Alma Mater Studiorum · Università di Bologna

SCHOOL OF SCIENCE

Master's Degree in Mathematics

Low-Rank Krylov Subspace Methods for the Solution of Non-Autonomous ODEs via the $\star$-Algebra

Master's Thesis in Numerical Analysis

Supervisors:
Prof. Davide Palitta
Prof. Stefano Pozza

Candidate:
Gabriele Emilio Maria Marconi

Academic Year 2024/2025

Alla mia famiglia

Introduction

ODE systems represent one of the fundamental languages in mathematical modeling. They can describe the evolution in time of physical, chemical, biological, and economic quantities, and they arise from many different contexts, from dynamical systems to numerical simulations of microscopic phenomena. In the linear case, in many relevant applications, one has to deal with systems of the form

$$u'(t) = A(t)u(t), \quad u : \mathbb{R} \to \mathbb{C}^N,$$

where the matrix $A(t)$ depends explicitly on time (non-autonomous case). For constant coefficients, or more generally, when the coefficient matrix commutes with itself at different times, the solution has a simple closed form given through the matrix exponential. However, in general, the solution can be expressed by a more complex object, known as *time-ordered exponential (TOE)*, which requires more delicate strategies from both a theoretical and a numerical point of view.

An example related to real-world problems in chemistry is the spin dynamics during nuclear magnetic resonance (NMR). In this type of model, the quantum evolution of the system is described by a non-autonomous Schrödinger equation, where $A(t) = -i\mathcal{H}(t)$, with $\mathcal{H}(t)$ denoting the time-dependent Hamiltonian. One aspect that makes this simulation particularly challenging is the growth of the state-space dimension, which depends on the number of spins as $N = 2^{N_s}$. Thus, for moderate values of N_s we have limitations in terms of memory and computational cost. This motivates the necessity of developing some faster methods that allow us to handle larger dimensions, remaining reliable and comparable to pre-existing methods.

In this thesis, we adopt a recent and promising strategy introduced in [1], [2] to solve non-autonomous systems: the $\star$-algebra, an algebra of bivariate distributions based on a convolution-like operation known as $\star$-product. The *TOE* in that framework can be obtained through the notion of $\star$-resolvent, which transforms a differential problem into an algebraic one. As explained in Chapter 1, this reformulation is not only elegant but allows us to obtain, after discretization, a linear problem in the usual matrix algebra. Indeed, the discretization, which is based on orthonormal Legendre polynomials, allows us to approximate the $\star$-product of distributions with the product of the respective

coefficient matrices, as highlighted in Chapter 2. Therefore, as shown in Section 2.2.1, the solution of the ODE can be obtained by solving a linear system or, equivalently, a matrix equation with several potential advantages in terms of implementation and efficiency.

This motivates the second important theme of the thesis, discussed in Chapter 3: the numerical solution of linear multiterm matrix equations, i.e. problems of the form

$$\sum_{i=1}^{d} A_i X B_i = F, \tag{1}$$

which generalize well-known and widely studied classes such as Sylvester, Lyapunov, and Stein matrix equations. When the number of terms d exceeds two, equation (1) becomes extremely challenging. From the theoretical point of view, necessary and sufficient conditions for the uniqueness and for good conditioning of the solution are still lacking, while from the algorithmic point of view many classical approaches require care to remain competitive on large-scale instances, and for these reasons they represent an active research frontier. One of the most widely used approaches consists of rewriting the matrix equation as a linear system via vectorization and Kronecker products, but that may lead to matrices of prohibitive size, making it unfeasible in large-scale settings. Therefore, one essential goal is to construct, or modify, solvers that work in matrix form using the linear operator without forming large matrices and, at the same time, limiting the storage cost of the iterates.

Assuming that the right-hand side of the matrix equation has rank one $F = fg^\top$, we have that, in many applications, including our case of interest, the solution X also has low numerical rank. This leads to another important theme of the thesis, explained in Section 3.3: the improvement of the solvers via low-rank techniques and truncations. Indeed, we can use this low-rank approximation to represent the iterates of the methods in a factored form $X = UV^\top$, reducing memory and computational cost. However, step by step the rank might increase due to linear combinations and the application of the linear operator, making compression strategies (e.g., "skinny" QR and truncated SVD decompositions) necessary. In this context, this thesis develops matrix-form variants of Krylov subspace methods for non-symmetric problems (in particular, BiCG and BiCGSTAB) and their low-rank counterparts, with a focus on their numerical stability and on the trade-off between compression and convergence speed.

Finally, to link the numerical methods with a concrete application context, Chapter 4 is devoted to numerical experiments on a spin dynamics model for NMR. Starting from real data, we solve the non-autonomous system via the $\star$-based procedure, applying the different Krylov methods, both in standard and low-rank form, to the resulting multiterm linear matrix equation. The results are compared by evaluating the accuracy, the residual

norms, the convergence speed, the computational cost, the runtime, and the behavior as the problem size increases, for different choices of model and discretization parameters.

Contents

Chapter 1

$\star$-Algebra Reformulation of Non-Autonomous ODEs

Consider the matrix-valued function $\tilde{A}(t) \in \mathbb{C}^{N\times N}$ analytic on the interval $\mathcal{I} = [a, b] \subset \mathbb{R}$, i.e., a matrix whose entries are complex-valued analytic functions in time, and let $\tilde{U}(t)$ be the solution of the non-autonomous system of ordinary differential equations

$$\frac{d}{dt}\tilde{U}_s(t) = \tilde{A}(t)\tilde{U}_s(t), \quad \tilde{U}_s(s) = I_N, \quad t \in [s, b] \subset \mathcal{I},$$

where I_N is the identity $N \times N$ matrix. The solution is an $N \times N$ matrix-valued function, known as the *time-ordered exponential* (*TOE*) of $\tilde{A}(t)$, and in general (we will see more details in Section 1.4) the *TOE* does not have a simple explicit expression in terms of $\tilde{A}(t)$. The goal of this first chapter is to summarize novel results on the so-called $\star$-product, a non-commutative convolution-like product, which allows us to express the *TOE* in a compact form that, as shown in Chapter 2, can be discretized to obtain a linear system in the usual matrix algebra. To achieve this, after introducing some preliminary notions concerning distribution theory in Section 1.1, we will rigorously define the $\star$-product and the resulting $\star$-algebra, describing its main features and properties in Section 1.2. Then we will reformulate the solution of the scalar problem (Section 1.3) and the matrix problem previously stated (Section 1.4) using the $\star$-algebra. Introducing the scalar case first will help to simplify the discussion when moving to the matrix case, since, as we will see, the extension is straightforward. The material introduced and developed in [1], [2], [3], [4], [5], [6] provides the foundation for this chapter; at the same time, this thesis revises some calculations and proofs, presenting them step by step.

1.1 Preliminaries

We first recall some notions from the theory of distributions that will be essential for the following chapters. The theory of distributions, formally introduced by *Laurent*

Schwartz in 1950, extends the classical concept of functions and allows differentiation and other operations to be defined in a generalized sense. This section is based on [7], which we take as our main reference. For simplicity, distributions are introduced here in the real-valued setting. All definitions and results can be extended in a straightforward way to the complex case, which is the framework relevant for the applications considered later, by replacing $\mathbb{R}$ with $\mathbb{C}$ and considering continuous $\mathbb{C}$-linear functionals on the space of complex-valued test functions.

Definition 1.1 (**Distribution**). *A distribution (or generalized function) on an open set $\Omega \subset \mathbb{R}^n$ is a continuous linear functional*

$$T : \mathcal{D}(\Omega) \to \mathbb{R},$$

where $\mathcal{D}(\Omega) = C_0^\infty(\Omega)$ denotes the space of test functions, i.e., infinitely differentiable functions with compact support in Ω. We denote with $\mathcal{D}'(\Omega)$ the space of distributions on Ω.

Remark 1.2. Every locally integrable function $f \in L^1_{\text{loc}}(\Omega)$ defines a distribution T_f by

$$T_f(\varphi) = \langle T_f, \varphi \rangle = \int_\Omega f(x)\, \varphi(x)\, dx, \quad \forall \varphi \in \mathcal{D}(\Omega).$$

Definition 1.3 (**Dirac delta distribution**). *The Dirac delta distribution, denoted by δ, is the distribution on an open set $\Omega \subset \mathbb{R}^n$ defined by*

$$\delta(\varphi) = \varphi(0), \quad \forall \varphi \in \mathcal{D}(\Omega),\ 0 \in \Omega,$$

Hence,

$$\langle \delta, \varphi \rangle = \int_\Omega \delta(x)\, \varphi(x)\, dx = \varphi(0).$$

More generally, the delta distribution centered at a point $x_0 \in \Omega$ is defined as

$$\delta_{x_0}(\varphi) = \varphi(x_0).$$

Definition 1.4 (**Derivatives of a distribution**). *Let $\Omega \subseteq \mathbb{R}^n$ be an open set and let $T \in \mathcal{D}'(\Omega)$ be a distribution. For each $i = 1, \ldots, n$, the partial derivative of T with respect to x_i is the distribution $\partial_i T \in \mathcal{D}'(\Omega)$ defined by*

$$\langle \partial_i T, \varphi \rangle = -\langle T, \partial_i \varphi \rangle, \quad \forall \varphi \in \mathcal{D}(\Omega).$$

If $T = T_f$ is the regular distribution associated with a locally integrable function $f \in L^1_{loc}(\Omega)$, then

$$\langle \partial_i T_f, \varphi \rangle = -\int_\Omega f(x)\, \partial_i \varphi(x)\, dx.$$

More generally, for any multi-index $\alpha \in \mathbb{N}^n$,

$$\langle \partial^\alpha T, \varphi \rangle = (-1)^{|\alpha|} \langle T, \partial^\alpha \varphi \rangle, \quad \forall \varphi \in \mathcal{D}(\Omega).$$

If $T = T_f$ is the regular distribution associated with a locally integrable function $f \in L^1_{\text{loc}}(\Omega)$, then

$$\langle \partial^\alpha T_f, \varphi \rangle = (-1)^{|\alpha|} \int_\Omega f(x)\, \partial^\alpha \varphi(x)\, dx.$$

Definition 1.5 (**Derivatives of the Dirac delta distribution**). *Let $\Omega \subseteq \mathbb{R}^n$ be an open set and $x_0 \in \Omega$. The derivative of the Dirac delta distribution δ_{x_0} with respect to x_i is the distribution $\partial_i \delta_{x_0}$ defined by*

$$\langle \partial_i \delta_{x_0}, \varphi \rangle = -\langle \delta_{x_0}, \partial_i \varphi \rangle = -\partial_i \varphi(x_0), \quad \forall \varphi \in \mathcal{D}(\Omega).$$

In integral form, this can be written as

$$\int_\Omega \partial_i \delta_{x_0}(x)\, \varphi(x)\, dx = -\int_\Omega \delta_{x_0}(x)\, \partial_i \varphi(x)\, dx = -\partial_i \varphi(x_0).$$

More generally, for any multi-index $\alpha \in \mathbb{N}^n$,

$$\langle \partial^\alpha \delta_{x_0}, \varphi \rangle = (-1)^{|\alpha|}\, \partial^\alpha \varphi(x_0).$$

Definition 1.6 (**Heaviside function**). *The Heaviside function is defined as:*

$$\Theta(t-s) = \begin{cases} 0, & \text{if } t < s, \\ 1, & \text{if } t \geq s. \end{cases}$$

Hence, $\Theta(t-s)$ represents a switch that turns on at time $t = s$. It is useful when we are considering an input function $\tilde{f}(t)$ that turns on only after a time s, indeed, we can write $f(t,s) = \tilde{f}(t)\Theta(t-s)$, which will be equal to $\tilde{f}(t)$ for $t \geq s$, 0 otherwise. The parameter s is sometimes called the *activation time*.

Proposition 1.7 (**Heaviside derivatives**). *It holds that, in the distributional sense,*

$$\partial_x \Theta(x) = \delta(x).$$

Moreover, for all $k \in \mathbb{N}$,

$$\partial_x^k \Theta(x) = \delta^{(k-1)}(x).$$

Proof. We have that, for all $\varphi \in C_0^\infty(\mathbb{R})$,

$$\begin{aligned} \langle \partial_x \Theta, \varphi \rangle &= -\int_{-\infty}^{\infty} \Theta(x) \partial_x \varphi(x) dx = -\int_0^\infty 1 \cdot \partial_x \varphi(x) dx \\ &= -[\varphi(x)]_0^\infty = \varphi(0) = \langle \delta, \varphi \rangle, \end{aligned}$$

that is

$$\partial_x \Theta = \delta.$$

Let us prove the more general case, for all $k \in \mathbb{N}$,

$$\begin{aligned}\langle \partial_x^k \Theta, \varphi \rangle &= \langle \partial_x(\partial_x^{k-1}\Theta), \varphi \rangle = -\langle \partial_x^{k-1}\Theta, \partial_x \varphi \rangle \\ &= \cdots = (-1)^{k-1}\langle \partial_x \Theta, \partial_x^{k-1}\varphi \rangle \\ &= (-1)^{k-1}\langle \delta, \partial_x^{k-1}\varphi \rangle = \langle \delta^{(k-1)}, \varphi \rangle \quad \forall \varphi \in C_0^\infty(\mathbb{R}).\end{aligned}$$

Thus,

$$\partial_x^k \Theta = \delta^{(k-1)}, \text{ for all } k \geq 1.$$

□

In summary, the distributional framework introduced here will serve as the foundation for the description and analysis of the ⋆-algebra developed in the next section.

1.2 The ⋆-Product: Definition and Properties

We start by defining the set $\mathcal{A}(\mathcal{I})$ of bivariate distributions for which there exists a finite integer k such that

$$f(t,s) = \tilde{f}_0(t,s)\Theta(t-s) + \tilde{f}_1(t,s)\delta(t-s) + \cdots + \tilde{f}_k(t,s)\delta^{(k)}(t-s),$$

where $\tilde{f}_0(t,s), \cdots, \tilde{f}_k(t,s)$ are analytic functions in both t, s over the interval $\mathcal{I} = [a,b]$, $\Theta(t-s)$ is the Heaviside function, $\delta(t-s)$ stands for the Dirac delta distribution δ_s, and $\delta'(t-s), \cdots, \delta^{(k)}(t-s)$ are its derivatives. We also define the subset $\mathcal{A}_\Theta(\mathcal{I}) \subset \mathcal{A}(\mathcal{I})$ formed by distributions of the kind $f(t,s) = \tilde{f}_0(t,s)\Theta(t-s)$. Over this set, we can define an algebra with the product defined below, which will be described in detail later. It was first introduced in [3], and was subsequently extended and formalized in [8].

Definition 1.8 (**⋆-product**). *The ⋆-product of $f, g \in \mathcal{A}(\mathcal{I})$ is the non-commutative product defined as*

$$(f \star g)(t,s) := \int_{\mathcal{I}} f(t,\tau) g(\tau,s) d\tau \in \mathcal{A}(\mathcal{I}).$$

This product is an extension of convolution between functions. Throughout the following, we will omit the argument (t,s) of the distributions whenever it is clear from the context.

Definition 1.9 (**⋆-resolvent**). *The ⋆-resolvent of $h \in \mathcal{A}_\Theta(\mathcal{I})$ is defined as*

$$R_\star(h) := \sum_{j=0}^{\infty} h^{\star j} = h^{\star 0} + h^{\star 1} + h^{\star 2} + \cdots + h^{\star j} + \cdots,$$

where, by convention, $h^{\star 0}(t,s) = \delta(t-s)$, $h^{\star 1}(t,s) = h(t,s)$ *and* $h^{\star j}(t,s) = h(t,s) \star \cdots \star h(t,s)$ j *times, for* $j \geq 2$, $(t,s) \in \mathcal{I} \times \mathcal{I}$.

