4.2 il numero di operazioni effettivamente eseguite dai nostri due algoritmi sono minori di quelle eseguite dall'algoritmo di Gauss (linea tratteggiata) come previsto dalla teoria, anche se questo vantaggio non è determinante viste le esigue dimensioni della matrice.

Un risultato interessante è invece rappresentato sul grafico degli errori riportato nella Figura 4.3. Esso mostra infatti che l'errore dell'algoritmo di Schur senza pivoting cresce troppo rapidamente rendendone sconsigliabile l'uso. Per questo

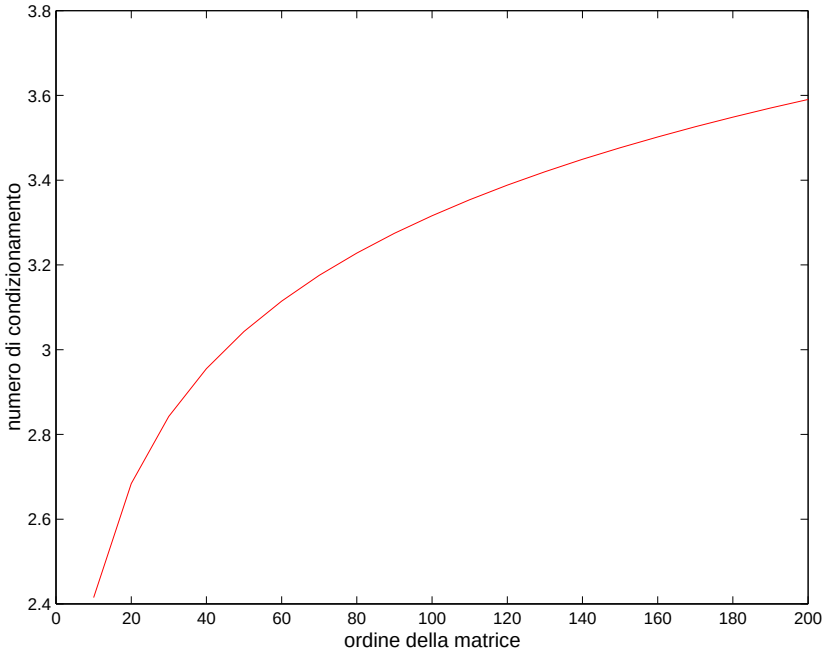

Figura 4.1: condizionamento della prima matrice di Cauchy

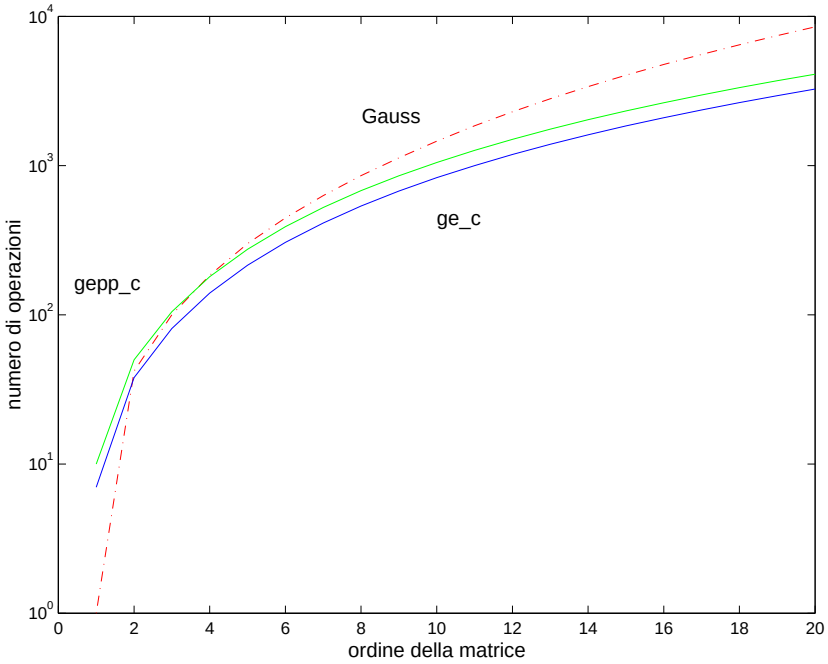

Figura 4.2: complessità computazionale, Schur vs. Gauss

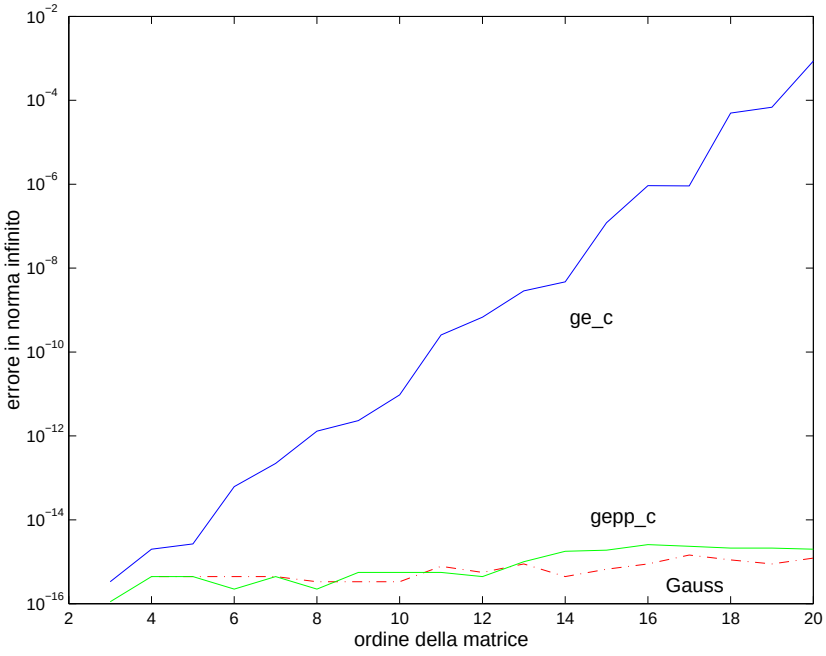

Figura 4.3: errori per il primo sistema di Cauchy

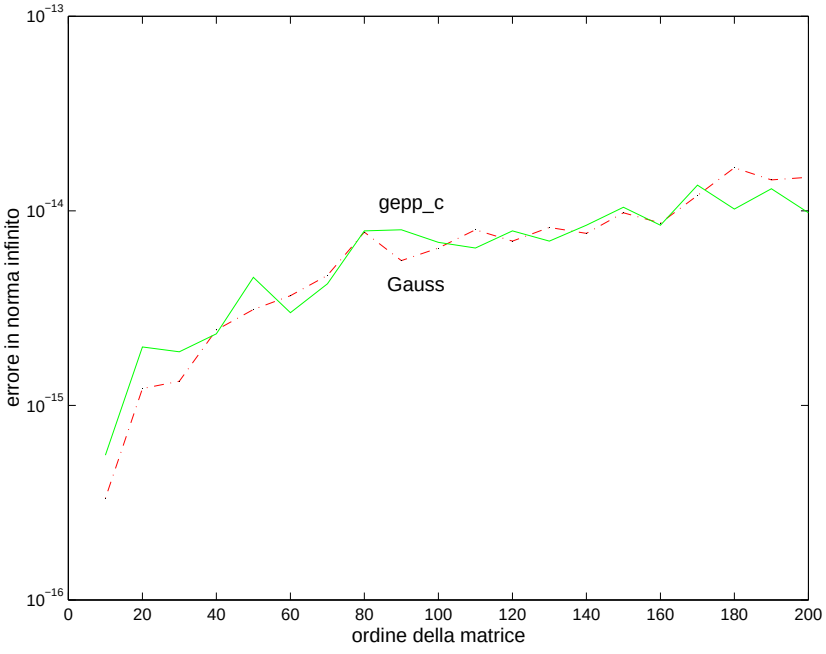

Figura 4.4: errori per il primo sistema di Cauchy

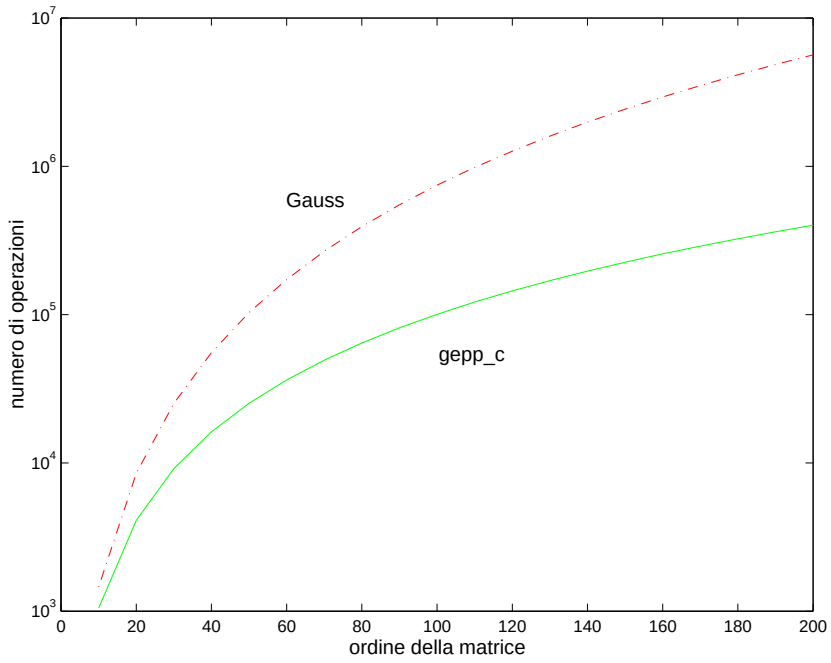

Figura 4.5: complessità computazionale, Schur vs. Gauss

motivo, nel test successivo non sono riportati i risultati ottenuti con questo algoritmo. La Figura 4.4 riporta gli errori misurati confrontando l'algoritmo con pivoting `gepp_c` e quello di Gauss per dimensioni moderatamente elevate e illustra la sostanziale equivalenza dei due algoritmi sotto l'aspetto della stabilità. Ben diverso è invece il costo computazionale, illustrato nella Figura 4.5. Da queste due ultime figure appare chiaro che si ottiene un vantaggio computazionale senza perdere in accuratezza rispetto alla soluzione.

Come secondo esempio è stata utilizzata la matrice di Hilbert, nota per il suo cattivo condizionamento come già evidenziato nel Paragrafo 3.2. Essa è definita da

$$H_{ij} = \frac{1}{i+j-1}, \quad i,j = 1,\ldots,n$$

ed è, oltre che una matrice di Hankel, anche una matrice di Cauchy caratterizzata dai vettori $\mathbf{t}$ e $\mathbf{s}$ aventi componenti $t_i = i$ e $s_j = 1 - j$.

