UNIVERSITÀ DI PISA

Dipartimento di Matematica
Corso di Laurea in Matematica

Tesi di Laurea Triennale

Metodi numerici *low-rank* per l'equazione di Sylvester

Relatore:
Prof. Dario Andrea Bini

Candidato:
Ivan Bioli

Anno Accademico 2020/2021

Alla mia famiglia: Mamma, Babbo, Ettore e Cesare,
a nonna Gabriella e nonno Gianfranco,
a nonna Lidia e nonno Adriano,
a Zia, Zio e Edoardo,
a Zio Piero,
ai miei amici,
a Flaminia per questi tre anni

Indice

Sommario

Considereremo l'equazione matriciale lineare di Sylvester $AX + XB = C$, con A, B matrici quadrate reali o complesse, C di dimensioni compatibili e incognita X. L'obiettivo di questo elaborato è quello di fornire una panoramica dei principali strumenti numerici per la risoluzione dell'equazione di Sylvester, con una particolare attenzione al caso in cui la matrice dei coefficienti C ammette una fattorizzazione di rango basso. Viene proposto un algoritmo per aggiornare velocemente la soluzione di tali equazioni quando i coefficienti A, B, C vengono sottoposti a modifiche di rango basso, con una applicazione alle Equazioni di Riccati Algebriche. Tramite la sperimentazione numerica viene dimostrato il vantaggio degli approcci che sfruttano la fattorizzazione *low-rank* dei coefficienti.

1. Introduzione

1.1 Introduzione al problema

Considereremo l'equazione matriciale lineare

$$AX + XB = C$$

con A, B matrici quadrate reali o complesse, C di dimensioni compatibili e incognita X. Tale equazione, per matrici dei coefficienti generiche, è detta equazione di Sylvester. Se invece $B = A^*$ e $C = C^*$, dove A^* è la trasposta coniugata di A, l'equazione prende il nome di equazione di Lyapunov e la soluzione, se esiste, può essere scelta hermitiana. Tali equazioni sono state largamente trattate in letteratura durante il secolo scorso, sia dal punto di vista teorico che computazionale, e sono tutt'ora oggetto di ricerca per le numerose applicazioni in cui compaiono, tra cui l'analisi della stabilità e la riduzione dimensionale di sistemi dinamici e la risoluzione di equazioni alle derivate parziali.

In questo elaborato ci si è concentrati più sull'aspetto computazionale che su quello teorico, con l'obiettivo di fornire una panoramica dei principali strumenti numerici per la risoluzione dell'equazione di Sylvester e un esempio di applicazione di tali tecniche, con particolare attenzione al caso in cui la matrice dei coefficienti C ammette una fattorizzazione di rango basso. Nel caso di problemi di taglia moderata l'algoritmo di Bartels-Stewart [1], basato sulla decomposizione di Schur, si dimostra tutt'oggi tra i più efficienti per la risoluzione dell'equazione di Sylvester nonostante sia stato introdotto già nel 1972. Tuttavia, in alcune applicazioni le matrici dei coefficienti A, B e C sono di taglia grande e non affrontabili con l'algoritmo di Bartels-Stewart. Se la matrice C ammette però una fattorizzazione di rango basso, questa può essere sfruttata per la risoluzione del problema attraverso algoritmi iterativi. Si noti tuttavia che il concetto di taglia "moderata" o "grande" non è assoluto, problemi che hanno una taglia che li rende oggi facilmente trattabili erano invece intrattabili soltanto qualche decina di anni fa. Il resto dell'elaborato è organizzato come segue.

Nel Capitolo 1 vengono introdotte alcune notazioni e alcuni risultati preliminari che saranno utili durante l'intero elaborato.

Nel Capitolo 2, dopo aver fornito condizioni teoriche per garantire esistenza e unicità della soluzione, viene presentato l'algoritmo di Bartels-Stewart. Successivamente viene discussa la approssimabilità della soluzione X dell'equazione di Sylvester tramite matrici di rango basso, condizione necessaria per poter applicare solutori low-rank per il calcolo di approssimazioni della soluzione X tramite matrici di rango basso. In particolare vengono trattati i metodi proiettivi, basati sugli Spazi di Krylov a blocchi, e il metodo Alternating-Direction-Implicit (ADI).

Nel Capitolo 3 viene trattato il problema degli aggiornamenti di rango basso. Data X_0 soluzione dell'equazione di Sylvester $A_0X + B_0X = C_0$, si vuole calcolare un aggiornamento

δX tale che $X := X_0 + \delta X$ sia soluzione dell'equazione perturbata $(A_0 + \delta A)X + X(B_0 + \delta B) = C_0 + \delta C$. Si trova che l'aggiornamento δX deve risolvere un'equazione di Sylvester con membro destro di rango basso a cui si possono applicare i solutori trattati nel precedente capitolo. Viene presentato l'algoritmo introdotto in [19].

Il Capitolo 4 tratta l'applicazione delle tecniche descritte nei capitoli precedenti all'Equazione di Riccati Algebrica Continua, cioè un'equazione matriciale della forma $XA + A^*X - XBX = C$ dove $A, B \in \mathbb{C}^{n\times n}$ e X è l'incognita. Viene presentato il metodo di Newton, illustrando come ottenere le iterate del suddetto metodo tramite approssimazioni successive di rango basso nel caso in cui la matrice dei coefficienti B ammetta una fattorizzazione di rango basso.

Nel Capitolo 5 viene concentrata la sperimentazione numerica relativa a tutti i metodi proposti nel corso dell'elaborato. L'implementazione degli algoritmi è stata svolta in MATLAB, con i listings necessari riportati in appendice . I risultati della sperimentazione numerica si rivelano consistenti con quanto ci si aspettava teoricamente: gli algoritmi low-rank si dimostrano più efficienti in termini di tempo di esecuzione, ma raggiungono una precisione inferiore rispetto a solutori basati sull'algoritmo di Bartels-Stewart.

1.2 Notazioni e risultati preliminari

Dove non specificato diversamente considereremo matrici reali, quadrate e di dimensione finita, mentre useremo $\mathbb{K}$ dove si potrebbe indifferentemente inserire $\mathbb{R}$ o $\mathbb{C}$. Con $\mathrm{Spec}(A)$ indicheremo lo spettro (insieme degli autovalori) della matrice A, mentre con A^T, A^* rispettivamente la trasposta e la trasposta coniugata di A. Scriveremo inoltre $\rho(A) = \max_{\lambda\in\mathrm{Spec}(A)}|\lambda|$ per il raggio spettrale di A.

Il vettore e_i denota l'i-esimo vettore della base canonica, la cui dimensione sarà chiara dal contesto, così come per la matrice identità I. Dato un vettore $x \in \mathbb{K}^n$ indicheremo con $\|x\| = \|x\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2}$ la sua norma euclidea (o norma 2). Analogamente, date una norma vettoriale $\|\cdot\|$ e una matrice $A = (a_{i,j})_{i=1,\dots m, j=1\dots n}$ con $\|A\|$ indicheremo la norma matriciale indotta dalla norma vettoriale $\|\cdot\|$, definita da $\|A\| = \max_{\|x\|=1} \|Ax\|$. In particolare la norma indotta dalla norma euclidea è tale che $\|A\|_2 = \rho(AA^*)^{1/2}$ [7]. Infine la norma di Frobenius di A è definita da $\|A\|_F^2 = \sum_{i,j}|a_{i,j}|^2$.

Introduciamo adesso delle definizioni e delle notazioni meno ovvie delle precedenti.

Definizione 1.1. *Una matrice A si dice stabile se tutte le soluzioni del sistema di Equazioni Differenziali Ordinarie (E.D.O.) lineari a coefficienti costanti $\dot{u}(t) = Au(t)$ sono infinitesime per $t \to +\infty$. Si mostra facilmente che questo è equivalente a richiedere che tutti gli autovalori di A abbiano parte reale negativa.*

Notazione. *La notazione $A \succ 0$ ($A \succeq 0$) indica che A è una matrice hermitiana definita (semidefinita) positiva. La notazione è analoga nel caso di matrici definite o semidefinite negative.*

Scriveremo inoltre $A \succ B$ ($A \succeq B$) per indicare $A - B \succ 0$ ($A - B \succeq 0$).

Notazione. *Data una matrice qualsiasi A indicheremo con $\mathcal{R}(A)$ o* range(A) *il suo range, cioè il sottospazio generato dalle sue colonne. In generale, dati vettori $v_1, \dots, v_m$ indicheremo con* Span$(v_1, \dots, v_m)$ *il sottospazio generato da $v_1, \dots, v_m$.*

1.2.1 Prodotto di Kronecker

Definizione 1.2 (Prodotto di Kronecker)**.** *Siano $A \in \mathbb{K}^{n_A \times m_A}$ e $B \in \mathbb{K}^{n_B \times m_B}$, il prodotto di Kronecker è definito da*

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1m_A}B \\ a_{21}B & a_{22}B & \cdots & a_{2m_A}B \\ \vdots & & & \vdots \\ a_{n_A1}B & a_{n_A2}B & \cdots & a_{n_Am_A}B \end{bmatrix} \in \mathbb{K}^{n_An_B \times m_Am_B}.$$

Data la matrice X, le cui colonne sono $x_i \in \mathbb{K}^m$ $i = 1, \ldots, n$, l'operatore vec *impila le colonne di X una sopra l'altra, cioè:*

$$\mathrm{vec}\,(X) = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{K}^{mn \times 1}.$$

Riassumiamo nella seguente proposizione alcune proprietà del prodotto di Kronecker [9].

Proposizione 1.1. *Alcune proprietà del prodotto di Kronecker:*

(i) $\mathrm{vec}(AXB) = (B^T \otimes A)\mathrm{vec}(X)$.

(ii) $(I \otimes B)(A \otimes I) = A \otimes B$.

(iii) $(A \otimes B)^T = A^T \otimes B^T$.

(iv) Se A, B, C, D sono matrici di dimensioni compatibili allora

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD).$$

In particolare se A, B sono matrici quadrate invertibili $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.
Se $A = S\hat{A}S^{-1}, B = R\hat{B}R^{-1}$ si ha $A \otimes B = (S \otimes R)(\hat{A} \otimes \hat{B})(S \otimes R)^{-1}$.

(v) Se $A \in \mathbb{K}^{n \times n}$ e $B \in \mathbb{K}^{m \times m}$, gli autovalori di $A \otimes B$ sono dati dai prodotti degli autovalori di A con gli autovalori di B (in tutti i modi possibili), cioè $\lambda \in \mathrm{Spec}(A \otimes B)$ se e solo se $\lambda = \lambda_A\lambda_B$ dove $\lambda_A \in \mathrm{Spec}(A), \lambda_B \in \mathrm{Spec}(B)$.

In particolare gli autovalori di $I_m \otimes A + B \otimes I_n$ sono tutti e soli i $\lambda \in \mathbb{K}$ della forma $\lambda = \lambda_A + \lambda_B$.

1.2.2 Decomposizione ai valori singolari

La decomposizione ai valori singolari (*Singular Values Decomposition, SVD*) è un ingrediente fondamentale per la determinazione della miglior approssimazione di rango fissato di una generica matrice A. Questa ci tornerà particolarmente utile nel seguito per "comprimere matrici", memorizzandone una approssimazione di rango basso in maniera efficiente. Riportiamo in seguito alcuni risultati classici riguardanti la SVD, per una dimostrazione si veda [9].

Teorema 1.2 (Esistenza della SVD)**.** *Sia $A \in \mathbb{C}^{m \times n}$, allora esistono due matrici unitarie $U \in \mathbb{C}^{m \times m}$, $V \in \mathbb{C}^{n \times n}$ tali che*

$$A = U\Sigma V^* \tag{1.1}$$

dove la matrice $\Sigma \in \mathbb{R}^{m\times n}$ *ha elementi* σ_{ij} *nulli per* $i \neq j$ *e per* $i = j$ *ha elementi* $\sigma_{ii} = \sigma_i$*, con:*

$$\sigma_1 \geq \cdots \geq \sigma_p \geq 0, \qquad p = \min\{m, n\}.$$

La decomposizione (1.1) è detta decomposizione ai valori singolari *della matrice A mentre i valori* $\sigma_i,\ i = 1, \ldots, p$ *sono detti* valori singolari *di A. Indicate con* $u_i,\ i = 1, \ldots, m$ *e* $v_i,\ i = 1, \ldots, n$ *le colonne rispettivamente di U e di V, i vettori* $u_i,\ v_i,\ i = 1, \ldots, p$ *sono detti rispettivamente* vettori singolari sinistri *e* vettori singolari destri *di A. La matrice* Σ *è univocamente determinata, mentre non lo sono le matrici U e V.*

Teorema 1.3. *Sia* $A \in \mathbb{C}^{m\times n}$ *e sia* $A = U\Sigma V^*$ *la sua decomposizione ai valori singolari, dove*

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > \sigma_{k+1} = \ldots \sigma_p = 0, \qquad p = \min\{m, n\}.$$

Allora, nelle notazioni del teorema precedente, valgono le seguenti proprietà:

(i)

$$A = U_k \Sigma_k V_k^* = \sum_{i=1}^{k} \sigma_i\, u_i\, v_i^*$$

dove $U_k = \left[u_1 \middle| \ldots \middle| u_k\right]$, $V_k = \left[v_1 \middle| \ldots \middle| v_k\right]$ *e* $\Sigma_k = diag(\sigma_1, \ldots \sigma_k)$.

(ii) Il nucleo di A è generato da $v_{k+1}, \ldots, v_n$.

(iii) L'immagine di A è generata da $u_1, \ldots, u_k$.

(iv) Gli autovalori non nulli della matrice A^*A *sono* $\sigma_i^2,\ i = 1, \ldots, k$*, pertanto*

$$\|A\|_F^2 = \sum_{i=1}^{k} \sigma_i^2, \qquad \|A\|_2 = \sigma_1.$$

(v) Se $m = n$ *e la matrice A è normale, allora* $\sigma_i = |\lambda_i|,\ i = 1, \ldots, n$*, dove i* λ_i *sono gli autovalori di A, e i vettori singolari destri e sinistri coincidono con gli autovettori di A.*

Teorema 1.4 (Teorema di Eckart-Young)**.** *Sia* $A \in \mathbb{C}^{m\times n}$ *e sia* $A = U\Sigma V^*$ *la sua decomposizione ai valori singolari, dove*

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > \sigma_{k+1} = \ldots \sigma_p = 0, \qquad p = \min\{m, n\}$$

e sia r un intero positivo minore o uguale a k. Indicando con

$$A_r = \sum_{i=1}^{r} \sigma_i\, u_i\, v_i^* \tag{1.2}$$

e con $\mathcal{S} = \{B \in \mathbb{C}^{m\times n} \ :\ rk(B) \leq r\}$ *si ha:*

$$\min_{B\in\mathcal{S}} \|A - B\|_2 = \|A - A_r\| = \sigma_{r+1}. \tag{1.3}$$

Il precedente teorema ci permette dunque di "comprimere" una matrice A, calcolandone la SVD e memorizzando soltanto i primi r valori/vettori singolari invece dell'intera matrice. Tale tecnica prende il nome di *TSVD (Truncated Singluar Value Decomposition)*. Il valore di taglio r può essere fissato in base al fattore di compressione che vogliamo ottenere oppure in base alla qualità della compressione che vogliamo raggiungere. Nel primo caso scegliamo r in maniera tale che $r/n < \tau$ con τ fattore di compressione. Nel secondo caso invece possiamo scegliere il taglio in maniera tale che $\sigma_i/\sigma_1 < \varepsilon$ per $i > r$ per un valore fissato di ε opportunamente piccolo.

1.2.3 Il teorema del MiniMax di Courant-Fischer

Sia $A = A^* \in \mathbb{R}^{n\times n}$ una matrice hermitiana, allora possiamo definire il quoziente di Rayleigh associato alla matrice A come la funzione $r_A : \mathbb{C}^{n\times n} \setminus \{0\} \to \mathbb{R}$ definita da

$$r_A(x) = \frac{x^*Ax}{x^*x}. \tag{1.4}$$

Si noti che tale funzione è ben definita (cioè ha effettivamente codominio $\mathbb{R}$) poichè la matrice A è hermitiana.

Se gli autovalori di A sono ordinati in modo che

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

con corrispondenti autovettori $v_1, v_2, \ldots, v_n$, si può mostrare facilmente che λ_1 è il massimo di r_A ed è assunto in corrispondenza di v_1, mentre λ_n è il minimo di r_A ed è assunto in corrispondenza di v_n. Un risultato più preciso viene fornito dal seguente teorema, per la cui dimostrazione si rimanda a [9].

Teorema 1.5 (Teorema del MiniMax di Courant-Fischer)**.** *Sia $A \in \mathbb{C}^{n\times n}$ una matrice hermitiana con autovalori $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Allora risulta*

$$\lambda_{n-k+1} = \min_{\substack{V_k\subseteq\mathbb{R}^n \\ \dim V_k = k}} \max_{\substack{x\in V_k \\ x\neq 0}} r_A(x)$$

$$\lambda_k = \max_{\substack{V_k\subseteq\mathbb{R}^n \\ \dim V_k = k}} \min_{\substack{x\in V_k \\ x\neq 0}} r_A(x).$$

Un interessante corollario è il seguente:

Corollario 1.5.1. *Sia $A \in \mathbb{C}^{n\times n}$ una matrice hermitiana con* $\mathrm{Spec}(A) \subseteq J$ *con $J \subseteq \mathbb{R}$ intervallo e sia $V \in \mathbb{C}^{n\times k}$ tale che $V^*V = I_k$. Allora* $\mathrm{Spec}(V^*AV) \subseteq J$.

Dimostrazione. Siano $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ gli autovalori di A. Deve valere $[\lambda_n, \lambda_1] \subseteq J$ e in generale potrebbe valere anche l'uguaglianza. Dunque possiamo supporre senza perdita di generalità che $J = [\lambda_n, \lambda_1]$. Siano inoltre $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_k$ gli autovalori di $B := V^*AV$, ci è sufficiente mostrare che $\lambda_n \leq \mu_n \leq \mu_1 \leq \lambda_1$. Essendo B hermitiana per il Teorema 1.5

$$\mu_1 = \max_{\substack{U_1\subseteq\mathbb{R}^k \\ \dim U_1 = 1}} \min_{\substack{x\in U_1 \\ x\neq 0}} r_B(x).$$

Supponiamo che tale massimo sia raggiunto in corrispondenza del sottospazio $\widehat{U}_1$, allora detto $\widehat{V}_1 = \{y \in \mathbb{R}^n \ : \ y = Vx, x \in \widehat{U}_1\}$ si ha $\dim\widehat{V}_1 = 1$ e

$$\mu_1 = \min_{\substack{x\in\widehat{U}_1 \\ x\neq 0}} \frac{x^*Bx}{x^*x} = \min_{\substack{x\in\widehat{U}_1 \\ x\neq 0}} \frac{(Vx)^*A(Vx)}{(Vx)^*(Vx)} = \min_{\substack{y\in\widehat{V}_1 \\ y\neq 0}} \frac{y^*Ay}{y^*y} \leq \max_{\substack{V_1\subseteq\mathbb{R}^n \\ \dim V_1 = 1}} \min_{\substack{y\in V_1 \\ y\neq 0}} r_A(y) = \lambda_1$$

dove l'ultima uguaglianza deriva anch'essa dal Teorema 1.5. Applicando un ragionamento identico alla matrice $-A$ si prova la disuguaglianza $\mu_n \geq \lambda_n$. □

1.2.4 Spazi di Krylov

Data $A \in \mathbb{C}^{n\times n}$ e $b \in \mathbb{C}^n$. Lo spazio

$$\mathcal{K}_m(A, b) = \mathrm{Span}\{b, Ab, A^2b, \ldots, A^{m-1}b\}$$

è detto m-esimo Spazio di Krylov.

Gli spazi di Krylov sono fondamentali per la risoluzione di sistemi lineari di grandi dimensioni e sparse, tanto che i più comuni solutori lineari usano proprio gli Spazi di Krylov.

Una procedura standard per determinare una base ortonormale di uno Spazio di Krylov è il metodo di Arnoldi. Tale metodo, salvo interruzioni forzate, dopo m passi fornisce una base di $\mathcal{K}_m(A, b)$. Si considera $q_1 = b/\|b\|_2$ e poi per $k = 1, 2, \ldots, m-1$ si pone:

$$\begin{aligned} h_{i,k} &= q_i^* A q_k, \quad i = 1, \ldots, k, \\ w_k &= A q_k - \sum_{i=1}^{k} h_{i,k} q i, \\ h_{k+1,k} &= \|w_k\|_2, \\ q_{k+1} &= w_k / h_{k+1,k}. \end{aligned} \tag{1.5}$$

Se al passo k-esimo si ottiene $w_k = 0$ allora lo spazio di Krylov $\mathcal{K}_m(A, b)$ è A-invariante.

La descrizione di cui sopra dell'Algoritmo di Arnoldi non è numericamente stabile in aritmetica floating point. Tuttavia, una algoritmo leggermente modificato, cioè l'Algoritmo di Arnoldi con modifica di Gram-Schmidt, produce gli stessi vettori ortonormali ma con proprietà di stabilità numerica nettamente migliori.

Posto $V_m = [q_1, \ldots, q_m] \in \mathbb{C}^{n \times m}$ e detta $H_{m+1,m} \in \mathbb{C}^{(m+1) \times m}$ la matrice di Hessenberg tale che $(H_{m+1,m})_{ij} = h_{ij}$, se indichiamo con H_m la sottomatrice principale di testa di $H_{m+1,m}$ vale:

$$A V_m = V_{m+1} H_{m+1,m} = V_m H_m + h_{m+1,m} q_{m+1} e_m^T$$

Si veda [27] per ulteriori approfondimenti sugli Spazi di Krylov e per le dimostrazioni omesse.