Remark 1.10. Note that $R_\star(h)$ is well defined (i.e., convergent) for every $h \in \mathcal{A}_\Theta(\mathcal{I})$. Indeed, for $j \geq 1$,

$$h^{\star j}(t,s) = \int_{\mathcal{I}^{j-1}} h(t,\tau_1) \cdots h(\tau_{j-1},s) d\tau_1 \cdots d\tau_{j-1},$$

and since $h \in \mathcal{A}_\Theta(\mathcal{I})$, we can write $h(\tau_i, \tau_{i+1}) = \tilde{h}(\tau_i, \tau_{i+1})\Theta(\tau_i - \tau_{i+1})$. Hence

$$h^{\star j}(t,s) = \int_{\mathcal{I}^{j-1}} \tilde{h}(t,\tau_1) \cdots \tilde{h}(\tau_{j-1},s)\Theta(t-\tau_1) \cdot \Theta(\tau_{j-1}-s) d\tau_1 \cdots d\tau_{j-1}.$$

Now, since $\tilde{h}$ is analytic, hence continuous, over $\mathcal{I}\times\mathcal{I}$ that is a compact set, by Weierstrass theorem we have that each $\left|\tilde{h}(t,s)\right| \leq M$ for $M := \max_{(t,s)\in I\times I} \left|\tilde{h}(t,s)\right| \geq 0$ and since $\Theta(t-\tau_1)\cdots\Theta(\tau_{j-1}-s) = 1$ if $s \leq \tau_{j-1} \leq \cdots \leq \tau_1 \leq t$ and 0 otherwise, we can estimate the integral as:

$$\left|h^{\star j}(t,s)\right| \leq M^j \Theta(t-s) \int_{s\leq\tau_{j-1}\leq\cdots\leq\tau_1\leq t} d\tau_1 \cdots d\tau_{j-1} = \Theta(t-s)\frac{M^j(t-s)^{j-1}}{(j-1)!},$$

which converges to zero for $j \to +\infty$. Hence, the $\star$-resolvent series is convergent since

$$\begin{aligned} |R_\star(h)| &:= \delta(t-s) + \left|\sum_{j=1}^{\infty} h^{\star j}\right| = \delta(t-s) + \left|\sum_{j=0}^{\infty} h^{\star j+1}\right| \\ &\leq \delta(t-s) + \Theta(t-s)M\sum_{j=0}^{\infty}\frac{M^j(t-s)^j}{j!} = \delta(t-s) + \Theta(t-s)M\exp(M(t-s)). \end{aligned}$$

Proposition 1.11. *The* $\star$*-algebra* $(\mathcal{A}(\mathcal{I}), +, \star, \delta)$ *is a unitary, associative algebra on* $\mathbb{C}$*, with the identity element with respect to* $\star$*-multiplication given by the Dirac delta distribution* $1_\star = \delta(t-s)$*. Moreover* $(\mathcal{A}_\Theta(\mathcal{I}) \cup \{\delta\}, +, \star, \delta)$ *is a unitary, associative sub-algebra of* $\mathcal{A}(\mathcal{I})$.

Proof. We prove the second part of the statement; the proof of the first part can be found in [8].

1. Firstly we prove that $\mathcal{A}_\Theta(\mathcal{I})$ is a vector space on $\mathbb{C}$.

   - Closure with respect to the sum: if $f = \tilde{f}\Theta$ and $g = \tilde{g}\Theta$, then

     $$(f+g)(t,s) = (\tilde{f}(t,s) + \tilde{g}(t,s))\Theta(t-s) \in \mathcal{A}_\Theta(\mathcal{I}), \quad (t,s) \in \mathcal{I}\times\mathcal{I}.$$

   - Closure with respect to the multiplication by a scalar: for $\alpha \in \mathbb{C}$, $(\alpha f)(t,s) = (\alpha\tilde{f}(t,s))\Theta(t-s) \in \mathcal{A}_\Theta(\mathcal{I})$.

- Other standard properties such as the commutativity of the sum, the existence of the unitary additive element 0, the additive inverse $-f$, are immediate.

2. Now we check the closure of the $\star$-product on $\mathcal{A}_\Theta(\mathcal{I})$:

$$\begin{aligned}(f\star g)(t,s) &= \tilde{f}(t,s)\Theta(t-s)\star\tilde{g}(t,s)\Theta(t-s)\\ &= \int_{\mathcal{I}}\tilde{f}(t,\tau)\Theta(t-\tau)\tilde{g}(\tau,s)\Theta(\tau-s)d\tau\\ &= \Theta(t-s)\int_s^t\tilde{f}(t,\tau)\tilde{g}(\tau,s)d\tau\\ &= \Theta(t-s)\tilde{F}(t,s)\in\mathcal{A}_\Theta(\mathcal{I}),\end{aligned}$$

indeed $\tilde{F}(t,s)$ is still analytic since $\tilde{f},\tilde{g}$ are also analytic.

3. The bilinearity of the $\star$-product follows directly from the linearity of the integral. We have, for $f=\tilde{f}\Theta$, $g=\tilde{g}\Theta$, $h=\tilde{h}\Theta$ and for $\alpha,\beta\in\mathbb{C}$,

$$\begin{aligned}((\alpha f+\beta h)\star g)(t,s) &= \int_{\mathcal{I}}(\alpha\tilde{f}(t,\tau)\Theta(t-\tau)+\beta\tilde{h}(t,\tau)\Theta(t-\tau))\tilde{g}(\tau,s)\Theta(\tau-s)d\tau\\ &= \alpha\int_{\mathcal{I}}\tilde{f}(t,\tau)\Theta(t-\tau)\tilde{g}(\tau,s)\Theta(\tau-s)d\tau+\cdots\\ &+\beta\int_{\mathcal{I}}\tilde{h}(t,\tau)\Theta(t-\tau)\tilde{g}(\tau,s)\Theta(\tau-s)d\tau\\ &= \alpha(f\star g)(t,s)+\beta(h\star g)(t,s).\end{aligned}$$

Clearly by the same argument, bilinearity in the other variable also holds.

4. For $f,g,h\in\mathcal{A}_\Theta(\mathcal{I})$ as above, we have:

$$\begin{aligned}(f\star(g\star h))(t,s) &= \int_{\mathcal{I}}f(t,\tau)(g\star h)(\tau,s)d\tau=\int_{\mathcal{I}}f(t,\tau)\left(\int_{\mathcal{I}}g(\tau,\sigma)h(\sigma,s)d\sigma\right)d\tau\\ &= \int_{\mathcal{I}}\tilde{f}(t,\tau)\Theta(t-\tau)\left(\int_{\mathcal{I}}\tilde{g}(\tau,\sigma)\tilde{h}(\sigma,s)\Theta(\tau-\sigma)\Theta(\sigma-s)d\sigma\right)d\tau.\end{aligned}$$

Let us define $\Phi_{t,s}(\tau,\sigma)=\tilde{f}(t,\tau)\,\tilde{g}(\tau,\sigma)\,\tilde{h}(\sigma,s)\,\Theta(t-\tau)\,\Theta(\tau-\sigma)\,\Theta(\sigma-s)$. Since $\tilde{f},\tilde{g},\tilde{h}$ are analytic on $\mathcal{I}\times\mathcal{I}$, they are continuous and hence bounded on the compact set $\mathcal{I}\times\mathcal{I}$, thanks to the Weierstrass theorem. The product $\Theta(t-\tau)\Theta(\tau-\sigma)\Theta(\sigma-s)$ is the characteristic function of the triangular region $D(t,s)=\{(\tau,\sigma)\in\mathcal{I}\times\mathcal{I}:\ s\leq\sigma\leq\tau\leq t\}$, which has finite measure. Therefore, $|\Phi_{t,s}|$ is bounded by a constant on a set of finite measure, and hence $\Phi_{t,s}\in L^1(\mathcal{I}\times\mathcal{I})$. By Fubini's theorem the order of integration can be exchanged, obtaining:

$$\int_{\mathcal{I}}\left(\int_{\mathcal{I}}\tilde{f}(t,\tau)\Theta(t-\tau)\tilde{g}(\tau,\sigma)\Theta(\tau-\sigma)d\tau\right)\tilde{h}(\sigma,s)\Theta(\sigma-s)d\sigma=\int_{\mathcal{I}}(f\star g)(t,\sigma)h(\sigma,s)d\sigma.$$

Hence,

$$(f\star(g\star h))(t,s)=((f\star g)\star h)(t,s)=(f\star g\star h)(t,s).$$

5. Distributivity with respect to the sum has already been proved in (3).

6. Let us verify that $\delta(t,s) = \delta(t-s) \in \mathcal{A}(\mathcal{I})$ is the $\star$-multiplication identity element:

$$f(t,s) \star \delta(t-s) = \int_{\mathcal{I}} \tilde{f}(t,\tau)\Theta(t-\tau)\delta(\tau-s)d\tau = \tilde{f}(t,s)\Theta(t-s) = f(t,s),$$

by the definition of the Dirac delta distribution.

□

Proposition 1.12 (**Main properties of the $\star$-product**). *The following properties hold:*

1. *Under certain assumptions on $f \in \mathcal{A}_\Theta(\mathcal{I})$ (see [3], [8]), there exists $f^{-\star}$, the $\star$-inverse of f: $f \star f^{-\star} = f^{-\star} \star f = \delta$.*

2. *δ' is the $\star$-inverse of Θ: $\delta' \star \Theta = \Theta \star \delta' = \delta$.*

3. *Dirac delta derivatives: $\delta^{(i)} \star \delta^{(j)} = \delta^{(i+j)}$.*

4. *$\star$-resolvent $R_\star(h) = (\delta - h)^{-\star}$, $h \in \mathcal{A}_\Theta(\mathcal{I})$.*

Proof. 1. The existence of the $\star$-inverse has been proved in [3], [8].

2. By the definition of the first derivative of the Dirac delta and applying the Leibniz derivation rule in the distributional sense, using Proposition 1.7, the following formula holds:

$$\begin{aligned}\delta'(t,s) \star f(t,s) &= \delta'(t-s) \star \tilde{f}(t,s)\Theta(t-s) \\ &= \int_{\mathcal{I}} \delta'(t-\tau)\tilde{f}(\tau,s)\Theta(\tau-s)d\tau \\ &= \Big(\partial_t \tilde{f}(t,s)\Big)\,\Theta(t-s) + \tilde{f}(s,s)\delta(t-s) \in \mathcal{A}(\mathcal{I}).\end{aligned}$$

It follows that

$$(\delta' \star \Theta)(t,s) = 0 \cdot \Theta(t-s) + \delta(t-s) = \delta(t-s) = 1_\star.$$

Therefore δ' and Θ are inverses of each other.

3. Let us consider a test function $\varphi \in C_0^\infty(\mathcal{I}\times\mathcal{I})$. By definition,

$$\langle \delta^{(i)} \star \delta^{(j)}, \varphi\rangle = \iiint_{\mathcal{I}\times\mathcal{I}\times\mathcal{I}} \delta^{(i)}(t-\tau)\,\delta^{(j)}(\tau-s)\,\varphi(t,s)\,dt\,ds\,d\tau.$$

Using the definition of Dirac's delta derivative distribution

$$\int_{\mathcal{I}} \delta^{(k)}(x-x_0)\,f(x)\,dx = (-1)^k f^{(k)}(x_0),$$

first in the variable t and then in the variable τ, we get

$$\langle \delta^{(i)} \star \delta^{(j)}, \varphi \rangle = \int_{\mathcal{I}} (-1)^{i+j} \, \partial_t^{i+j} \varphi(t,s)\big|_{t=s} \, ds = \langle \delta^{(i+j)}(t-s), \varphi \rangle.$$

Since this holds for all $\varphi \in C_0^\infty(\mathcal{I} \times \mathcal{I})$, it follows that

$$\delta^{(i)} \star \delta^{(j)} = \delta^{(i+j)}.$$

4. Let the $\star$-resolvent of $h \in \mathcal{A}_\Theta(\mathcal{I})$ be defined as

$$R_\star(h) = \sum_{j=0}^{\infty} h^{\star j}, \qquad \text{with } h^{\star 0} = \delta, \; h^{\star(j+1)} = h \star h^{\star j}.$$

We want to prove that $R_\star(h)$ is the $\star$-inverse of $(\delta - h)$, that is,

$$(\delta - h) \star R_\star(h) = R_\star(h) \star (\delta - h) = \delta.$$

Consider the partial sums

$$S_N := \sum_{j=0}^{N} h^{\star j}.$$

By bilinearity and associativity of the $\star$-product, we have

$$(\delta - h) \star S_N = \delta \star S_N - h \star S_N = S_N - \sum_{j=0}^{N} h \star h^{\star j} = \sum_{j=0}^{N} h^{\star j} - \sum_{j=1}^{N+1} h^{\star j} = \delta - h^{\star(N+1)}.$$

Analogously,

$$S_N \star (\delta - h) = \delta - h^{\star(N+1)}.$$

For every $h \in \mathcal{A}_\Theta(\mathcal{I})$, as noted in Remark 1.10, we have the estimate

$$|h^{\star j}(t,s)| \leq \frac{M^{\,j}}{(j-1)!} (t-s)^{j-1} \Theta(t-s),$$

which implies $h^{\star(N+1)} \to 0$ as $N \to \infty$. Taking the limit, we obtain

$$(\delta - h) \star R_\star(h) = R_\star(h) \star (\delta - h) = \delta.$$

Therefore, the resolvent $R_\star(h)$ is indeed the $\star$-inverse of $(\delta - h)$:

$$R_\star(h) = (\delta - h)^{-\star}.$$

□

As we have seen, the $\star$-algebra defines a consistent non-commutative convolution structure endowed with clear algebraic properties. This framework will now be employed to express the solution of the scalar problem and the vector problem in a compact and convenient form.

1.3 Scalar ODE Reformulation

In order to better understand the more relevant case, namely the vector (or matrix) case that arises when dealing with a system of ordinary differential equations, let us start by considering the following non-autonomous scalar ODE, for every initial time $s \in \mathcal{I} = [a, b] \subset \mathbb{R}$ and analytic complex-valued scalar function $\tilde{f}$,

$$\frac{d}{dt}\tilde{u}_s(t) = \tilde{f}(t)\tilde{u}_s(t), \quad \tilde{u}_s(s) = 1, \quad t \in [s, b] \subset \mathcal{I},$$

in the unknown scalar function $\tilde{u}_s(t) : [s, b] \to \mathbb{C}$. To be more precise, this equation represents a family of ordinary differential equations, since it depends on the initial time instant $s \in \mathcal{I}$. We can therefore interpret our solution $\tilde{u}_s(t)$ as a function defined on the entire domain $\mathcal{I} \times \mathcal{I}$:

$$u(t, s) : \mathcal{I} \times \mathcal{I} \to \mathbb{C}, \qquad u(t, s) = \tilde{u}_s(t)\,\Theta(t - s).$$

Including the Heaviside function $\Theta(t - s)$ in the expression of the solution allows us to define the latter over the entire interval $\mathcal{I}$, since for $t < s$ the function is set to zero. Now, our solution, viewed as

$$u(t, s) = \tilde{u}_s(t)\,\Theta(t - s) \in \mathcal{A}_\Theta(\mathcal{I}),$$

actually represents a distribution, as discussed in the previous section. Therefore, here we are committing a slight abuse of notation since the function $\tilde{u}_s(t)$ is not defined for $t < s$. This scalar case has been mainly discussed in [6], where more information can be found.

In this case, we can easily find an analytic expression for the solution

$$\tilde{u}_s(t) = \exp\left(\int_s^t \tilde{f}(\tau)\,d\tau\right).$$

Indeed, by separating the variables in the differential equation

$$\frac{d}{dt}\tilde{u}_s(t) = \tilde{f}(t)\tilde{u}_s(t), \quad \tilde{u}_s(s) = 1,$$

we obtain

$$\frac{\tilde{u}_s'(t)}{\tilde{u}_s(t)} = \tilde{f}(t).$$

Integrating both sides from s to t gives

$$\int_s^t \frac{\tilde{u}_s'(\tau)}{\tilde{u}_s(\tau)}\,d\tau = \int_s^t \tilde{f}(\tau)\,d\tau.$$

The left-hand side equals $\log \tilde{u}_s(t) - \log \tilde{u}_s(s)$; since $u(s, s) = 1$, we get

$$\log \tilde{u}_s(t) = \int_s^t \tilde{f}(\tau)\,d\tau.$$

Exponentiating both sides yields the desired expression:

$$\tilde{u}_s(t) = \exp\left(\int_s^t \tilde{f}(\tau)d\tau\right).$$

However, by exploiting the properties and tools of the ⋆-algebra introduced in the previous chapter, the solution to this problem can also be expressed in an alternative form via ⋆-multiplication.

Let us now rewrite the solution in terms of the ⋆-product. We recall our problem

$$\tilde{u}_s'(t) = \tilde{f}(t)\tilde{u}_s(t), \quad \tilde{u}_s(s) = 1, \quad t, s \in I,\ t \geq s,$$

that is,

$$u'(t,s) = \tilde{f}(t)u(t,s), \quad u(s,s) = 1, \quad t, s \in I, \quad \textit{for}\ \ u(t,s) = \tilde{u}_s(t)\Theta(t-s). \tag{1.1}$$

Proposition 1.13. *The solution of the problem* (1.1) *is given, in terms of the ⋆-algebra, by:*

$$u(t,s) = (R_\star(f) \star \Theta)(t,s),$$

with $f(t,s) = \tilde{f}(s)\Theta(t-s)$.