La Figura 4.6 illustra l'andamento del numero di condizionamento di questa matrice. Nella Figura 4.7, invece, possiamo osservare che gli errori hanno un andamendo piuttosto simile per i due algoritmi, e non si ha un vantaggio in termini di complessità in quanto l'altissimo condizionamento di questa matrice non consente di risolvere sistemi lineari di dimensione superiore a 15. L'applicazione

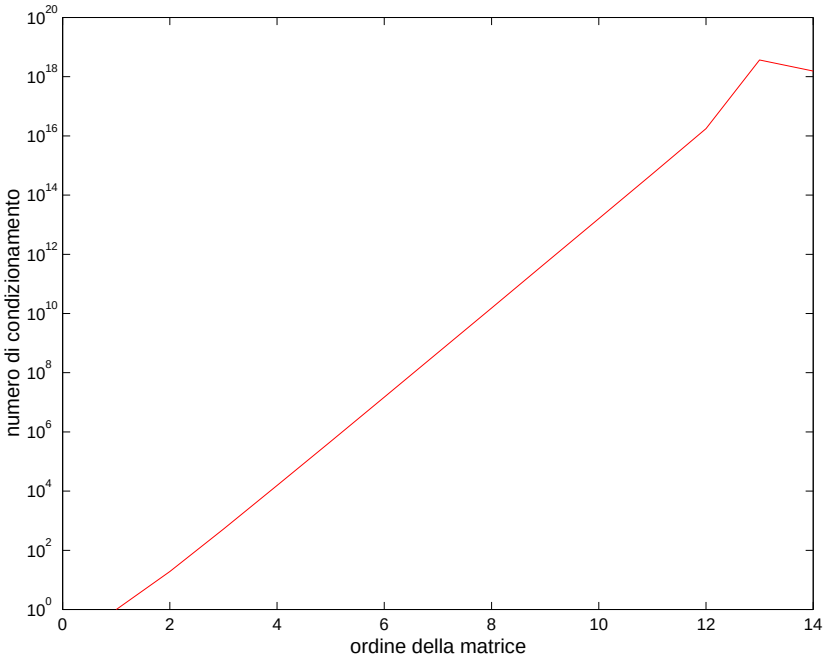

Figura 4.6: condizionamento della matrice di Hilbert

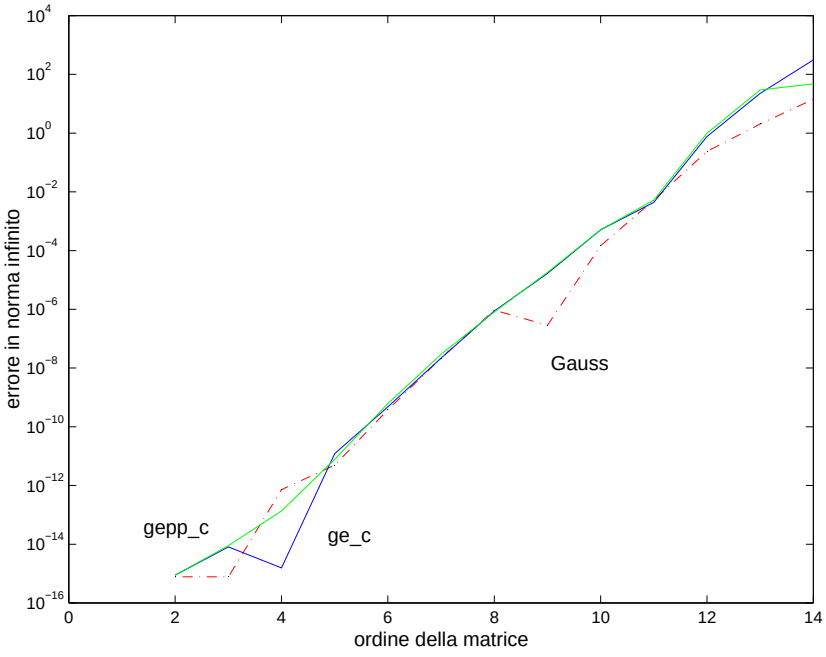

Figura 4.7: errori per il sistema di Hilbert

di un algoritmo veloce non porta quindi sostanziali vantaggi in presenza di una matrice molto mal-condizionata.

4.5 Conversione di struttura

In [3] si osserva che il pivoting parziale può essere incorporato all'interno di un algoritmo veloce non solo per le matrici Cauchy-like, ma anche per ogni classe di matrici con struttura di spostamento definita da (4.11) e caratterizzata da una matrice F diagonale. Infatti è stato dimostrato che è possibile trasformare un insieme di matrici dotate di struttura di spostamento in un altro insieme di matrici strutturate. Le tecniche per trasformare matrici Vandermonde-like e Cauchy-like in Toeplitz-like mediante la trasformata veloce di Fourier (*FFT*) sono state utilizzate in [8] per ridurre la complessità dei prodotti matrice per vettore in presenza di matrici con struttura di spostamento. La trasformazione da Toeplitz-like a Cauchy-like è stata invece introdotta in [5].

Incorporando il pivoting parziale nell'algoritmo di Schur generalizzato si ottiene, quindi, un algoritmo di fattorizzazione veloce e stabile, oltre che per le matrici Cauchy-like, anche per le matrici Toeplitz-like e Vandermonde-like. Questo algoritmo non è dunque ristretto ad un'unica classe di matrici regolari, ma è valido per un'arbitraria matrice invertibile in ognuna delle tre classi di matrici appena menzionate. Alla luce di queste considerazioni e del fatto che risulta conveniente applicare questo algoritmo alle matrici Cauchy-like, in [3] sono state proposte una varietà di formule che trasformano le classi base di matrici con struttura di spostamento in Cauchy-like. Queste formule richiedono unicamente la trasformata discreta veloce di Fourier (*FFT*) delle 2α colonne dei generatori G e B^*, che è un'operazione accurata, richiede un basso costo computazionale e inoltre preserva il condizionamento della matrice.