2. L'equazione di Sylvester

Una equazione di Sylvester è un'equazione matriciale lineare della forma:

$$AX + XB = C \tag{2.1}$$

dove $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{m\times m}$, $C \in \mathbb{C}^{n\times m}$ e l'incognita è $X \in \mathbb{C}^{n\times m}$. Si noti che in generale le taglie di A e B possono essere molto diverse, cosa che influenza l'efficienza degli algoritmi risolutivi.

L'equazione (2.1) è detta equazione di Sylvester poiché la sua prima apparizione è storicamente associata al lavoro di J.J. Sylvester in [31]. Se $B = A^*$ e $C = C^*$ è chiamata equazione di Lyapunov in onore di A.M. Lyapunov e dei suoi contributi in materia. Le equazioni matriciali lineari, in particolare le due di cui sopra, sono state argomento di un gran numero di studi a partire dal primo '900 e successivamente dagli anni '50 e '60 del secolo scorso. Tale campo è tutt'oggi costantemente studiato e sviluppato, stimolato soprattutto da numerose applicazioni in svariati ambiti della teoria del controllo e dei sistemi dinamici.

In questo capitolo, basato principalmente su [30, 10], ci proponiamo di esporre i risultati che saranno di maggiore utilità nello sviluppo dell'argomento principale di questo elaborato, fornendo al contempo riferimenti per eventuali approfondimenti.

2.1 Esistenza e unicità della soluzione

Usando il prodotto di Kroenecker e le proprietà nella Proposizione 1.1, l'equazione (2.1) può essere riscritta come

$$\mathcal{A}x = c \qquad \text{dove} \qquad \begin{array}{c} \mathcal{A} = I_m \otimes A + B^T \otimes I_n, \\ x = \operatorname{vec}(X), \quad c = \operatorname{vec}(C). \end{array} \tag{2.2}$$

Nonostante la riscrittura dell'equazione (2.1) come sistema lineare sia piuttosto semplice dal punto di vista formale, questo produce un considerevole aumento della taglia, con $\mathcal{A} \in \mathbb{C}^{nm\times nm}$. Pertanto non è conveniente applicare metodi iterativi o diretti al sistema definito dall'equazione (2.2) quando m e n assumono valori elevati. In particolare una implementazione di un algoritmo risolutivo come nel Listato 2.1 sarebbe del tutto inefficiente, richiedendo $\mathcal{O}((mn)^3)$ operazioni aritmetiche se ad esempio il sistema viene risolto tramite metodi diretti come quello di Gauss con pivot parziale.

La formulazione di cui in (2.2) risulta però particolarmente utile nella dimostrazione del seguente teorema.

Teorema 2.1. *L'equazione* (2.1) *ammette soluzione unica per ogni scelta di* $C \in \mathbb{C}^{n\times m}$ *se e solo se* $\operatorname{Spec}(A)$ *e* $\operatorname{Spec}(-B)$ *sono disgiunti.*

Listato 2.1: *Algoritmo per l'equazione di Sylvester basato sulla formulazione vettoriale tramite prodotto di Kroenecker.*

```
function X = sylv_vectorial(A,B,C)
% X=SYLV_VECTORIAL(A,B,C) risolve l'equazione di Sylvester AX + XB = C
    sfruttando la formulazione come sistema lineare tramite prodotto di
    Kroenecker
% A, B, C: matrici dei coefficienti (di taglie compatibili)
% X soluzione
[m,n] = size(C);
H = kron(eye(n), A) + kron(B.', eye(m));
Cvec = reshape(C,m*n,1);
Xvec = H\Cvec;
X = reshape(Xvec,m,n);
```

Dimostrazione. Dalla espressione (2.2), la soluzione esiste unica per ogni scelta di C se e solo se la matrice $\mathcal{A}$ è non singolare. Dalla Proposizione 1.1 otteniamo che questo è vero se e solo se $\lambda_A + \lambda_B \neq 0$ per ogni $\lambda_A \in \mathrm{Spec}(A), \lambda_B \in \mathrm{Spec}(B)$ e cioè la tesi. □

Nel seguito della trattazione assumeremo che tale condizione venga soddisfatta, di modo che sia assicurato che il problema ammetta soluzione unica.

Esistono svariate formule chiuse per ricavare la soluzione X dell'equazione (2.1), per una panoramica si vedano ad esempio [30, 21]. Tuttavia nessuna di queste formule può essere usata numericamente a causa di problemi di efficienza o di stabilità numerica. In letteratura sono stati sviluppati numerosi metodi per la risoluzione numerica, divisi in tre macro-categorie: metodi di tipo proiettivo (basati principalmente sugli spazi di Krylov), metodi basati su sequenze di aggiornamenti (come il metodo Alternating-Direction-Implicit) o metodi ricorsivi per matrici sparse con particolari strutture. La convergenza dei primi due approcci dipende fortemente dalle proprietà spettrali delle matrici dei coefficienti.

2.2 Metodi per matrici di taglia medio-bassa

Un metodo per risolvere problemi di dimensione medio-bassa è stato introdotto nel 1972 da Bartels e Stewart in [1] e, con qualche modifica, è ancora oggi il metodo più utilizzato per questo tipo di problemi. L'idea di tale algoritmo è quella di sfruttare la decomposizione di Schur (reale o complessa, a seconda del caso) delle matrici A e B per generare una equazione equivalente che permetta di sfruttare la forma quasi-triangolare superiore delle matrici di Schur. Nel caso generale di A e B matrici complesse, l'algoritmo può essere riassunto come nell'Algoritmo 1.

Algoritmo 1 Algoritmo di Bartels-Stewart

1: Calcola le forme normali di Schur: $A^* = URU^*$, $B = VSV^*$
2: Risolvi in Y: $R^*Y + YS = U^*CV$
3: $X = UYV^*$

È chiaro che l'equazione che andiamo a risolvere sia equivalente a quella iniziale, dato che è ottenuta semplicemente da $UR^*U^*X + XVSV^* = C$ moltiplicando a sinistra per U^* e a destra per V. Resta però da chiarire come risolvere tale problema. A tale scopo, indichiamo con $\widehat{C}$ la matrice $\widehat{C} := U^*VC$. Essendo R ed S triangolari superiori, l'equazione

Listato 2.2: *Implementazione dell'Algoritmo di Bartels-Stewart nel caso complesso.*

```
function X = sylv_bartels_stewart(A,B,C)
% X=SYLV_BARTELS_STEWART(A,B,C) risolve l'equazione di Sylvester AX + XB =
    C usando l'algoritmo di Bartels-Stewart basato sulla decomposizione di
    Schur complessa
% A, B, C: matrici dei coefficienti
% X : soluzione
[n,m] = size(C);
[U,R] = schur(A','complex');
R = R';
[V,S] = schur(B,'complex');
C = U'*C*V;
X = zeros(n,m);
X(1,:) = C(1,:)/(R(1,1)*eye(m) + S);
for k = 2:n
    v = C(k,:) - R(k,1:k-1)*X(1:k-1,:);
    X(k,:) = v/(S + R(k,k)*eye(m));
end
X = U*X*V';
```

letta sull'elemento (k, j) diventa:

$$\overline{R_{kk}}Y_{kj} + Y_{kj}S_{jj} = \widehat{C}_{kj} - \sum_{i=1}^{k-1} \overline{R_{ik}}Y_{ij} - \sum_{i=1}^{j-1} Y_{ki}S_{ij}\,. \tag{2.3}$$

Questo ci permette di ricavare gli elementi di Y in sequenza a partire dall'elemento Y_{11} e proseguendo per righe.

Nel caso di matrici A, B reali possiamo sfruttare la forma normale di Schur reale, con S e R quasi triangolari superiori con blocchi 1×1 o 2×2 sulla diagonale. Se i blocchi di R^* e S sono di dimensioni compatibili, possiamo analogamente a prima ricavare gli elementi di Y in sequenza andando a risolvere equazioni di Sylvester di taglia al più 2×2.

Il costo complessivo di tale algoritmo è di circa $10n^3+10m^3+\frac{5}{2}(n^2m+nm^2)$ operazioni aritmetiche, si veda [15] per un resoconto dettagliato. Tale costo è fortemente influenzato dal costo del calcolo della decomposizione di Schur, che nel caso di una matrice $k \times k$ richiede circa $10k^3$ operazioni. Un metodo proposto da Golub, Nash e Van Loan nel 1979 in [15] permette di ridurre tale costo, sfruttando la decomposizione di Hessenberg per A (con un costo di circa $\frac{5}{3}n^3$ operazioni) al posto di quella di Schur, che viene invece comunque calcolata per B. In tal caso il costo complessivo è di circa $\frac{5}{3}n^3 + 10m^3 + 5n^2m + \frac{5}{2}nm^2$ operazioni, dunque notevolmente più basso soprattutto se $n > m$.

Una implementazione in MATLAB dell'Algoritmo di Bartels e Stewart può essere quella del Listato 2.2.

2.3 Approssimabilità a rango basso

In principio, sotto le ipotesi di applicabilità, il metodo di Bartels-Stewart appena presentato funziona per qualsiasi matrici di qualsiasi taglia. In pratica però, date le limitazioni hardware dei calcolatori, funziona solo per matrici di taglia moderata. Questo non è un concetto assoluto, dato che chiaramente la "taglia moderata" cresce con il progresso tecnologico, ma può essere considerata come la più grande taglia per cui gli algoritmi standard per la moltiplicazione di matrici hanno un tempo di esecuzione ragionevole. Tuttavia i problemi di taglia grande hanno solitamente due caratteristiche principali, che possono essere sfruttate da algoritmi iterativi: sparsità e rango basso rispetto alla taglia.

In questa sezione, discuteremo quando la soluzione dell'equazione di Sylvester (2.1) ammetta una buona approssimazione di rango basso, richiamando alcuni risultati noti in letteratura. Supponiamo di avere:

$$AX + XB = C, \qquad \operatorname{rank}(C) \leq k. \tag{2.4}$$

Diremo che X ammette una ε-approssimazione di rango r se esiste una matrice Y di rango al più r tale che $\|X - Y\|_2 \leq \varepsilon$. Per il Teorema 1.4 di Eckart-Young, questo è equivalente a richiedere che l'$(r+1)$-esimo valore singolare $\sigma_{r+1}(X)$ sia dominato da ε (oppure che la taglia di X sia più piccola di $r+1$). Il minimo valore di r è detto ε-rango, inoltre se ε è la precisione di macchina l'ε-rango è detto rango numerico.

In letteratura sono presenti svariati risultati sul decadimento dei valori singolari delle soluzioni di equazioni del tipo di (2.4), si vedano ad esempio [3, 11] e le relative citazioni in materia. In particolare da alcune considerazioni in [3] si può ricavare il seguente risultato.

Proposizione 2.2. *Siano A, B matrici hermitiane definite positive con spettro contenuto nell'intervallo $[a, b]$ dove $0 < a < b < \infty$. Allora la soluzione X dell'equazione* (2.4) *soddisfa:*

$$\frac{\sigma_{1+kh}(X)}{\|X\|} \leq 4\rho^{-h} \qquad \rho := \exp\left(\frac{\pi^2}{\log(4b/a)}\right). \tag{2.5}$$

Una generalizzazione al caso di matrici diagonalizzabili viene presentata in [11]. Indicati con $\kappa_{eig}(A)$, $\kappa_{eig}(B)$ i numeri di condizionamento delle matrici degli autovettori rispettivamente di A e B, viene proposto il limite superiore (ottimale, anche se non asintoticamente)

$$\frac{\sigma_{1+kh}(X)}{\|X\|} \leq 4\kappa_{eig}(A)\kappa_{eig}(B)\rho^{-h}. \tag{2.6}$$

Chiaramente i due risultati precedenti non coprono il caso generale (richiedono addirittura che le matrici A e B siano quadrate e diagonalizzabili), ma ci fanno capire che se k è piccolo allora X ammette una ε-approssimazione con ε che decade a zero esponenzialmente all'aumentare di h. Osserviamo inoltre che la separazione degli autovalori di A e B ha un effetto sul fattore di convergenza fortemente mitigato dalla presenza del logaritmo. Risulta dunque ragionevole aspettarsi che, anche nel caso generale, se $\operatorname{rank}(C)$ è molto piccolo allora la soluzione possa essere ben approssimata da una matrice di rango basso.

2.4 Metodi proiettivi

Sia $\mathcal{V}$ un sottospazio di $\mathbb{C}^n$ di dimensione k e supponiamo che le colonne di $V_k \in \mathbb{C}^{n\times k}$ costituiscano una base ortonormale di $\mathcal{V}$. L'idea dei metodi basati su proiezioni è quella di cercare una approssimazione X_k della soluzione tale che $\mathcal{R}(X_k) \subset \mathcal{V}$, cerchiamo cioè $Y_k \in \mathbb{C}^{k\times m}$ in maniera tale che $X_k := V_k Y_k$ verifichi:

$$R_k := AX_k + X_k B - C \approx 0.$$

In maniera analoga a quanto viene fatto, ad esempio, nel Metodo del Gradiente Coniugato, un'idea può essere quella di imporre una qualche condizione di ortogonalità sul residuo R_k. Ad esempio se imponiamo l'ortogonalità delle colonne della matrice R_k rispetto al sottospazio di approssimazione $\mathcal{V}$ otteniamo la condizione di Galerkin:

$$0 = V_k^* R_k = V_k^* A V_k Y_k + Y_k B - V_k^* C.$$

Supponendo che k sia molto più piccolo della taglia di A, ci troviamo a dover risolvere un'equazione di Sylvester in cui la prima matrice dei coefficienti ha taglia fortemente ridotta

rispetto a quella iniziale. Tale metodo è quindi applicabile efficientemente se A ha taglia grande e B ha taglia piccola, cioè in una situazione che può essere visualizzata come sotto:

$$\begin{bmatrix} \quad A \quad \end{bmatrix}\begin{bmatrix} X \end{bmatrix} + \begin{bmatrix} X \end{bmatrix}\begin{bmatrix} B \end{bmatrix} = \begin{bmatrix} C \end{bmatrix}.$$

Si noti che altri tipi di condizioni possono risultare estremamente naturali, come la minimizzazione del residuo rispetto a una qualche norma matriciale o l'ortogonalità rispetto a un qualche altro spazio, ma noi non considereremo queste alternative nonostante siano state spesso analizzate in letteratura.

Quando sia A che B hanno taglia molto grande, occorre ridurre la taglia di entrambe le matrici dei coefficienti per poi operare una risoluzione del tipo descritto nell'Algoritmo 1. Se C ammette una decomposizione di rango basso $C = C_1 C_2^*$ con $C_1 \in \mathbb{C}^{n\times s}$, $C_2 \in \mathbb{C}^{m\times s}$ e $s \ll \min\{m, n\}$, quanto detto nella Sezione 2.3 ci porta a pensare che la soluzione dell'equazione (2.1) possa essere ben approssimata con una matrice di rango basso, rendendo i metodi di tipo proiettivo estremamente interessanti. Si noti che l'approssimazione di X si rende necessaria anche per ragioni di risparmio di memoria, dato che se n e m sono molto grandi non è possibile memorizzare l'intera matrice $X \in \mathbb{R}^{n\times m}$ e diventa necessario usare una qualche tecnica per ridurre lo spazio di memoria utilizzato.

Generalizzando quanto visto in precedenza, siano $\mathcal{V}$ e $\mathcal{W}$ due sottospazi di $\mathbb{C}^n$, in principio non necessariamente della stessa dimensione, tali che $\mathcal{V}$ non sia ortogonale a C_1 e $\mathcal{W}$ non sia ortogonale a C_2. Supponiamo che le $k \ll n$ colonne di V_k siano una base ortonormale di $\mathcal{V}$ e le $j \ll m$ colonne di W_j siano una base ortonormale di $\mathcal{W}$. Cerchiamo una approssimazione di rango basso della soluzione del tipo

$$\widetilde{X} = V_k Y W_j^* \approx X. \tag{2.7}$$

Detto $R := C_1 C_2^* - A\widetilde{X} - \widetilde{X}B$ il residuo associato a tale approssimazione, possiamo imporre anche in questo caso una condizione di ortogonalità. Vedendo $\tilde{x} = \mathrm{vec}(\widetilde{X}) = (W_j \otimes V_k)\mathrm{vec}(Y)$ come una soluzione approssimata di (2.2) e imponendo l'ortogonalità del residuo $c - \mathcal{A}\tilde{x}$ rispetto allo spazio $\mathcal{R}(W_j \otimes V_k)$ si ottiene

$$(W_j \otimes V_k)^*(c - \mathcal{A}\tilde{x}) \quad \iff \quad V_k^* R W_j = 0. \tag{2.8}$$

Sostituendo a R la sua definizione otteniamo l'equazione di Sylvester di taglia ridotta:

$$V_k^* A V_k Y + Y W_j^* B W_j = (V_k^* C_1)(W_j^* C_2)^*. \tag{2.9}$$

Per il Teorema 2.1, l'equazione ammette soluzione unica per qualsiasi scelta del membro destro se e solo se gli spettri di $V_k^* A V_k$ e $-W_j^* B W_j$ sono disgiunti. Per il Corollario 1.5.1 tale condizione può essere assicurata assumendo che le matrici A e B siano hermitiane tali che $\mathrm{Spec}(A) \subseteq [a, b]$, $\mathrm{Spec}(-B) \subseteq [c, d]$ e $[a, b] \cap [c, d] = \emptyset$.

Tale algoritmo può essere a priori applicato per una qualsiasi scelta dei sottospazi $\mathcal{V}, \mathcal{W}$. Però, l'efficienza del metodo e il costo necessario a determinare i nuovi vettori $\hat{v}, \hat{w}$ dipendono fortemente da $\mathcal{V}$ e $\mathcal{W}$. Solitamente la scelta è analoga per entrambi gli spazi e pertanto ci limiteremo a discutere soltanto la scelta di $\mathcal{V}$.

Per primo Saad in [28] ha proposto l'uso degli spazi di Krylov per determinare un'approssimazione di rango basso della soluzione dell'equazione di Lyapunov, che analizzeremo più a fondo nella Sezione 2.6. Uno dei problemi principali dei metodi di tipo proiettivo è che le basi V_k e W_k devono essere memorizzate per fornire l'approssimazione finale. Questo pone limiti piuttosto stretti sulla dimensione dei due spazi di Krylov quando A e B sono molto grandi, cosa che non rende efficiente il metodo nel caso si usino spazi di Krylov

Algoritmo 2 Metodo proiettivo

1: Ortogonalizza le colonne di C_1 per ottenere $v_1 = V_1$
2: Ortogonalizza le colonne di C_2 per ottenere $w_1 = W_1$
3: **for** $k = 1, 2, \dots$ **do**
4: Risolvi in Y_k l'equazione: $V_k^* A V_k Y + Y W_j^* B W_j = (V_k^* C_1)(W_j^* C_2)^*$
5: **if** converged **then return** V_k, Y_k, W_j
6: **end if**
7: Calcola $\hat{v}, \hat{w}$ per gli spazi $\mathcal{V}, \mathcal{W}$ scelti
8: Rendi $\hat{v}, \hat{w}$ ortogonali rispettivamente a $\{v_1, \dots v_k\}$ e $\{w_1, \dots w_k\}$
9: Ortogonalizza le colonne di $\hat{v}, \hat{w}$ per ottenere v_{k+1}, w_{k+1}
10: $V_{k+1} = [V_k, v_{k+1}]$, $W_{k+1} = [W_k, w_{k+1}]$
11: **end for**

Standard. In letteratura ci si è dunque concentrati sulla ricerca di alternative che permettessero, seppur con un costo più alto, di ottenere buone approssimazioni con spazi di dimensione più piccola. Le prestazioni di questi spazi di Krylov sono notevolmente migliori e permettono di risolvere il problema in maniera più efficiente. Oltre allo spazio di Krylov Standard, elenchiamo alcune delle possibili scelte che si possono fare per $\mathcal{V}$ a partire da A e C_1. Similmente si può ricavare $\mathcal{W}$ usando B^* e C_2.

1. Spazio di Krylov Standard (a blocchi):

$$\mathcal{V} = \mathcal{K}_k^{\square}(A, C_1) := \text{range}([C_1, AC_1, A^2C_1, \dots, A^kC_1]).$$

2. Spazio di Krylov Razionale (a blocchi):

$$\mathcal{V} = \text{range}([(A + \sigma_1 I)^{-1} C_1, (A + \sigma_2 I)^{-1}(A + \sigma_1 I)^{-1} C_1, \dots])$$

con $\{\sigma_i\}$ successione che assicuri la non singolarità delle matrici shiftate.

3. Spazio di Krylov Globale (a blocchi)

$$\mathcal{V} = \left\{ \sum_{i=0}^{k-1} \gamma_i A^i C_1, \quad \gamma_i \in \mathbb{R} \right\} = \text{Span}\left\{ C_1, AC_1, A^2C_1, \dots \right\}$$

dove le combinazioni lineari vengono fatte tra blocchi.

4. Spazio di Krylov Esteso (a blocchi)

$$\mathcal{V} = \mathcal{EK}_k(A, C_1) := \mathcal{K}_k^{\square}(A, C_1) + \mathcal{K}_k^{\square}(A^{-1}, A^{-1}C_1).$$

Si noti che questi metodi sono tutti collegati in un qualche senso, ad esempio gli spazi di Krylov Standard possono essere formalmente visti come spazi di Krylov Razionali dove $\sigma_i = \infty \; \forall i$, oppure gli spazi di Krylov Estesi possono essere visti come spazi di Krylov Razionali in cui σ_i vale alternatamente 0 e ∞. Gli spazi di Krylov Globali sono invece spazi di Krylov Standard in cui però viene ridotto il numero di gradi di libertà.

2.4.1 Alcuni dettagli implementativi

In questa sezione accenneremo a dettagli che, pur non essendo necessari alla comprensione dell'Algoritmo 2 dal punto di vista teorico, risultano indispensabili per una efficiente implementazione. Analizzeremo inoltre in dettaglio il caso dell'implementazione tramite spazi di Krylov Estesi, per la loro semplice formulazione coniugata ad una ottima efficienza.