Proof. Since $u(t,s) \in \mathcal{A}_\Theta(\mathcal{I})$, we can compute, using Leibniz rule for distributions:

$$\begin{aligned}
\delta'(t-s) \star u(t,s) &= \delta'(t-s) \star \tilde{u}_s(t)\Theta(t-s) \\
&= \tilde{u}_s'(t)\Theta(t-s) + \tilde{u}_s(s)\delta(t-s) \\
&= \tilde{f}(t)u(t,s) + \tilde{u}_s(s)\delta(t-s) \\
&= \tilde{f}(t)(\delta \star u)(t,s) + \tilde{u}_s(s)\delta(t-s) \\
&= \tilde{f}(t)\delta(t-s) \star u(t,s) + \tilde{u}_s(s)\delta(t-s).
\end{aligned}$$

Multiplying on the left both sides of the equation by Θ:

$$\Theta(t-s) \star \delta'(t-s) \star u(t,s) = \Theta(t-s) \star (\tilde{f}(t)\delta(t-s) \star u(t,s) + \tilde{u}_s(s)\delta(t-s))$$

That is, since for Proposition 1.12 $\Theta \star \delta' = \delta$,

$$u(t,s) - \Theta(t-s) \star \tilde{f}(t)\delta(t-s) \star u(t,s) = \Theta(t-s) \star \tilde{u}_s(s)\delta(t-s). \tag{1.2}$$

Now we separately compute the terms:

$$(\Theta \star \tilde{f}\delta)(t,s) = \int_{\mathcal{I}} \Theta(t-\tau)\tilde{f}(\tau)\delta(\tau-s)d\tau = \Theta(t-s)\tilde{f}(s),$$

$$(\Theta \star \tilde{u}_s(s)\delta)(t,s) = \tilde{u}_s(s)\int_{\mathcal{I}} \Theta(t-\tau)\delta(\tau-s)d\tau = \tilde{u}_s(s)\Theta(t-s) = \Theta(t-s),$$

where we have used the definition of δ and the initial condition $u_s(s) = 1$. Replacing those expressions in equation (1.2), we obtain the following:

$$u(t,s) - \tilde{f}(s)\Theta(t-s) \star u(t,s) = \Theta(t-s) \iff (\delta(t-s) - \tilde{f}(s)\Theta(t-s)) \star u(t,s) = \Theta(t-s).$$

Hence, denoting $f(t,s) = \tilde{f}(s)\Theta(t-s)$,

$$\begin{aligned} u(t,s) &= (\delta(t-s) - \tilde{f}(s)\Theta(t-s))^{-\star} \star \Theta(t-s) \\ &= (R_\star(f))(t,s) \star \Theta(t-s) \\ &= (R_\star(f) \star \Theta)(t,s). \end{aligned}$$

□

Remark 1.14. Analogously, we can express the solution as follows:

$$\begin{aligned} u(t,s) &= \Theta(t-s) \star (\delta(t-s) - \tilde{f}(t)\Theta(t-s))^{-\star} \\ &= (\Theta \star R_\star(f_t))(t,s), \end{aligned}$$

where $f_t(t,s) = \tilde{f}(t)\Theta(t-s)$. By Proposition 1.12 we get

$$\begin{aligned} (R_\star(f) \star \Theta)(t,s) &= (\delta(t-s) - \tilde{f}(s)\Theta(t-s))^{-\star} \star \Theta(t-s) \\ &= (\delta'(t-s) \star (\delta(t-s) - \tilde{f}(s)\Theta(t-s)))^{-\star} \\ &= (\delta'(t-s) - \delta'(t-s) \star \tilde{f}(s)\Theta(t-s))^{-\star} \\ &= \Theta \star \delta' \star (\delta' - \delta' \star \tilde{f}(s)\Theta)^{-\star} \\ &= \Theta \star (\delta - \delta' \star \tilde{f}(s)\Theta \star \Theta)^{-\star} \\ &= \Theta \star (\delta - \tilde{f}(t)\Theta)^{-\star} \\ &= (\Theta \star R_\star(f_t))(t,s), \end{aligned}$$

where we have used the property,

$$\delta' \star \tilde{f}(s)\Theta \star \Theta = \tilde{f}(s)\delta \star \Theta = \int_{\mathcal{I}} \tilde{f}(\tau)\delta(t-\tau)\Theta(\tau - s)d\tau = \tilde{f}(t)\Theta(t-s) = f_t(t,s).$$

1.4 Extension to the Vector Case

As we mentioned earlier, we can extend what we have said in the previous sections for the scalar problem to the vector and matrix cases, which naturally arise when dealing with systems of differential equations. This extension is straightforward, and the generalization turns out to be quite intuitive, although it requires some care. This section builds upon [1], [4] which serve as our main references in this context. Let us begin by recalling the matrix problem:

$$\frac{d}{dt}\tilde{U}_s(t) = \tilde{A}(t)\tilde{U}_s(t), \quad \tilde{U}_s(s) = I_N, \quad t \in [s,b] \subset \mathcal{I}, \tag{1.3}$$

where $\tilde{A}(t)$ is an $N \times N$ analytic complex matrix-valued function on $\mathcal{I}$, I_N is the identity $N \times N$ matrix. The solution $\tilde{U}_s(t)$ is an $N \times N$ matrix-valued function known as *time-ordered exponential (TOE)*, and it does not generally admit a simple analytic expression. As we did for the scalar case, we can extend it to $\mathcal{I}\times\mathcal{I}$ by setting $U(t,s) := \tilde{U}_s(t)\,\Theta(t-s)$.

It is known that, when $\tilde{A}(t)$ commutes with itself at different times, i.e., $\tilde{A}(\tau_1)\tilde{A}(\tau_2) = \tilde{A}(\tau_2)\tilde{A}(\tau_1)$, for all $\tau_1, \tau_2 \in \mathcal{I}$, the *TOE* can be easily expressed as:

$$\tilde{U}_s(t) = \exp\left(\int_s^t \tilde{A}(\tau)d\tau\right).$$

Indeed, if $B(t) = \int_s^t \tilde{A}(\tau)d\tau$, since $\tilde{A}(t)$ commutes with itself, also $B(t)$ will commute, giving:

$$\frac{d}{dt}\exp(B(t)) = B'(t)\exp(B(t)) = \tilde{A}(t)\exp(B(t)).$$

In general, however, $\tilde{U}_s(t)$ has no known simple expression in terms of $\tilde{A}(t)$.

Remark 1.15. Note that condition $\tilde{U}_s(s) = I_N$ is not restrictive, since, given a matrix $B \in \mathbb{C}^{N\times N}$, the matrix-valued function $\tilde{V}_s(t) := \tilde{U}_s(t)B$ solves the ODE

$$\frac{d}{dt}\tilde{V}_s(t) = \tilde{A}(t)\tilde{V}_s(t), \quad \tilde{V}_s(s) = B, \quad t \in [s,b] \subset \mathcal{I}.$$

Indeed, since B is not time-dependent,

$$\frac{d}{dt}\tilde{V}_s(t) = \frac{d}{dt}\tilde{U}_s(t)B = \tilde{A}(t)\tilde{U}_s(t)B = \tilde{A}(t)\tilde{V}_s(t),$$

and

$$\tilde{V}_s(s) = \tilde{U}_s(s)B = I_N B = B,$$

therefore the result follows from the uniqueness of the solution of the initial value problem.

Let us begin by defining the extension of the $\star$-product and its properties to the matrix (vector) case.

Definition 1.16 ($\star$-product). *If $A_1(t,s) \in \mathcal{A}(\mathcal{I})^{N\times L}$, $A_2(t,s) \in \mathcal{A}(\mathcal{I})^{L\times M}$, i.e., generally rectangular, matrices whose elements are distributions in $\mathcal{A}(\mathcal{I})$,*

$$(A_1 \star A_2)(t,s) := \int_{\mathcal{I}} A_1(t,\tau)A_2(\tau,s)d\tau.$$

Remark 1.17. Equivalently, considering the square matrices and developing the matrix product inside the integral, for $A(t,s) = [a_{ij}]_{i,j=1}^N$ and $B(t,s) = [b_{ij}]_{i,j=1}^N$ with $a_{ij}, b_{ij} \in \mathcal{A}(\mathcal{I})$, we obtain

$$\begin{aligned}(A_1 \star A_2)_{ij}(t,s) = \Big[\int_I A_1(t,\tau)\,A_2(\tau,s)\,d\tau\Big]_{ij} &= \sum_{k=1}^N \int_I a_{ik}(t,\tau)\,b_{kj}(\tau,s)\,d\tau \\ &= \sum_{k=1}^N a_{ik}(t,\cdot)\star b_{kj}(\cdot,s).\end{aligned}$$

Proposition 1.18. *$(\mathcal{A}(\mathcal{I})^{N\times N}, +, \star, \delta I_N)$ is a unitary, associative algebra in $\mathbb{C}$, with the identity element with respect to $\star$-multiplication given by $1_\star = \delta(t-s)I_N$. Moreover $(\mathcal{A}_\Theta(\mathcal{I})^{N\times N} \cup \{\delta I_N\}, +, \star, \delta I_N)$ is a unitary, associative sub-algebra of $\mathcal{A}(\mathcal{I})$.*

Proof. The same computations as those carried out in the proof of Proposition 1.11 apply here, where if $A(t,s) = \tilde{A}(t)\,\Theta(t-s)$ and $B(t,s) = \tilde{B}(t)\,\Theta(t-s)$ are two elements of $\mathcal{A}_\Theta(\mathcal{I})^{N\times N}$, with $\tilde{A}(t)$ and $\tilde{B}(t)$ analytic complex matrix-valued functions, then $\tilde{A}(t) + \tilde{B}(t) = (\tilde{A}+\tilde{B})(t)$ is the standard matrix sum, and $\alpha\tilde{A}(t) = (\alpha\tilde{A})(t)$ for $\alpha \in \mathbb{C}$ is the standard elementwise scalar multiplication. □

We now show how the solution to our problem can be expressed using the $\star$-algebra on matrices.

Definition 1.19 ($\star$-resolvent). *The $\star$-resolvent of $A(t,s) \in (\mathcal{A}_\Theta(\mathcal{I}))^{N\times N}$ is defined as*

$$R_\star(A) := \sum_{j=0}^{\infty} A^{\star j} = A^{\star 0} + A^{\star 1} + A^{\star 2} + \cdots + A^{\star j} + \cdots = (\delta I_N - A)^{-\star},$$

where, by convention, $A^{\star 0}(t,s) = \delta(t-s)I_N$, that is the $\star$- product identity in that class, $A^{\star 1}(t,s) = A(t,s)$ and $A^{\star j}(t,s) = A(t,s) \star \cdots \star A(t,s)$ j times, for $j \geq 2$, $(t,s) \in \mathcal{I}\times\mathcal{I}$.

Proposition 1.20. *The solution of the problem* (1.3) *can be given as:*

$$U(t,s) = \Theta(t-s) \star R_\star(A)(t,s), \tag{1.4}$$

where $A(t,s) = \tilde{A}(t)\Theta(t-s)$ and $R_\star(A)$ is the $\star$-resolvent of A.

Proof. The same computations are shown in Proposition 1.13 and Remark 1.14; it suffices to replace u and f by U and A, recalling that the $\star$-product identity for matrices is $\delta(t-s)I_N$. □

Remark 1.21. In practical situations, the initial time s of the evolution is fixed ($s = 0$ for example), and the initial condition is given as a vector $v \in \mathbb{C}^N$:

$$\frac{d}{dt}\tilde{u}(t) = \tilde{A}(t)\tilde{u}(t), \quad \tilde{u}(0) = v, \quad t \in \mathcal{I}. \tag{1.5}$$

By Remark 1.15 the solution of such a problem, with an initial condition different from the identity I_N, is given by $\tilde{u}(t) = \tilde{U}(t)v$, which corresponds in the $\star$-algebra to $u = U\star v\delta$, where $u(t,s) = \tilde{u}(t)\Theta(t-s)$ and $U(t,s) = \tilde{U}(t)\Theta(t-s)$. Using Proposition 1.20 the solution of the problem (1.5) can be obtained by solving the $\star$-linear equation

$$\tilde{u}(t) = u(t,0), \quad u = \Theta \star x, \quad (\delta I_N - \tilde{A}\Theta) \star x = v\delta,\ t \in \mathcal{I}.$$

Although the expression (1.4) is compact, the ⋆-resolvent definition hides an infinite series of nested integrals. Therefore, at first sight, it does not seem like a convenient expression. In the next chapter, we will effectively solve this problem by showing that it is possible to approximate the ⋆-product by the usual matrix-matrix product between matrices. Consequently, the expression (1.4) can be approximated relatively efficiently by solving a linear system. In summary, the ⋆-algebra provides a unified and compact framework for expressing the solution of both scalar and matrix non-autonomous linear ODEs as an algebraic equation. This formulation will serve as the foundation for the discretization strategy presented in the next chapter.

Chapter 2

Discretization of the $\star$-Product and the Resulting Linear System

In the previous chapter, we expressed the solution of the ODE using a linear equation in the $\star$-algebra. However, this formulation is not particularly convenient, since the definition of the $\star$-resolvent hides an apparently infinite number of nested integrals. The goal of this chapter is therefore to transform these $\star$-algebra operations into operations in the usual matrix algebra, by discretizing the problem using (shifted and rescaled) orthonormal Legendre polynomials, although other approaches are being studied, such as changing the polynomial basis to a Fourier basis, for instance. After introducing some preliminary notions, we will present an important result showing that the $\star$-product between two functions can be approximated by the classical matrix-matrix product of their corresponding *Legendre coefficient matrices*. This will allow us to compute the solution in a simple and computationally cheap way, by solving an ordinary linear system in the matrix algebra. As in the previous chapter, we will distinguish the scalar case (Section 2.1) from the matrix case (Section 2.2) to make the exposition clearer and to show that the transition from one to the other is straightforward. Finally, in *Section* 2.2.1, we will see how, in the vector case, the linear system obtained from the discretization can be rearranged into a matrix equation, and how this reformulation can provide certain advantages. In the next chapter, we will discuss numerical methods for solving this matrix equation. The main references for this chapter are, in general, [2] for the polynomial discretization and the computation of the coefficients, [6] for the scalar case, and [1] for the vector case and the reformulation of the problem as a matrix equation.

2.1 The Scalar Case

Consider a sequence of orthonormal polynomial functions $\{p_k\}_k$ over our bounded interval $\mathcal{I}$, i.e.,

$$\int_{\mathcal{I}} p_k(\tau)p_l(\tau)d\tau = \begin{cases} 0, & \text{if } k \neq l, \\ 1, & \text{if } k = l, \end{cases}$$

so that $\{p_k\}_k$ is a basis for the space of analytic functions on $\mathcal{I}$. Note that the functions p_k are not in $\mathcal{A}(\mathcal{I})$, hence we cannot (formally) $\star$-multiply them. Consider a function $f \in \mathcal{A}_\Theta(\mathcal{I})$, namely $f(t,s) = \tilde{f}(t,s)\Theta(t-s)$, that is piecewise analytic since it has a discontinuity for $t = s$. We can use the basis $\{p_k\}_k$ to expand f into the series

$$f(t,s) = \sum_{k=0}^{\infty}\sum_{l=0}^{\infty} f_{k,l}p_k(t)p_l(s), \quad t \neq s, \quad t,s \in \mathcal{I}, \tag{2.1}$$

with coefficients

$$f_{k,l} = \int_{\mathcal{I}}\int_{\mathcal{I}} f(\tau,\rho)p_k(\tau)p_l(\rho)d\rho d\tau.$$

In this thesis, the basis $\{p_k\}_k$ is chosen as the sequence of shifted and rescaled orthonormal Legendre polynomials over $\mathcal{I} = [a,b]$ (e.g., [9]), but different choices can be made, for instance by using the Fourier basis.

If we consider the following infinite-size matrix and vector

$$F := \begin{pmatrix} f_{0,0} & f_{0,1} & \cdots \\ f_{1,0} & f_{1,1} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \quad \varphi(\tau) := \begin{pmatrix} p_0(\tau) \\ p_1(\tau) \\ \vdots \end{pmatrix},$$

by the definition of matrix-vector product, we get the matrix representation of the series:

$$f(t,s) = \varphi(t)^\top F\varphi(s), \quad t,s \in \mathcal{I},\ t \neq s,$$

where F is known as the *coefficient matrix*. An algorithm for computing the coefficients of $\tilde{f}(t)\Theta(t-s)$ has been developed in [2], where further details on the discretization based on Legendre polynomials can be found.

Proposition 2.1. *Consider $f, g \in \mathcal{A}_\Theta(\mathcal{I})$ and their respective infinite-size coefficient matrices F, G in the Legendre basis. The following properties hold:*

1. *If $f + g = h \in \mathcal{A}_\Theta(\mathcal{I})$, then its coefficient matrix is $H = F + G$.*
2. *If $f \star g = h \in \mathcal{A}_\Theta(\mathcal{I})$, then, assuming the matrix product is well defined, its coefficient matrix is $H = FG$.*
3. *The coefficient matrix of the $\star$-identity element $1_\star := \delta(t-s)$ is the identity matrix I.*

4. *If we consider $R_\star(f)$, the $\star$-resolvent of f, its associated matrix of coefficients is $R_\star(F) = (I - F)^{-1}$.*

Proof. From the Legendre expansion of f and g, we have $f(t,s) = \varphi(t)^\top F \varphi(s)$, and $g(t,s) = \varphi(t)^\top G \varphi(s)$, for $t, s \in \mathcal{I},\ t \neq s$.

1.
$$\begin{aligned} h = f + g &= \begin{pmatrix} p_0(t) & p_1(t) & \cdots \end{pmatrix} F \begin{pmatrix} p_0(s) \\ p_1(s) \\ \vdots \end{pmatrix} + \begin{pmatrix} p_0(t) & p_1(t) & \cdots \end{pmatrix} G \begin{pmatrix} p_0(s) \\ p_1(s) \\ \vdots \end{pmatrix} \\ &= \begin{pmatrix} p_0(t) & p_1(t) & \cdots \end{pmatrix} (F + G) \begin{pmatrix} p_0(s) \\ p_1(s) \\ \vdots \end{pmatrix}, \end{aligned}$$

   hence $H = F + G$.