Consideriamo una matrice R che soddisfi l'equazione di spostamento di Sylvester (4.11) per certe matrici F ed A, e siano T_1 e T_2 due matrici invertibili. Allora la matrice

$$\hat{R} = T_1 R T_2 \tag{4.18}$$

soddisfa l'equazione di Sylvester

$$\nabla_{\{F,A\}}(\hat{R}) = \hat{F}\hat{R} - \hat{R}\hat{A} = \hat{G}\hat{B}$$

dove

$$\hat{F} = T_1 F T_1^{-1}$$
$$\hat{A} = T_2^{-1} A T_2$$
$$\hat{G} = T_1 G$$
$$\hat{B} = B T_2.$$

Questo ci consente di modificare la forma delle matrici F e A nell'equazione di spostamento, e perciò di trasformare una classe di matrici strutturate in un'altra classe. Le matrici T_1 e T_2 hanno il solo requisito di essere facilmente invertibili e di poter essere applicate ad un vettore con basso costo computazionale.

Nell'applicare questa procedura alla soluzione di un sistema lineare bisogna naturalmente tener conto delle trasformazioni che essa impone sul termine noto e sulla soluzione del sistema. Infatti, sostituendo la (4.18) nel sistema

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

si ottiene

$$(T_1 R T_2) \cdot (T_2^{-1}\mathbf{x}) = T_1\mathbf{b},$$

ossia il sistema lineare

$$\hat{R}\mathbf{y} = \hat{\mathbf{b}}$$

con $\hat{\mathbf{b}} = T_1\mathbf{b}$, la cui soluzione è legata a quella del sistema di partenza mediante la relazione

$$\mathbf{x} = T_2\mathbf{y}.$$

4.5.1 Conversione da Toeplitz-like a Cauchy-like

Teorema 4.5 *Sia $R \in \mathbb{C}^{n\times n}$ una matrice Toeplitz-like che soddisfa l'equazione*

$$\nabla_{\{Z_1, Z_{-1}\}}(R) = Z_1 \cdot R - R \cdot Z_{-1} = G \cdot B, \tag{4.19}$$

dove $G \in \mathbb{C}^{n\times\alpha}$ e $B \in \mathbb{C}^{\alpha\times n}$. Allora

$$C = \mathcal{F} \cdot R \cdot D_0^{-1} \cdot \mathcal{F}^*$$

è una matrice Cauchy-like con

$$\nabla_{\{D_1, D_{-1}\}}(C) = D_1 \cdot C - C \cdot D_{-1} = \hat{G} \cdot \hat{B}.$$

La matrice $\mathcal{F}$ è la matrice normalizzata di Fourier

$$\mathcal{F} = \frac{1}{\sqrt{n}} \left[\mathrm{e}^{\frac{2\pi \mathrm{i}}{n}(k-1)(j-1)} \right]_{1\le k,j\le n} \tag{4.20}$$

mentre

$$D_1 = \mathrm{diag}(1, \mathrm{e}^{\frac{2\pi \mathrm{i}}{n}}, \ldots, \mathrm{e}^{\frac{2\pi \mathrm{i}}{n}(n-1)})$$
$$D_{-1} = \mathrm{diag}(\mathrm{e}^{\frac{\pi \mathrm{i}}{n}}, \mathrm{e}^{\frac{3\pi \mathrm{i}}{n}}, \ldots, \mathrm{e}^{\frac{(2n-1)\pi \mathrm{i}}{n}})$$
$$D_0 = \mathrm{diag}(1, \mathrm{e}^{\frac{\pi \mathrm{i}}{n}}, \ldots, \mathrm{e}^{\frac{(n-1)\pi \mathrm{i}}{n}}).$$

I nuovi generatori sono dati da

$$\hat{G} = \mathcal{F} \cdot G, \quad \hat{B} = B \cdot D_0^{-1} \cdot \mathcal{F}^*. \tag{4.21}$$

Dimostrazione. La tesi segue immediatamente dalle fattorizzazioni

$$Z_1 = \mathcal{F}^* \cdot D_1 \cdot \mathcal{F} \quad \text{e} \quad Z_{-1} = D_0^{-1} \cdot \mathcal{F}^* \cdot D_{-1} \cdot \mathcal{F} \cdot D_0, \tag{4.22}$$

sostituendo le relazioni (4.22) nella (4.19) e moltiplicando per $\mathcal{F}$ a sinistra e per $D_0^{-1} \cdot \mathcal{F}^*$ a destra. □

Supponiamo ora di dover risolvere il sistema lineare

$$T\mathbf{x} = \mathbf{b} \tag{4.23}$$

dove T è una matrice Toeplitz-like che verifica l'equazione di Sylvester (4.19) e dotata dei generatori G e B. Sappiamo dal teorema precedente che $C = \mathcal{F} T D_0^{-1} \mathcal{F}^*$ è la matrice Cauchy-like trasformata della matrice T. Quindi, attraverso questa trasformazione, il sistema (4.23) diventa

$$(\mathcal{F} T D_0^{-1} \mathcal{F}^*) \cdot (\mathcal{F} D_0 \mathbf{x}) = \mathcal{F}\mathbf{b}$$

cioè

$$C\mathbf{y} = \hat{\mathbf{b}}$$

dove

$$\hat{\mathbf{b}} = \mathcal{F}\mathbf{b}$$
$$\mathbf{x} = M\mathbf{y} = (D_0^{-1} \mathcal{F}^*) \cdot \mathbf{y}.$$

La procedura di conversione di struttura da Toeplitz-like a Cauchy-like è stata implementata in un programma Matlab allo scopo di poter applicare l'algoritmo di Schur generalizzato ad una matrice di Toeplitz. Il programma prende in input i due vettori che definiscono la matrice di Toeplitz e il vettore $\mathbf{b}$ contenente i termini noti del sistema e restituisce i quattro parametri fondamentali della matrice Cauchy-like risultante, ossia i vettori $\mathbf{t}$ e $\mathbf{s}$ e le matrici $\Phi = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_n]$ e $\Psi = [\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n]$, unitamente al nuovo vettore dei termini noti $\hat{\mathbf{b}}$ e alla matrice