Come spazi di Krylov useremo nelle successive implementazioni degli spazi di Krylov Estesi. In particolare, supponiamo che l'equazione sia della forma $AX + XB = C_1C_2^*$ dove C_1 e C_2 hanno $s \ll \min\{n, m\}$ colonne. Considereremo V_k e W_k basi ortonormali dei sottospazi:

$$\mathcal{V}_k = \mathcal{EK}_k(A, C_1) = \text{Span}\left\{C_1, A^{-1}C_1, AC_1, A^{-2}C_1, \ldots, A^{k-1}C_1, A^{-k}C_1\right\},$$
$$\mathcal{W}_k = \mathcal{EK}_k(B^*, C_2) = \text{Span}\left\{C_2, (B^*)^{-1}C_2, B^*C_2, (B^*)^{-2}C_2, \ldots, (B^*)^{k-1}C_2, (B^*)^{-k}C_2\right\}$$

per qualche k che soddisfi $2ks < \min\{m, n\}$. Queste due matrici possono essere ottenute tramite processi di tipo Arnoldi a blocchi. In particolare, si rivela particolarmente efficiente per il nostro caso il Processo di Arnoldi Esteso a blocchi *(Extended Block Arnoldi Process, EBA)* proposto in [17]. Tale processo è integrato nello pseudocodice dell'Algoritmo 3 e permette di ricavare due successioni di matrici $Q_1^V, \ldots, Q_k^V \in \mathbb{C}^{n\times 2s}$ e $Q_1^W, \ldots, Q_k^W \in \mathbb{C}^{m\times 2s}$ tali che le colonne della matrice $V_k = [Q_1^V\, Q_2^V\, \cdots\, Q_k^V]$ formino una base ortonormale di $\mathcal{V}_k$ e le colonne di $W_k = [Q_1^W\, Q_2^W\, \cdots\, Q_k^W]$ formino una base ortonormale di $\mathcal{W}_k$.

Richiamiamo brevemente il funzionamento del metodo di Arnoldi a blocchi per lo Spazio di Krylov Esteso $\mathcal{EK}(A, C_1)$ omettendo per semplicità di notazione l'apice V. Calcoleremo anche una matrice di Hessenberg a blocchi $H_{k+1,k} \in \mathbb{C}^{2(k+1)s\times 2ks}$ che permette di ricavare le matrici

$$\widetilde{A} = V_k^*AV_k, \quad \widetilde{B} = W_k^*BW_k, \quad \widetilde{C} = (V_k^*C_1)(W_k^*C_2)^*$$

dell'equazione compressa direttamente dai coefficienti generati durante il processo di tipo Arnoldi e senza bisogno di ulteriori moltiplicazioni per A, B o C. Si veda [29] per ulteriori approfondimenti.

Supponiamo che C_1 abbia rango massimo, allora Q_1 è ottenibile dalla fattorizzazione QR ridotta di $[C_1, A^{-1}C_1] = Q_1R_1$. Supponiamo adesso di aver già costruito $Q_1, \ldots, Q_m$ e $H_{m,m-1}$, partizioniamo Q_m in due blocchi di s colonne ciascuno $Q_m = [Q^{(+)}, Q^{(-)}]$ e calcoliamo la matrice

$$\widetilde{Q} = [AQ^{(+)}, A^{-1}Q^{(-)}].$$

Allora l'ultima colonna a blocchi della matrice $H_{m,m-1}$ è $[h_{1,m}^*\, h_{2,m}^*\, \cdots\, h_{m+1,m}^*]^*$ con i blocchi $2s \times 2s$ ottenuti come

$$h_{j,m} = Q_j^*\widetilde{Q} \qquad j = 1, \ldots, m$$

e $h_{m+1,m}$ e Q_{m+1} sono ottenuti tramite la fattorizzazione QR ridotta

$$Q_{m+1}h_{m+1,m} = \widetilde{Q} - \sum_{i=1}^{m} Q_i h_{i,m}.$$

Similmente si può operare per $\mathcal{EK}(B^*, C_2)$.

Riassumiamo quanto detto nello pseudocodice dell'Algoritmo 3, dove per ragioni di efficienza in termini di memoria usiamo una unica matrice $\widetilde{V}$ per calcolare le varie Q_i. Non memorizziamo inoltre la matrice $H_{k+1,k}$ poiché, per ragioni di semplicità, abbiamo scelto di calcolare direttamente le matrici $\widetilde{A}, \widetilde{B}, \widetilde{C}$.

In generale le matrici V_k (analogamente per le matrici W_k) calcolate come nell'Algoritmo 3 possono diventare di rango numerico non massimo. Questo può accadere sia a causa della presenza di un sottospazio A-invariante, sia per la ridondanza di alcune delle informazioni generate. Nella nostra trattazione precedente avevamo però supposto che V_k avesse colonne ortonormali e dunque in particolare linearmente indipendenti. Occorre pertanto eliminare alcune delle colonne della matrice V_k, ad esempio effettuando una fattorizzazione

Algoritmo 3 Metodo di Krylov Esteso per l'equazione di Sylvester $AX + XB = C_1C_2^*$

procedure LOW-RANK-SYLV-KRYLOV(A, B, C_1, C_2)
  $[V_1, -] = \texttt{qr}([C_1, A^{-1}C_1])$, $[W_1, -] = \texttt{qr}([C_2, (B^*)^{-1}C_2])$
  for $k = 1, 2, \ldots$ **do**
    $\widetilde{A} = V_k^* A V_k$, $\widetilde{B} = W_k^* B W_k$, $\widetilde{C} = (V_k^* C_1)(W_k^* C_2)^*$
    $X_k = \text{BARTELS-STEWART}(\widetilde{A}, \widetilde{B}, \widetilde{C})$
    if converged **then**
      return $\widetilde{X} = V_k X_k W_k^*$
    end if
    Seleziona le ultime $2s$ colonne: $V_k = [V^{(0)}, V^{(+)}, V^{(-)}]$, $W_k = [W^{(0)}, W^{(+)}, W^{(-)}]$
    $\widetilde{V} = [AV^{(+)}, A^{-1}V^{(-)}]$, $\widetilde{W} = [B^*W^{(+)}, (B^*)^{-1}W^{(-)}]$
    for $i = 1, \ldots, k$ **do**
      $H^V = (Q_i^V)^* \widetilde{V}$
      $\widetilde{V} = \widetilde{V} - (Q_i^V)H^V$
      $H^W = (Q_i^W)^* \widetilde{W}$
      $\widetilde{W} = \widetilde{V} - (Q_i^W)H^W$
    end for
    $[\widetilde{V}, -] = \texttt{qr}(\widetilde{V})$, $[\widetilde{W}, -] = \texttt{qr}(\widetilde{W})$
    $V_{k+1} = [V_k, \widetilde{V}]$, $W_{k+1} = [W_k, \widetilde{W}]$
  end for
end procedure

QR di V_k con tecniche di pivoting e poi scartando le colonne relative a elementi diagonali della matrice triangolare che siano dominate da un certo valore soglia. Si vedano [12, 16] per dettagli implementativi.

Il criterio di arresto è solitamente basato sulla norma 2 o di Frobenius del residuo $R = C_1C_2^* - A\widetilde{X} - \widetilde{X}B$. In particolare si può richiedere che le iterazioni si arrestino quando viene verificata la condizione:

$$\|C_1C_2^* - A\widetilde{X} - \widetilde{X}B\| \leq \varepsilon \cdot \|\widetilde{X}\|$$

con ε fissato opportunamente piccolo. Anche se X ha rango basso, in generale R è densa e quindi risulta inefficiente calcolarla esplicitamente al solo fine di verificare la condizione di arresto. Si vedano [29, 17, 30] per tecniche con cui verificare tale condizione in maniera efficiente.

Alcuni ultimi dettagli:

- Per questioni di efficienza e stabilità numerica, nell'Algoritmo 3 invece di moltiplicare per A^{-1} e $(B^*)^{-1}$ si risolvono sistemi lineari. A tale scopo viene calcolata una fattorizzazione LU (sparsa) di A e B una sola volta all'inizio dell'algoritmo.
- Una volta che l'esecuzione è terminata, si potrebbe eseguire una ulteriore compressione della soluzione tramite la TSVD di $\widetilde{X}$.

2.5 Metodi ADI

L'iterazione *Alternating-Direction-Implicit* (ADI) è stata introdotta per la prima volta nel 1955 in [25] e proposta per risolvere equazioni di Sylvester di grandi dimensioni da Ellner e Wachspress in [14]. Nella sua formulazione originaria il metodo ADI è applicabile a

una matrice densa X, ma in pratica vengono usate versioni basate su fattorizzazioni per risparmiare memoria. Nel seguito della presentazione di questo metodo, assumeremo per semplicità di esposizione che A e B abbiano autovalori con parte reale positiva. In realtà, come vedremo in seguito, il metodo si applica in casi più generali.

Possiamo riscrivere l'equazione (2.1) come:

$$(qI + A)X(qI + B) - (qI - A)X(qI - B) = 2qC, \qquad q \neq 0. \tag{2.10}$$

Se $q > 0$, $(qI + A)$ e $(qI + B)$ sono invertibili e possiamo moltiplicare per le loro inverse ottenendo:

$$X - (qI + A)^{-1}(qI - A)X(qI - B)(qI + B)^{-1} = 2q(qI + A)^{-1}C(qI + B)^{-1}. \tag{2.11}$$

Posto $\mathcal{A} = (qI + A)^{-1}(qI - A)$, $\mathcal{B} = (qI - B)(qI + B)^{-1}$ e $\mathcal{C} = 2q(qI + A)^{-1}C(qI + B)^{-1}$, l'equazione (2.11) può essere riscritta come $X - \mathcal{A}X\mathcal{B} = \mathcal{C}$, nota anche come equazione di Stein. Osserviamo inoltre che se A ha autovalori λ con parte reale positiva e $q > 0$, gli autovalori di $\mathcal{A} = (qI + A)^{-1}(qI - A)$ sono dati da $(q - \lambda)/(q + \lambda)$ e quindi hanno valore assoluto minore di 1, cioe $\rho(\mathcal{A}) < 1$. Con un ragionamento del tutto analogo possiamo dire che $\rho(\mathcal{B}) < 1$. Pertanto la serie $X = \sum_{k=0}^{\infty} \mathcal{A}^k \mathcal{C} \mathcal{B}^k$ è convergente ed è una soluzione dell'equazione di Stein (2.11). Queste considerazioni portano in maniera naturale alla seguente sequenza di approssimazioni successive:

$$X_0 = \mathcal{C}, \qquad X_{k+1} = \mathcal{C} + \mathcal{A}X_k\mathcal{B}. \tag{2.12}$$

Tale approccio può essere generalizzato usando rispettivamente due parametri $p, q > 0$ per A e B, portando a un'equazione della forma

$$X - \mathcal{A}(p, q)X\mathcal{B}(p, q) = \mathcal{C}(p, q)\,,$$

dove $\mathcal{A}(p, q) = (pI + A)^{-1}(qI - A)$, $\mathcal{B}(p, q) = (pI - B)(qI + B)^{-1}$ e $\mathcal{C}(p, q) = (p + q)(pI + A)^{-1}C(qI + B)^{-1}$. Per poter eseguire un'iterazione del tipo della precedente, occorre scegliere i parametri p e q in maniera che il raggio spettrale di $\mathcal{A}(p, q)$ e $\mathcal{B}(p, q)$ sia minore di 1. Per migliorare la velocità di convergenza inoltre è conveniente scegliere i parametri in maniera tale che tale raggio sia minimizzato: se ad esempio A, B sono entrambe simmetriche con spettro contenuto rispettivamente in $[a, b]$ e $[c, d]$, questo corrisponde a risolvere il problema di minimax ADI

$$\min_{p,q>0} \max_{s\in[a,b],\, t\in[c,d]} \left| \frac{(q - s)(p - t)}{(q + s)(q + t)} \right|.$$

Una ulteriore generalizzazione si ottiene scegliendo diversi p, q ad ogni interazione, ottenendo una successione di parametri $\{(p_j, q_j)\}$. Tipicamente, essendo il problema di minimizzazione troppo costoso da risolvere, un numero K di parametri viene scelto a priori e ripetuto ciclicamente. Il fattore di convergenza di K iterazioni accorpate diventa:

$$\prod_{j=1}^{K} \max_{s\in[a,b],\, t\in[c,d]} \left| \frac{(q_j - s)(p_j - t)}{(q_j + s)(q_j + t)} \right|.$$

Seguendo un'idea sviluppata in precedenza per l'equazione di Lyapunov, in [5] è stata proposta una versione fattorizzata del metodo ADI, che permette di risparmiare memoria fornendo una soluzione approssimata di rango basso nel caso anche $C = C_1C_2^*$ sia di rango basso. Approfondiremo questo nel caso dell'equazione di Lyapunov nella Sezione 2.6, dato che i cambiamenti che occorre fare al metodo sono sostanzialmente tecnici e che

questo metodo non fornisce sostanziali miglioramenti in termini di efficienza rispetto al metodo proiettivo con spazi di Krylov Razionali (ma anche rispetto al caso di spazi di Krylov Estesi). Per un approfondito confronto teorico tra i due metodi si vedano [2, 13], accenniamo soltanto che, anche nel caso di poli ottimali, il metodo ADI non fornisce risultati molto migliori rispetto ai metodi di Krylov Razionali, mentre nel caso di poli non ottimali i risultati sono notevolmente peggiori.

2.6 Un caso particolare: l'equazione di Lyapunov

L'equazione di Lyapunov è un caso particolare dell'equazione di Sylvester in cui $B = A^*$ e $C = C^*$ è hermitiana:

$$AX + XA^* = C. \tag{2.13}$$

L'equazione (2.13) viene in realtà chiamata equazione di Lyapunov *continua*, per distinguerla dall'equazione di Lyapunov *discreta*. Queste equazioni sono particolarmente importanti in teoria del controllo e nell'analisi di sistemi dinamici lineari rispettivamente continui e discreti.

Per il Teorema 2.1, la soluzione dell'equazione (2.13) esiste ed è unica se e solo se $\lambda_i + \overline{\lambda_j} \neq 0$ per tutti gli autovalori λ_i, λ_j di A. Ad esempio, se A è stabile (cioè tutti gli autovalori hanno parte reale negativa) tale condizione è assicurata e dunque la soluzione esiste ed è unica. Si noti che la stabilità di A è una proprietà spesso necessaria nelle applicazioni alla teoria del controllo, pertanto non è considerata come una grande restrizione per la risoluzione dell'equazione di Lyapunov.

Si può inoltre mostrare che se A è stabile e $C \prec 0$ ($C \preceq 0$) allora $X \succ 0$ ($X \succeq 0$); in questo caso si parla equazione di Lyapunov *stabile*.

Si osservi inoltre che l'equazione di Lyapunov è simmetrica, cioè se X risolve l'equazione (2.13) anche X^* la risolve. Pertanto, sotto ipotesi di esistenza e unicità, la soluzione è hermitiana.

2.6.1 Metodi numerici

Il metodo standard per la risoluzione dell'equazione (2.13) quando A ha taglia ridotta non presenta sostanziali differenze rispetto al caso dell'equazione di Sylvester. Infatti essendo $B = A^*$ il costo della riduzione alla forma normale di Schur viene dimezzato nell'Algoritmo 1 (di Bartels-Stewart). Una implementazione in MATLAB nel caso complesso è fornita nel Listato 2.3.

Analogamente a quanto discusso per l'equazione di Sylvester, quando A ha grandi dimensioni ci si focalizza sulla ricerca di approssimazioni che siano numericamente interessanti e permettano di risparmiare memoria. Nel caso del problema stabile ad esempio, si ricerca una approssimazione di rango basso della soluzione della forma $\widetilde{X} = ZZ^*$, in modo che soltanto la matrice Z venga calcolata e memorizzata (con Z che ha un numero di colonne molto minori rispetto a m, n). Accenneremo a questi metodi nei seguenti paragrafi fornendo inoltre riferimenti per eventuali approfondimenti. Nel seguito di questa sezione, supporremo che C ammetta una fattorizzazione di rango basso $C = -C_1C_1^*$, $C_1 \in \mathbb{C}^{n\times s}$.

Metodi Proiettivi

Come nel caso dell'equazione di Sylvester possiamo ottenere un metodo proiettivo imponendo, ad esempio, la condizione di Galerkin sul residuo rispetto a un qualche spazio di approssimazione. In particolare dall'equazione (2.9) con $k = j$, $\mathcal{V} = \mathcal{W}$, $V_k = W_j$ e $C_1 = C_2$

Listato 2.3: *Implementazione dell'Algoritmo di Bartels-Stewart per l'equazione di Lyapunov nel caso complesso.*

```
function X = lyap_bartels_stewart(A,C)
% X = LYAP_BARTELS_STEWART(A,C) risolve l'equazione di Sylvester AX + XA* =
    C usando l'algoritmo di Bartels-Stewart basato sulla decomposizione di
    Schur complessa
% A,C: matrici dei coefficienti
% X: soluzione
[n,~] = size(C);
[U,R] = schur(A','complex');
C = U'*C*U;
X = zeros(n,n);
X(1,:) = C(1,:)/(conj(R(1,1))*eye(n) + R);
for k = 2:n
    v = C(k,:) - R(1:k-1,k)'*X(1:k-1,:);
    X(k,:) = v/(R + conj(R(k,k))*eye(n));
end
X = U*X*U';
```

otteniamo l'equazione di Lyapunov ridotta:

$$V_k^* A V_k Y_k + Y_k V_k^* A^* V_k = -V_k^* C_1 (V_k^* C_1)^* \tag{2.14}$$

che se risolta permette di ottenere la soluzione approssimata $X_k = V_k Y_k V_k^*$.

Nel caso del problema stablie, dato che Y_k è semidefinita positiva e di rango numerico basso, si può calcolarne una TSVD del tipo $Y_k = LL^*$ e memorizzare soltanto il fattore $Z_k = V_k L$. Ancora una volta, occorre assumere che valgano le ipotesi del Teorema 2.1 per assicurare la risolubilità di (2.14). Queste sono assicurate nel caso in cui $V_k^* A V_k$ sia stabile, cosa garantita se ad esempio A è assunta definita negativa. Queste ipotesi non sono necessarie e sono una restrizione, dato che l'equazione (2.13) ammette soluzione unica anche assumendo soltanto che A sia stabile e non necessariamente definita negativa. Tuttavia, nella pratica i metodi proiettivi funzionano anche senza questa ulteriore assunzione. Si vedano [24, 30] per una trattazione più esaustiva.

In [29] è stato proposto un metodo particolarmente efficiente basato sull'uso di spazi di Krylov estesi a blocchi $\mathcal{EK}_k^{\square}(A, C_1)$. Tale metodo si comporta bene anche se confrontato con metodi ADI o di Arnoldi, sia dal punto di vista teorico che sperimentale.

Metodo ADI

Nel caso dell'equazione di Lyapunov il metodo ADI di cui nella Sezione 2.5 si semplifica notevolmente, portando alla seguente successione X_k ottenibile in due passi:

$$\begin{aligned} X_0 &= 0 \\ (A + s_j I) X_{j-\frac{1}{2}} &= C_1 C_1^* - X_{j-1}(A^* - s_j I) \\ (A + s_j I) X_j &= C_1 C_1^* - (X_{j-\frac{1}{2}})^* (A^* - s_j I), \quad j = 1, \dots, k \end{aligned} \tag{2.15}$$

con gli shift $\{s_1, s_2, \dots\}$ che sono complessi (in generale), adoperati ciclicamente e non sono autovalori di A. Nel caso le matrici A, C siano reali, assumeremo inoltre che i parametri s_j siano reali o appaiano in coppie complesse coniugate. Si può mostrare che per ogni j la matrice X_j è hermitiana, anche se $X_{j-\frac{1}{2}}$ potrebbe non esserlo. Inoltre può essere dimostrato che se X_j è reale e vengono adoperati due parametri consecutivi s_{j+1}, s_{j+2} complessi coniugati, allora X_{j+2} è reale.