2.
$$\begin{aligned} h = f \star g &= \int_{\mathcal{I}} \begin{pmatrix} p_0(t) & p_1(t) & \cdots \end{pmatrix} F \begin{pmatrix} p_0(\tau) \\ p_1(\tau) \\ \vdots \end{pmatrix} \begin{pmatrix} p_0(\tau) & p_1(\tau) & \cdots \end{pmatrix} G \begin{pmatrix} p_0(s) \\ p_1(s) \\ \vdots \end{pmatrix} d\tau \\ &= \begin{pmatrix} p_0(t) & p_1(t) & \cdots \end{pmatrix} F \left(\int_{\mathcal{I}} \begin{pmatrix} p_0(\tau)p_0(\tau) & p_0(\tau)p_1(\tau) & \cdots \\ p_1(\tau)p_0(\tau) & p_1(\tau)p_1(\tau) & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} d\tau \right) G \begin{pmatrix} p_0(s) \\ p_1(s) \\ \vdots \end{pmatrix}. \end{aligned}$$

   Due to the orthonormality of our polynomial basis $\{p_k\}_k$, the matrix above in the middle equals the identity matrix
$$\int_{\mathcal{I}} \begin{pmatrix} p_0(\tau)p_0(\tau) & p_0(\tau)p_1(\tau) & \cdots \\ p_1(\tau)p_0(\tau) & p_1(\tau)p_1(\tau) & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} d\tau = \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 1 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}.$$

   Thus, we get
$$h = f \star g = \begin{pmatrix} p_0(t) & p_1(t) & \cdots \end{pmatrix} FG \begin{pmatrix} p_0(s) \\ p_1(s) \\ \vdots \end{pmatrix},$$

   that is, the coefficient matrix for h is $H = FG$, under the assumption that this matrix product is well defined.

3. Consider the Dirac delta $\delta(t-s)$ and its coefficient matrix $F = (f_{k,\ell})_{k,\ell \geq 0}$ in the expansion
$$\delta(t-s) \ \sim \sum_{k,\ell \geq 0} f_{k,\ell}\, p_k(t)\, p_\ell(s),$$

By the defining property of the Dirac delta distribution,

$$f_{k,\ell} = \int_I \int_I \delta(\tau - \rho)\, p_k(\tau)\, p_\ell(\rho)\, d\rho\, d\tau = \int_I p_k(\tau)\, p_\ell(\tau)\, d\tau = \delta_{k\ell} = \begin{cases} 0, & \text{if } k \neq l, \\ 1, & \text{if } k = l, \end{cases}$$

where in the last equality we used the orthonormality of $\{p_k\}$ on $\mathcal{I}$.
Hence $F = (\delta_{k\ell})_{k,\ell \geq 0} = I$.

4. We have proved in Proposition 1.12, that the $\star$-resolvent of f can be written as $R_\star(f) = (\delta - f)^{-\star}$. Moreover, we know that $R_\star(f)$ can be written as δ plus an element from $\mathcal{A}_\Theta$. Therefore, using (1), (2), (3) above, the claim follows. See also [5].

□

More generally, there is a map between $\{\mathcal{A}_\Theta(\mathcal{I}) \cup \delta, +, \star\}$ and a subalgebra of ∞ matrices [5]. However, since we cannot work numerically with infinite matrices, we need to truncate them in order to obtain a finite-dimensional linear system in the usual matrix algebra. The series of truncated expansions can be written in matrix form, for $t, s \in \mathcal{I}$, $t \neq s$, and some $M > 0$:

$$f_M(t,s) = \sum_{k=0}^{M-1} \sum_{l=0}^{M-1} f_{k,l} p_k(t) p_l(s) = \varphi_M(t)^\top F_M \varphi_M(s),$$

with

$$F_M := \begin{pmatrix} f_{0,0} & f_{0,1} & \cdots & f_{0,M-1} \\ f_{1,0} & f_{1,1} & \cdots & f_{1,M-1} \\ \vdots & \vdots & \ddots & \vdots \\ f_{M-1,0} & f_{M-1,1} & \cdots & f_{M-1,M-1} \end{pmatrix} \quad \varphi_M(\tau) := \begin{pmatrix} p_0(\tau) \\ p_1(\tau) \\ \vdots \\ p_{M-1}(\tau) \end{pmatrix}.$$

We can also rewrite the properties of the Proposition 2.1 in terms of finite-size truncated matrices, for example, if $h = f \star g$, we can approximate the coefficient matrix $H \approx H_M := F_M G_M$, where we recall that H is the matrix containing the coefficients $h_{k,l} = \sum_{j=0}^{\infty} f_{k,j} g_{j,l}$. This approximation is affected by a truncation error. Therefore, fixing k and l, if the magnitude of the product $f_{k,j} \cdot g_{j,l}$ does not decay quickly enough for $j \to \infty$, then the truncation error $(H)_{k,l} - (H_M)_{k,l}$ can be too large for practical purposes. Luckily, since $f \in \mathcal{A}_\Theta(\mathcal{I})$, if we choose the shifted orthonormal Legendre polynomials, it was proved in [2] that F_M and G_M are numerically banded, i.e., most of their significant entries are concentrated around the main diagonal, while the elements far from the diagonal are really close to zero. Hence M does not need to be large to reach a small truncation error in the approximation.

Example 2.2 (**Truncating a (numerically) banded coefficient matrix**). Let us consider the coefficient matrix for the Heaviside function Θ, that is $T \in \mathbb{C}^{\infty\times\infty}$ tridiagonal infinite-size with entries

$$T = \begin{pmatrix} \alpha_1 & \gamma_1 & 0 & 0 & \cdots \\ \beta_2 & \alpha_2 & \gamma_2 & 0 & \cdots \\ 0 & \beta_3 & \alpha_3 & \gamma_3 & \ddots \\ \vdots & & \ddots & \ddots & \ddots \end{pmatrix},$$

When truncating the infinite matrix to its leading $M \times M$ block T_M, the entry $(M, M+1) = \gamma_M$ is omitted. As a result, for a coefficient vector $c = (c_1, c_2, \dots)^\mathsf{T}$ we have

$$(Tc)_M = \beta_M\, c_{M-1} + \alpha_M\, c_M + \gamma_M\, c_{M+1}, \qquad (T_M c_{1:M})_M = \beta_M\, c_{M-1} + \alpha_M\, c_M,$$

with the notation $c_{1:M} = (c_1, c_2, \dots, c_M)^\mathsf{T}$ standing for the 1 to M components of a vector. Hence, the truncation error is localized in the last component:

$$\|(Tc)_{1:M} - T_M c_{1:M}\| = |(Tc)_M - (T_M c_{1:M})_M| = |\gamma_M\, c_{M+1}|.$$

To reduce this error's propagation, we set to zero all matrix entries with column index larger than M, starting from the last affected row (row M) downwards (see Figure 2.1).

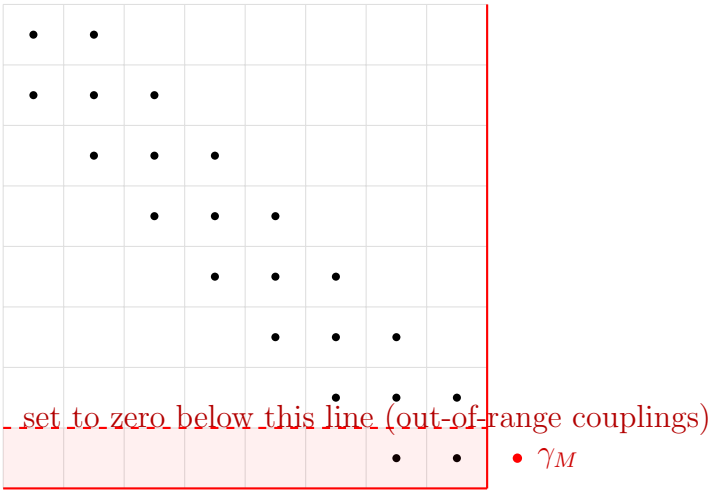


Figure 2.1: Truncation to size M of an infinite tridiagonal matrix with α_i (main), γ_i (upper), β_i (lower). The element $(M, M+1) = \gamma_M$ lies outside the $M \times M$ block (red dot). Consequently, $(Tc)_M = (T_M c_{1:M})_M + \gamma_M c_{M+1}$, i.e., the truncation error is confined to the last component. Below the dashed line we set to zero out-of-range couplings.

In general if $F = [F_{ij}]$ is a numerically banded coefficient matrix (like the ones arising from the discretization of our problem) with upper bandwidth β in the sense that $|F_{ij}| \lesssim \varepsilon$ for $j - i > \beta$ (beyond numerical tolerance), we will truncate to size M and *zero* all entries (i, j) with $j > M - 1$ and $i \geq M - \beta$ (i.e., from the row where the upper band

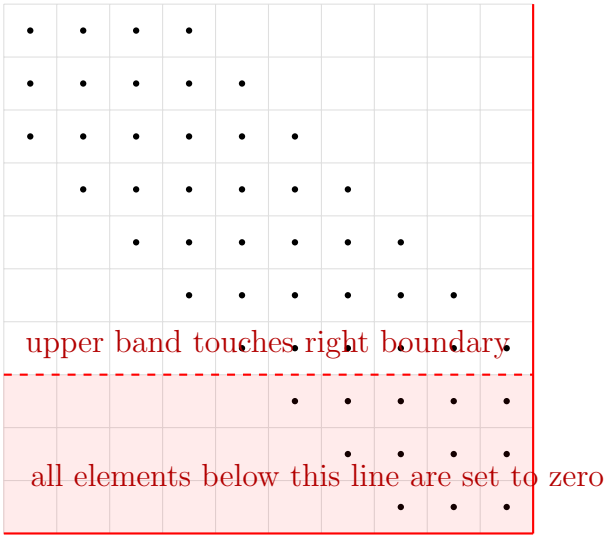


Figure 2.2: Numerically banded matrix (upper bandwidth $\beta = 3$). When truncating at size M, the upper band touches the right boundary at row $M - \beta$. All elements below this horizontal line are set to zero, keeping the truncation error localized and preventing accumulation.

meets the right boundary downward), as shown in Figure 2.2. This procedure helps to keep the truncation error controlled and to reduce its accumulation [2].

We must now discuss the convergence behavior of the series in eq. (2.1). Indeed, since f is discontinuous for $t = s$, the expansion may not converge quickly (or not converge at all) to $f(t, s)$, for every $t, s \in \mathcal{I}$. Nevertheless, fixing $s = a$, the left endpoint of the interval, the univariate function $f(t, a) = \tilde{f}(t)\Theta(t-a) = \tilde{f}(t)$, is analytic, hence smooth, over $\mathcal{I} = [a, b]$. Therefore,

$$f(t, a) = \sum_{k=0}^{\infty} a_k p_k(t) = \sum_{k=0}^{\infty} p_k(t) \sum_{l=0}^{\infty} f_{k,l} p_l(a).$$

As a consequence, we can approximate the function $f(t, a)$ by the expression $f(t, a) = \varphi_M(t)^\top F_M \varphi_M(a)$, and expect to reach a small enough accuracy for a relatively small M.

Let us now return to our problem and try to discretize the expression of the solution via Legendre polynomials, as described above. We recall that:

$$\frac{d}{dt}u(t, s) = \tilde{f}(t)u(t, s), \quad u(s, s) = 1, \quad t, s \in \mathcal{I}, \quad \textit{for} \quad u(t, s) = \tilde{u}_s(t)\Theta(t - s), \tag{2.2}$$

with its solution in the $\star$-algebra given, as highlighted in Remark 1.14, by

$$u(t, s) = \Theta(t - s) \star R_\star(f).$$

Using Proposition 2.1, we see that the coefficient matrix U_M related to the function above $u(t, s) \approx \varphi_M(t)^\top U_M \varphi_M(s)$, can be approximated by

$$U_M = T_M(I_M - F_M)^{-1},$$

where T_M is the coefficient matrix of $\Theta(t-s)$, and F_M is the coefficient matrix of $f(t,s) = \tilde{f}(t)\Theta(t-s)$, with $\tilde{f}(t)$ from problem (2.2). Since $u(t,s) \in \mathcal{A}_\Theta(\mathcal{I})$, for $s = a$ we can approximate the solution of (2.2) by the formula

$$u(t,a) = u_a(t) = \varphi_M(t)^\top U_M \varphi_M(a) = \varphi_M(t)^\top T_M (I_M - F_M)^{-1} \varphi_M(a).$$

Hence, if we solve the linear system

$$(I_M - F_M)x = \varphi_M(a),$$

the vector $u_M = T_M x$ will contain the approximated expansion coefficients of the solution $u_a(t)$, which will be constructed by

$$u_a(t) = \varphi_M(t)^\top u_M, \quad \text{for every } t \in \mathcal{I}.$$

Assuming the linear system can be computed exactly, in [2] it was proved that the truncation error can be kept at a desired precision as long as M is large enough. We remark that $u_a(t) = \sum_{i=0}^{M-1} \hat{c}_i p_i(t)$, with the coefficients $\hat{c}_0, ..., \hat{c}_{M-1}$ components of the vector $u_M = T_M x$, approximation of the first M Legendre coefficients of the solution $u(t)$. Therefore the resulting method is a *spectral method*, with a super-linear convergence asymptotically with M.

2.2 Vector-Valued Discretization

The approximation in the scalar case can be easily extended to the matrix one. Indeed, if we consider $A(t,s) = [a_{i,j}(t,s)]_{i,j=1}^N \in \mathcal{A}_\Theta(\mathcal{I})^{N\times N}$, $N \times N$ matrix with elements $a_{ij}(t,s) \in \mathcal{A}_\Theta(\mathcal{I})$, then for each a_{ij}, we can compute the related (truncated) coefficient matrices $F_M^{(i,j)}$ as we did for the scalar case, obtaining the block matrix

$$\mathcal{A}_M = \begin{bmatrix} \boxed{F_M^{(1,1)}} & \cdots & \boxed{F_M^{(1,N)}} \\ \vdots & \ddots & \vdots \\ \boxed{F_M^{(N,1)}} & \cdots & \boxed{F_M^{(N,N)}} \end{bmatrix} \in \mathbb{C}^{MN\times MN}. \tag{2.3}$$

Proposition 2.3. *Given the $N \times N$ matrices $A(t,s), B(t,s), C(t,s) \in \mathcal{A}_\Theta(\mathcal{I})^{N\times N}$ such that $C(t,s) = A(t,s) \star B(t,s)$, and let their coefficient matrices be, respectively, $\mathcal{A}_M, \mathcal{B}_M, \mathcal{C}_M$. Then, analogously to the scalar case, $\mathcal{C}_M$ is approximated by*

$$\mathcal{C}_M \approx \mathcal{A}_M \mathcal{B}_M.$$

Proof. Let

$$A(t,s) = [a_{ij}(t,s)]_{i,j=1}^N, \qquad B(t,s) = [b_{ij}(t,s)]_{i,j=1}^N,$$

with $a_{ij}, b_{ij} \in \mathcal{A}_\Theta(\mathcal{I})$, and define

$$C(t,s) = A(t,s) \star B(t,s).$$

By definition of the ⋆-product for matrices, as noted in Remark 1.17, we have

$$C_{ij}(t,s) = \sum_{k=1}^{N} a_{ik}(t,\cdot) \star b_{kj}(\cdot,s), \qquad i,j = 1,\dots,N. \tag{2.4}$$

For each pair (i,j) we expand a_{ij} and b_{ij} in the chosen orthonormal basis on $\mathcal{I}$ (for instance, the shifted Legendre polynomials), and denote by

$$F_M^{(i,j)},\ G_M^{(i,j)} \in \mathbb{C}^{M\times M},$$

the corresponding truncated coefficient matrices. Collecting all blocks together, we obtain

$$\mathcal{A}_M = \begin{bmatrix} F_M^{(1,1)} & \cdots & F_M^{(1,N)} \\ \vdots & \ddots & \vdots \\ F_M^{(N,1)} & \cdots & F_M^{(N,N)} \end{bmatrix}, \qquad \mathcal{B}_M = \begin{bmatrix} G_M^{(1,1)} & \cdots & G_M^{(1,N)} \\ \vdots & \ddots & \vdots \\ G_M^{(N,1)} & \cdots & G_M^{(N,N)} \end{bmatrix}.$$

From the scalar case (Proposition 2.1) we know that, at the level of coefficients, the ⋆-product corresponds to the matrix product of the truncated coefficient matrices:

$$a_{ik} \star b_{kj} \approx \varphi_M(t)^\top F_M^{(i,k)}\, G_M^{(k,j)} \varphi_M(s). \tag{2.5}$$

Applying formula (2.5) to each term of eq. (2.4), the truncated coefficients of block C_{ij} satisfy

$$H_M^{(i,j)} \ \approx\ \sum_{k=1}^{N} F_M^{(i,k)}\, G_M^{(k,j)}.$$

This is precisely the rule for the multiplication of block matrices. Defining $\mathcal{C}_M = [H_M^{(i,j)}]_{i,j}$, we thus have

$$\mathcal{C}_M \approx \mathcal{A}_M\, \mathcal{B}_M.$$

□

As a consequence, also in the matrix case, the ⋆-algebra can be approximated by the usual matrix algebra, as summarized in the last two columns of Table 2.1.