M che consente di trasformare la soluzione $\mathbf{y}$ del sistema Cauchy-like nella soluzione $\mathbf{x}$ del sistema di partenza. Nel corso dei calcoli si opera esclusivamente sui generatori delle matrici in gioco senza costruire esplicitamente la matrici dei due sistemi e quelle dei passi intermedi. La matrice di Fourier $\mathcal{F}$, per semplicità, è stata usata esplicitamente, ma una versione ottimizzata di questo algoritmo dovrebbe ricorrere alla *FFT* per ridurre la complessità computazionale e l'occupazione di memoria.

```
function [t,s,PHI,PSI,M,b1] = trasfoTC(tt,ss,b)
if nargin ~= 3
   error( 'errato numero di parametri')
end
if ss(1) ~= tt(1)
   ss(1) = tt(1);
end
tt=tt(:);
ss=ss(:);
n = size(tt,1);
F = fmat(n)' / sqrt(n);
w = exp(2*pi*i/n);
u = exp(pi*i/n);
D1 = diag(w.^[0:n-1].');
Dm1 = diag(u.^[1:2:2*n-1].');
D0 = diag(u.^[0:n-1].');

% generatori della Toeplitz
G = [[tt(1);tt(2:n)+ss(n:-1:2)] [1;zeros(n-1,1)]];
B = [zeros(1,n-1) 1;tt(n:-1:2)'-ss(2:n)' tt(1)];

% generatori della Cauchy-like
G1 = F * G;
B1 = B * D0' * F';

% parametri delle Cauchy-like
t = diag(D1);
s = diag(Dm1);
PHI = G1';
PSI = B1;
M = D0' * F';
b1 = F * b;
```

All'interno del programma compare una chiamata ad una piccola funzione che

consente la costruzione della matrice di Fourier, e che riportiamo di seguito.

```
function F = fmat(n)
f = 2*pi/n;
x = 0:n-1;
w = i * f * x';
F = exp(-w*x);
```

4.5.2 Conversione da Vandermonde-like a Cauchy-like

Teorema 4.6 *Sia $R \in \mathbb{C}^{n\times n}$ una matrice Vandermonde-like che soddisfa l'equazione*

$$\nabla_{\{D_{\mathbf{x}},Z_1\}}(R) = D_{\mathbf{x}} \cdot R - R \cdot Z_1 = G \cdot B, \tag{4.24}$$

dove $G \in \mathbb{C}^{n\times\alpha}$ e $B \in \mathbb{C}^{\alpha\times n}$. Siano $\mathcal{F}$ la matrice di Fourier normalizzata (4.20) e

$$D_1 = \mathrm{diag}(1, \mathrm{e}^{\frac{2\pi \mathrm{i}}{n}}, \ldots, \mathrm{e}^{\frac{2\pi \mathrm{i}}{n}(n-1)}).$$

Allora $R \cdot \mathcal{F}^*$ è una matrice Cauchy-like tale che

$$\nabla_{\{D_{\mathbf{x}},D_1\}}(R \cdot \mathcal{F}^*) = D_{\mathbf{x}} \cdot (R \cdot \mathcal{F}^*) - (R \cdot \mathcal{F}^*) \cdot D_1 = G \cdot (B \cdot \mathcal{F}^*). \tag{4.25}$$

Dimostrazione. È sufficiente sostituire nella (4.24) la fattorizzazione di Z_1 riportata in (4.22) e moltiplicare l'equazione a destra per $\mathcal{F}^*$. □

Consideriamo ora il sistema lineare

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

dove V è una matrice Vandermonde-like dotata di generatori G e B. Con la trasformazione riportata nel teorema il sistema diventa

$$(V\mathcal{F}^*) \cdot (\mathcal{F}\mathbf{x}) = \mathbf{b}$$

cioè

$$C\mathbf{y} = \mathbf{b},$$

dove C è una matrice Cauchy-like e $\mathbf{x} = M\mathbf{y} = \mathcal{F}^*\mathbf{y}$.

Il programma che converte una matrice di Vandermonde in una Cauchy-like accetta in input il vettore che genera la Vandermonde e il vettore dei termini noti $\mathbf{b}$ e restituisce i parametri della Cauchy-like e la matrice di trasformazione M.

```
function [t,s,PHI,PSI,M,b1] = trasfoVC(tt,b)
if ~any(nargin==[1 2])
   error( 'errato numero di parametri')
end
n = size(tt,1);
F = fmat(n)' / sqrt(n);
w = exp(-2*pi*i/n);
D1 = diag(w.^[0:n-1].');
Dx = diag(tt);

% generatori della Toeplitz
G = [tt.^(n)-1];
B = [zeros(1,n-1) 1];

% generatori della Cauchy-like
G1 = G;
B1 = B * F';

% parametri delle Cauchy-like
t = diag(Dx);
s = diag(D1');
PHI = G1';
PSI = B1;
M = F';
b1 = b;
```

4.5.3 Risultati numerici

In questo paragrafo intendiamo fare un confronto tra gli algoritmi veloci studiati nel Capitolo 2 per le matrici di Toeplitz e Vandermonde e la procedura di trasformazione in Cauchy-like accoppiata all'algoritmo di Schur.

Il primo confronto riguarda l'algoritmo di Levinson. Supponiamo, quindi, di voler risolvere il sistema lineare

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

in cui T è una matrice di Toeplitz simmetrica definita positiva. Le matrici test utilizzate sono le stesse introdotte nel Paragrafo 2.1.1, cioè le matrici *KMS* e *PROLATE*. e il termine noto è quello corrispondente a soluzioni unitarie.