Come già osservato nella sezione 2.5 l'efficienza del metodo ADI, in termini di numero di iterazioni svolte, dipende fortemente dalla scelta dei parametri di shift. In pratica, occorre determinare un numero di passi N e dei parametri $s_1, \ldots, s_N$ in maniera da rendere piccola la quantità:

$$\prod_{i=1}^{N} \max_{x \in \mathcal{W}} \left| \frac{x - s_i}{x + s_i} \right|$$

dove $\mathcal{W} \in \mathbb{C}$ è un sottoinsieme del piano complesso contenente lo spettro di A o lo spettro stesso, mentre per $i = 1, \ldots, N$ o s_i è reale oppure esiste $j \in \{1, \ldots, N\}$ tale che $\overline{s}_j = s_i$. Tale problema è stato un argomento di ricerca largamente trattato, tuttavia una soluzione esplicita è nota soltanto in casi molto particolari e spesso, quando è nota, è costosa dal punto di vista computazionale. Soltanto con l'approccio euristico in [26] il calcolo di parametri *sub-ottimali* è diventato un problema numericamente affrontabile. L'idea è quella di cercare la soluzione di un problema semplificato, cioè:

$$\min_{p_1, \ldots, p_N \in \mathcal{P}} \prod_{i=1}^{N} \max_{x \in \mathcal{P}} \left| \frac{x - p_i}{x + p_i} \right|$$

dove $\mathcal{P}$ è un insieme finito di approssimazioni degli autovalori estremi di A. Tali approssimazioni possono essere ottenute eseguendo un certo numero di iterazioni del metodo di Arnoldi di A e A^{-1} a partire da un vettore casuale r, ottenendo:

$$\begin{aligned} AV_m &= V_m H_m + h_{m+1,m} v_m e_m^* \\ A^{-1} W_m &= W_m K_m + k_{m+1,m} w_m e_m^* \end{aligned}$$

e una buona approssimazione degli autovalori estremi di A è data dagli autovalori di H_m e dagli inversi degli autovalori di W_m (detti anche *valori di Ritz*). Il calcolo di tali autovalori non è eccessivamente costoso essendo H_m, W_m matrici di Hessenberg. Sfortunatamente tale idea funziona soltanto se $\mathcal{P}$ è un sottoinsieme del semipiano complesso sinistro. Un possibile pseudocodice per tale metodo è dato nell'Algoritmo 4, dove dato $\mathcal{P} = \{p_1, \ldots, p_k\}$ definiamo la funzione $s_{\mathcal{P}}(t)$ come

$$s_{\mathcal{P}}(t) = \left| \frac{(t - p_1) \cdots (t - p_k)}{(t + p_1) \cdots (t + p_k)} \right|.$$

L'idea chiave per ottenere buone prestazioni anche nel caso di matrici di taglia molto grande è quella di mantenere le approssimazioni in forma fattorizzata durante le iterazioni. Quest'idea è stata discussa e implementata da Penzl in [26], portando a un algoritmo *LR-ADI* (Low Rank ADI) che calcola soltanto i fattori Z_j della soluzione approssimata $X_j = Z_j Z_j^*$. Assumendo che A sia *c-stabile*, cioè con autovalori tutti con parte reale positiva o tutti con parte reale negativa, e che i parametri di shift abbiano tutti parte reale negativa, i fattori Z_j possono essere calcolati come:

$$\begin{aligned} Z_1 &= \sqrt{-2\mathrm{Re}(s_1)}(A + s_1 I)^{-1} C_1 \\ Z_{j+1} &= [(A - s_j I)(A + s_j I)^{-1} Z_j, \sqrt{-2\mathrm{Re}(s_j)}(A + s_j I)^{-1} C_1]. \end{aligned} \tag{2.16}$$

Si noti che il numero di colonne di Z_j ad ogni iterazione aumenta di tante colonne quante ne ha C_1, cioè di $\mathrm{rank}(C)$ colonne. L'efficienza di questo metodo è collegata a quello che Penzl chiama il *low-rank phenomenon* della soluzione X, cioè la tendenza dei valori singolari di X a decadere molto velocemente verso la precisione di macchina, rendendo possibile una approssimazione di rango basso della soluzione.

Algoritmo 4 Calcolo dei parametri sub-ottimali per il metodo ADI

procedure ADI-SUBOPTIMAL(A, l_0, k_+, k_-) ▷ OUTPUT: $\mathcal{P}$
  Scegli $r \in \mathbb{R}^n$
  Esegui k_+ passi del metodo di Arnoldi per (A, r) e calcola i valori di Ritz $\mathcal{R}_+$
  Esegui k_- passi del metodo di Arnoldi per (A^{-1}, r) e calcola i valori di Ritz $\mathcal{R}_-$
  $\mathcal{R} = \{\rho_1, \ldots, \rho_{(k_-+k_+)}\} := \mathcal{R}_+ \cup (1/\mathcal{R}_-)$
  if $\mathcal{R} \not\subset \mathbb{C}_-$ **then** STOP
  end if
  Calcola i tale che $\max_{t\in\mathcal{R}} s_{\{\rho_i\}}(t) = \min_{\rho\in\mathcal{R}} \max_{t\in\mathcal{R}} s_{\{\rho\}}(t)$ e inizializza

$$\mathcal{P} := \begin{cases} \{\rho_i\} & \text{se } \rho_i \in \mathbb{R} \\ \{\rho_i, \overline{\rho}_i\} & \text{altrimenti} \end{cases}$$

  while $\mathrm{card}(\mathcal{P}) < l_0$ **do**
    Calcola i tale che $s_{\mathcal{P}}(\rho_i) = \max_{t\in\mathcal{R}} s_{\mathcal{P}}(t)$ e aggiorna

$$\mathcal{P} := \begin{cases} \mathcal{P} \cup \{\rho_i\} & \text{se } \rho_i \in \mathbb{R} \\ \mathcal{P} \cup \{\rho_i, \overline{\rho}_i\} & \text{altrimenti} \end{cases}$$

  end while
end procedure

In [23] Li e White hanno proposto una riscrittura del metodo *LR-ADI* che permette di ridurre notevolmente il costo computazionale. Vengono calcolate delle matrici $\widehat{Z}_j$ tali che $\widehat{Z}_j\widehat{Z}_j^* = Z_jZ_j^* = X_j$. Tali fattori vengono ottenuti tramite:

$$\begin{aligned} \widehat{U}_1 &= \sqrt{-2\mathrm{Re}(s_1)}(A + s_1 I)^{-1}C_1, \quad \widehat{Z}_1 = \widehat{U}_1 \\ \widehat{U}_k &= \frac{\sqrt{-2\mathrm{Re}(s_k)}}{\sqrt{-2\mathrm{Re}(s_{k-1})}}(I - (s_k + \overline{s}_{k-1})(A + s_k I)^{-1})\widehat{U}_{k-1} \\ \widehat{Z}_k &= [\widehat{Z}_{k-1}\, \widehat{U}_k], \quad k = 2, 3, \ldots \end{aligned} \tag{2.17}$$

Tale metodo prende il nome di *CF-ADI (Cholesky Factor ADI)*.

Per concludere, spieghiamo come scegliere il numero di iterazioni del metodo *CF-ADI*. Nella pratica, usando parametri sub-ottimali, il metodo dimostra convergenza superlineare e viene usato come un vero e proprio metodo iterativo, con un criterio di arresto adeguato. Poiché l'aggiornamento $\widehat{U}_k$ diventa piccolo quando viene raggiunta una buona approssimazione, Benner, Li e Penzl in [4] hanno proposto il criterio $\|\widehat{U}_k\|_F / \|\widehat{Z}_k\|_F \leq \varepsilon$ per un valore fissato di ε opportunamente piccolo. Si noti inoltre che è possibile calcolare $\|Z_k\|_F$ iterazione dopo iterazione dato che vale la relazione

$$\|\widehat{Z}_k\|_F^2 = \mathrm{tr}(\widehat{Z}_k^*\widehat{Z}_k) = \mathrm{tr}(\widehat{Z}_{k-1}^*\widehat{Z}_{k-1} + \widehat{U}_k^*\widehat{U}_k) = \|\widehat{Z}_{k-1}\|_F^2 + \|\widehat{U}_k\|_F^2.$$

3. Aggiornamenti di Rango Basso

Supponiamo di avere X_0 soluzione dell'equazione di Sylvester

$$A_0X_0 + X_0B_0 = C_0\,, \tag{3.1}$$

cerchiamo di calcolare una correzione δX tale che $X_0 + \delta X$ risolva l'equazione perturbata

$$(A_0 + \delta A)(X_0 + \delta X) + (X_0 + \delta X)(B_0 + \delta B) = C_0 + \delta C \tag{3.2}$$

dove le perturbazioni $\delta A, \delta B, \delta C$ hanno rango molto più piccolo di $\min\{m, n\}$.

3.1 Algoritmo per il calcolo della correzione

In questa sezione descriveremo un algoritmo per il calcolo della sola correzione δX elaborato in [19] e particolarmente interessante non solo di per sé, ma anche per alcune applicazioni che andremo ad analizzare in seguito.

Sottraendo l'equazione (3.1) dalla (3.2) si ottiene:

$$(A_0 + \delta A)\delta X + \delta X(B_0 + \delta B) = \delta C - \delta AX_0 - X_0\delta B. \tag{3.3}$$

Questa è un'equazione di Sylvester in cui il termine a destra ha sempre rango basso, dominato da $k := \operatorname{rank}(\delta A) + \operatorname{rank}(\delta B) + \operatorname{rank}(\delta C)$. Questo ci permette di risolvere questa equazione con solutori specifici per il caso in cui il termine a destra ha rango basso, come il Metodo degli spazi di Krylov Estesi o il Metodo CF-ADI presentati nel Capitolo 2. Si noti che tale approccio è di scarsa utilità se C_0 ha rango comparabile a quello delle perturbazioni. In tal caso è più efficiente risolvere direttamente la (3.2).

Il nostro approccio può essere riassunto nello pseudocodice dell'Algoritmo 5, nel prosieguo ci concentreremo sul come eseguire ciascuna delle operazioni richieste.

Algoritmo 5 Algoritmo per la risoluzione dell'equazione perturbata (3.2)

```
1: procedure UPDATE-SYLV(A_0, δA, B_0, δB, C_0, δC, X_0)
2:     Calcola U, V tali che δC − δAX_0 − X_0δB = UV*
3:     δX = LOW-RANK-SYLV(A_0 + δA, B_0 + δB, U, V)
4:     return δX
5: end procedure
```

3.1.1 Passo 1: Fattorizzazione di rango basso del membro destro

Supponiamo di avere a disposizione delle fattorizzazioni di rango basso delle perturbazioni

$$\delta A = U_A V_A^*, \quad \delta B = U_B V_B^*, \quad \delta C = U_C V_C^*\,,$$

allora possiamo ottenere una fattorizzazione $\delta C - \delta A X_0 - X_0 \delta B = UV^*$ ponendo

$$U = [U_C, -U_A, -X_0 U_B], \qquad V = [V_C, X_0^* V_A, V_B] \tag{3.4}$$

con U e V che hanno entrambe k colonne.

Il costo computazionale e di conseguenza l'efficienza dei solutori low-rank per l'equazione di Sylvester dipendono fortemente dal rango del termine noto UV^*. Pertanto è consigliabile operare una compressione dei fattori descritti in (3.4) al fine di decrementare il rango. Per fare questo ci viene incontro la TSVD descritta nella Sezione 1.2.2. Calcoliamo inizialmente una decomposizione QR dei fattori $U = Q_U R_U$ e $V = Q_V R_V$ dove $Q_U \in \mathbb{R}^{m\times s}$, $Q_V \in \mathbb{R}^{n\times s}$ hanno colonne ortonormali e $R_U, R_V \in \mathbb{R}^{s\times s}$ sono triangolari superiori. Calcoliamo poi la SVD della matrice $s \times s$ $R_U R_V^*$, mantenendo soltanto i primi $\tilde{s} \leq s$ valori singolari tali che $\sigma_{\tilde{s}+1}/s_1 \leq \tau_\sigma$ per una tolleranza τ_σ scelta a dall'utente. Dette U_R e V_R le matrici contenenti i primi $\tilde{s}$ vettori singolari sinistri e destri rispettivamente e detta $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_{\tilde{s}})$ poniamo:

$$\widetilde{U} := Q_U U_R \sqrt{\Sigma}\,, \qquad \widetilde{V} := Q_V V_R \sqrt{\Sigma}\,.$$

Per il Teorema 1.4 si ha $\frac{\|\widetilde{U}\widetilde{V}^* - UV^*\|_2}{\|UV^*\|_2} \leq \tau_\sigma$ e possiamo pertanto sostituire la fattorizzazione UV^* con $\widetilde{U}\widetilde{V}^*$.

Il calcolo della fattorizzazione finale $\widetilde{U}\widetilde{V}^*$ è dominato dal costo delle due fattorizzazioni QR e della SVD, pertanto il costo totale è un $\mathcal{O}((m+n)s^2 + s^3)$. Nelle nostre ipotesi $s \ll \min\{m, n\}$ e dunque tale metodo appare particolarmente interessante.

3.1.2 Passo 2: Risoluzione dell'equazione per la correzione

L'equazione da risolvere per calcolare la correzione δX è della forma:

$$A\delta X + \delta X B = UV^* \tag{3.5}$$

dove $A = A_0 + \delta A$, $B = B_0 + \delta B$ e U, V hanno $s \ll \min\{m, n\}$ colonne. Per le considerazioni fatte nella Sezione 2.3 risulta ragionevole supporre che la soluzione δX della (3.5) ammetta una approssimazione di rango basso. I solutori più comuni per questo tipo di equazioni sono basati sui metodi descritti nel capitolo precedente, principalmente Metodi di Krylov Estesi e Metodi ADI.

Supponiamo di aver ricavato, tramite uno qualsiasi dei metodi di cui sopra, una approssimazione $\widetilde{\delta X}$ della soluzione tale che

$$\|A\widetilde{\delta X} + \widetilde{\delta X} B - UV^*\|_2 \leq \tau_\delta.$$

Se l'approssimazione $\widetilde{X}_0$ della soluzione dell'equazione di partenza soddisfa

$$\|A\widetilde{X}_0 + \widetilde{X}_0 B - C\|_2 \leq \tau_0$$

allora dalla disuguaglianza triangolare la soluzione $\widetilde{X} = \widetilde{X}_0 + \widetilde{\delta X}$ dell'equazione perturbata verifica:

$$\|(A + \delta A)\widetilde{X} + \widetilde{X}(B + \delta B) - (C + \delta C)\|_2 \leq \tau_0 + \tau_\delta.$$

Pertanto, per evitare di eseguire del lavoro inutile, è consigliabile scegliere la condizione di arresto per l'algoritmo risolutivo per l'equazione (3.5) in maniera tale che τ_δ non sia più piccolo di τ_0.

3.2 Equazioni di Lyapunov Stabili

Consideriamo adesso il caso particolare dell'equazione di Lyapunov stabile $A_0X_0+X_0A_0^* = C_0$, dove $A_0 \in \mathbb{C}^{n\times n}$ è stabile e $C_0 \preceq 0$. Supponiamo inoltre che anche l'equazione perturbata $A(X_0 + \delta X) + (X_0 + \delta X)A^* = C_0 + \delta C$ con $A = A_0 + \delta A$ abbia le stesse proprietà, con quindi $X_0, (X_0 + \delta X) \succeq 0$. In generale, questo non assicura che l'equazione per la correzione

$$A\delta X + \delta X A^* = \delta C - \delta A X_0 - X_0 \delta A^* \tag{3.6}$$

abbia le stesse proprietà poiché il termine a destra potrebbe essere non definito. Pertanto non possono essere applicati tutti quei solutori low-rank per equazioni di Lyapunov stabili. Questo problema può essere risolto attraverso un partizionamento.

Supponiamo di avere delle fattorizzazioni di rango basso del tipo $\delta A = U_A V_A^*$, $\delta C = U_C \Sigma_C U_C^*$ con Σ_C hermitiana. Allora possiamo scrivere il membro destro come:

$$\begin{aligned} &\delta C - \delta A X_0 - X_0 \delta A^* = \widetilde{U}\Sigma\widetilde{U}^* \\ &\widetilde{U} = [U_C, U_A, X_0 V_A], \qquad \Sigma = \begin{bmatrix} \Sigma_C & 0 & 0 \\ 0 & 0 & -I \\ 0 & -I & 0 \end{bmatrix}. \end{aligned} \tag{3.7}$$

Calcoliamo poi una fattorizzazione QR di $\widetilde{U} = Q_{\widetilde{U}} R_{\widetilde{U}}$ e una decomposizione spettrale di $R_{\widetilde{U}} \Sigma R_{\widetilde{U}}^*$ del tipo:

$$R_{\widetilde{U}} \Sigma R_{\widetilde{U}}^* = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} D_1 & 0 \\ 0 & -D_2 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}$$

con D_1, D_2 matrici diagonali con elementi diagonali positivi. Una volta posto $U_1 = Q_{\widetilde{U}} Q_1 \sqrt{D_1}, U_2 = Q_{\widetilde{U}} Q_2 \sqrt{D_2}$ possiamo riscrivere $R_{\widetilde{U}} \Sigma R_{\widetilde{U}}^* = U_1 U_1^* - U_2 U_2^*$. Pertanto basta risolvere le due equazioni di Lyapunov stabili

$$A\delta X_1 + \delta X_1 A^* = -U_1 U_1^*, \qquad A\delta X_2 + \delta X_2 A^* = -U_2 U_2^*$$

e si ricava $\delta X = \delta X_2 - \delta X_1$.

4. Applicazioni all'Equazione di Riccati Algebrica

Le Equazioni di Riccati Algebriche sono un argomento largamente trattato negli ultimi anni in letteratura, soprattutto per la continua domanda, proveniente dalle applicazioni, di algoritmi efficienti per la loro risoluzione. Sono inoltre di nostro interesse poichè sono un eccellente esempio in cui vi è necessità di risolvere equazioni di Lyapunov perturbate in sequenza. Per una trattazione più approfondita delle Equazioni di Riccati Algebriche e delle loro proprietà numeriche si veda [10], noi ci limiteremo a richiamare soltanto quanto strettamente necessario alla realizzazione della sperimentazione numerica di nostro interesse.

Una equazione di Riccati algebrica non simmetrica (*Nonsymmetric Algebraic Riccati Equation, NARE*) è un'equazione matriciale della forma

$$XA + DX - XBX = C \tag{4.1}$$

dove $X \in \mathbb{C}^{m\times n}$ è l'incognita e dove i coefficienti sono $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$, $C \in \mathbb{C}^{m\times n}$ e $D \in \mathbb{C}^{m\times m}$.

Una equazione di Riccati algebrica continua (*Continuous-Time Algebraic Riccati Equation, CARE*) è una NARE in cui le matrici dei coefficienti sono quadrate, cioè $m = n$, e tali che $C = C^*, B = B^*$ e $D = A^*$. Una CARE può essere quindi scritta come:

$$XA + A^*X - XBX = C. \tag{4.2}$$

L'equazione è simmetrica e la soluzione di interesse è X hermitiana.

Assumiano inoltre che $B \in \mathbb{C}^{n\times n}$ sia hermitiana semidefinita positiva e che $C \in \mathbb{C}^{n\times n}$ sia hermitiana semidefinita negativa. Inoltre consideriamo il caso in cui la coppia (A, B) è stabilizzabile, cioè tale che esista $X_0 \in \mathbb{C}^{n\times n}$ tale che $A - BX_0$ è stabile. Sotto tali ipotesi, si può mostrare che esiste un'unica soluzione $X_+ \in \mathbb{C}^{n\times n}$ hermitiana semidefinita positiva di (4.2) tale che $A - BX_+$ è stabile [20]. Questa è detta la soluzione *stabilizzante*.

4.1 Metodo di Newton

In [18] Kleinman ha proposto l'uso del metodo di Newton per la risoluzione di (4.2). Il metodo di Newton classico è applicato per la risoluzione di equazioni scalari del tipo $f(x) = 0$, con f differenziabile con continuità almeno in un intorno della soluzione $\alpha \in \mathbb{C}$, e produce una successione definita ricorsivamente da $x_{k+1} = x_k - f(x_k)/f'(x_k)$. Se x_0 è opportunamente scelto (cioè è abbastanza vicino alla soluzione α), la convergenza della successione $\{x_k\}_{k\in\mathbb{N}}$ a α è quadratica se α è uno zero semplice di f, lineare se $f'(\alpha) = 0$ ma f' è non nulla in un intorno di α.

Un approccio analogo può essere replicato per la risoluzione di equazioni del tipo $F(X) = 0$ dove $F : \mathbb{E} \to \mathbb{E}$ è un operatore differenziabile di uno spazio di Banach. In tal caso la successione è definita da

$$X_{k+1} = X_k - \left(F'_{X_k}\right)^{-1} [F(X_k)], \qquad X_0 \in \mathbb{E} \tag{4.3}$$

dove F'_X è la derivata di Fréchet di F nel punto X.

Nel nostro caso abbiamo $F(X) = XA + A^*X - XBX - C$ e dunque l'azione di F'_X su una qualsiasi matrice H è della forma

$$F'_X[H] = HA + A^*H - HBX - XBH = (A^* - XB)H + H(A - BX).$$

Poiché l'iterazione (4.3) può essere riscritta come $F'_{X_k}[X_{k+1}] = F'_{X_k}[X_k] - F(X_k)$, per ottenere X_{k+1} occorre risolvere l'equazione matriciale

$$(A^* - X_kB)X_{k+1} + X_{k+1}(A - BX_k) = C - X_kBX_k \tag{4.4}$$

o alternativamente detto $H_k = X_{k+1} - X_k$ si risolve

$$(A^* - X_kB)H_k + H_k(A - BX_k) = -F(X_k) \tag{4.5}$$

e poi si calcola $X_{k+1} = X_k + H_k$. Si osservi che in entrambi i casi se X_0 è hermitiana allora $\{X_k\}_k$ è una successione di matrici hermitiane e ad ogni passo si risolve un'equazione di Lyapunov.

Sotto l'ipotesi che un'unica soluzione stabilizzante X_+ della CARE (4.2) esista, il metodo di Newton può essere usato per calcolarla. In particolare, sotto ipotesi standard, ogni scelta di X_0 stabilizzante (cioè tale che $A - BX_0$ sia stabile) permette di generare una successione $\{X_k\}_k$ convergente a X_+ in maniera monotona. Questi risultati sono sintetizzati nel seguente teorema, per la cui dimostrazione si rimanda a [22].

Teorema 4.1. *Supponiamo che $B \succeq 0$, $C \preceq 0$ e che le coppie (A, B) e $(A^*, -C)$ siano stabilizzabili. Se X_0 è una matrice hermitiana tale che $A - BX_0$ è stabile, allora il metodo di Newton genera una successione di matrici hermitiane $\{X_k\}_{k\geq 0}$ tale che:*

(i) $A - BX_k$ *è stabile per ogni* k

(ii) $X_1 \succeq X_2 \succeq \cdots \succeq X_+$

(iii) $\lim_{k\to\infty} X_k = X_+$

(iv) per ogni norma matriciale $\|\cdot\|$ *esiste una costante* $c > 0$ *tale che*

$$\|X_{k+1} - X_+\| \leq c \cdot \|X_k - X_+\|^2 \qquad k \geq 0.$$

Se A è già stabile, una scelta naturale è $X_0 = 0$. Nel caso generale il problema di determinare X_0 non è banale, si veda [10] per una trattazione approfondita.