$\star$-operations	Matrix operations	$\star$-operations	Matrix operations
$f(t,s) \in \mathcal{A}_\Theta(\mathcal{I})$	$F_M \in \mathbb{C}^{M\times M}$	$A(t,s) \in \mathcal{A}_\Theta(\mathcal{I})^{N\times N}$	$\mathcal{A}_M \in \mathbb{C}^{MN\times MN}$
$q = f \star g$	$Q_M = F_M G_M$	$C = A \star B$	$\mathcal{C}_M = \mathcal{A}_M \mathcal{B}_M$
$f + g$	$F_M + G_M$	$A + B$	$\mathcal{A}_M + \mathcal{B}_M$
$1_\star := \delta(t-s)$	I_M, identity matrix	$1_\star := \delta(t-s) I_N$	I_{MN}, identity matrix
$f^{-\star}$	F_M^{-1}	$A^{-\star}$	$\mathcal{A}_M^{-1}$
$R_\star(f) := (1_\star - f)^{-\star}$	$R(F_M) := (I - F_M)^{-1}$	$R_\star(A) := (1_\star - A)^{-\star}$	$R(\mathcal{A}_M) := (I_{MN} - \mathcal{A}_M)^{-1}$

Table 2.1: The $\star$-algebra operations and the corresponding matrix algebra operations after discretization, scalar case (first two columns) and matrix case (last two columns) [1].

We now introduce the definition of the Kronecker product and some of its fundamental properties, which will be useful for the description of the solution to the vector problem.

Definition 2.4 (**Kronecker product**). *Let $A = [a_{ij}] \in \mathbb{C}^{m\times n}$ and $B = [b_{ij}] \in \mathbb{C}^{p\times q}$. The* Kronecker product *of A and B, denoted by $A \otimes B$, is the block matrix of size $(mp) \times (nq)$ defined as*

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix} \in \mathbb{C}^{(mp)\times(nq)}.$$

Proposition 2.5 (**Kronecker product properties**). *Let A, B, C, D be real (or complex) matrices of compatible dimensions. Then:*

1. *Linearity.*

$$(A + B) \otimes C \;=\; A \otimes C \;+\; B \otimes C, \qquad A \otimes (C + D) \;=\; A \otimes C \;+\; A \otimes D.$$

2. *Mixed-product property. If the usual matrix products AB and CD are well defined, then*

$$(AB) \otimes (CD) \;=\; (A \otimes C)(B \otimes D).$$

3. *Vectorization identity. For $A \in \mathbb{R}^{m\times n}$, $X \in \mathbb{R}^{n\times p}$, and $B \in \mathbb{R}^{p\times q}$,*

$$\operatorname{vec}(AXB) \;=\; (B^\top \otimes A)\, \operatorname{vec}(X),$$

where $\operatorname{vec}(X)$ denotes the vectorization of X, i.e., the vector obtained by stacking the columns of X into a single vector, one on top of each other.

Proof. (1) Linearity. Write $A = [a_{ij}]$, $B = [b_{ij}]$, and let C have compatible size. By the block definition of the Kronecker product,

$$(A+B)\otimes C = \begin{bmatrix} (a_{11}+b_{11})C & \cdots & (a_{1n}+b_{1n})C \\ \vdots & \ddots & \vdots \\ (a_{m1}+b_{m1})C & \cdots & (a_{mn}+b_{mn})C \end{bmatrix}$$
$$= \begin{bmatrix} a_{11}C & \cdots & a_{1n}C \\ \vdots & \ddots & \vdots \\ a_{m1}C & \cdots & a_{mn}C \end{bmatrix} + \begin{bmatrix} b_{11}C & \cdots & b_{1n}C \\ \vdots & \ddots & \vdots \\ b_{m1}C & \cdots & b_{mn}C \end{bmatrix},$$

which equals $A\otimes C + B\otimes C$. The proof of $A\otimes(C+D) = A\otimes C + A\otimes D$ is analogous.

(2) Mixed-product property. Let $A\in\mathbb{R}^{m\times n}$, $B\in\mathbb{R}^{n\times r}$, $C\in\mathbb{R}^{p\times q}$, and $D\in\mathbb{R}^{q\times s}$. Using the entrywise characterization $(A\otimes C)_{(i,\alpha),(j,\beta)} = a_{ij}\,c_{\alpha\beta}$ and ordinary matrix multiplication,

$$\big[(A\otimes C)(B\otimes D)\big]_{(i,\alpha),(j,\beta)} = \sum_{k=1}^{n}\sum_{\gamma=1}^{q}(A\otimes C)_{(i,\alpha),(k,\gamma)}\,(B\otimes D)_{(k,\gamma),(j,\beta)} = \sum_{k,\gamma} a_{ik}c_{\alpha\gamma}\,b_{kj}d_{\gamma\beta}.$$

Rearranging the sums gives

$$\Big(\sum_{k} a_{ik}b_{kj}\Big)\Big(\sum_{\gamma} c_{\alpha\gamma}d_{\gamma\beta}\Big) = (AB)_{ij}\,(CD)_{\alpha\beta} = \big[(AB)\otimes(CD)\big]_{(i,\alpha),(j,\beta)},$$

hence $(A\otimes C)(B\otimes D) = (AB)\otimes(CD)$.

(3) Vectorization identity. Let $A\in\mathbb{R}^{m\times n}$, $X\in\mathbb{R}^{n\times p}$, $B\in\mathbb{R}^{p\times q}$, and recall that $\mathrm{vec}(\cdot)$ stacks columns. The (i,k)-entry of AXB is

$$(AXB)_{ik} = \sum_{j=1}^{n}\sum_{\ell=1}^{p} a_{ij}\,x_{j\ell}\,b_{\ell k}.$$

On the other hand, by the Kronecker entry rule,

$$\big[(B^{\top}\otimes A)\,\mathrm{vec}(X)\big]_{(i,k)} = \sum_{j,\ell}(B^{\top}\otimes A)_{(i,k),(j,\ell)}\,x_{j\ell} = \sum_{j,\ell}(B^{\top})_{k\ell}\,A_{ij}\,x_{j\ell} = \sum_{j,\ell} a_{ij}\,x_{j\ell}\,b_{\ell k}.$$

The coordinates coincide for all i,k, hence $\mathrm{vec}(AXB) = (B^{\top}\otimes A)\,\mathrm{vec}(X)$. □

We recall now our differential problem in the matrix form, i.e.,

$$\frac{d}{dt}U(t,s) = \tilde{A}(t)U(t,s), \quad U(s,s) = I_N, \quad \text{for } t,s\in\mathcal{I}, \tag{2.6}$$

with solution written in the ⋆-algebra as

$$U(t,s) = \Theta(t-s)\star R_{\star}(A)(t,s). \tag{2.7}$$

The matrix-valued function $U(t,s)$ is composed of elements from $\mathcal{A}_\Theta(\mathcal{I})$, therefore, we can define the related (truncated) coefficient matrix $\mathcal{U}_M$ as in formula (2.3). Then the expression (2.7) can be approximated by

$$\mathcal{U}_M = (I_N \otimes T_M)(I_{MN} - \mathcal{A}_M)^{-1},$$

where $\otimes$ is the Kronecker product, T_M is the coefficient matrix of $\Theta(t-s)$, and $\mathcal{A}_M$ is the coefficient matrix of $A(t,s) = \tilde{A}(t)\Theta(t-s)$ with $\tilde{A}(t)$ from equation (2.6).

Moreover, we can approximate the solution of the problem (2.6) for $s = a$ by the formula:

$$\begin{aligned} U(t,a) &= (I_N \otimes \varphi_M(t)^\top)(I_N \otimes T_M)(I_{MN} - \mathcal{A}_M)^{-1}(I_N \otimes \varphi_M(a)) \\ &= (I_N \otimes \varphi_M(t)^\top T_M)(I_{MN} - \mathcal{A}_M)^{-1}(I_N \otimes \varphi_M(a)), \end{aligned}$$

where in the second equality we have used the Kronecker property stated in Proposition 2.5. Note that, as we explained for the scalar case, the approximation converges quickly enough to the solution only when s is the left endpoint of the interval $\mathcal{I}$, i.e., $s = a$.

Remark 2.6. As noted in Remark 1.21, in some practical situation the initial time of the evolution is fixed and set equal to zero, $s = a = 0$, and the initial condition is given as a vector $v \in \mathbb{C}^N$,

$$\frac{d}{dt}\tilde{u}(t) = \tilde{A}(t)\tilde{u}(t), \quad \tilde{u}(0) = v, \quad t \in \mathcal{I}. \tag{2.8}$$

Therefore, using the expression above, the solution to such a problem is given by

$$\begin{aligned} u(t) = U(t,0)v &= (I_N \otimes \varphi_M(t)^\top T_M)(I_{MN} - \mathcal{A}_M)^{-1}(I_N \otimes \varphi_M(0))v \\ &= (I_N \otimes \varphi_M(t)^\top T_M)(I_{MN} - \mathcal{A}_M)^{-1}(v \otimes \varphi_M(0)). \end{aligned}$$

Then, solving the linear system

$$(I_{MN} - \mathcal{A}_M)x = v \otimes \varphi_M(0), \tag{2.9}$$

one can approximate the solution of the problem (2.8) in terms of its Legendre coefficients $u_M := (I_N \otimes T_M)x$, that is,

$$\begin{aligned} u(t) &= (I_N \otimes \varphi_M(t)^\top T_M)(I_{MN} - \mathcal{A}_M)^{-1}(v \otimes \varphi_M(0)) \\ &= (I_N \otimes \varphi_M(t)^\top T_M)x \\ &= (I_N \otimes \varphi_M(t)^\top)(I_N \otimes T_M)x \\ &= (I_N \otimes \varphi_M(t)^\top)u_M. \end{aligned}$$

Therefore, once the Legendre coefficients of the solution have been computed, the solution itself can be reconstructed at any given time $t \in \mathcal{I}$ by multiplying on the left

by $I_N \otimes \varphi_M(t)^\top$. The method is global: it does not require a discretization of the time interval, as the solution is expressed in terms of its Legendre expansion coefficients, i.e., in the "frequency" domain.

The discretization of the ⋆-product in the vector case has allowed us to reformulate the continuous problem as a linear system in the standard matrix algebra. By expanding the entries of $A(t, s)$ in the Legendre basis and assembling the global coefficient matrix $\mathcal{A}_M \in \mathbb{C}^{MN \times MN}$, we have finally obtained a formulation expressed entirely in terms of familiar objects. The resulting linear system, while simple in form, can be extremely large. Since Kronecker products are involved, the system's size grows as $MN \times MN$, and in practical scenarios both N (the size of the original ODE system) and M (the number of expansion coefficients required for a prescribed accuracy) may be sizeable, leading to memory requirements and computational costs that render any direct solution prohibitively expensive. Therefore, even though we have succeeded in expressing the problem as a linear system, this system cannot be solved via a direct numerical approach. The matrix, despite its structure, cannot be explicitly formed nor stored for realistic choices of M and N. It is thus necessary to reformulate the problem once again, passing from its vectorized representation to a matrix equation, drastically reducing both computational cost and memory usage.

In the next section, we will show that such a reformulation not only avoids the explicit matrix construction and the expensive operations, but also enables the use of more efficient iterative methods exploiting the matrix structure. This final reformulation is therefore essential to make the discretized ⋆-product in the vector case numerically feasible.

2.2.1 From Discretization to Matrix Equations

Consider the matrix-valued function $\tilde{A}(t)$ in expression (2.6) in the form

$$\tilde{A}(t) = \sum_{k=1}^{d} A_k \tilde{f}_k(t), \tag{2.10}$$

with $\tilde{f}_1, ..., \tilde{f}_d$ distinct scalar functions and $A_1, ..., A_d$ constant matrices, $d > 0$. Note that, since $\tilde{A}(t) = [\tilde{a}_{ij}(t)]_{i,j=1}^N \in \mathcal{A}_\Theta(\mathcal{I})^{N \times N}$ with $\tilde{a}_{ij}$ analytic, we can always approximate it by a truncated expansion in a basis $\{\tilde{f}_k\}$, obtaining:

$$\tilde{A}(t) = [\tilde{a}_{ij}(t)]_{i,j=1}^N = \sum_{k=1}^{d} [c_{ij}^{(k)}]_{i,j} \tilde{f}_k(t) = \sum_{k=1}^{d} A_k \tilde{f}_k(t).$$

In many applications, as we will see, d is small. Then exploiting expression (2.10), the block coefficient matrix in (2.3) of $A(t,s) = \tilde{A}(t)\Theta(t-s)$ becomes

$$\mathcal{A}_M = \sum_{k=1}^{d} A_k \otimes F_M^{(k)}, \tag{2.11}$$

with $F_M^{(k)}$ the coefficient matrix of $\tilde{f}_k(t)$.

Proposition 2.7. *In the notation previously introduced for problem* (2.8)*, the solution* x *of the linear system* (2.9)

$$(I_{MN} - \mathcal{A}_M)x = v \otimes \varphi_M(0),$$

can be rewritten as

$$X - \sum_{k=1}^{d} F_M^{(k)} X A_k^\top = \varphi_M(0)v^\top, \quad x = vec(X). \tag{2.12}$$

Proof. By replacing $\mathcal{A}_M$ with its expression in (2.11) and denoting $x = vec(X)$, we get

$$\left(I_{MN} - \sum_{k=1}^{d} A_k \otimes F_M^{(k)}\right) vec(X) = v \otimes \varphi_M(0).$$

Then, using the fact that $v \otimes \varphi_M(0) = vec(\varphi_M(0)v^\top)$ we obtain

$$vec(X) - \sum_{k=1}^{d} (A_k \otimes F_M^{(k)}) vec(X) = vec(\varphi_M(0)v^\top).$$

Thanks to the property stated in Proposition 2.5, we can rewrite as

$$vec(X) - \sum_{k=1}^{d} vec(F_M^{(k)} X A_k^\top) = vec(\varphi_M(0)v^\top),$$

and by linearity and invertibility of the *vec* map the claim follows

$$X - \sum_{k=1}^{d} F_M^{(k)} X A_k^\top = \varphi_M(0)v^\top.$$

□

Remark 2.8. The matrix equation (2.12) has a rank-1 right-hand side $\varphi_M(0)v^\top$. This suggests that the solution X may have a low numerical rank. Indeed, as observed in [1], varying k, the numerical rank of X typically increases slowly with the size of X. This low-rank property is of fundamental importance and will be exploited in the numerical methods used to solve the matrix equation, helping us to achieve gains in terms of memory usage and computational cost.

As we mentioned earlier, in terms of computational cost, it is preferable to compute these matrix-matrix products of size $M \times M$, $N \times N$, or $M \times N$, rather than trying to solve a linear system of size $MN \times MN$, which in some cases cannot even be formed due to memory limitations.

In the literature, there already exist direct and iterative methods for solving matrix equations that have only two terms on the left-hand side, such as *Sylvester, Stein* and *Lyapunov equations*. In the next chapter, we will discuss the numerical methods used to solve multiterm linear equations in general, which involve $d + 1$ terms on the left-hand side, like our matrix equation (2.12), which can be seen as a *generalized Stein equation*.

Chapter 3

Numerical Methods for Multiterm Matrix Equations

In the previous chapter, we have shown that the $\star$-approach applied to a system of non-autonomous ordinary differential equations, after discretization, leads to the solution of a linear system, which can also be formulated as a matrix equation. For this reason, in the present chapter, we introduce, in Section 3.1, the more general class of multiterm matrix equations, highlighting the differences with respect to more classical and well-known special cases. In this section, we refer to [10], which provides an overview of the main methods and theoretical results for the solution of matrix equations, with a focus on Sylvester and Lyapunov-type problems. After reformulating the multiterm matrix equation as a linear system, we then discuss a numerical solution strategy. In particular, Section 3.2 is devoted to the presentation of *Krylov subspace methods*, with a particular focus on the BiCG and BiCGSTAB algorithms, first in the vector setting and subsequently in the matrix formulation. We take [11] as the main reference for that section.

3.1 Linear Multiterm Matrix Equations

As underlined in [10], linear matrix equations have received considerable attention since the early 1900s, and their study is still ongoing because they arise from many different applications, from statistics to dynamical systems and control theory to, as we will see in the next chapter, nuclear magnetic resonance in chemistry, for example. We consider linear multiterm matrix equations of the form

$$\sum_{i=1}^{d} A_i X B_i = f g^\top, \tag{3.1}$$

where $A_i \in \mathbb{R}^{n\times n}$ and $B_i \in \mathbb{R}^{m\times m}$ are given matrices, $X \in \mathbb{R}^{n\times m}$ is the unknown matrix and $f \in \mathbb{R}^n$, $g \in \mathbb{R}^m$ are given vectors. Therefore we focus on the particular case where the right-hand-side is a rank-one matrix.