Le Figure 4.8, 4.9 e 4.10 riportano gli errori misurati applicando gli algoritmi di Gauss, Levinson e l'algoritmo di Schur, preceduto dalla trasformazione in

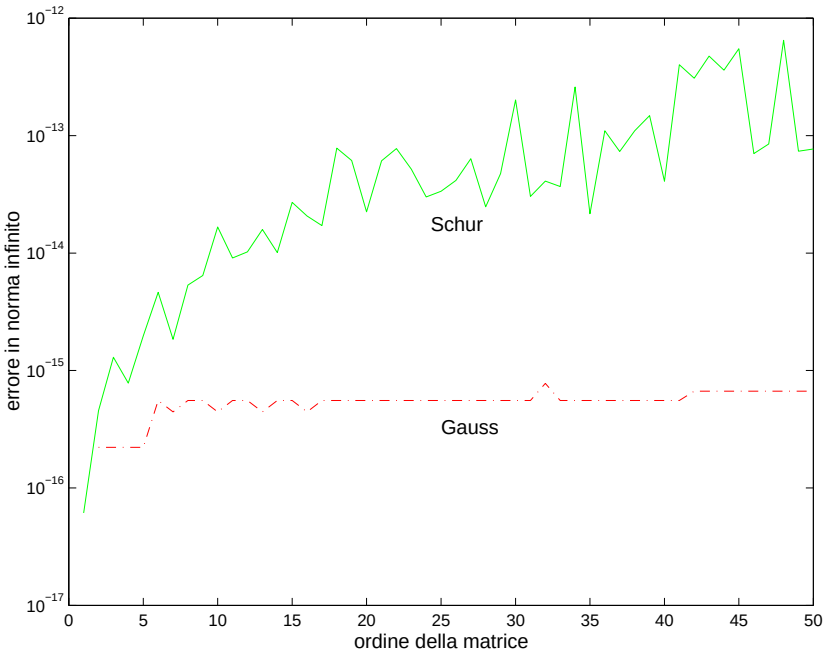

Figura 4.8: errori, *KMS*, $\rho = 1/2$

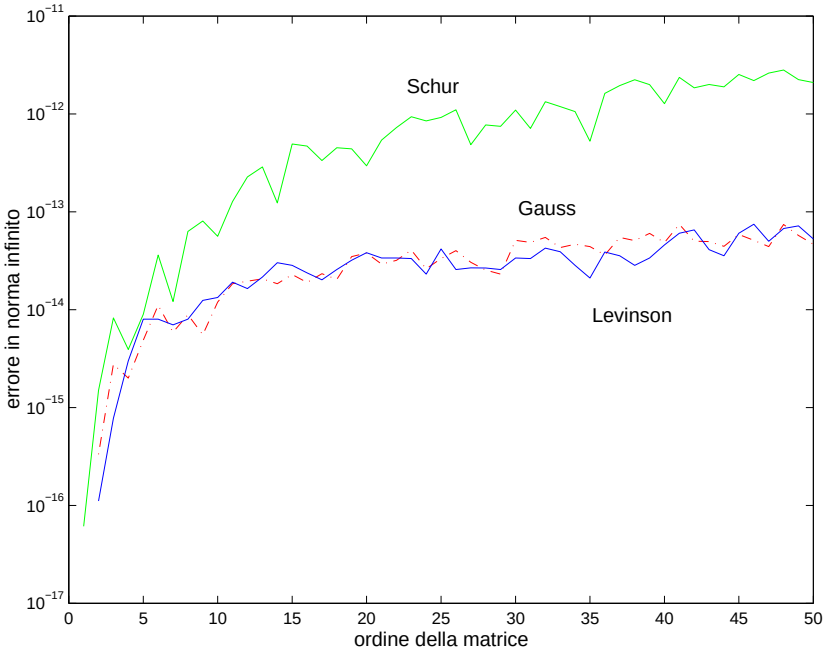

Figura 4.9: errori, *KMS*, $\rho = 9/10$

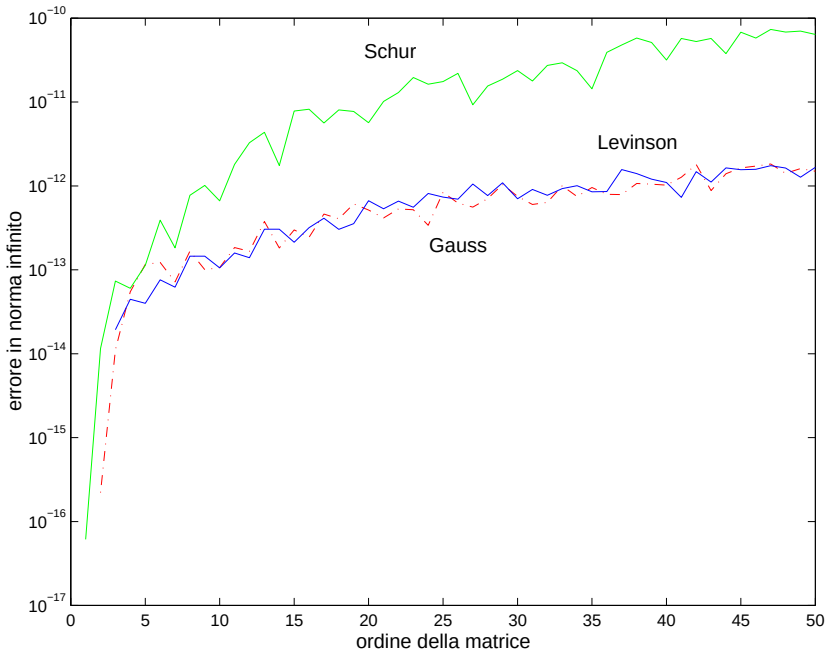

Figura 4.10: errori, *KMS*, $\rho = 99/100$

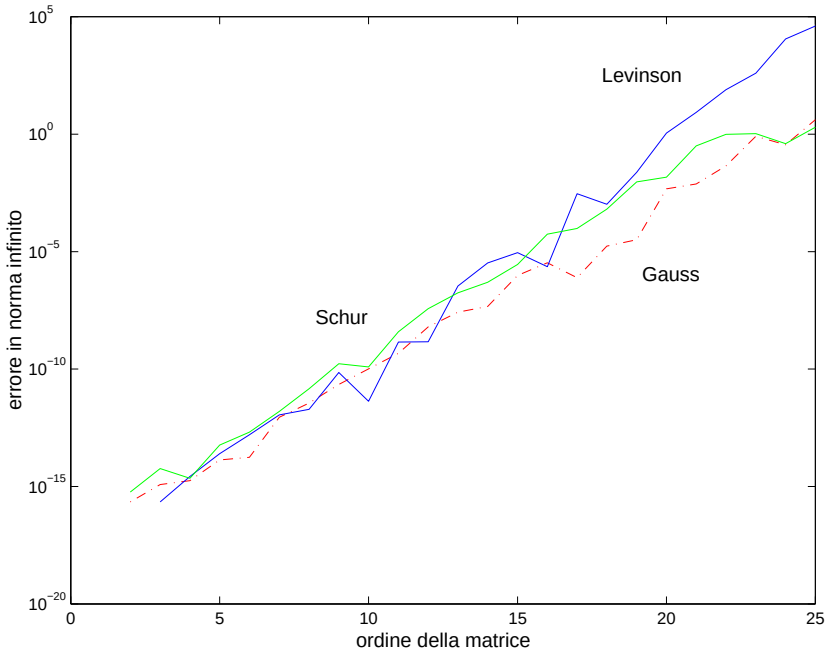

Figura 4.11: errori, *PROLATE*, $w = 0.25$

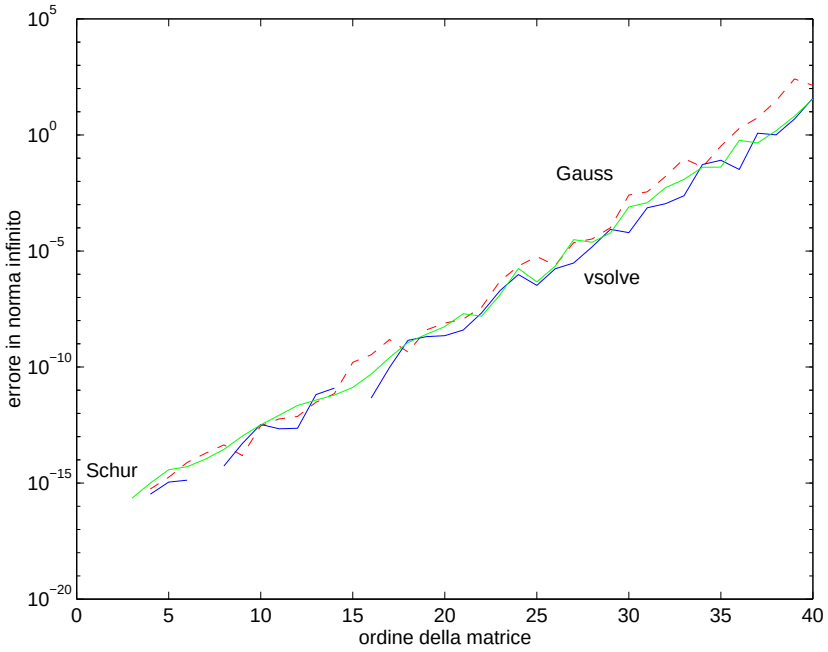

Figura 4.12: confronto Gauss - Schur - `vsolve`, I caso

Cauchy-like, alle matrici KMS, per i tre valori del parametro $\rho = 1/2,\ 9/10,\ 99/100$. Come si vede in questi esempi l'algoritmo di Schur generalizzato, pur producendo risultati ad un livello di accuratezza accettabile, è risultato meno preciso degli algoritmi di Gauss e Levinson. Per le matrici *PROLATE* con parametro $w = 0.25$, invece, la situazione è differente, come mostrato in Figura 4.11. Infatti, in questo caso l'algoritmo di Schur mantiene lo stesso grado di affidabilità di quello di Gauss pur con minore sforzo computazionale, mentre l'algoritmo di Levinson risente maggiormente della crescita del condizionamento.

Bisogna comunque ricordare che l'algoritmo di Schur è applicabile a qualsiasi matrice non singolare, mentre quello di Levinson è limitato a matrici definite positive, le quali sono fattorizzabili LU senza pivoting di colonna. Si sa invece, come già osservato alla fine del Paragrafo 2.1, che la generalizzazione dell'algoritmo di Levinson per matrici qualsiasi può risultare instabile, ed è probabilmente in questo caso che l'algoritmo di Schur risulta ad esso superiore.

Lo stesso confronto è stato effettuato per le matrici di Vandermonde mediante le matrici test utilizzate nel Paragrafo 2.2.1. Le Figure 4.12 e 4.13 riportano i risultati ottenuti con l'algoritmo di Gauss, l'algoritmo di interpolazione (2.10) e l'algoritmo di Schur applicato dopo la trasformazione in Cauchy-like, e mostrano che i risultati forniti dai tre algoritmi sono del tutto equivalenti.