4.1.1 Approssimazione successive di rango basso

In alcune applicazioni, il coefficiente B ha rango basso e può essere espresso come $B = B_U B_U^*$ dove B_U ha poche colonne. In tal caso, esplicitando i prodotti dalla (4.4) si ottiene:

$$A^*X_{k+1} + X_{k+1}A - C = X_kBX_{k+1} + X_{k+1}BX_k - X_kBX_k$$

e dunque nella (4.5) possiamo scrivere il membro destro come:

$$\begin{aligned} -F(X_k) &= X_k B X_k - X_k A - A^* X_k + C = \\ &= X_k B X_k - X_{k-1} B X_k - X_k B X_{k-1} + X_{k-1} B X_{k-1} = \\ &= (X_k - X_{k-1}) B (X_k - X_{k-1}). \end{aligned}$$

Pertanto, dopo il calcolo (eventualmente costoso) di X_1 possiamo calcolare gli incrementi $H_k := X_{k+1} - X_k$ attraverso la risoluzione di

$$(A^* - X_k B) H_k + H_k (A - B X_k) = H_{k-1} B H_{k-1}. \tag{4.6}$$

A tale scopo possiamo usare uno dei solutori per l'Equazione di Lyapunov presentati nel Capitolo 2, sfruttando il fatto che il membro destro ammette una fattorizzazione di rango basso $H_{k-1} B H_{k-1} = H_{k-1} B_U B_U^* H_{k-1}$. Si noti che, al contrario della situazione più generale descritta nella Sezione 3.2, in questo caso il membro destro è sempre definito positivo e pertanto non è necessario il partizionamento.

Tale approccio è riassunto nell'Algoritmo 6.

Algoritmo 6 Metodo di Newton per approssimazioni successive di rango basso per la risoluzione dell'equazione $XA + A^*X - XB_U B_U^* X = C$

procedure LOW-RANK-NEWTON-RICCATI(A, BU, C, X_0)
  $A_0 = A^* - X_0 B_U B_U^*$
  $C_0 = C - X_0 B_U B_U^* X_0$
  $X_1 =$ SOLVE-LYAP(A_0, C_0) ▷ Solutore qualsiasi per l'equazione di Lyapunov
  $H_0 = X_1 - X_0$
  for $k = 1, 2, \ldots$ **do**
    $A_k = A^* - X_k B_U B_U^*$
    $C_U = H_{k-1} B_U$
    $H_k =$ LOW-RANK-SOLVE-LYAP(A_k, C_U)
    $X_{k+1} = X_k + H_k$
    if $\|H_{k+1}\| / \|X_k + 1\| < tol$ **then**
      return X_{k+1}
    end if
  end for
end procedure

5. Sperimentazione numerica

Riportiamo le metodologie e i risultati della sperimentazione numerica effettuata per confrontare i metodi proposti in precedenza. La sperimentazione è organizzata come segue. Inizialmente viene trattato il caso dell'equazione di Sylvester, con il problema della divergenza del metodo basato su Spazi di Krylov Estesi a blocchi a causa di errori di round-off. Vengono poi confrontati dal punto di vista della complessità in tempo e della precisione raggiunta gli algoritmi proposti, vale a dire l'algoritmo di Bartels-Stewart e i solutori low-rank basati su Spazi di Krylov. Passando poi all'equazione di Lyapunov, viene aggiunto al confronto anche il metodo ADI. In secondo luogo vengono illustrati i risultati sperimentali relativi agli algoritmi di update per l'equazione di Sylvester e per l'equazione di Lyapunov stabile, con i vari metodi confrontati ad ogni passo di aggiornamento in termini di precisione raggiunta e tempo di esecuzione. La sperimentazione si conclude con il confronto delle possibili implementazioni dell'algoritmo di Newton per la risoluzione di CARE. I risultati della sperimentazione numerica si rivelano consistenti con quanto ci si aspettava teoricamente: gli algoritmi low-rank si dimostrano più efficienti in termini di tempo di esecuzione, ma raggiungono una precisione inferiore rispetto a solutori basati sull'algoritmo di Bartels-Stewart.

Per effettuare i test è stato usato MATLAB R2019b su PC con processore Intel® Core™ i7 Q720 1.60GHz, 8 GB di RAM e Windows 10 Home (64 bit).

5.1 Solutori low-rank

Consideriamo l'equazione di Sylvester

$$AX + XB = C$$

dove $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{m\times m}$, $C \in \mathbb{C}^{n\times m}$ e supponiamo che C ammetta una fattorizzazione di rango basso del tipo $C = C_1 C_2^*$ con $C_1 \in \mathbb{C}^{n\times s}$, $C_2 \in \mathbb{C}^{m\times s}$ e $s \ll \min\{m, n\}$. Una ipotesi che assicura l'applicabilità dei metodi proposti, in particolare dei metodi basati su spazi di Krylov, è che A, B siano entrambe stabili. Considereremo dunque, per semplicità, il caso in cui A e B sono matrici simmetriche definite negative. Inoltre nel prosieguo, data $\widetilde{X}$ approssimazione della soluzione della suddetta equazione di Sylvester, indicheremo con errore relativo commesso (o per semplicità talvolta anche solo errore) la quantità

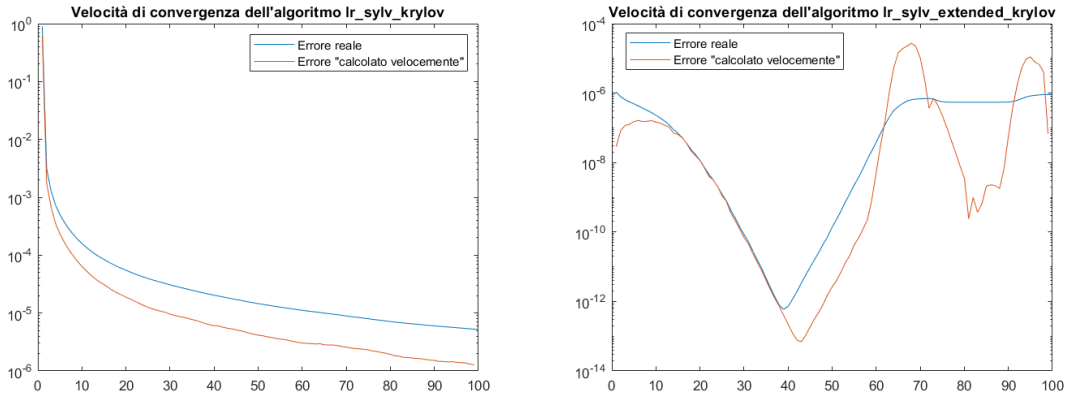
$$\texttt{err} = \frac{\|A\widetilde{X} + \widetilde{X}B - C\|_F}{\|\widetilde{X}\|_F \cdot (\|A\|_F + \|B\|_F)}.$$

5.1.1 Metodi di Krylov: condizione di arresto

Come già sottolineato nella Sezione 2.4.1, il calcolo diretto del residuo $R = C_1 C_2^* - A\widetilde{X} - \widetilde{X}B$ al solo fine di verificare la condizione di arresto è inefficiente, dato che R è in generale

una matrice densa. In [29, 17, 30] sono presentate delle tecniche che permettono di calcolare l'errore in maniera più efficiente, anche se operando in aritmetica floating point ci possiamo aspettare per errori di round off le due quantità calcolate non siano esattamente coincidenti. Si vedano i Listati B.3 e B.4 per una implementazione di tali tecniche.

Figura 5.1: *Plot in scala semilogaritmica di errore reale e errore "calcolato velocemente" per i metodi di Krylov. Parametri* $m = n = 1000$, $s = 1$.



(a) *Spazi di Krylov Standard*

(b) *Spazi di Krylov Estesi*

Osservando la Figura 5.1, si può notare come la velocità di convergenza del Metodo di Krylov Standard sia piuttosto bassa dopo le prime iterazioni. Il metodo basato su Spazi di Krylov Estesi ha invece una velocità di convergenza notevolmente migliore, salvo poi incontrare un "rimbalzo" dell'errore che porta il metodo a divergere. Tale problema di divergenza è dovuto a errori di round off ed è un fenomeno tipico delle iterazioni che sono instabili. Il vettore degli errori converge a zero fino a quando sta dentro un opportuno sottospazio, in questo caso l'ortogonale allo Spazio di Krylov Esteso (a blocchi). Nel momento in cui esce dal sottospazio allora diverge. Accade quindi che in teoria la successione dovrebbe stare dentro il sottospazio e quindi convergere, ma per gli errori di round off a un certo punto un elemento della successione cade fuori dal sottospazio (anche di poco) e quindi diverge. Per ovviare parzialmente a questo problema possiamo mettere un controllo per fermare le iterazioni quando si verifica un aumento dell'errore non nelle prime iterazioni. Il nuovo andamento dell'errore è mostrato in Figura 5.2 e si può notare che il problema è stato risolto soltanto parzialmente, dato che l'errore "calcolato velocemente" sembra aumentare qualche iterazione dopo rispetto a quello reale. Questo causa sia una perdita precisione di qualche ordine di grandezza, sia una peggiore efficienza in termini di tempo calcolo dato che vengono svolte diverse iterazioni in più del dovuto.

5.1.2 Confronto tra metodi per l'equazione di Sylvester

Le Tabelle 5.1,5.2,5.3 e 5.4 comparano i metodi proposti per la risoluzione dell'equazione di Sylvester con la funzione `sylvester` di MATLAB al variare dei parametri m, n e s. Gli algoritmi sono stati confrontati, eseguendo test su N istanze differenti e facendo una media dei risultati ottenuti, in termini di:

- `it`: numero medio di iterazioni per istanza (se applicabile);
- `time`: tempo medio di esecuzione di un istanza, in secondi;
- `it_time`: tempo medio di esecuzione di un'iterazione, in secondi (se applicabile);

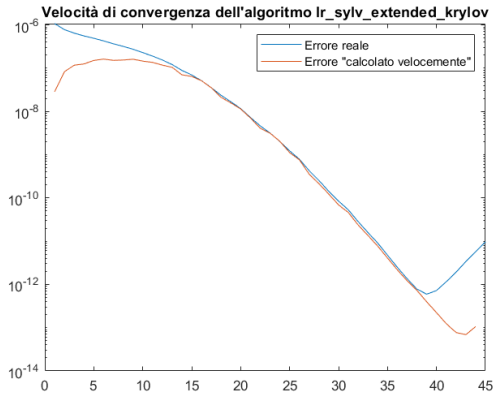


Figura 5.2: *Plot in scala semilogaritmica di errore reale e errore "calcolato velocemente" per il Metodo di Krylov Esteso con condizione di arresto modificata. Stessi parametri del test precedente.*

- `err`: errore relativo medio, calcolato come $\|AX+XB-C\|_F/(\|X\|_F\cdot(\|A\|_F+\|B\|_F))$.

Tabella 5.1: $n = m = 100$, $A, B \prec 0$, $s = 1$, `itmax` $= 100$, `tol` $= 10^{-13}$, $N = 20$

Algoritmo	`it`	`time`	`it_time`	`err`
`sylvester`		0.007		3.784×10^{-17}
Bartels-Stewart		0.020		3.784×10^{-17}
Krylov Standard	100	1.102	0.011	8.067×10^{-11}
Krylov Esteso	21.8	0.052	0.002	1.192×10^{-9}

L'80% delle istanze del Metodo di Krylov Esteso sono state arrestate perché l'errore ha iniziato ad aumentare.

Tabella 5.2: $n = m = 1000$, $A, B \prec 0$, $s = 1$, `itmax` $= 100$, `tol` $= 10^{-13}$, $N = 10$

Algoritmo	`it`	`time`	`it_time`	`err`
`sylvester`		1.311		3.176×10^{-17}
Bartels-Stewart		11.283		3.176×10^{-17}
Krylov Standard	100	2.653	0.026	4.998×10^{-6}
Krylov Esteso	40.5	1.615	0.040	5.158×10^{-12}

Il 10% delle istanze del Metodo di Krylov Esteso sono state arrestate perché l'errore ha iniziato ad aumentare.

Tabella 5.3: $n = m = 1000$, $A, B \prec 0$, $s = 5$, `itmax` $= 100$, `tol` $= 10^{-13}$, $N = 10$

Algoritmo	`it`	`time`	`it_time`	`err`
`sylvester`		1.430		3.580×10^{-17}
Bartels-Stewart		12.135		3.580×10^{-17}
Krylov Standard	100	82.349	0.823	3.400×10^{-6}
Krylov Esteso	23.9	4.031	0.167	2.667×10^{-11}

Il 20% delle istanze del Metodo di Krylov Esteso sono state arrestate perché l'errore ha iniziato ad aumentare.

Tabella 5.4: $n = 2000, m = 3000$, $A, B \prec 0$, $s = 1$, `itmax` $= 100$, `tol` $= 10^{-13}$, $N = 5$

Algoritmo	`it`	`time`	`it_time`	`err`
`sylvester`		19.985		2.550×10^{-17}
Krylov Esteso	29.4	19.790	0.665	3.433×10^{-12}

Nessuna istanza del Metodo di Krylov Esteso è stata arrestata perché l'errore ha iniziato ad aumentare.

Si osserva che per valori "piccoli" di n e m (nel nostro caso per $n = m = 100$) il metodo più efficiente tra quelli proposti è l'algoritmo di Bartels-Stewart, che risulta essere il migliore sia dal punto di vista del tempo di esecuzione che per la precisione raggiunta. La precisione raggiunta è la stessa della funzione `sylvester` di MATLAB, anche se il tempo medio di risoluzione di un'istanza è circa 3 volte maggiore.

Al crescere di n e m si osserva come l'algoritmo di Bartels-Stewart, pur rimanendo il migliore per precisione raggiunta, perde notevolmente efficienza per quanto riguarda la complessità in tempo. L'algoritmo basato sugli Spazi di Krylov Standard, avendo una velocità di convergenza molto bassa dopo le prime iterazioni (come mostrato in Figura 5.1a), non riesce a convergere alla precisione voluta entro il numero massimo di iterazioni `itmax` $= 100$. L'algoritmo basato su Spazi di Krylov Estesi invece riesce a raggiungere una buona precisione in un tempo minore rispetto all'algoritmo di Bartels-Stewart.

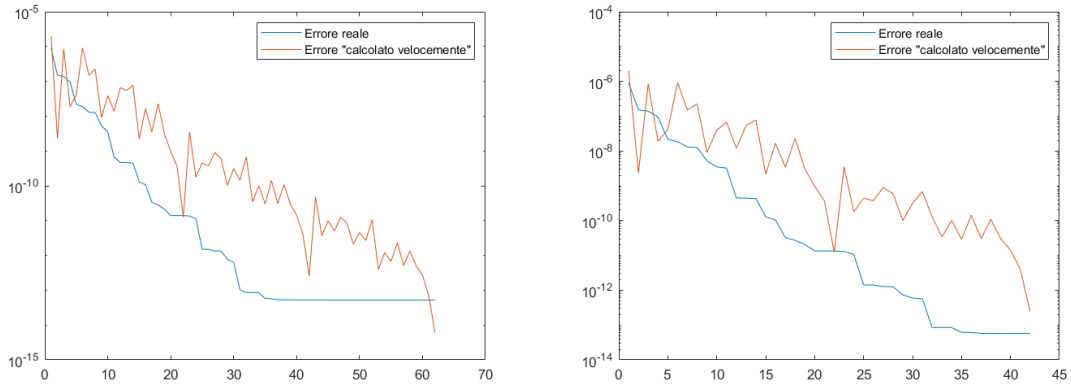
Come atteso dall'analisi teorica svolta in precedenza, al crescere di s aumenta la complessità in tempo delle iterazioni degli algoritmi basati sugli Spazi di Krylov. In particolare nel caso $n = m = 1000$ passando da $s = 1$ a $s = 5$ il tempo di esecuzione di un'iterazione viene più che quadruplicato, anche se parte del tempo necessario a risolvere un'istanza viene recuperato svolgendo un minor numero di iterazioni. Un altro fenomeno interessante che si verifica al crescere del parametro n è una "stabilizzazione" delle iterazioni, con il fenomeno del "rimbalzo" dell'errore che si verifica molto più raramente nei casi $n = m = 1000$ e $n = 2000, m = 3000$ rispetto al caso $n = m = 100$.

5.1.3 Metodo ADI: condizioni di arresto

Come già detto nel paragrafo dedicato, come condizione di arresto viene imposta una condizione sull'aggiornamento della soluzione. Usiamo in particolare la condizione $\|\widehat{U}_k\|_F/(\|\widehat{Z}_k\|_F \cdot 2\|A\|_F) \leq \varepsilon$ per un valore fissato di ε opportunamente piccolo, con $\|Z_k\|_F$ che viene calcolata iterazione dopo iterazione. A priori non vi è alcun legame tra l'errore commesso e tale condizione imposta e l'errore che viene effettivamente commesso e infatti l'andamento di $\|\widehat{U}_k\|_F/(\|\widehat{Z}_k\|_F \cdot 2\|A\|_F) \leq \varepsilon$ è piuttosto caotico rispetto a quello dell'errore, come mostrato in Figura 5.3.

Dalla Figura 5.3a si vede anche come l'errore realmente commesso si stabilizzi dopo un certo numero di passi, mentre la quantità calcolata come sopra continui ad oscillare e decrescere anche dopo tale stabilizzazione. Eseguendo diverse prove sperimentali, è risultato che il valore ottimale per la tolleranza `tol` è circa $\texttt{tol}_{\texttt{ADI}} = 5 \times 10^{-13}$: tale valore permette di non arrestare eccessivamente presto le iterazioni e al contempo di limitare il numero di iterazioni svolte inutilmente con l'errore reale ormai stabilizzato. Il comportamento con tale soglia di tolleranza è rappresentato in Figura 5.3b.

Figura 5.3: *Plot in scala semilogaritmica di errore reale e errore "calcolato velocemente" per il Metodo ADI. Parametri:* $n = 1000, s = 1$



(a) $\texttt{tol} = 10^{-14}$ (b) $\texttt{tol} = 5 \times 10^{-13}$

5.1.4 Confronto tra metodi per l'equazione di Lyapunov

Test analoghi a quelli per l'equazione di Sylvester sono stati condotti per l'equazione di Lyapunov

$$AX + XA^* = C$$

con $A \in \mathbb{C}^{n\times n}$, $C \in \mathbb{C}^{n\times n}$, $C = C_1 C_1^*$ con $C_1 \in \mathbb{C}^{n\times s}$ e $s \ll n$. Sono stati comparati i solutori proposti con la funzione `lyap` di MATLAB. I risultati sono riportati nelle Tabelle 5.5, 5.6, 5.7 e 5.8, dove con N_{ADI} indichiamo il numero di parametri di shift calcolati per il metodo CF-ADI.

Tabella 5.5: $n = 100 \quad A \prec 0, \quad s = 1, \quad \texttt{itmax} = 100, \quad \texttt{tol}_{\texttt{KRYL}} = 10^{-13}, \quad \texttt{tol}_{\texttt{ADI}} = 5 \times 10^{-13}, \quad N = 20, \quad N_{ADI} = 20$

Algoritmo	it	time	it_time	err
lyap		0.005		3.342×10^{-17}
Bartels-Stewart		0.015		3.377×10^{-17}
CF-ADI	41.6	0.035	0.0008	3.694×10^{-15}
Krylov Esteso	21.6	0.035	0.0016	9.856×10^{-10}

Il 70% delle istanze del Metodo di Krylov Esteso sono state arrestate perché l'errore ha iniziato ad aumentare.

Tabella 5.6: $n = 1000 \quad A \prec 0, \quad s = 1, \quad \texttt{itmax} = 100, \quad \texttt{tol}_{\texttt{KRYL}} = 10^{-13}, \quad \texttt{tol}_{\texttt{ADI}} = 5 \times 10^{-13}, \quad N = 10, \quad N_{ADI} = 20$

Algoritmo	it	time	it_time	err
lyap		1.000		3.094×10^{-17}
Bartels-Stewart		10.857		3.099×10^{-17}
CF-ADI	42.4	3.430	0.081	4.516×10^{-14}
Krylov Esteso	41.7	0.841	0.021	2.300×10^{-11}

Il 20% delle istanze del Metodo di Krylov Esteso sono state arrestate perché l'errore ha iniziato ad aumentare.

Tabella 5.7: $n = 1000 \quad A \prec 0, \quad s = 5, \quad \texttt{itmax} = 100, \quad \texttt{tol}_{\texttt{KRYL}} = 10^{-13}, \quad \texttt{tol}_{\texttt{ADI}} = 5 \times 10^{-13}, \quad N = 10, \quad N_{ADI} = 20$

Algoritmo	it	time	it_time	err
lyap		1.345		3.547×10^{-17}
Bartels-Stewart		13.962		3.548×10^{-17}
CF-ADI	42.4	4.629	0.109	5.318×10^{-15}
Krylov Esteso	23.3	2.490	0.107	1.348×10^{-11}

Nessuna istanza del Metodo di Krylov Esteso è stata arrestata perché l'errore ha iniziato ad aumentare.

Tabella 5.8: $n = 2000 \quad A \prec 0, \quad s = 1, \quad \texttt{itmax} = 100, \quad \texttt{tol}_{\texttt{KRYL}} = 10^{-13}, \quad \texttt{tol}_{\texttt{ADI}} = 5 \times 10^{-13}, \quad N = 5, \quad N_{ADI} = 20$

Algoritmo	it	time	it_time	err
lyap		8.25		2.722×10^{-17}
CF-ADI	42.6	24.791	0.581	1.463×10^{-14}
Krylov Esteso	27.8	7.640	0.275	1.742×10^{-12}

Nessuna istanza del Metodo di Krylov Esteso è stata arrestata perché l'errore ha iniziato ad aumentare.

Analogamente a quanto visto per l'equazione di Sylvester, anche per l'equazione di Lyapunov se n è piccolo il solutore migliore tra quelli proposti è l'algoritmo di Bartels-Stewart, sia per precisione raggiunta che per tempo di esecuzione. In aggiunta l'algoritmo CF-ADI raggiunge una precisione notevolmente migliore rispetto a quella dell'algoritmo basato su Spazi di Krylov estesi.