For $d = 2$, under certain conditions on the coefficient matrices, equation (3.1) has a unique solution with available elegant and explicit closed forms. Several efficient algorithms have been implemented for the solution of problem (3.1), especially in the case of Stein (A_1 and B_1 are equal to the identity matrix), Sylvester (B_1 and A_2 are equal to the identity matrix) and Lyapunov (in addition $B_2 = A_1^\top$) equations, where the literature is particularly rich since they have been extensively studied over the past decades. Many computational approaches and numerical methods for this type of equations can be found in [10].

On the other hand, for the more general case $d > 2$, this is no longer true. Indeed, the multiterm matrix equation (3.1) is considerably more difficult to analyze, and necessary and sufficient conditions for the existence and uniqueness of the solution X, formulated in terms of the coefficient matrices $\{A_i\}$ and $\{B_i\}$, are hard to derive, except for some very special cases [12]. While, from a theoretical viewpoint, the importance of explicitly taking into account the structure of the problem has been acknowledged [13], the same cannot be said for computational strategies, especially in the large-scale setting. The algorithmic approach most commonly employed for problem (3.1) consists of transforming the matrix equation into a vectorized form by means of the Kronecker product. However, the need for low-complexity solution methods becomes particularly compelling whenever one or both of the matrices A_i and B_i are of large sizes. Kronecker-based formulations were eventually abandoned as core methods for the problem in the case $d = 2$, once algorithms with computational complexity growing only as a modest power of the coefficient matrices' dimensions became available. The development of efficient numerical solvers for problem (3.1) in the more general case therefore represents the next frontier in the study of linear matrix equations, and for that reason it is the main focus of this chapter.

Let us now try to reformulate the matrix problem as a linear system by means of vectorization, exploiting the definition of the Kronecker product. By the properties underlined in Proposition 2.5 and following the steps made in the proof of Proposition 2.7, we have that:

$$\sum_{i=1}^{d} A_i X B_i = fg^\top \iff (\sum_{i=1}^{d} B_i^\top \otimes A_i)vec(X) = g \otimes f.$$

Hence, defining the matrix

$$L := \sum_{i=1}^{d} B_i^\top \otimes A_i \in \mathbb{R}^{nm\times nm},$$

the problem (3.1) is equivalent to solving the linear system of dimension nm

$$Lvec(X) = g \otimes f. \tag{3.2}$$

For large matrix dimensions, explicitly forming the coefficient matrix L is computationally infeasible due to memory constraints, and for this reason, in many applications the coefficient matrix of the vectorized linear system is not constructed; instead, its action on a vector is computed through the original linear matrix equation. Indeed, we have seen that if we define the linear operator

$$\mathcal{L}(X) := \sum_{i=1}^{d} A_i X B_i,$$

since $vec(\mathcal{L}(X)) = Lvec(X)$, solving the linear system (3.2) is equivalent to

$$\mathcal{L}(X) = fg^\top.$$

Notice that forming the vectorized coefficient matrix $L \in \mathbb{R}^{nm \times nm}$ is prohibitively expensive: in the dense case it would require $O\big((nm)^2\big)$ storage (and comparable cost to manipulate). However, applying the associated matrix operator $\mathcal{L}(X)$ to $X \in \mathbb{R}^{n \times m}$ can be carried out without forming L, at a cost $O\big(d(n^2m + nm^2)\big)$ in the dense case. Furthermore, if the matrices A_i and B_i are sparse or structured, the application of $\mathcal{L}$ can be carried out at a lower cost, which can be expressed in terms of its number of nonzero elements. For this reason, we seek iterative methods that require only the action of the operator, i.e., the evaluation of $\mathcal{L}(X)$, rather than the explicit construction of L. This naturally leads to *Krylov subspace methods* for non-symmetric linear systems, which we will describe in the next section.

3.2 Krylov Subspace Methods

In the present section, following [11], we introduce a class of iterative *projection methods* that are particularly effective for solving linear systems of the form

$$Ax = b, \quad A \in \mathbb{R}^{n \times n},\ b \in \mathbb{R}^n,\ x \in \mathbb{R}^n,$$

especially when the coefficient matrix A is large, sparse, and possibly non-symmetric. These methods are known as *Krylov subspace methods*. We will first describe them in the general setting of linear systems, and then move on to the matrix case, since, as we said before, the linear system shown in equation (3.2) is often computationally intractable and we want to replace its coefficient matrix with the action of a linear matrix operator $\mathcal{L}$. In both cases, we will focus on the Biconjugate Gradient (BiCG) algorithm and on its stabilized version, BiCGSTAB.

In a general *projection method*, at iteration m one looks for an approximation x_m such that

$$x_m \in x_0 + \mathcal{S}_m, \qquad r_m = b - Ax_m \perp \mathcal{C}_m,$$

where $\mathcal{S}_m$ denotes the search space and $\mathcal{C}_m$ the constraint space. In Krylov methods, the search space is chosen as a Krylov subspace associated with the matrix A and the initial residual $r_0 = b - Ax_0$ of dimension m, which is defined as

$$\mathcal{K}_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\},$$

while the constraint space could be chosen equal or different from the search space, depending on whether A is symmetric ($\mathcal{C}_m = \mathcal{S}_m$) or not ($\mathcal{C}_m \neq \mathcal{S}_m$). The various Krylov subspace methods differ according to the choice of these spaces. Therefore, these methods construct approximate solutions by projecting the original problem onto a sequence of nested subspaces generated by repeated applications of the matrix A to the initial residual. Their main advantage lies in the fact that they only require matrix-vector products, making them well suited for large-scale problems where direct methods would be computationally prohibitive.

An orthogonal basis of the Krylov subspace $\mathcal{K}_m$ could be built in general by Arnoldi's procedure, or by Lanczos's one in the symmetric case. In exact arithmetic, a variant of Arnoldi's algorithm is given by Algorithm 1.

Algorithm 1 Arnoldi [11].

1: Choose a vector v_1, such that $\|v_1\|_2 = 1$
2: For $j = 1, 2, \ldots, m$ Do:
3: Compute $h_{ij} = (Av_j, v_i) = v_i^\top Av_j$ for $i = 1, 2, \ldots, j$
4: Compute $w_j := Av_j - \sum_{i=1}^{j} h_{ij}v_i$
5: $h_{j+1,j} = \|w_j\|_2$
6: If $h_{j+1,j} = 0$ then Stop
7: $v_{j+1} = w_j/h_{j+1,j}$
8: EndDo

At each step, Algorithm 1 multiplies the previous Arnoldi vector v_j by A and then orthonormalizes the resulting vector w_j against all previous vectors v_i using a standard Gram-Schmidt procedure. The algorithm ends if the vector w_j computed in line 4 vanishes.

Remark 3.1. If we assume that Algorithm 1 does not stop before the m-th step, then the vectors $v_1, v_2, ..., v_m$ form an orthonormal basis of the Krylov subspace $\mathcal{K}_m(A, v_1)$.

Proposition 3.2. *[11] Denote by V_m the $n \times m$ matrix with column vectors $v_1, \ldots, v_m$, by $\bar{H}_m$ the $(m+1) \times m$ Hessenberg matrix whose nonzero entries h_{ij} are defined by*

Algorithm 1, and by H_m the matrix obtained from $\bar{H}_m$ by deleting its last row. Then the following relations hold:

$$AV_m = V_m H_m + w_m e_m^\top \tag{3.3}$$

$$= V_{m+1}\bar{H}_m, \tag{3.4}$$

$$V_m^\top A V_m = H_m. \tag{3.5}$$

Proof. Relation (3.4) follows from the equality

$$Av_j = \sum_{i=1}^{j+1} h_{ij} v_i, \qquad j = 1, 2, \ldots, m, \tag{3.6}$$

which is readily derived from lines 4, 5, and 7 of Algorithm 1. Relation (3.3) is a matrix reformulation of (3.6). Finally, relation (3.5) follows by multiplying both sides of (3.3) by $V_m^\top$ and making use of the orthonormality of the vectors $\{v_1, \ldots, v_m\}$. □

In practical situations, due to the presence of round-off, much can be gained by using the Modified Gram-Schmidt algorithm instead of the standard Gram-Schmidt algorithm, obtaining the more reliable formulation given by Algorithm 2.

Algorithm 2 Arnoldi-Modified Gram-Schmidt [11].

1: Choose a vector v_1 of norm 1
2: For $j = 1, 2, \ldots, m$ Do:
3: Compute $w_j := Av_j$
4: For $i = 1, \ldots, j$ Do:
5: $h_{ij} = (w_j, v_i)$
6: $w_j := w_j - h_{ij} v_i$
7: EndDo
8: $h_{j+1,j} = \|w_j\|_2$
9: If $h_{j+1,j} = 0$ Stop
10: $v_{j+1} = w_j / h_{j+1,j}$
11: EndDo

Another algorithm for producing bases of Krylov subspaces is the Lanczos biorthogonalization algorithm, which can be viewed as an extension of the symmetric Lanczos algorithm to non-symmetric matrices. We have already seen the Arnoldi procedure for that case; however, the non-symmetric Lanczos algorithm is conceptually quite different from Arnoldi's method, since it relies on biorthogonal sequences rather than orthogonal ones. The non-symmetric Lanczos algorithm produces a pair of biorthogonal bases for $\mathcal{K}_m(A, v_1)$ and $\mathcal{K}_m(A^\top, w_1)$; one possible algorithm for that procedure is given by

Algorithm 3. If we denote V_m and W_m the tall matrices whose columns are the vectors of the bases of $\mathcal{K}_m(A, v_1) = Span\{v_1, ..., v_m\}$ and $K_m(A^\top, w_1) = Span\{w_1, ..., w_m\}$, respectively, and we define the matrix

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{pmatrix},$$

then the following proposition holds.

Proposition 3.3. *[11] If the algorithm does not break down before step m, then the vectors v_i, $i = 1, \ldots, m$, and w_j, $j = 1, \ldots, m$, form a biorthogonal system, i.e.,*

$$(v_j, w_i) = \delta_{ij}, \qquad 1 \leq i, j \leq m, \quad \textit{or equivalently} \quad W_m^\top V_m = I_m$$

Moreover, the following relations hold:

$$AV_m = V_m T_m + \delta_{m+1} v_{m+1} e_m^\top, \tag{3.7}$$

$$A^\top W_m = W_m T_m^\top + \beta_{m+1} w_{m+1} e_m^\top, \tag{3.8}$$

$$W_m^\top A V_m = T_m. \tag{3.9}$$

Algorithm 3 Lanczos Biorthogonalization Procedure (Non-Symmetric Lanczos) [11].

1: Choose two vectors v_1, w_1 such that $(v_1, w_1) = 1$
2: Set $\beta_1 = \delta_1 = 0$, $w_0 = v_0 = 0$
3: For $j = 1, 2, \ldots, m$ Do:
4: $\quad \alpha_j = (Av_j, w_j)$
5: $\quad \hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
6: $\quad \hat{w}_{j+1} = A^\top w_j - \alpha_j w_j - \delta_j w_{j-1}$
7: $\quad \delta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})^{1/2}$
8: $\quad$ If $\delta_{j+1} = 0$ Stop
9: $\quad \beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}$
10: $\quad w_{j+1} = \hat{w}_{j+1}/\beta_{j+1}$
11: $\quad v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}$
12: EndDo

Proof. The biorthogonality of the vectors v_i and w_i will be proved by induction. By assumption, $(v_1, w_1) = 1$. Assume now that the vectors $v_1, \ldots, v_j$ and $w_1, \ldots, w_j$ are

biorthogonal, and let us prove that the vectors $v_1, \ldots, v_{j+1}$ and $w_1, \ldots, w_{j+1}$ are biorthogonal.

First, we show that $(v_{j+1}, w_i) = 0$ for $i \leq j$. When $i = j$, we have

$$(v_{j+1}, w_j) = \delta_{j+1}^{-1}\big[(Av_j, w_j) - \alpha_j(v_j, w_j) - \beta_j(v_{j-1}, w_j)\big].$$

The last inner product in the above expression vanishes by the induction hypothesis. The two remaining terms cancel each other out by the definition of α_j and the fact that $(v_j, w_j) = 1$.

Consider now the inner product (v_{j+1}, w_i) with $i < j$:

$$\begin{aligned}(v_{j+1}, w_i) &= \delta_{j+1}^{-1}\big[(Av_j, w_i) - \alpha_j(v_j, w_i) - \beta_j(v_{j-1}, w_i)\big] \\ &= \delta_{j+1}^{-1}\big[(v_j, A^\top w_i) - \beta_j(v_{j-1}, w_i)\big] \\ &= \delta_{j+1}^{-1}\big[(v_j, \beta_{i+1}w_{i+1} + \alpha_i w_i + \delta_i w_{i-1}) - \beta_j(v_{j-1}, w_i)\big].\end{aligned}$$

For $i < j-1$, all inner products in the above expression vanish by the induction hypothesis. For $i = j-1$, we obtain

$$\begin{aligned}(v_{j+1}, w_{j-1}) &= \delta_{j+1}^{-1}\big[(v_j, \beta_j w_j + \alpha_{j-1}w_{j-1} + \delta_{j-1}w_{j-2}) - \beta_j(v_{j-1}, w_{j-1})\big] \\ &= \delta_{j+1}^{-1}\big[\beta_j(v_j, w_j) - \beta_j(v_{j-1}, w_{j-1})\big] = 0.\end{aligned}$$

It can be shown in an analogous way that $(v_i, w_{j+1}) = 0$ for $i \leq j$. Finally, by construction, $(v_{j+1}, w_{j+1}) = 1$. This completes the induction proof.

The proof of the matrix relations (3.7)-(3.9) is similar to that of the relations (3.3)-(3.5) in Arnoldi's method.

□

The relations (3.7)-(3.9) allow us to interpret the algorithm. The matrix T_m is the projection of A obtained from an oblique projection process onto $\mathcal{K}_m(A, v_1)$ and orthogonal to $\mathcal{K}_m(A^\top, w_1)$. Similarly, $T_m^\top$ represents the oblique projection of $A^\top$ onto $\mathcal{K}_m(A^\top, w_1)$ and orthogonal to $\mathcal{K}_m(A, v_1)$. From a practical point of view, the Lanczos algorithm has a significant advantage over Arnoldi's method because it requires only a few vectors to store, if no reorthogonalization is performed. Specifically, six vectors of length n are needed, plus some storage for the tridiagonal matrix, no matter how large m is. Arnoldi's method makes long recurrences for the updates, while non-symmetric Lanczos uses short recurrences, meaning that it only requires the previous two iterates for the updates and not all the previous ones. On the other hand, there are potentially more opportunities for serious breakdown with the non-symmetric Lanczos method, since in Arnoldi's method only *lucky* breakdowns can happen.