Anche in questo caso, quindi, l'effetto del pivoting non ha portato vantaggi

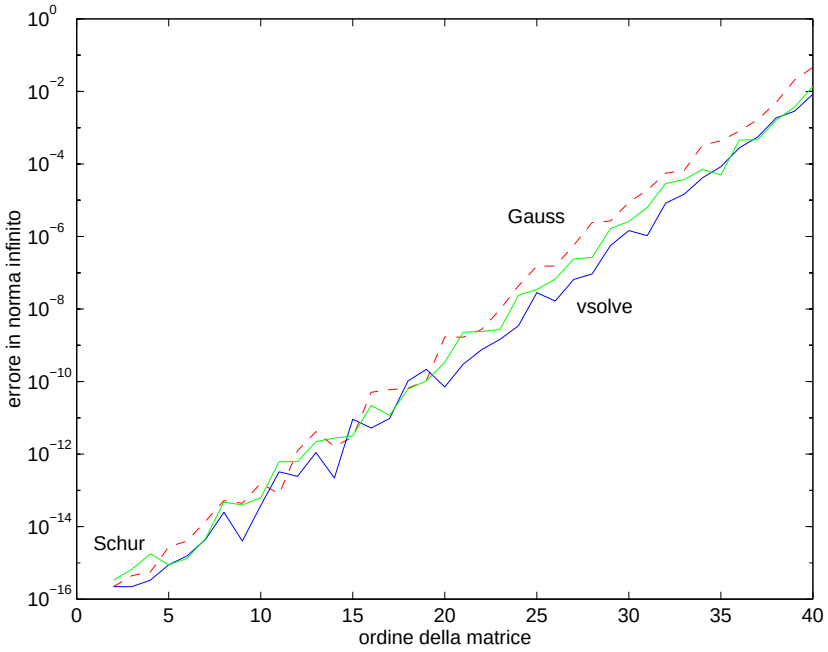

Figura 4.13: confronto Gauss - Schur - `vsolve`, II caso

in termini di accuratezza nè per l'algoritmo di Gauss nè per l'algoritmo di Schur. Quest'ultimo, però, si è dimostrato affidabile tanto quanto il primo ed è quindi ad esso preferibile, vista la minore complessità.

In conclusione, l'algoritmo di Schur generalizzato unisce le stesse proprietà di stabilità dell'algoritmo di Gauss alla complessità propria di un algoritmo veloce. Inoltre risulta applicabile a tutte le matrici dotate di struttura di spostamento che possano essere convertite in Cauchy-like con basso costo computazionale, e questo ne fa un algoritmo veloce *general purpose* per tutti i problemi in cui intervengano matrici strutturate.

Bibliografia

[1] D. Bini, M. Capovani, and O. Menchi. *Metodi Numerici per l'Algebra Lineare*. Zanichelli, Bologna, 1988.

[2] B. Friedlander, M. Morf, T Kailath, and L. Ljung. New inversion formulas for matrices classified in terms of their distance from Toeplitz matrices. *Linear Algebra and its Applications*, 27:31–60, 1979.

[3] I. Gohberg, T. Kailath, and V. Olshevsky. Fast gaussian elimination with partial pivoting for matrices with displacement structure. *Mathematics of Computation*, 64(212):1557–1576, 1995.

[4] I. Gohberg and I. Koltracht. Mixed, componentwise, and structured condition numbers. *SIAM J. Matrix Anal. Appl.*, 14:688–704, 1993.

[5] I. Gohberg and V. Olshevsky. Complexity of multiplication with vectors for structured matrices. *Linear Algebra Appl.*, 202:163–192, 1994.

[6] G.H. Golub and Van Loan C.F. *Matrix Computations*. The John Hopkins University Press, Baltimore, third edition, 1996.

[7] T.N.T. Goodman, C.A. Micchelli, G. Rodriguez, and S. Seatzu. On the limiting profile arising from orthonormalizing shifts of exponentially decaying functions. *IMA J. Num. Anal.*, 18(3):331–354, 1998.

[8] G. Heinig. Inversion of generalized Cauchy matrices and other classes of structured matrices. In *Linear Algebra in Signal Processing*, volume 69 of *IMA volumes in Mathematics and its Application*, pages 95–114. IMA, 1994.

[9] G. Heinig, P. Jankowski, and K. Rost. Fast inversion algorithms of Toeplitz-plus-Hankel matrices. *Numer. Math.*, 52:665–682, 1988.

[10] G. Heinig and K. Rost. *Algebraic methods for Toeplitz-like matrices and operators*, volume 13 of *Operator Theory: Advances and Applications*. Birkhäuser, Basel-Boston, 1984.

[11] N.J. Higham. The Test Matrix Toolbox for Matlab (Version 3.0). Technical Report 276, UMIST, 1995.

[12] T. Kailath, S.Y. Kung, and M. Morf. Displacement ranks of matrices and linear equations. *J. Math. Anal. Appl.*, 68:395–407, 1979.

[13] T. Kailath and A.H. Sayed. Fast algorithms for generalized displacement structures. In H. Kimura and S. Kodama, editors, *Recent Advances in Mathematical Theory of Systems, Control, Network and Signal Processing II, Proc. MTNS-91*, pages 27–32, Japan, 1992. Mita Press.

[14] The MathWorks, Inc., Natick, MA. *Matlab ver. 5.3*, 1999.

[15] T.J. Rivlin. *An Introduction to the Approximation of Function*. Dover Publications, Inc., New York, 1981.

[16] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*, volume 12 of *Texts in Applied Mathematics*. Springer-Verlag, New York, second edition, 1991.

[17] J.M. Varah. The prolate matrix. *Linear Algebra Appl.*, 187:269–278, 1993.