Al crescere di n, in particolare nel caso $n = 1000$, l'algoritmo basato su Spazi di Krylov estesi risulta il migliore per tempo di risoluzione di un'istanza, perdendo però in termini di precisione raggiunta. L'algoritmo di Bartels-Stewart invece, pur avendo performance nettamente peggiori in tempo di esecuzione, rimane il migliore per precisione raggiunta (comparabile con la funzione `lyap` di MATLAB). Il metodo CF-ADI ha prestazioni intermedie in entrambi i campi. Inoltre passando da $s = 1$ a $s = 5$ le performance degli algoritmi CF-ADI e di Bartels-Stewart restano sostanzialmente inalterate. Aumenta invece la complessità delle iterazioni del metodo di Krylov, mentre diminuisce il numero di iterazioni effettuate.

Facendo crescere ulteriormente n fino al valore $n = 2000$ l'algoritmo di Bartels-Stewart diventa inutilizzabile e per questa ragione non ne abbiamo riportato le performance. Anche l'algoritmo CF-ADI perde notevolemente efficienza in termini di tempo di esecuzione, mentre la precisione raggiunta rimane comunque migliore rispetto al Metodo di Krylov Esteso.

Interessante notare come anche nel caso dell'equazione di Lyapunov le iterazioni del Metodo di Krylov Esteso non vengano quasi mai arrestate a causa di un aumento dell'errore per $n \geq 1000$, mentre nel caso $n = 100$ il 70% delle iterazioni vengono arrestate per questo motivo.

5.2 Algoritmi di update

Le tecniche descritte nel Capitolo 3 per il calcolo della soluzione dell'equazione perturbata

$$(A_0 + \delta A)(X_0 + \delta X) + (X_0 + \delta X)(B_0 + \delta B) = C_0 + \delta C$$

con le fattorizzazioni di rango basso

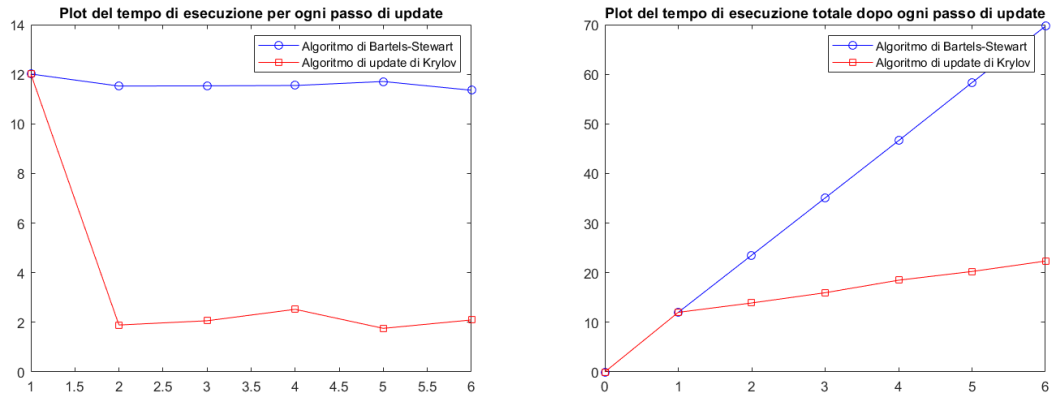
$$\delta A = U_A V_A^*, \quad \delta B = U_B V_B^*, \quad \delta C = U_C V_C^*,$$

sono state inizialmente testate su problemi di taglia $n = m = 1000$, con $A_0, B_0 \prec 0$. Sono stati considerati cinque updates successivi di rango 1 (vale a dire $U_A, V_A, U_B, V_B, U_C, V_C \in \mathbb{R}^{1000}$) per ogni test. Sono stati comparati gli algoritmi che sfruttano la fattorizzazione di rango basso degli update con un nuovo calcolo della soluzione del problema. Essendo in generale le matrici dei coefficienti $(A_0 + \delta A), (B_0 + \delta B)$ e $(C_0 + \delta C)$ di rango non basso, non possono essere sfruttati solutori low-rank e la risoluzione diretta della nuova equazione deve essere effettuata tramite l'algoritmo di Bartels-Stewart. I risultati, frutto della media effettuata su $N = 5$ test, sono riportati graficamente nelle Figure 5.4, 5.5, 5.6 e 5.7. In particolare, nelle Figure 5.4 e 5.6 sono riportati i dati relativi al tempo di risoluzione di ciascun update, mentre nelle Figure 5.5 e 5.7 viene riportato l'errore relativo commesso ad ogni passo di update.

Per quanto riguarda l'equazione di Sylvester le Figure 5.4 e 5.5 mostrano che, dopo un primo passo comune a entrambi i metodi per calcolare la soluzione X_0 dell'equazione di partenza, l'algoritmo di update basato su Spazi di Krylov Estesi impiega un tempo notevolmente inferiore a calcolare la soluzione aggiornata rispetto all'algoritmo di Bartels-Stewart. Chiaramente, le differenze in termini di tempo totale di calcolo si fanno ancora più marcate all'aumentare del numero di passi di update da calcolare.

Tuttavia, queste performance migliori dal punto di vista del tempo di calcolo comportano una notevole perdita di precisione, come mostrato in Figura 5.5. Andando a risolvere ogni volta la nuova equazione tramite l'algoritmo di Bartels-Stewart, la precisione raggiunta rimane sempre più o meno la stessa, mentre si può notare che usando l'algoritmo basato sugli Spazi di Krylov Estesi ad ogni passo di update si ha un peggioramento dell'errore relativo commesso. In particolare, si ha una notevole perdita di precisione nell'eseguire il primo update, mentre a partire dal secondo update si ha una stabilizzazione dell'errore relativo attorno al valore 10^{-10}.

Figura 5.4: *Tempo di esecuzione degli algoritmi di update per l'equazione di Sylvester*



(a) *Tempo di esecuzione di ciascun passo di update*

(b) *Tempo di esecuzione totale dopo ogni passo di update*

Per quanto riguarda l'equazione di Lyapunov, abbiamo effettuato dei test nel caso di equazioni di Lyapunov stabili sfruttando uno splitting come descritto nella Sezione 3.2. Come mostrato nelle Figure 5.6 e 5.7 le differenze tra l'algoritmo di Bartels-Stewart e l'algoritmo di update basato su Spazi di Krylov estesi sono sostanzialmente le stesse dell'equazione di Sylvester, con il primo che raggiunge una precisione notevolmente migliore impiegando però un tempo molto più alto. L'algoritmo di update basato sul Metodo CF-ADI invece mostra performance intermedie sia per l'errore commesso che per il tempo di esecuzione.

5.3 Metodo di Newton per equazioni di Riccati algebriche

Come descritto nel Capitolo 4, il metodo di Newton per la risoluzione di Equazioni di Riccati Algebriche del tipo

$$XA + A^*X - XBX = C,$$

con $A, B, C \in \mathbb{R}^{n\times n}$, $B \succeq 0$, $C \preceq 0$ e (A, B) stabilizzabile, richiede di risolvere equazioni di Lyapunov perturbate in sequenza. Se B ammette una fattorizzazione di rango basso $B_U B_U^*$, $B_U \in \mathbb{R}^{n\times s}$, abbiamo visto che è possibile sfruttare solutori low-rank per il calcolo dell'update. Risulta abbastanza naturale dunque chiedersi per quali valori di n e s è conveniente usare solutori low-rank invece dell'algoritmo di Bartels-Stewart, sia in termini di tempo di calcolo che per precisione raggiunta.

Abbiamo dunque confrontato, al variare di n e s, tre possibili implementazioni del Metodo di Newton:

- `newton_care_v1`: esegue il metodo di Newton calcolando ad ogni passo l'update $H = X_{k+1} - X_k$ con l'algoritmo di Bartels-Stewart;
- `newton_care_v2`: esegue il metodo di Newton calcolando ad ogni passo direttamente X_{k+1} (non l'update) con l'algoritmo di Bartels-Stewart;
- `newton_care_ADI`: esegue il metodo di Newton calcolando ad ogni passo l'update con l'algoritmo di CF-ADI.

Gli algoritmi di cui sopra sono stati confrontati, eseguendo la media su N test, in termini di

- `it`: numero medio di iterazioni svolte;
- `time`: tempo medio di risoluzione di un'istanza;
- `err`: errore relativo medio sulla soluzione finale, calcolato come

$$\texttt{err} = \frac{\|XA + A^*X - XBX - C\|_F}{\|X\|_F \cdot (\|A\|_F + \|B\|_F)};$$

- `not_ris`: percentuale di istanze non risolte. Sono considerate non risolte tutte le istanze che, avendo fissato $\texttt{tol} = 10^{-13}$, vengono risolte con una precisione inferiore a 10^{-6}.

Inoltre, per fare in modo che venissero soddisfatte le ipotesi sui teoremi di convergenza del metodo di Newton, abbiamo considerato il caso in cui $A \prec 0$ e si può quindi prendere $X_0 = 0$. I risultati sono riportati nelle seguenti tabelle e grafici.

Figura 5.5: *Precisione raggiunta ad ogni passo dagli algoritmi di update per l'equazione di Sylvester*

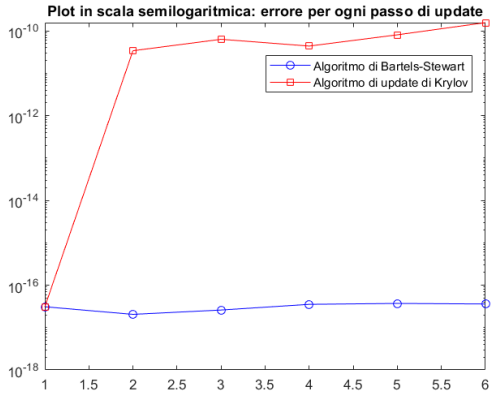
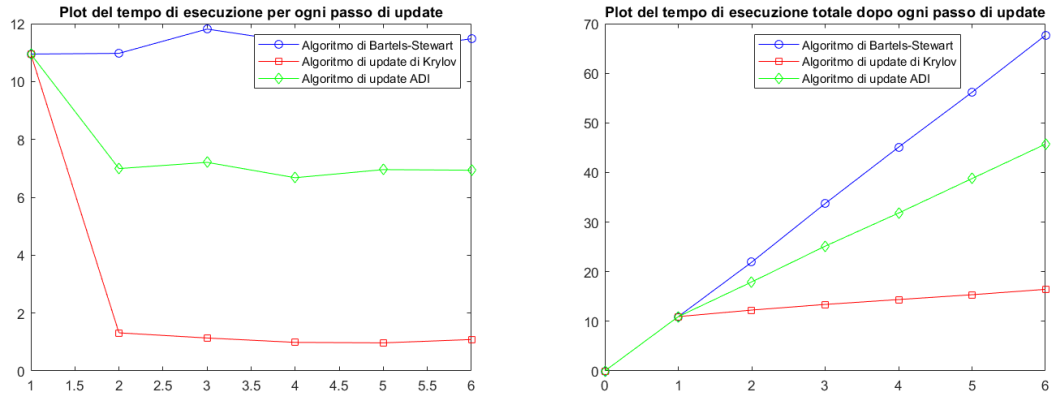


Figura 5.6: *Tempo di esecuzione degli algortmi di update per l'equazione di Lyapunov stabile*



(a) *Tempo di esecuzione di ciascun passo di update*

(b) *Tempo di esecuzione totale dopo ogni passo di update*

Figura 5.7: *Precisione raggiunta ad ogni passo dagli algoritmi di update per l'equazione di Lyapunov*

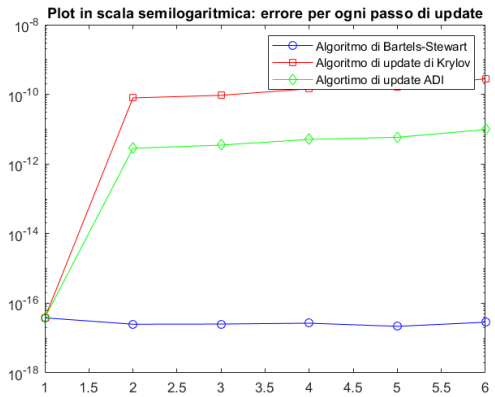


Tabella 5.9: $n = 100 \quad A \prec 0, \quad s = 1, \quad \texttt{itmax}_{\texttt{newton}} = 40, \quad \texttt{itmax}_{\texttt{ADI}} = 100, \quad \texttt{tol} = 10^{-13}, \quad N = 10 \quad N_{ADI} = 20$

Algoritmo	it	time	err	not_ris
newton_care_v1	13	0.705	2.363×10^{-16}	0 %
newton_care_v2	13	0.671	3.161×10^{-16}	0 %
newton_care_ADI	12	0.705	6.680×10^{-16}	20 %

Tabella 5.10: $n = 200 \quad A \prec 0, \quad s = 1, \quad \texttt{itmax}_{\texttt{newton}} = 40, \quad \texttt{itmax}_{\texttt{ADI}} = 100, \quad \texttt{tol} = 10^{-13}, \quad N = 10 \quad N_{ADI} = 20$

Algoritmo	it	time	err	not_ris
newton_care_v1	16.2	3.706	2.904×10^{-16}	0 %
newton_care_v2	16	3.523	3.213×10^{-16}	0 %
newton_care_ADI	14.8	2.126	7.963×10^{-15}	20 %

Tabella 5.11: $n = 300 \quad A \prec 0, \quad s = 1, \quad \texttt{itmax}_{\texttt{newton}} = 40, \quad \texttt{itmax}_{\texttt{ADI}} = 100, \quad \texttt{tol} = 10^{-13}, \quad N = 5 \quad N_{ADI} = 20$

Algoritmo	it	time	err	not_ris
newton_care_v1	17.8	12.215	1.214×10^{-16}	0 %
newton_care_v2	17.8	11.832	1.770×10^{-16}	0 %
newton_care_ADI	9.4	5.034	7.414×10^{-15}	20 %

Tabella 5.12: $n = 400 \quad A \prec 0, \quad s = 1, \quad \texttt{itmax}_{\texttt{newton}} = 40, \quad \texttt{itmax}_{\texttt{ADI}} = 100, \quad \texttt{tol} = 10^{-13}, \quad N = 5 \quad N_{ADI} = 20$

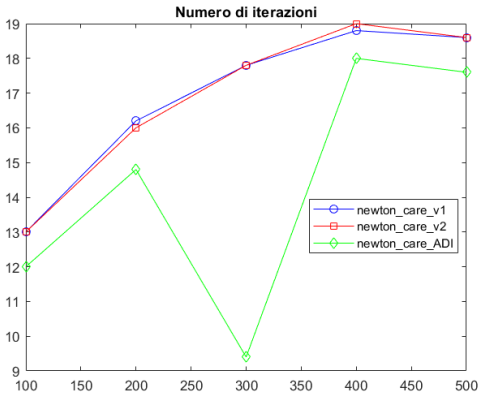
Algoritmo	it	time	err	not_ris
newton_care_v1	18.8	31.913	2.529×10^{-16}	0 %
newton_care_v2	19	31.652	2.664×10^{-16}	0 %
newton_care_ADI	18	13.877	1.374×10^{-14}	20 %

Tabella 5.13: $n = 500 \quad A \prec 0, \quad s = 1, \quad \texttt{itmax}_{\texttt{newton}} = 40, \quad \texttt{itmax}_{\texttt{ADI}} = 100, \quad \texttt{tol} = 10^{-13}, \quad N = 5 \quad N_{ADI} = 20$

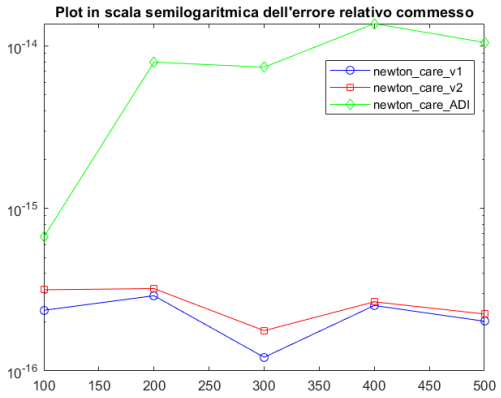
Algoritmo	it	time	err	not_ris
newton_care_v1	18.6	58.913	2.023×10^{-16}	0 %
newton_care_v2	18.6	57.028	2.242×10^{-16}	0 %
newton_care_ADI	17.6	17.687	1.053×10^{-14}	0 %

Per $n = 100$ e $s = 1$ l'algoritmo basato sul metodo ADI è quello con le performance peggiori. La precisione raggiunta e il tempo di esecuzione sono sostanzialmente equivalenti per tutti e tre gli algoritmi proposti, ma un'istanza su 5 non è stata risolta adoperando il

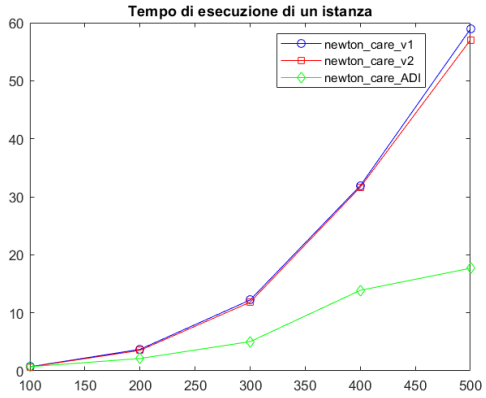
Figura 5.8: *Performance dell'algoritmo di Newton al variare della taglia del problema*



(a) *Numero medio di iterazioni*



(b) *Errore relativo medio commesso*



(c) *Tempo di risoluzione medio di un'istanza*

metodo `newton_care_ADI` mentre gli altri due algoritmi sono riusciti a risolverla. Avevamo infatti già osservato che per valori piccoli di n l'algoritmo di Bartels-Stewart risulta essere la scelta più efficiente.

Mantenendo fisso $s = 1$ si osserva che a partire da $n = 200$ l'algoritmo basato sul metodo CF-ADI risulta più efficiente per quanto riguarda il tempo di esecuzione, con le differenze che si fanno ancor più marcate al crescere del parametro n, fino al caso $n = 500$ il cui il tempo di risoluzione di un'istanza è circa un terzo di quello impiegato da `newton_care_v1` e `newton_care_v2`. L'errore relativo commesso adoperando il metodo `newton_care_ADI` è sempre maggiore, con la differenza che si fa più marcata al crescere di n ma rimane sempre entro limiti ragionevoli dato il consistente guadagno in termini di tempo di esecuzione. L'algoritmo `newton_care_ADI` tuttavia non riesce a risolvere tutte le istanze.

Possiamo dunque concludere, considerando sia il tempo di esecuzione che l'errore relativo commesso, che nel caso di valori di n piccoli gli algoritmi basati sul metodo di Bartels-Stewart sono i più efficienti. Per valori di n sufficientemente grandi invece risulta più conveniente usare solutori basati sul metodo CF-ADI. Risulta naturale a questo punto considerare il comportamento dei tre diversi algoritmi mantenendo fissa la taglia del problema $n = 300$ e facendo aumentare il rango della matrice $B = B_U B_U^*$.

Tabella 5.14: $n = 300 \quad A \prec 0, \quad s = 1, \quad \texttt{itmax}_{\texttt{newton}} = 40, \quad \texttt{itmax}_{\texttt{ADI}} = 100, \quad \texttt{tol} = 10^{-13}, \quad N = 10 \quad N_{ADI} = 20$

Algoritmo	it	time	err	not_ris
newton_care_v1	17.2	15.540	1.144×10^{-16}	0 %
newton_care_v2	17.2	14.923	1.445×10^{-16}	0 %
newton_care_ADI	9	8.960	2.659×10^{-15}	20 %

Tabella 5.15: $n = 300 \quad A \prec 0, \quad s = 2, \quad \texttt{itmax}_{\texttt{newton}} = 40, \quad \texttt{itmax}_{\texttt{ADI}} = 100, \quad \texttt{tol} = 10^{-13}, \quad N = 10 \quad N_{ADI} = 20$

Algoritmo	it	time	err	not_ris
newton_care_v1	20	16.429	1.937×10^{-16}	0 %
newton_care_v2	20	16.282	5.593×10^{-16}	0 %
newton_care_ADI	10.8	7.237	1.292×10^{-13}	20 %

Tabella 5.16: $n = 300 \quad A \prec 0, \quad s = 5, \quad \texttt{itmax}_{\texttt{newton}} = 40, \quad \texttt{itmax}_{\texttt{ADI}} = 100, \quad \texttt{tol} = 10^{-13}, \quad N = 10 \quad N_{ADI} = 20$

Algoritmo	it	time	err	not_ris
newton_care_v1	20.4	17.802	6.936×10^{-17}	0 %
newton_care_v2	20.4	17.474	1.138×10^{-15}	0 %
newton_care_ADI	15.4	13.420	2.266×10^{-12}	20 %

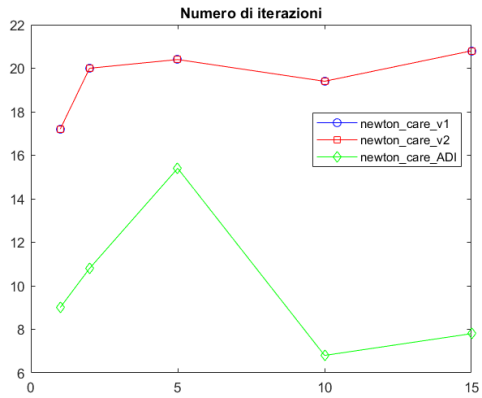
Tabella 5.17: $n = 300 \quad A \prec 0, \quad s = 10, \quad \texttt{itmax}_{\texttt{newton}} = 40, \quad \texttt{itmax}_{\texttt{ADI}} = 100, \quad \texttt{tol} = 10^{-13}, \quad N = 10 \quad N_{ADI} = 20$

Algoritmo	it	time	err	not_ris
newton_care_v1	19.4	18.450	7.040×10^{-17}	0 %
newton_care_v2	19.4	18.051	1.735×10^{-15}	0 %
newton_care_ADI	6.8	14.604	8.264×10^{-11}	60 %

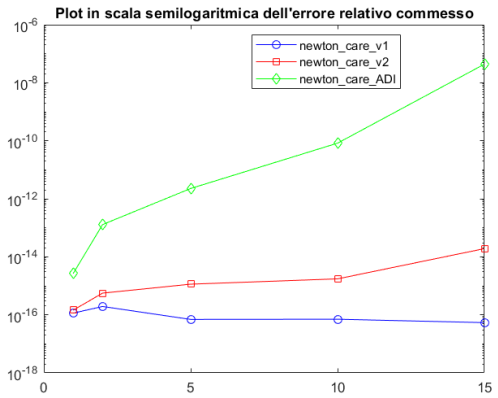
Tabella 5.18: $n = 300 \quad A \prec 0, \quad s = 15, \quad \texttt{itmax}_{\texttt{newton}} = 40, \quad \texttt{itmax}_{\texttt{ADI}} = 100, \quad \texttt{tol} = 10^{-13}, \quad N = 10 \quad N_{ADI} = 20$

Algoritmo	it	time	err	not_ris
newton_care_v1	20.8	19.852	5.381×10^{-17}	0 %
newton_care_v2	20.8	19.575	1.911×10^{-14}	0 %
newton_care_ADI	7.8	15.498	4.378×10^{-8}	60 %

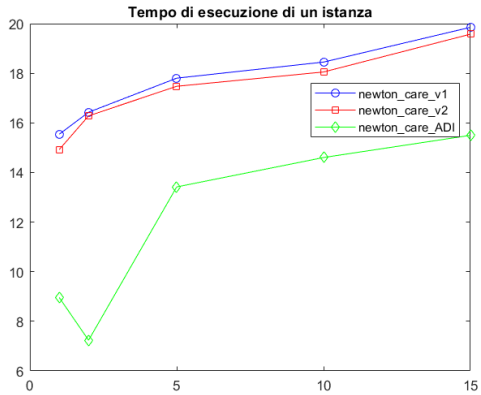
Figura 5.9: *Performance dell'algoritmo di Newton per* $n = 300$ *al variare della rango* s *di* B



(a) *Numero medio di iterazioni*



(b) *Errore relativo medio commesso*



(c) *Tempo di risoluzione medio di un'istanza*

Per valori molto piccoli di s, vale a dire $s = 1$ e $s = 2$, l'algoritmo basato sul metodo ADI risulta notevolmente migliore in termini di tempo di risoluzione di un'istanza è competitivo per quanto riguarda l'errore commesso, anche se abbiamo nuovamente il problema che alcune delle istanze non vengono risolte. Al crescere di s invece aumenta notevolmente la complessità in tempo e anche la precisione raggiunta peggiora, fino al caso $s = 15$ in cui appare ormai chiaro che il metodo `newton_care_ADI` è totalmente inefficiente rispetto agli altri due proposti, dato che impiega un tempo non molto inferiore ma raggiunge una precisione nettamente peggiore e non risolve la maggior parte delle istanze.