3.2.1 The Biconjugate Gradient Method (BiCG)

The Biconjugate gradient method (BiCG) is a suitable Krylov subspace method to solve the linear system $Ax = b$, when the matrix A is non-symmetric. Its mathematical characterization is given by choosing $\mathcal{S}_m = K_m(A, r_0)$ and $\mathcal{C}_m = K_m(A^\top, \tilde{r}_0)$, where $r_0 = b - Ax_0$ is the initial residual associated with the initial guess x_0 and $\tilde{r}_0$ is a *shadow* initial residual selected by the user, typically chosen so that $(\tilde{r}_0, r_0) \neq 0$, used to build the Krylov subspace associated with $A^\top$. We can derive it starting from Algorithm 3; indeed, if we denote V_m and W_m the bases for $K_m(A, r_0)$ and $K_m(A^\top, \tilde{r}_0)$ respectively, due to our choice of search space and constraint space, we have

$$x_m = x_0 + V_m y_m, \quad r_m = b - Ax_m = r_0 - AV_m y_m,$$

and imposing the orthogonality condition

$$W_m^\top r_m = 0 \iff W_m^\top r_0 - W_m^\top A V_m y_m = 0.$$

Now, using the relation (3.9) from Proposition 3.3, we get $T_m y_m = W_m^\top r_0$, and choosing $v_1 = \frac{r_0}{\|r_0\|}$, $w_1 = v_1$, we can rewrite everything as

$$\begin{cases} T_m y_m = W_m^\top \|r_0\| e_1, \\ x_m = x_0 + V_m y_m. \end{cases}$$

Let us consider the LU decomposition of T_m,

$$T_m = L_m U_m = \begin{pmatrix} 1 & & & & \\ \ell_2 & 1 & & & \\ & \ell_3 & 1 & & \\ & & \ddots & \ddots & \\ & & & \ell_m & 1 \end{pmatrix} \begin{pmatrix} u_1 & \beta_2 & & & \\ & u_2 & \beta_3 & & \\ & & \ddots & \ddots & \\ & & & u_{m-1} & \beta_m \\ & & & & u_m \end{pmatrix};$$

comparing the two sides of the expression above, we have for the first diagonal entry, $u_1 = \alpha_1$, while for $j = 2, \ldots, m$ one has

$$\ell_j = \frac{\delta_j}{u_{j-1}},$$

and

$$u_j = \alpha_j - \ell_j \beta_j.$$

We can write now

$$\begin{aligned} x_m = x_0 + V_m y_m &= x_0 + V_m T_m^{-1} e_1 \|r_0\| \\ &= x_0 + V_m U_m^{-1} L_m^{-1} e_1 \|r_0\|, \end{aligned}$$

defining $P_m = V_m U_m^{-1}$ and z_m the solution of the lower triangular system $L_m z_m = e_1 \|r_0\|$ we get

$$x_m = x_0 + P_m z_m = x_0 + P_{m-1} z_{m-1} + \alpha_m p_m = x_{m-1} + \alpha_m p_m,$$

where α_m indicates the last component of the vector z_m. Rescaling p_m, we obtain

$$x_{m+1} = x_m + \alpha_m p_m.$$

Therefore, the residual update rewrites as

$$r_{m+1} = b - A x_{m+1} = b - A(x_m + \alpha_m p_m) = r_m - \alpha_m A p_m. \tag{3.10}$$

For the dual system, analogously, after defining the matrix $\tilde{P}_m = W_m L_m^{-T}$, we will have

$$\tilde{r}_{m+1} = \tilde{r}_m - \alpha_m A^\top \tilde{p}_m.$$

The scalar α_m is obtained imposing the Petrov-Galerkin condition $(\tilde{p}_m, r_{m+1}) = 0$; substituting the expression for the residual (3.10) we get

$$(\tilde{p}_m, r_m) - \alpha_m (\tilde{p}_m, A p_m) = 0 \iff \alpha_m = \frac{(\tilde{p}_m, r_m)}{(\tilde{p}_m, A p_m)}.$$

From the definition $P_m = V_m U_m^{-1} \iff V_m = P_m U_m$, if we consider the last column of V_m, v_m, then, since U_m is bidiagonal,

$$v_m = p_{m-1}\beta_m + p_m u_m \iff p_m = \frac{1}{u_m}(v_m - \beta_m p_{m-1}).$$

And since r_m is proportional to v_m, we can fix p_{m+1} so that

$$p_{m+1} = r_{m+1} + \beta_m p_m.$$

In the same manner, the definition of $\tilde{P}_m = W_m L_m^{-T}$, since L_m^{-T} is an upper bidiagonal matrix, leads to a two terms recurrence for the dual direction

$$\tilde{p}_{m+1} = \tilde{r}_{m+1} + \beta_m \tilde{p}_m,$$

with the same scalar β_m, since it comes from the same LU factorization of T_m. Since, from

$$(\tilde{p}_m, r_m) = (\tilde{r}_m, r_m) + \beta_{m-1}(\tilde{p}_{m-1}, r_m) = (\tilde{r}_m, r_m),$$

we obtain

$$\alpha_m = \frac{(\tilde{r}_m,\ r_m)}{(\tilde{p}_m,\ A p_m)} = \frac{(r_m, \tilde{r}_m)}{(A p_m, \tilde{p}_m)}.$$

Let us carry now the derivation of β_m. We have seen that from the BiCG recurrences we have

$$r_{m+1} = r_m - \alpha_m A p_m, \qquad \tilde{r}_{m+1} = \tilde{r}_m - \alpha_m A^T \tilde{p}_m. \tag{3.11}$$

Taking the inner product of $\tilde{r}_{m+1}$ with r_{m+1} gives

$$(\tilde{r}_{m+1}, r_{m+1}) = (\tilde{r}_m, r_{m+1}) - \alpha_m(\tilde{p}_m, Ar_{m+1}). \tag{3.12}$$

Moreover, using $p_{m+1} = r_{m+1} + \beta_m p_m$ and the bi-orthogonality $(Ap_{m+1}, \tilde{p}_m) = 0$ (for $m \neq m+1$), we obtain

$$0 = (Ap_{m+1}, \tilde{p}_m) = (Ar_{m+1}, \tilde{p}_m) + \beta_m(Ap_m, \tilde{p}_m) \implies (\tilde{p}_m, Ar_{m+1}) = -\beta_m(\tilde{p}_m, Ap_m). \tag{3.13}$$

Substituting (3.13) into (3.12) yields

$$(\tilde{r}_{m+1}, r_{m+1}) = (\tilde{r}_m, r_{m+1}) + \alpha_m\beta_m(\tilde{p}_m, Ap_m).$$

Finally, by the definition of α_m,

$$\alpha_m = \frac{(r_m, \tilde{r}_m)}{(Ap_m, \tilde{p}_m)} \qquad \Longrightarrow \qquad \alpha_m(\tilde{p}_m, Ap_m) = (\tilde{r}_m, r_m),$$

and by bi-orthogonality of the residuals, $(\tilde{r}_m, r_{m+1}) = 0$. Therefore,

$$(\tilde{r}_{m+1}, r_{m+1}) = \beta_m(\tilde{r}_m, r_m) \qquad \Longrightarrow \qquad \beta_m = \frac{(\tilde{r}_{m+1}, r_{m+1})}{(\tilde{r}_m, r_m)}.$$

Based on the derivations above, the BiCG method can be finally summarized as shown in algorithm 4. If a dual system with $A^\top$ is being solved, then in line 1 the dual residual

Algorithm 4 Biconjugate Gradient (BiCG) [11].

1: Compute $r_0 := b - Ax_0$. Choose $\tilde{r}_0$ such that $(r_0, \tilde{r}_0) \neq 0$.
2: Set $p_0 := r_0$, $\tilde{p}_0 := \tilde{r}_0$.
3: For $j = 0, 1, \ldots,$ until convergence Do:
4: $\quad \alpha_j := (r_j, \tilde{r}_j)/(Ap_j, \tilde{p}_j)$
5: $\quad x_{j+1} := x_j + \alpha_j p_j$
6: $\quad r_{j+1} := r_j - \alpha_j Ap_j$
7: $\quad \tilde{r}_{j+1} := \tilde{r}_j - \alpha_j A^\top \tilde{p}_j$
8: $\quad \beta_j := (r_{j+1}, \tilde{r}_{j+1})/(r_j, \tilde{r}_j)$
9: $\quad p_{j+1} := r_{j+1} + \beta_j p_j$
10: $\quad \tilde{p}_{j+1} := \tilde{r}_{j+1} + \beta_j \tilde{p}_j$
11: EndDo

$\tilde{r}_0$ should be defined as

$$\tilde{r}_0 = \tilde{b} - A^\top \tilde{x}_0,$$

and the update to the dual approximate solution,

$$\tilde{x}_{j+1} := \tilde{x}_j + \alpha_j \tilde{p}_j,$$

must be inserted after line 5.

We now want to understand how to adapt the BiCG algorithm to the matrix setting in order to solve our multiterm linear matrix equation (3.1), which we know is equivalent to solving the linear system (3.2) $L\,\mathrm{vec}(X) = \mathrm{vec}(fg^\top) = g \otimes f$ introduced in the previous section, without forming the coefficient matrix L, whose size would be prohibitively large. Instead, we apply the method directly to the associated matrix linear operator, which we recall is

$$\mathcal{L}(X) = \sum_{i=1}^{k} A_i X B_i.$$

First of all, we define its adjoint operator

$$\mathcal{L}^*(Y) = \sum_{i=1}^{k} A_i^\top Y B_i^\top,$$

where we will have ^{H} instead of $^\top$ in the complex case. The vectors $x, r, \tilde{r}, p, \tilde{p}$ in Algorithm 4 will be substituted by the matrices $X, R, \tilde{R}, P, \tilde{P}$, the euclidean scalar product $(\cdot,\cdot)$ by the Frobenius one, defined as $\langle U, V\rangle := \mathrm{trace}(U^\top V)$ (or $\mathrm{trace}(U^H V)$ in the complex setting) and the matrix-vector multiplication Ap or $A^\top \tilde{p}$ will be replaced by the application of the operator $\mathcal{L}(P)$ and its adjoint $\mathcal{L}^*(\tilde{P})$. These modifications lead to the construction of Algorithm 5.

Algorithm 5 Matrix-form Biconjugate Gradient (BiCG) for $\mathcal{L}(X) = fg^\top$.

1: Set $F := fg^\top$.
2: Compute $R_0 := F - \mathcal{L}(X_0)$. Choose $\tilde{R}_0$ such that $\langle R_0, \tilde{R}_0\rangle_F \neq 0$.
3: Set $P_0 := R_0$, $\tilde{P}_0 := \tilde{R}_0$.
4: For $m = 0, 1, \ldots,$ until convergence Do:
5: $\quad \alpha_m := \langle R_m, \tilde{R}_m\rangle_F / \langle \mathcal{L}(P_m), \tilde{P}_m\rangle_F$
6: $\quad X_{m+1} := X_m + \alpha_m P_m$
7: $\quad R_{m+1} := R_m - \alpha_m \mathcal{L}(P_m)$
8: $\quad \tilde{R}_{m+1} := \tilde{R}_m - \overline{\alpha}_m \mathcal{L}^*(\tilde{P}_m)$
9: $\quad \beta_m := \langle R_{m+1}, \tilde{R}_{m+1}\rangle_F / \langle R_m, \tilde{R}_m\rangle_F$
10: $\quad P_{m+1} := R_{m+1} + \beta_m P_m$
11: $\quad \tilde{P}_{m+1} := \tilde{R}_{m+1} + \overline{\beta}_m \tilde{P}_m$
12: EndDo

3.2.2 Transpose-Free Variants: CGS and BiCGSTAB

Each step of the Biconjugate Gradient (BiCG) algorithm requires a matrix-vector product with both A and A^T. However, note that the auxiliary vectors p_i^* or w_j generated

using A^T do not directly contribute to the approximate solution. Rather, they are only used to compute the scalars required by the iterations α_j and β_j. This naturally raises the question of whether it is possible to avoid the explicit use of A^T while still producing iterates closely related to those of BiCG. One motivation is that, in some applications, A is available only through approximations and not in explicit form; in such cases, the transpose A^T is typically not accessible.

The Conjugate Gradient Squared algorithm (CGS) was developed mainly to avoid using the transpose of A in the BiCG method and to gain faster convergence for roughly the same computational cost. The derivation of the method relies on simple algebra only, starting from Algorithm 4. For the detailed computations for the derivation of the algorithm to solve $Ax = b$, which for completeness, we report in Algorithm 6, we refer to [11]. We can observe that the algorithm involves no matrix-vector products with A^T.

Algorithm 6 Conjugate Gradient Squared (CGS) [11].

1: Compute $r_0 := b - Ax_0$; choose r_0^* (arbitrary) such that $(r_0, r_0^*) \neq 0$.
2: Set $p_0 := u_0 := r_0$.
3: For $j = 0, 1, 2, \ldots$, until convergence Do:
4: $\quad \alpha_j := (r_j, r_0^*)/(Ap_j, r_0^*)$
5: $\quad q_j := u_j - \alpha_j A p_j$
6: $\quad x_{j+1} := x_j + \alpha_j(u_j + q_j)$
7: $\quad r_{j+1} := r_j - \alpha_j A(u_j + q_j)$
8: $\quad \beta_j := (r_{j+1}, r_0^*)/(r_j, r_0^*)$
9: $\quad u_{j+1} := r_{j+1} + \beta_j q_j$
10: $\quad p_{j+1} := u_{j+1} + \beta_j(q_j + \beta_j p_j)$
11: EndDo

Instead, at each iteration it performs two matrix-vector products with A. In principle, this suggests that the method may converge in roughly half as many iterations as BiCG. Thus, the essential effect is to trade products with A^T for additional, and typically more useful, work with A.

The CGS algorithm is based on squaring the residual polynomial and, in the presence of irregular convergence, this may cause a significant accumulation of rounding errors, or even overflow. The Biconjugate Gradient Stabilized (BiCGSTAB) method is a variant of CGS designed to alleviate these difficulties. A new parameter ω_j is introduced to stabilize or smooth the convergence of the method. Following the computations made in [11], we can derive Algorithm 7 to solve the linear system $Ax = b$.

As we did previously for BiCG, we now show how the BiCGSTAB algorithm adapts to the matrix setting to solve our matrix equation $\mathcal{L}(X) = fg^\top$ (see Algorithm 8). Note

Algorithm 7 Biconjugate Gradient Stabilized (BiCGSTAB) [11].

Compute $r_0 := b - Ax_0$; choose r_0^* (arbitrary) such that $(r_0, r_0^*) \neq 0$.
Set $p_0 := r_0$.
For $j = 0, 1, \ldots,$ until convergence Do:
    $\alpha_j := (r_j, r_0^*)/(Ap_j, r_0^*)$
    $s_j := r_j - \alpha_j Ap_j$
    $\omega_j := (As_j, s_j)/(As_j, As_j)$
    $x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
    $r_{j+1} := s_j - \omega_j As_j$
    $\beta_j := \dfrac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \dfrac{\alpha_j}{\omega_j}$
    $p_{j+1} := r_{j+1} + \beta_j\big(p_j - \omega_j Ap_j\big)$
EndDo

that since BiCGSTAB is a transpose-free method, we do not need the adjoint operator $\mathcal{L}^*(Y)$.

Algorithm 8 Matrix-form BiCGSTAB for $\mathcal{L}(X) = fg^\top$.

Set $F := fg^\top$.
Compute $R_0 := F - \mathcal{L}(X_0)$; choose R_0^* (arbitrary) such that $\langle R_0, R_0^*\rangle_F \neq 0$.
Set $P_0 := R_0$.
For $m = 0, 1, \ldots,$ until convergence Do:
    $\alpha_m := \langle R_m, R_0^*\rangle_F \big/ \langle \mathcal{L}(P_m), R_0^*\rangle_F$
    $S_m := R_m - \alpha_m \mathcal{L}(P_m)$
    $\omega_m := \langle \mathcal{L}(S_m), S_m\rangle_F \big/ \langle \mathcal{L}(S_m), \mathcal{L}(S_m)\rangle_F$
    $X_{m+1} := X_m + \alpha_m P_m + \omega_m S_m$
    $R_{m+1} := S_m - \omega_m \mathcal{L}(S_m)$
    $\beta_m := \dfrac{\langle R_{m+1}, R_0^*\rangle_F}{\langle R_m, R_0^*\rangle_F} \dfrac{\alpha_m}{\omega_m}$
    $P_{m+1} := R_{m+1} + \beta_m\big(P_m - \omega_m \mathcal{L}(P_m)\big)$
EndDo

In terms of computational cost, BiCG performs one application of A and one of A^T per iteration (or $\mathcal{L}$ and $\mathcal{L}^*$ in the matrix setting), while BiCGSTAB is transpose-free, but requires two applications of A per iteration. Hence in practice, BiCGSTAB often exhibits a smoother and more robust convergence behavior, while BiCG may converge irregularly and is more sensitive to breakdown; consequently, BiCGSTAB is frequently preferred when the adjoint is unavailable or when stability is a concern, although the most effective choice remains problem-dependent.

In large-scale problems, the iterates of both Algorithms 5 and 8 may become expensive to store and manipulate in full form. This motivates the use of low-rank representations and truncation strategies, which are introduced in the next section and will be exploited to keep the computational cost and memory requirements under control.

3.3 Low-Rank Compression Strategies

We recall our linear multiterm matrix equation

$$\sum_{i=1}^{d} A_i X B_i = f g^\top, \tag{3.14}$$

and since the right-hand side $fg^\top$ has rank one, it is natural to expect that, for many cases of practical interest, the solution X has a low numerical rank, as observed numerically in [1]. However, this behavior is not guaranteed in general and depends primarily on the spectral properties and the conditioning of the linear operator. In general, if we have a matrix $C \in \mathbb{R}^{n\times m}$ with a low rank $r \ll \min\{m, n\}$, we can approximate it as

$$C \approx UV^\top,$$

where $U \in \mathbb{R}^{n\times r}$ and $V \in \mathbb{R}^{m\times r}$. This low-rank representation reduces both storage and computational costs, since we do not need to store the entire large matrix C, but only the factors U and V. Therefore, low-rank decompositions allow us to consider larger problem sizes without running into memory issues during allocations or computations. For this reason, when applying an iterative method to solve problem (3.14), we aim to preserve the low-rank structure of the matrix iterates. However, in the BiCGSTAB algorithm 8, some steps may increase the rank, for example, matrix linear combinations and the application of the operator $\mathcal{L}$. Therefore, these operations must be performed in a smart way to not increase the rank while keeping the low-rank representation during the iterations. In what follows, we assume that the initial guess $X_0 \in \mathbb{R}^{n\times m}$ admits a low-rank representation, or can be well approximated by one, namely

$$X_0 \approx X_0^{(1)} \big(X_0^{(2)}\big)^\top, \qquad X_0^{(1)} \in \mathbb{R}^{n\times r_0}, \quad X_0^{(2)} \in \mathbb{R}^{m\times r_0},$$

with $r_0 \ll \min\{n, m\}$. Already at the beginning of BiCGSTAB, we need to compute the initial residual

$$R_0 = fg^\top - \mathcal{L}(X_0) = fg^\top - A_1 X_0 B_1 - \ldots - A_d X_0 B_d,$$

and that leads to rank growth. For this reason, we form R_0 in a factored form by applying a compression step based on a skinny QR factorization followed by a truncated SVD.