I metodi `newton_care_v1` e `newton_care_v2` sono sostanzialmente equivalenti per numero di iterazioni svolte e tempo di risoluzione di un'istanza, ma si può osservare come la precisione raggiunta dal metodo `newton_care_v1` sia migliore. Risulta quindi più efficiente andare a calcolare ad ogni passo l'update $H_k = X_{k+1} - X_k$ rispetto al calcolo diretto di X_{k+1}.

Appendici

A. Listings di base

Listato A.1: *Truncated Singular Value Decomposition (TSVD).*

```
function [U,Sigma,V] = tsvd(A,tol,k)
%   [U,SIGMA,V] = TSVD(A,TOL,K) calcola la Truncated Singular Value
%   Decomposition della matrice A, eliminando i vettori/valori singolari
%   relativi ai i valori singolari sigma_r tali che r > k o
%   sigma_r/sigma_1 < TOL.
%   INPUT:
%       - A: matrice di cui calcolare la TSVD
%       - TOL: tolleranza relativa per il troncamento dei valori singolari
%       - k: numero massimo di valori singolari mantenuti
%   OUTPUT:
%       - U,Sigma,V: matrici tali che Atilde = U*Sigma*V' fornisce
%         l'approssimazione di A cercata

%Gestione nel caso in cui k<=0 o k>min(m,n)
[m,n] = size(A);
if (k <= 0 || k > min(m,n))
    disp('Warning: k <= 0 o k > min(m,n), impostato di default k = min(m,n)
        ');
    k = min(m,n);
end

%Determinazione dell'indice per il troncamento
[U,Sigma,V] = svd(A,'econ');
sigma1 = Sigma(1,1);
r = find((diag(Sigma)/sigma1 < tol),1);
if ~isempty(r)
    k = min(k,r);
end

%Esecuzione del troncamento
U = U(:,1:k);
Sigma = Sigma(1:k,1:k);
V = V(:,1:k);
end
```

B. Listings per solutori low-rank

B.1 Metodi di Krylov

Listato B.1: *Metodo di Arnoldi.*

```
function [Q,H] = Arnoldi(A,r,m)
%   [Q,H] = ARNOLDI(A,r,M) esegue M iterazioni del processo di Arnoldi a
%   partire da una matrice A N-by-N e un vettore iniziale r (che non
%   necessariamente deve avere norma-2 unitaria). Per M < N produce in
%   output un matrice Q N-by-(M+1) con colonne ortonormali e una matrice di
%   Hessenberg superiore H (M+1)-by-M tale che:
%       A*Q(:,1:M) = Q(:,1:M)*H(1:M,1:M) + H(M+1,M)*Q(:,M+1)*e_M'
%   dove e_M e' l'M-esima colonna della matrice identita' di taglia M.

[n,~]=size(A);
r = r/norm(r);
Q = zeros(n,m);
Q(:,1) = r;
H = zeros(m+1,m);
tol = 1e-16;

for k=1:m
    z = A*Q(:,k);
    for i=1:k
        H(i,k) = Q(:,i)'*z;
        z = z - H(i,k)*Q(:,i);
    end
    if k < n
       H(k+1,k) = norm(z);
       if (H(k+1,k) < tol)
           return
       end
       Q(:,k+1) = z/H(k+1,k);
   end
end
```

Listato B.2: *Metodo di Arnoldi applicato all'inversa di una matrice.*

```
function [Q,H] = ArnoldiInv(A,r,m)
%   [Q,H] = ARNOLDIINV(A,r,M) esegue M iterazioni del processo di Arnoldi a
%   partire dalla matrice A^(-1) N-by-N e da un vettore iniziale r (che non
%   necessariamente deve avere norma-2 unitaria). Per M < N produce in
%   output un matrice Q N-by-(M+1) con colonne ortonormali e una matrice di
%   Hessenberg superiore H (M+1)-by-M tale che:
%       A^(-1)*Q(:,1:M) = Q(:,1:M)*H(1:M,1:M) + H(M+1,M)*Q(:,M+1)*e_M'
%   dove e_M e' l'M-esima colonna della matrice identita' di taglia M.

```

```
[n,~]=size(A);
r = r/norm(r);
Q = zeros(n,m);
Q(:,1) = r;
H = zeros(m+1,m);
tol = 1e-16;

for k=1:m
    z = A\Q(:,k);                %Non viene calcolata inv(A) ma si risolvono
        sitemi lineari
    for i=1:k
        H(i,k) = Q(:,i)'*z;
        z = z - H(i,k)*Q(:,i);
    end
    if k < n
       H(k+1,k) = norm(z);
       if (H(k+1,k) < tol)
           return
       end
       Q(:,k+1) = z/H(k+1,k);
   end
end
```

Listato B.3: *Metodo di Krylov per l'equazione di Sylvester basato su Spazi di Krylov Standard.*

```
function [X,k] = lr_sylv_krylov(A,B,C1,C2,itmax,tol)
%   [X,K] = LR_SYLV_KRYLOV(A,B,C1,C2,ITMAX,TOL) risolve
%   l'equazione di Sylvester A*X + B*X = C1*C2' per mezzo dell'algoritmo
%   proiettivo basato sugli Spazi di Krylov Standard.
%   INPUT:
%       - A,B,C1,C2: matrici dei coefficienti
%       - ITMAX: numero massimo di iterazioni
%       - TOL: valore soglia per la condizione di arresto
%   OUTPUT:
%       - X: soluzione dell'equazione
%       - K: numero di iterazioni svolte

%Operazioni preliminari
[~,s] = size(C1);
[V,~] = qr(A*C1,0);
[W,~] = qr(B'*C2,0);
normA = norm(A,'fro');
normB = norm(B,'fro');

for k = 1:itmax
    %Formulazione e risoluzione del problema di taglia ridotta
    C1tilde = V'*C1;
    C2tilde = W'*C2;
    Ctilde = C1tilde*C2tilde';
    %Controllo della convergenza
    if (k > 1)
        err = norm(HV*Y*[eye((k-1)*s),zeros((k-1)*s,s)]+[eye((k-1)*s);zeros
            (s,(k-1)*s)]*Y*HW'-Ctilde,'fro')/(norm(Y,'fro')*(normA+normB));
        if (err < tol)
            X = V(:,1:end-s)*Y*W(:,1:end-s)';
            return
        end
    end

    Atilde = V'*A*V;
    Btilde = W'*B*W;
```

```
    Y = sylv_bartels_stewart(Atilde,Btilde,Ctilde);

    %Aggiornamento dello spazio di Krylov
    %Partizionamento di V e W per ottenere Vtilde e Wtilde
    Vtilde = A*V(:,end-s+1:end);
    Wtilde = B'*W(:,end-s+1:end);

    %Ortogonalizzazione di Vtilde risepetto a V e di Wtilde rispetto a W
    %Poiche' le colonne di V e W sono ortonormali conviene calcolare come
    %sotto invece che con:
    %   Vtilde = Vtilde-V*V'*Vtilde
    %   Wtilde = Wtilde-W*W'*Wtilde
    for i = 1:k
        HV(((i-1)*s+1):(i*s),((k-1)*s+1):(k*s)) = V(:,((i-1)*s+1):(i*s))'*
            Vtilde;
        Vtilde = Vtilde - V(:,((i-1)*s+1):(i*s))*HV(((i-1)*s+1):(i*s),((k
            -1)*s+1):(k*s));
        HW(((i-1)*s+1):(i*s),((k-1)*s+1):(k*s)) = W(:,((i-1)*s+1):(i*s))'*
            Wtilde;
        Wtilde = Wtilde - W(:,((i-1)*s+1):(i*s))*HW(((i-1)*s+1):(i*s),((k
            -1)*s+1):(k*s));
    end

    %Aggiornamento di V e W
    [Vtilde,HV((k*s+1):((k+1)*s),((k-1)*s+1):(k*s))]=qr(Vtilde,0);
    V = [V,Vtilde];
    [Wtilde,HW((k*s+1):((k+1)*s),((k-1)*s+1):(k*s))]=qr(Wtilde,0);
    W = [W,Wtilde];
end
disp('Warning: raggiunto il numero massimo di iterazioni');
X = V(:,1:end-s)*Y*W(:,1:end-s)';
```

Listato B.4: *Metodo di Krylov per l'equazione di Sylvester basato su Spazi di Krylov Estesi.*

```
function [X,k,flag] = lr_sylv_extended_krylov(A,B,C1,C2,itmax,tol)
%   [X,K,FLAG] = LR_SYLV_EXTENDED_KRYLOV(A,B,C1,C2,ITMAX,TOL) risolve
%   l'equazione di Sylvester A*X + X*B = C1*C2' per mezzo dell'algoritmo
%   proiettivo basato sugli Spazi di Krylov Estesi.
%   INPUT:
%       - A,B,C1,C2: matrici dei coefficienti
%       - ITMAX: numero massimo di iterazioni
%       - TOL: valore soglia per la condizione di arresto
%   OUTPUT:
%       - X: soluzione dell'equazione
%       - K: numero di iterazioni svolte
%       - FLAG: vale 0 o 1 e indica se le iterazioni sono state arrestate
%         perche' l'errore ha iniziato a crescere


%Operazioni preliminari
[~,s] = size(C1);
[LA,UA] = lu(A);
[LB,UB] = lu(B');
H = zeros(2*s);
itmin = 20;
errprec = 0;
err = 0;
flag = 0;
normA = norm(A,'fro');
normB = norm(B,'fro');

```

```
%Calcolo di V1 e W1
[V,~] = qr([A*C1,UA\(LA\C1)],0);
[W,~] = qr([B'*C2,UB\(LB\C2)],0);

for k = 1:itmax
    %Formulazione e risoluzione del problema di taglia ridotta
    Atilde = V'*A*V;
    Btilde = W'*B*W;
    C1tilde = V'*C1;
    C2tilde = W'*C2;
    Ctilde = C1tilde*C2tilde';
    Y = sylv_bartels_stewart(Atilde,Btilde,Ctilde);

    %Controllo della convergenza
    m = k-1;
    if k>1
        errprec = err;
        err = sqrt(norm(Atilde(2*m*s+1:2*m*s+s,2*(m-1)*s+1:2*m*s)*Y(2*(m-1)
            *s+1:2*m*s,:),'fro')^2+norm(Btilde(2*(m-1)*s+1:2*m*s,2*m*s+1:2*
            m*s+s)'*Y(:,2*(m-1)*s+1:2*m*s)','fro')^2)/(norm(Y,'fro')*(normA
            +normB));
        if (err < tol || (errprec < err && k > itmin))
            if (errprec < err && k > itmin)
                disp('Warning: iterazioni arrestate per probabili errori di
                    round-off');
                flag = 1;
            end
            X = V*Y*W';
            return
        end
    end

    %Aggiornamento dello spazio di Krylov
    %Partizionamento di V e W per ottenere Vtilde e Wtilde
    Vplus = V(:,end-2*s+1:end-s);
    Vmin = V(:,end-s+1:end);
    Vtilde = [A*Vplus,UA\(LA\Vmin)];
    Wplus = W(:,end-2*s+1:end-s);
    Wmin = W(:,end-s+1:end);
    Wtilde = [B'*Wplus,UB\(LB\Wmin)];

    %Ortogonalizzazione di Vtilde risepetto a V e di Wtilde rispetto a W
    %Poiche' le colonne di V e W sono ortonormali conviene calcolare come
    %sotto invece che con:
    %   Vtilde = Vtilde-V*V'*Vtilde
    %   Wtilde = Wtilde-W*W'*Wtilde
    for i = 1:k
        H = V(:,((i-1)*2*s+1):(i*2*s))'*Vtilde;
        Vtilde = Vtilde - V(:,((i-1)*2*s+1):(i*2*s))*H;
        H = W(:,((i-1)*2*s+1):(i*2*s))'*Wtilde;
        Wtilde = Wtilde - W(:,((i-1)*2*s+1):(i*2*s))*H;
    end

    %Aggiornamento di V e W
    [Vtilde,~]=qr(Vtilde,0);
    V = [V,Vtilde];
    [Wtilde,~]=qr(Wtilde,0);
    W = [W,Wtilde];
end
disp('Warning: raggiunto il numero massimo di iterazioni');
X = V(:,1:end-2*s)*Y*W(:,1:end-2*s)';
```

Listato B.5: *Metodo di Krylov per l'equazione di Lyapunov basato su Spazi di Krylov Estesi.*

```
function [X,k,flag] = lr_lyap_extended_krylov(A,C1,itmax,tol)
%   [X,K,FLAG] = LR_LYAP_EXTENDED_KRYLOV(A,C1,ITMAX,TOL) risolve
%   l'equazione di Sylvester A*X + X*A' = C1*C1' per mezzo dell'algoritmo
%   proiettivo basato sugli Spazi di Krylov Estesi.
%   INPUT:
%       - A,B,C1: matrici dei coefficienti
%       - ITMAX: numero massimo di iterazioni
%       - TOL: valore soglia per la condizione di arresto
%   OUTPUT:
%       - X: soluzione dell'equazione
%       - K: numero di iterazioni svolte
%       - FLAG: vale 0 o 1 e indica se le iterazioni sono state arrestate
%         perche' l'errore ha iniziato a crescere


%Operazioni preliminari
[~,s] = size(C1);
[LA,UA] = lu(A);
H = zeros(2*s);
itmin = 20;
errprec = 0;
err = 0;
flag = 0;
normA = norm(A,'fro');

%Calcolo di V1 e W1
[V,~] = qr([A*C1,UA\(LA\C1)],0);

for k = 1:itmax
    %Formulazione e risoluzione del problema di taglia ridotta
    Atilde = V'*A*V;
    C1tilde = V'*C1;
    Ctilde = C1tilde*C1tilde';
    Y = lyap_bartels_stewart(Atilde,Ctilde);

    %Controllo della convergenza
    m = k-1;
    if k>1
        errprec = err;
        err = sqrt(2)*norm(Atilde(2*m*s+1:2*m*s+s,2*(m-1)*s+1:2*m*s)*Y(2*(m
            -1)*s+1:2*m*s,:),'fro')/(norm(Y,'fro')*2*normA);
        if (err < tol || (errprec < err && k > itmin))
            if (errprec < err && k > itmin)
                disp('Warning: iterazioni arrestate per probabili errori di
                    round-off');
                flag = 1;
            end
            X = V*Y*V';
            return
        end
    end

    %Aggiornamento dello spazio di Krylov
    %Partizionamento di V e W per ottenere Vtilde e Wtilde
    Vplus = V(:,end-2*s+1:end-s);
    Vmin = V(:,end-s+1:end);
    Vtilde = [A*Vplus,UA\(LA\Vmin)];

    %Ortogonalizzazione di Vtilde risepetto a V e di Wtilde rispetto a W
    %Poiche' le colonne di V e W sono ortonormali conviene calcolare come
```

```
    %sotto invece che con:
    %   Vtilde = Vtilde-V*V'*Vtilde
    %   Wtilde = Wtilde-W*W'*Wtilde
    for i = 1:k
        H = V(:,((i-1)*2*s+1):(i*2*s))'*Vtilde;
        Vtilde = Vtilde - V(:,((i-1)*2*s+1):(i*2*s))*H;
    end

    %Aggiornamento di V e W
    [Vtilde,~]=qr(Vtilde,0);
    V = [V,Vtilde];
end
disp('Warning: raggiunto il numero massimo di iterazioni');
X = V(:,1:end-2*s)*Y*V(:,1:end-2*s)';
```

B.2 Metodi ADI

Listato B.6: *Calcolo dei parametri sub-ottimali per il metodo ADI.*

```
function p = ADI_Suboptimal(A,c,kplus,kmin)
%   P = ADI_SUBOPTIMAL(A,C,KPLUS,KMIN) calcola dei parametri subottimali
%   per il metodo ADI secondo l'algoritmo euristico di T. Penzl.
%   Applicabile a matrici definite negative oppure tali che R sia contenuto
%   nel semipiano complesso sinistro.
%   INPUT:
%       - A: matrice dei coefficienti dell'equazione di Lyapunov
%       - C: numero di parametri da ottenere
%       - KPLUS: numero di iterazioni del metodo di Arnoldi per A
%       - KMIN: numero di iterazioni del metodo di Arnoldi per A^(-1)
%   Se KPLUS+KMIN<C vengono impostati di defalut KPLUS = KPLUS+C e
%   KMIN = KMIN +C.
%   OUTPUT:
%       - P: vettore contenente i parametri di shift

%Operazioni preliminari
[n,~]=size(A);
count = 0;
k = kplus+kmin;
p = zeros(c,1);
if (k<c)
    kplus = kplus+c;
    kmin = kmin+c;
    k = k+2*c;
    disp('Warning: kplus+kmin < c, impostato di defalut kplus = kplus+c e
        kmin = kmin+c');
end

%Definizione della funzione ausiliaria s
s = @(v,t) abs(prod(v-t)/prod(v+t));

%Calcolo dell'insieme R
r = rand(n,1);
[~,H] = Arnoldi(A,r,kplus);
[~,W] = ArnoldiInv(A,r,kmin);
Rplus = eig(H(1:kplus,1:kplus));
Rmin = 1./eig(W(1:kmin,1:kmin));
R = [Rplus;Rmin];
%Verifica che R sia contenuto nel semipiano sinistro
if ~all(real(R)<0)
```

```
    error('L''insieme R non e'' contenuto nel semipiano complesso sinistro');
end

%Passo iniziale
aux1 = zeros(k,k);
for i = 1:k
    for j = 1:k
        aux1(i,j) = abs((R(i)-R(j))/(R(i)+R(j)));
    end
end
[~,i] = min(max(aux1));
count = count+1;
p(count) = R(i);
R(R == p(count)) = [];
if ~isreal(p(count))
    count = count+1;
    p(count) = conj(p(count-1));
    R(R == p(count)) = [];
end

%Passi successivi al primo
while (count < c)
    imax = 1;
    val = s(p,R(1));
    for i = 2:k-count
        temp = s(p,R(i));
        if (temp > val)
            imax = i;
            val = temp;
        end
    end
    count = count+1;
    p(count) = R(imax);
    R(R == p(count)) = [];
    if ~isreal(p(count))
        count = count+1;
        p(count) = conj(p(count-1));
        R(R == p(count)) = [];
    end
end
```

Listato B.7: *Metodo Cholesky Factor ADI (CF-ADI).*

```
function [Z,k] = CF_ADI(A,C1,N,itmax,tol)
%   [Z,K] = CF_ADI(A,C1,N,ITMAX,TOL) risolve l'equazione di Lyapunov
%   A*X + X*A' + C1*C1' = 0 per mezzo dell'algoritmo Cholesky Factor ADI.
%   INPUT:
%       - A,C1: matrici dei coefficienti
%       - N: numero dei parametri di shift da adoperare
%       - ITMAX: numero massimo di iterazioni
%       - TOL: valore soglia per la condizione di arresto
%   OUTPUT:
%       - Z: matrice tale che X = Z*Z' risolve l'equazione
%       - K: numero di iterazioni svolte

[n,~] = size(A);
normA = norm(A,'fro');

%Iterazione 1
s = ADI_Suboptimal(A,N,2*N,2*N);
Z = sqrt(-2*real(s(1)))*((A+s(1)*eye(n))\C1);
```

```
19 U = Z;
20 normZ = norm(Z,'fro');
21
22 %Iterazioni successive alla prima
23 for k = 2:itmax
24     m = mod(k,N);
25     if (m == 0)
26         m = N;
27     end
28     prec = m-1;
29     if (prec == 0)
30         prec = N;
31     end
32     U = sqrt(-2*real(s(m)))/sqrt(-2*real(s(prec)))*(U-(s(m)+conj(s(prec)))
           *((A+s(m)*eye(n))\U));
33     Z = [Z,U];
34     %Aggiornamento della norma della soluzione e valutazione della
           condizione di arresto
35     normU = norm(U,'fro');
36     normZ = sqrt(normZ^2 + normU^2);
37     if (normU/(normZ*normA*2) < tol)
38         return
39     end
40 end
41 disp('Warning: raggiunto il numero massimo di iterazioni');
```

C. Listings per algoritmi di update

Listato C.1: *Algoritmo di update per aggiornamenti di rango basso: equazione di Sylvester.*

```
function deltaX = update_sylv_krylov(A,deltaA,B,deltaB,deltaC,X,tol,UA,VA,
    UB,VB,UC,VC)
%   DELTAX = UPDATE_SYLV_KRYLOV(A,DELTAA,B,DELTAB,DELTAC,X,TOL,UA,VA,UB,VB,
    UC,VC)
%   calcola una correzione DELTAX di rango basso tale che, data X soluzione
%   di AX+XB=C, allora (X+DELTAX) e' soluzione dell'equazione di Sylvester
%       (A+DELTAA)(X+DELTAX)+(X+DELTAX)(B+DELTAB)=(C+DELTAC)
%   usando come solutore per l'equazione di rango basso quello basato sugli
%   Spazi di Krylov Estesi.
%   INPUT:
%       -A,B: matrici dei coefficienti dell'equazione originaria
%       -DELTAA,DELTAB,DELTAC: matrici dei coefficienti della correzione
%       -X: soluzione dell'equazione originaria
%       -tol: tolleranza per la risoluzione dell'equazione della correzione
%       -UA,VA,UB,VB,UC,VC (OPZIONALI): matrici di fattorizzazione tali che
%        DELTAA = UA*VA', DELTAB = UB*VB' e DELTAC = UC*VC'
%   OUTPUT:
%       -DELTAX: correzione

%Parametri e operazioni preliminari
toltsvd1 = 1e-7;
toltsvd2 = 1e-7;
kA = -1;
kB = -1;
kC = -1;
kUV = -1;
itmax = 500;

%Se vengono date in input deltaA,deltaB e deltaC ma non le loro
%fattorizzazioni, allora UA,VA,UB,VB,UC,VC vengono calcolate con la tsvd
if (nargin == 7)
    [UA,SigmaA,VA] = tsvd(deltaA,toltsvd1,kA);
    UA = UA*sqrt(SigmaA);
    VA = VA*sqrt(SigmaA);
    [UB,SigmaB,VB] = tsvd(deltaB,toltsvd1,kB);
    UB = UB*sqrt(SigmaB);
    VB = VB*sqrt(SigmaB);
    [UC,SigmaC,VC] = tsvd(deltaC,toltsvd1,kC);
    UC = UC*sqrt(SigmaC);
    VC = VC*sqrt(SigmaC);
end

%Generazione e successiva compressione di U e V tali che:
%   deltaC-deltaA*X-X*deltaB = U*V'
```

```
U = [UC,-UA,-X*UB];
V = [VC,X'*VA,VB];
[QU,RU] = qr(U,0);
[QV,RV] = qr(V,0);
[UR,SigmaR,VR] = tsvd(RU*RV',toltsvd2,kUV);
U = QU*UR*sqrt(SigmaR);
V = QV*VR*sqrt(SigmaR);

%Risoluzione dell'equazione per la correzione con il metodo proiettivo
%basato su Spazi di Krylov Estesi
A = A+deltaA;
B = B+deltaB;
deltaX = lr_sylv_extended_krylov(A,B,U,V,itmax,tol);
```

Listato C.2: *Algoritmo di update per aggiornamenti di rango basso: equazione di Lyapunov stabile e metodo di Krylov.*

```
function deltaX = update_stable_lyap_krylov(A,deltaA,deltaC,X,tol,UA,VA,UC,
    SigmaC)
%   DELTAX = UPDATE_STABLE_LYAP_ADI(A,DELTAA,DELTAC,X,TOL,UA,VA,UC,SIGMAC)
%   calcola una correzione DELTAX di rango basso tale che, data X soluzione
%   di AX+XA'=C, allora (X+DELTAX) e' soluzione dell'equazione di Sylvester
%       (A+DELTAA)(X+DELTAX)+(X+DELTAX)(A+DELTAA)'=(C+DELTAC)
%   usando come solutore per l'equazione di rango basso quello basato sul
%   metodo ADI.
%   INPUT:
%       -A: matrice dei coefficienti dell'equazione originaria
%       -DELTAA,DELTAC: matrici dei coefficienti della correzione
%       -X: soluzione dell'equazione originaria
%       -tol: tolleranza per la risoluzione dell'equazione della correzione
%       -UA,VA,UC,SIGMAC (OPZIONALI): matrici di fattorizzazione tali che
%        DELTAA = UA*VA', DELTAC = UC*SIGMAC*UC'
%   OUTPUT:
%       -DELTAX: correzione

%Parametri e operazioni preliminari
toltsvd1 = 1e-7;
kA = -1;
kC = -1;
itmax = 1000;
N = 20;

%Se vengono date in input deltaA e deltaC ma non le loro fattorizzazioni,
%allora UA,VA,UC,SigmaC vengono calcolate con la tsvd
if (nargin == 5)
    [UA,SigmaA,VA] = tsvd(deltaA,toltsvd1,kA);
    UA = UA*sqrt(SigmaA);
    VA = VA*sqrt(SigmaA);
    [UC,SigmaC,~] = tsvd(deltaC,toltsvd1,kC);
end

%Partizionamento del membro di destra della equazione per il calcolo della
%correzione
[sC,~] = size(SigmaC);
[~,sA] = size(UA);
Utilde = [UC,UA,X*VA];
Sigma = [SigmaC,zeros(sC,2*sA);zeros(sA,sC+sA),-eye(sA);zeros(sA,sC),-eye(
    sA),zeros(sA,sA)];
[Qtilde,Rtilde] = qr(Utilde,0);
[Q,D] = eig(Rtilde*Sigma*Rtilde');
%Suddivisione in due equazioni di taglia ridotta
indexplus = find(diag(D)>0);
```

```
indexmin = find(diag(D)<0);
D1 = D(indexplus,indexplus);
Q1 = Q(:,indexplus);
D2 = -D(indexmin,indexmin);
Q2 = Q(:,indexmin);
U1 = Qtilde*Q1*sqrt(D1);
U2 = Qtilde*Q2*sqrt(D2);
A = A+deltaA;

%Calcolo della correzione
X1 = lr_lyap_extended_krylov(A,U1,itmax,tol);
X2 = lr_lyap_extended_krylov(A,U2,itmax,tol);
deltaX = X1-X2;
```

Listato C.3: *Algoritmo di update per aggiornamenti di rango basso: equazione di Lyapunov stabile e metodo ADI.*

```
function deltaX = update_stable_lyap_ADI(A,deltaA,deltaC,X,tol,UA,VA,UC,
    SigmaC)
%   DELTAX = UPDATE_STABLE_LYAP_ADI(A,DELTAA,DELTAC,X,TOL,UA,VA,UC,SIGMAC)
%   calcola una correzione DELTAX di rango basso tale che, data X soluzione
%   di AX+XA'=C, allora (X+DELTAX) e' soluzione dell'equazione di Sylvester
%       (A+DELTAA)(X+DELTAX)+(X+DELTAX)(A+DELTAA)'=(C+DELTAC)
%   usando come solutore per l'equazione di rango basso quello basato sul
%   metodo ADI
%   INPUT:
%       -A: matrice dei coefficienti dell'equazione originaria
%       -DELTAA,DELTAC: matrici dei coefficienti della correzione
%       -X: soluzione dell'equazione originaria
%       -tol: tolleranza per la risoluzione dell'equazione della correzione
%       -UA,VA,UC,SIGMAC (OPZIONALI): matrici di fattorizzazione tali che
%        DELTAA = UA*VA', DELTAC = UC*SIGMAC*UC'
%   OUTPUT:
%       -DELTAX: correzione


%Parametri e operazioni preliminari
toltsvd1 = 1e-7;
kA = -1;
kC = -1;
itmax = 1000;
N = 20;

%Se vengono date in input deltaA e deltaC ma non le loro fattorizzazioni,
%allora UA,VA,UC,SigmaC vengono calcolate con la tsvd
if (nargin == 5)
    [UA,SigmaA,VA] = tsvd(deltaA,toltsvd1,kA);
    UA = UA*sqrt(SigmaA);
    VA = VA*sqrt(SigmaA);
    [UC,SigmaC,~] = tsvd(deltaC,toltsvd1,kC);
end

%Partizionamento del membro di destra della equazione per il calcolo della
%correzione
[sC,~] = size(SigmaC);
[~,sA] = size(UA);
Utilde = [UC,UA,X*VA];
Sigma = [SigmaC,zeros(sC,2*sA);zeros(sA,sC+sA),-eye(sA);zeros(sA,sC),-eye(
    sA),zeros(sA,sA)];
[Qtilde,Rtilde] = qr(Utilde,0);
[Q,D] = eig(Rtilde*Sigma*Rtilde');
%Suddivisione in due equazioni di taglia ridotta
```

```
indexplus = find(diag(D)>0);
indexmin = find(diag(D)<0);
D1 = D(indexplus,indexplus);
Q1 = Q(:,indexplus);
D2 = -D(indexmin,indexmin);
Q2 = Q(:,indexmin);
U1 = Qtilde*Q1*sqrt(D1);
U2 = Qtilde*Q2*sqrt(D2);
A = A+deltaA;

%Calcolo della correzione
Z1 = CF_ADI(A,U1,N,itmax,tol);
Z2 = CF_ADI(A,U2,N,itmax,tol);
deltaX = Z2*Z2'-Z1*Z1';
```

D. Listings per Equazioni di Riccati Algebriche

Listato D.1: *Metodo di Newton per CARE con calcolo dell'update* $H_k = X_{k+1} - X_k$ *tramite l'algoritmo di Bartels-Stewart.*

```
function [X,k] = newton_care_v1(A,B,C,X0,tol,itmax)
%   [X,K] = NEWTON_CARE_V1(A,B,C,X0,TOL,ITMAX) risolve la CARE
%   XA + A'X - XBX = C per mezzo del metodo di Newton basato sul calcolo di
%   H_K = X_(K+1) - X_K
%   INPUT
%       - A, B, C: matrici dei coefficienti
%       - X0: approssimazione iniziale
%       - TOL: tolleranza per la condizione di arresto
%       - ITMAX: numero massimo di iterazioni
%   OUTPUT
%       - X : soluzione della CARE
%       - K: numero di iterazioni del metodo di Newton effettuate

X = X0;
err = 1;
k = 0;
normA = norm(A,'fro');
normB = norm(B,'fro');

while (err > tol && k < itmax)
    RX = X*A + A'*X - X*B*X - C;
    H = lyap_bartels_stewart(A'-X*B,-RX);
    X = X + H;
    err = norm(H,'fro')/(norm(X,'fro')*(normA+normB));
    k = k + 1;
end
if k == itmax
    disp('Warning: raggiunto il numero massimo di iterazioni')
end
```

Listato D.2: *Metodo di Newton per CARE con calcolo diretto di* X_{k+1} *tramite l'algoritmo di Bartels-Stewart.*

```
function [X,k] = newton_care_v2(A,B,C,X0,tol,itmax)
%   [X,K] = NEWTON_CARE_V1(A,B,C,X0,TOL,ITMAX) risolve la CARE
%   XA + A'X - XBX = C per mezzo del metodo di Newton basato sul calcolo
%   diretto di X_(k+1)
%   INPUT
%       - A, B, C: matrici dei coefficienti
%       - X0: approssimazione iniziale
```

```
%       - TOL: tolleranza per la condizione di arresto
%       - ITMAX: numero massimo di iterazioni
%   OUTPUT
%       - X : soluzione della CARE
%       - K: numero di iterazioni del metodo di Newton effettuate

X = X0;
err = 1;
k = 0;
normA = norm(A,'fro');
normB = norm(B,'fro');

while (err > tol && k < itmax)
    prec = X;
    RX = C - X*B*X;
    X = lyap_bartels_stewart(A'-X*B,RX);
    err = norm(prec-X,'fro')/(norm(X,'fro')*(normA+normB));
    k = k + 1;
end
if k == itmax
    disp('Warning: raggiunto il numero massimo di iterazioni')
end
```

Listato D.3: *Metodo di Newton per CARE con calcolo di update successivi di rango basso tramite l'algoritmo CF-ADI.*

```
function [X,k] = lr_newton_care_ADI(A,BU,C,X0,tol,itmax)
%   [X,K] = LR_NEWTON_CARE_ADI(A,BU,C,X0,TOL,ITMAX) risolve la CARE
%   X*A + A'*X - X*B*X = C per mezzo del metodo di Newton sfruttando la
%   fattorizzazione di rango basso di B = BU*BU' e il solutore di rango
%   basso basato sul metodo ADI.
%   INPUT
%       - A, BU, C: matrici dei coefficienti
%       - X0: approssimazione iniziale
%       - tol: tolleranza per la condizione di arresto
%       - itmax: numero massimo di iterazioni
%   OUTPUT
%       - X : soluzione della CARE
%       - K: numero di iterazioni del metodo di Newton effettuate

err = 1;
k = 0;
normA = norm(A,'fro');
normB = norm(BU,'fro')^2;
Ak = A'-X0*BU*(BU');
Ck = C-X0*BU*(BU')*X0;
X = lyap_bartels_stewart(Ak,Ck);
H = X-X0;
N = 20;
itmaxADI = 250;
while (err > tol && k < itmax)
    Ak = A'-X*BU*(BU');
    Z = CF_ADI(Ak,H*BU,N,itmaxADI,tol);                    %Si usa
        il metodo ADI per l'update
    H = -Z*Z';
    X = X + H;
    err = norm(H,'fro')/(norm(X,'fro')*(normA+normB));
    k = k + 1;
end
if k == itmax
    disp('Warning: raggiunto il numero massimo di iterazioni')
end
```

Bibliografia

[1] R. H. Bartels e G. W. Stewart. «Solution of the Matrix Equation AX + XB = C [F4]». In: *Commun. ACM* 15.9 (set. 1972), pp. 820–826. ISSN: 0001-0782. DOI: 10.1145/361573.361582.

[2] Bernhard Beckermann, Daniel Kressner e Christine Tobler. «An Error Analysis Of Galerkin Projection Methods For Linear Systems With Tensor Product Structure». In: *SIAM Journal on Numerical Analysis* 51 (gen. 2013). DOI: 10.1137/120900204.

[3] Bernhard Beckermann e Alex Townsend. «On the Singular Values of Matrices with Displacement Structure». In: *SIAM Journal on Matrix Analysis and Applications* 38.4 (2017), pp. 1227–1248. DOI: 10.1137/16M1096426.

[4] Peter Benner, Jing-Rebecca Li e Thilo Penzl. «Numerical solution of large-scale Lyapunov equations, Riccati equations, and linear-quadratic optimal control problems». In: *Numerical Linear Algebra with Applications* 15 (nov. 2008), pp. 755–777. DOI: 10.1002/nla.622.

[5] Peter Benner, Ren-Cang Li e Ninoslav Truhar. «On the ADI method for Sylvester equations». In: *Journal of Computational and Applied Mathematics* 233.4 (2009), pp. 1035–1045. ISSN: 0377-0427. DOI: https://doi.org/10.1016/j.cam.2009.08.108.

[6] Peter Benner e Jens Saak. «Numerical solution of large and sparse continuous time algebraic matrix Riccati and Lyapunov equations: A state of the art survey». In: *GAMM-Mitteilungen* 36 (ago. 2013). DOI: 10.1002/gamm.201310003.

[7] Dario Bini. *Dispense per il corso di Analisi Numerica*. Università di Pisa. 2020.

[8] Dario Bini. *Problemi di vibrazioni - Dispense per il corso di Calcolo Scientifico*. Università di Pisa. 2020.

[9] Dario Bini, Milvio Capovani e Ornella Menchi. *Metodi numerici per l'algebra lineare*. Zanichelli, 1988. ISBN: 88-08-06438-7.

[10] Dario A. Bini, Bruno Iannazzo e Beatrice Meini. *Numerical Solution of Algebraic Riccati Equations*. Society for Industrial e Applied Mathematics, 2011. DOI: 10.1137/1.9781611972092.

[11] Dario A. Bini, Stefano Massei e Leonardo Robol. «On the decay of the off-diagonal singular values in cyclic reduction». In: *Linear Algebra and its Applications* 519 (2017), pp. 27–53. ISSN: 0024-3795. DOI: https://doi.org/10.1016/j.laa.2016.12.027.

[12] Sebastian Birk. «Deflated Shifted Block Krylov Subspace Methods for Hermitian Positive Definite Matrices». In: 2018.

[13] V. Druskin, L. Knizhnerman e V. Simoncini. «Analysis of the Rational Krylov Subspace and ADI Methods for Solving the Lyapunov Equation». In: *SIAM Journal on Numerical Analysis* 49.5 (2011), pp. 1875–1898. DOI: 10.1137/100813257.

[14] Nancy S. Ellner e Eugene L. Wachspress. «Alternating Direction Implicit Iteration for Systems with Complex Spectra». In: *SIAM Journal on Numerical Analysis* 28.3 (1991), pp. 859–870. DOI: 10.1137/0728045.

[15] G. Golub, S. Nash e C. Van Loan. «A Hessenberg-Schur method for the problem AX + XB= C». In: *IEEE Transactions on Automatic Control* 24.6 (1979), pp. 909–913. DOI: 10.1109/TAC.1979.1102170.

[16] Martin H. Gutknecht. *BLOCK KRYLOV SPACE METHODS FOR LINEAR SYSTEMS WITH Multiple Right-hand Sides: An Introduction*. 2006.

[17] M. Heyouni. «Extended Arnoldi methods for large low-rank Sylvester matrix equations». In: *Applied Numerical Mathematics* 60.11 (2010). Special Issue: 9th IMACS International Symposium on Iterative Methods in Scientific Computing (IISIMSC 2008), pp. 1171–1182. ISSN: 0168-9274. DOI: https://doi.org/10.1016/j.apnum.2010.07.005.

[18] D. Kleinman. «On an iterative technique for Riccati equation computations». In: *IEEE Transactions on Automatic Control* 13.1 (1968), pp. 114–115. DOI: 10.1109/TAC.1968.1098829.

[19] Daniel Kressner, Stefano Massei e Leonardo Robol. «Low-Rank Updates and a Divide-And-Conquer Method for Linear Matrix Equations». In: *SIAM Journal on Scientific Computing* 41.2 (2019), A848–A876. DOI: 10.1137/17M1161038.

[20] Vladimír Kučera. «Algebraic Riccati Equation: Hermitian and Definite Solutions». In: *The Riccati Equation*. A cura di Sergio Bittanti, Alan J. Laub e Jan C. Willems. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 53–88. ISBN: 978-3-642-58223-3. DOI: 10.1007/978-3-642-58223-3_3.

[21] Peter Lancaster. «Explicit Solutions of Linear Matrix Equations». In: *SIAM Review* 12.4 (1970), pp. 544–566. ISSN: 00361445. DOI: 10.2307/2028490. URL: http://www.jstor.org/stable/2028490.

[22] Peter Lancaster e Leiba Rodman. *Algebraic Riccati Equations*. Oxford University Press, New York, gen. 2002. ISBN: 978-0-19-853795-3.

[23] Jing-Rebecca Li e Jacob White. «Low Rank Solution of Lyapunov Equations». In: *SIAM Journal on Matrix Analysis and Applications* 24.1 (2002), pp. 260–280. DOI: 10.1137/S0895479801384937.

[24] Yiding Lin e Valeria Simoncini. «Minimal residual methods for large scale Lyapunov equations». In: *Applied Numerical Mathematics* 72.Complete (2013), pp. 52–71. DOI: 10.1016/j.apnum.2013.04.004.

[25] D. W. Peaceman e H. H. Rachford Jr. «The Numerical Solution of Parabolic and Elliptic Differential Equations». In: *Journal of the Society for Industrial and Applied Mathematics* 3.1 (1955), pp. 28–41. DOI: 10.1137/0103003.

[26] Thilo Penzl. «A Cyclic Low-Rank Smith Method for Large Sparse Lyapunov Equations». In: *SIAM Journal on Scientific Computing* 21.4 (1999), pp. 1401–1418. DOI: 10.1137/S1064827598347666.

[27] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Second. Society for Industrial e Applied Mathematics, 2003. DOI: 10.1137/1.9780898718003. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898718003. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898718003.

[28] Yousef Saad e X V Gv. «Numerical Solution of Large Lyapunov Equations». In: *in Signal Processing, Scattering and Operator Theory, and Numerical Methods, Proc. MTNS-89*. Birkhauser, 1990, pp. 503–511.

[29] V. Simoncini. «A New Iterative Method for Solving Large-Scale Lyapunov Matrix Equations». In: *SIAM Journal on Scientific Computing* 29.3 (2007), pp. 1268–1288. DOI: 10.1137/06066120X.

[30] V. Simoncini. «Computational Methods for Linear Matrix Equations». In: *SIAM Review* 58.3 (2016), pp. 377–441. DOI: 10.1137/130912839.

[31] J. J. Sylvester. «Sur l'equations en matrices px = xq». In: *Comptes Rendus Acad. Sci. Paris* (1884), pp. 67–71, 115–116.