Indeed, we can first rewrite R_0 as:

$$R_0 = [f, A_1 X_0^{(1)}, ..., A_d X_0^{(1)}] \begin{bmatrix} 1 & 0 \\ 0 & -I_{r_0 d} \end{bmatrix} [g, B_1^\top X_0^{(2)}, ..., B_d^\top X_0^{(2)}]^\top,$$

where $I_{r_0 d}$ denotes the identity matrix of size $r_0 d$. Then a skinny QR factorization is performed on the left and right matrices;

$$Q_1 R_1 = [f, A_1 X_0^{(1)}, ..., A_d X_0^{(1)}],\ Q_1 \in \mathbb{R}^{n \times (r_0 d+1)},\ R_1 \in \mathbb{R}^{(r_0 d+1)\times(r_0 d+1)}$$
$$Q_2 R_2 = [g, B_1^\top X_0^{(2)}, ..., B_d^\top X_0^{(2)}],\ Q_2 \in \mathbb{R}^{m \times (r_0 d+1)},\ R_2 \in \mathbb{R}^{(r_0 d+1)\times(r_0 d+1)}.$$

We denote that the skinny QR factorization can be performed since these matrices are relatively small and can be allocated, since r_0 is small and d in many practical situations is small. Now, a truncated SVD is executed on the resulting core matrix, obtaining

$$\tilde{U}\tilde{\Sigma}\tilde{V}^\top = R_1 \begin{bmatrix} 1 & 0 \\ 0 & -I_{r_0 d} \end{bmatrix} R_2^\top,\ \tilde{\Sigma} \in \mathbb{R}^{k \times k},$$

where the truncation parameter k is chosen between the index of the last singular value above a given tolerance and a maximum rank that we do not want to exceed. Finally, R_0 is decomposed in a low-rank form as

$$R_0 = (Q_1 \tilde{U} \tilde{\Sigma}^{1/2})(Q_2 \tilde{V} \tilde{\Sigma}^{1/2})^\top = R_0^{(1)} (R_0^{(2)})^\top.$$

The same strategy will then be applied to the other intermediate quantities generated within the iterations to control the rank growth. For example, for the solution's update of line 8 of Algorithm 8, we have that, assuming that each matrix of the previous iterations is already in a low-rank form,

$$\begin{aligned} X_{m+1} &= X_m + \alpha_m P_m + \omega_m S_m \\ &= X_m^{(1)} (X_m^{(2)})^\top + \alpha_m P_m^{(1)} (P_m^{(2)})^\top + \omega_m S_m^{(1)} (S_m^{(2)})^\top \\ &= [X_m^{(1)}, P_m^{(1)}, S_m^{(1)}] \begin{bmatrix} I_{r_X} & 0 & 0 \\ 0 & \alpha_m I_{r_P} & 0 \\ 0 & 0 & \omega_m I_{r_S} \end{bmatrix} [X_m^{(2)}, P_m^{(2)}, S_m^{(2)}]^\top, \end{aligned}$$

where I_{r_X}, I_{r_P}, I_{r_S} are the identity matrices of sizes r_X, r_P and r_S, which denote the ranks of X_m, P_m and S_m, respectively. Note that, as before, since $r_X + r_P + r_S$ is small, we can perform a skinny QR decomposition on the left and right matrices and a truncated SVD on the resulting core matrix, obtaining

$$X_{m+1} = X_{m+1}^{(1)} (X_{m+1}^{(2)})^\top.$$

Each Frobenius inner product is computed in a smart way to not create full matrices, using the cyclic property of the trace. For example, for line 5 of Algorithm 8, if $R_m = R_m^{(1)}(R_m^{(2)})^\top$ and $R_0^* = R_0^{*(1)}(R_0^{*(2)})^\top$ are low-rank matrices, we have

$$\begin{aligned}\langle R_m, R_0^*\rangle_F &= \langle R_m^{(1)}(R_m^{(2)})^\top, R_0^{*(1)}(R_0^{*(2)})^\top\rangle_F \\ &= tr\left(R_m^{(2)}(R_m^{(1)})^\top R_0^{*(1)}(R_0^{*(2)})^\top\right) \\ &= tr\left(\left((R_m^{(1)})^\top R_0^{*(1)}\right)\left((R_0^{*(2)})^\top R_m^{(2)}\right)\right).\end{aligned}$$

Therefore, the products inside the trace are performed on smaller matrices. Applying these strategies to each line of Algorithm 8, we obtain a low-rank version of BiCGSTAB, that can be summarized in Algorithm 9, where the low-rank truncations and the applications of the linear operator are described in Algorithms 10 and 11.

Algorithm 9 Low-rank matrix-form BiCGSTAB for $\mathcal{L}(X) = fg^\top$ with truncations.

1: Let $\mathcal{L}(X) := \sum_{i=1}^d A_i X B_i$.
2: Assume $X_0 \approx X_0^{(1)}(X_0^{(2)})^\top$ is given in low-rank form.
3: Compute $(\mathcal{L}(X_0))^{(1)}, (\mathcal{L}(X_0))^{(2)} := \mathbf{ApplyOp}\big(X_0^{(1)}, X_0^{(2)}\big)$.
4: Compute $R_0^{(1)}, R_0^{(2)} := \mathbf{Trunc}\Big([f,\ (\mathcal{L}(X_0))^{(1)}],\ \mathrm{blkdiag}(1, -I),\ [g,\ (\mathcal{L}(X_0))^{(2)}]\Big)$.
5: Choose $R_0^* \approx (R_0^*)^{(1)}\big((R_0^*)^{(2)}\big)^\top$ such that $\langle R_0, R_0^*\rangle_F \neq 0$ (e.g., $R_0^* = R_0$).
6: Set $P_0^{(1)}, P_0^{(2)} := R_0^{(1)}, R_0^{(2)}$.
7: For $m = 0, 1, \ldots,$ until convergence Do:
8: Compute $(\mathcal{L}(P_m))^{(1)}, (\mathcal{L}(P_m))^{(2)} := \mathbf{ApplyOp}\big(P_m^{(1)}, P_m^{(2)}\big)$.
9: $\alpha_m := \langle R_m, R_0^*\rangle_F / \langle \mathcal{L}(P_m), R_0^*\rangle_F$.
10: Compute $S_m^{(1)}, S_m^{(2)} := \mathbf{Trunc}\Big([R_m^{(1)},\ (\mathcal{L}(P_m))^{(1)}],\ \mathrm{blkdiag}(I, -\alpha_m I),\ [R_m^{(2)},\ (\mathcal{L}(P_m))^{(2)}]\Big)$.
11: Compute $(\mathcal{L}(S_m))^{(1)}, (\mathcal{L}(S_m))^{(2)} := \mathbf{ApplyOp}\big(S_m^{(1)}, S_m^{(2)}\big)$.
12: $\omega_m := \langle \mathcal{L}(S_m), S_m\rangle_F / \langle \mathcal{L}(S_m), \mathcal{L}(S_m)\rangle_F$.
13: Compute $X_{m+1}^{(1)}, X_{m+1}^{(2)} := \mathbf{Trunc}\Big([X_m^{(1)},\ P_m^{(1)},\ S_m^{(1)}],\ \mathrm{blkdiag}(I, \alpha_m I, \omega_m I),\ [X_m^{(2)},\ P_m^{(2)},\ S_m^{(2)}]\Big)$.
14: Compute $R_{m+1}^{(1)}, R_{m+1}^{(2)} := \mathbf{Trunc}\Big([S_m^{(1)},\ (\mathcal{L}(S_m))^{(1)}],\ \mathrm{blkdiag}(I, -\omega_m I),\ [S_m^{(2)},\ (\mathcal{L}(S_m))^{(2)}]\Big)$.
15: $\beta_m := \dfrac{\langle R_{m+1}, R_0^*\rangle_F}{\langle R_m, R_0^*\rangle_F}\dfrac{\alpha_m}{\omega_m}$.
16: Compute $P_{m+1}^{(1)}, P_{m+1}^{(2)} := \mathbf{Trunc}\Big([R_{m+1}^{(1)},\ P_m^{(1)},\ (\mathcal{L}(P_m))^{(1)}],\ \mathrm{blkdiag}(I, \beta_m I, -\beta_m\omega_m I),$
$[R_{m+1}^{(2)},\ P_m^{(2)},\ (\mathcal{L}(P_m))^{(2)}]\Big)$.
17: EndDo

We can apply the same strategies and computations to the BiCG method in its matrix formulation (Algorithm 5), once we have constructed a function for the application of the adjoint operator $\mathcal{L}^*$; see Algorithm 12. Finally, we obtain the low-rank version of BiCG in Algorithm 13.

Algorithm 10 Trunc: truncation of $A = LCR^\top$ into a low-rank factorization $UV^\top$ with rank $\le r_{\max}$.

Skinny ***QR****:* $[Q_1, R_1] := \mathrm{qr}(L, 0)$ ▷ $L = Q_1 R_1$
Skinny ***QR****:* $[Q_2, R_2] := \mathrm{qr}(R, 0)$ ▷ $R = Q_2 R_2$
$G := R_1\, C\, R_2^\top$
Economy ***SVD****:* $[U_1, S, V_1] := \mathrm{svd}(G, \texttt{econ})$
$\ell := \mathrm{diag}(S)$ ▷ singular values in nonincreasing order
$i^\star := \max\{\, i : \ell_i/\ell_1 \ge \texttt{tol}_{\texttt{trunc}} \,\}$ ▷ rank selection
$k := \min(i^\star, r_{\max})$
$S_k := S(1\!:\!k, 1\!:\!k)$; $U_{1k} := U_1(:, 1\!:\!k)$; $V_{1k} := V_1(:, 1\!:\!k)$
$U := Q_1\, (U_{1k}\, S_k^{1/2})$
$V := Q_2\, (V_{1k}\, S_k^{1/2})$

Algorithm 11 ApplyOp: apply the low-rank operator $\mathcal{L}(P) = \sum_{i=1}^{d} A_i P B_i$ to $P = P_1 P_2^\top$ in factored form.

$d := \mathrm{numel}(\texttt{AA})$ ▷ $\texttt{AA} = \{A_1, \ldots, A_d\}$, $\texttt{BB} = \{B_1, \ldots, B_d\}$
$r := \mathrm{size}(P_1, 2)$
Initialize $LP_1 \in \mathbb{R}^{n \times (dr)}$ and $LP_2 \in \mathbb{R}^{m \times (dr)}$ to zeros
for $i = 1, 2, \ldots, d$ **do**
    $\texttt{cols} := (i-1)r + (1\!:\!r)$
    $LP_1(:, \texttt{cols}) := A_i\, P_1$ ▷ $A_i = \texttt{AA}\{i\}$
    $LP_2(:, \texttt{cols}) := B_i^\top\, P_2$ ▷ $B_i = \texttt{BB}\{i\}$
end for
$LP_1 = [A_1 P_1, \ldots, A_d P_1]$, $LP_2 = [B_1^\top P_2, \ldots, B_d^\top P_2]$. ▷ $\mathcal{L}(P) \approx LP_1\, LP_2^\top$

Algorithm 12 ApplyOpH: apply the adjoint low-rank operator $\mathcal{L}^*(P) = \sum_{i=1}^{d} A_i^H P B_i^H$ to $P = P_1 P_2^\top$ in factored form.

$d := \mathrm{numel}(\texttt{AA})$ ▷ $\texttt{AA} = \{A_1, \ldots, A_d\}$, $\texttt{BB} = \{B_1, \ldots, B_d\}$
$r := \mathrm{size}(P_1, 2)$
Initialize $LP_1 \in \mathbb{R}^{n \times (dr)}$ and $LP_2 \in \mathbb{R}^{m \times (dr)}$ to zeros
for $i = 1, 2, \ldots, d$ **do**
    $\texttt{cols} := (i-1)r + (1\!:\!r)$
    $LP_1(:, \texttt{cols}) := A_i^H\, P_1$ ▷ $A_i = \texttt{AA}\{i\}$
    $LP_2(:, \texttt{cols}) := B_i\, P_2$ ▷ $B_i = \texttt{BB}\{i\}$
end for
$LP_1 = [A_1^H P_1, \ldots, A_d^H P_1]$, $LP_2 = [B_1 P_2, \ldots, B_d P_2]$. ▷ $\mathcal{L}^*(P) \approx LP_1\, LP_2^\top$

Algorithm 13 Low-rank matrix-form BiCG for $\mathcal{L}(X) = fg^\top$ with truncations.

Let $\mathcal{L}(X) := \sum_{i=1}^{d} A_i X B_i$ and $\mathcal{L}^*(X) := \sum_{i=1}^{d} A_i^H X B_i^H$.
Assume $X_0 \approx X_0^{(1)}(X_0^{(2)})^\top$ is given in low-rank form.
Compute $(\mathcal{L}(X_0))^{(1)}, (\mathcal{L}(X_0))^{(2)} := \mathbf{ApplyOp}\big(X_0^{(1)}, X_0^{(2)}\big)$.
Compute $R_0^{(1)}, R_0^{(2)} := \mathbf{Trunc}\Big([f,\ (\mathcal{L}(X_0))^{(1)}],\ \mathrm{blkdiag}(1, -I),\ [g,\ (\mathcal{L}(X_0))^{(2)}]\Big)$.
Choose $R_0^* \approx (R_0^*)^{(1)}\big((R_0^*)^{(2)}\big)^\top$ (e.g., $R_0^* = R_0$).
Set $P_0^{(1)}, P_0^{(2)} := R_0^{(1)}, R_0^{(2)}$ and $(P_0^*)^{(1)}, (P_0^*)^{(2)} := (R_0^*)^{(1)}, (R_0^*)^{(2)}$.
For $m = 0, 1, \ldots,$ until convergence or $m = \texttt{maxit} - 1$ Do:
  $\rho_m := \langle R_m^*, R_m \rangle_F$.
  Compute $(\mathcal{L}(P_m))^{(1)}, (\mathcal{L}(P_m))^{(2)} := \mathbf{ApplyOp}\big(P_m^{(1)}, P_m^{(2)}\big)$.
  $\alpha_m := \rho_m \big/ \langle P_m^*, \mathcal{L}(P_m) \rangle_F$.
  Compute $X_{m+1}^{(1)}, X_{m+1}^{(2)} := \mathbf{Trunc}\Big([X_m^{(1)},\ P_m^{(1)}],\ \mathrm{blkdiag}(I, \alpha_m I),\ [X_m^{(2)},\ P_m^{(2)}]\Big)$.
  Compute $R_{m+1}^{(1)}, R_{m+1}^{(2)} := \mathbf{Trunc}\Big([R_m^{(1)},\ (\mathcal{L}(P_m))^{(1)}],\ \mathrm{blkdiag}(I, -\alpha_m I),\ [R_m^{(2)},\ (\mathcal{L}(P_m))^{(2)}]\Big)$.
  Compute $(\mathcal{L}^*(P_m^*))^{(1)}, (\mathcal{L}^*(P_m^*))^{(2)} := \mathbf{ApplyOpH}\big((P_m^*)^{(1)}, (P_m^*)^{(2)}\big)$.
  Compute $(R_{m+1}^*)^{(1)}, (R_{m+1}^*)^{(2)} := \mathbf{Trunc}\Big([(R_m^*)^{(1)},\ (\mathcal{L}^*(P_m^*))^{(1)}],\ \mathrm{blkdiag}(I, -\overline{\alpha}_m I),$
    $[(R_m^*)^{(2)},\ (\mathcal{L}^*(P_m^*))^{(2)}]\Big)$.
  $\rho_{m+1} := \langle R_{m+1}^*, R_{m+1} \rangle_F, \quad \beta_m := \rho_{m+1}/\rho_m$.
  Compute $P_{m+1}^{(1)}, P_{m+1}^{(2)} := \mathbf{Trunc}\Big([R_{m+1}^{(1)},\ P_m^{(1)}],\ \mathrm{blkdiag}(I, \beta_m I),\ [R_{m+1}^{(2)},\ P_m^{(2)}]\Big)$.
  Compute $(P_{m+1}^*)^{(1)}, (P_{m+1}^*)^{(2)} := \mathbf{Trunc}\Big([(R_{m+1}^*)^{(1)},\ (P_m^*)^{(1)}],\ \mathrm{blkdiag}(I, \overline{\beta}_m I),$
    $[(R_{m+1}^*)^{(2)},\ (P_m^*)^{(2)}]\Big)$.
EndDo

We observe that, in practical situations, in both Algorithms 9 and 13, the *computational* residual can be replaced by the *true* residual

$$R = fg^\top - \mathcal{L}(X),$$

computed as before in a low-rank form, since the residual obtained from the method's updates may be inaccurate due to truncation. Indeed, computing the true residual is important to avoid an excessive accumulation of truncation errors that may arise when using the computational one. Computing the true residual within the iterations of the algorithm also helps us to obtain more accurate solutions, reducing the final errors.

The low-rank formulation developed in this chapter is meant to keep the BiCGSTAB and BiCG iterates in a factored form to control the rank growth by means of truncations. This compression strategy significantly reduces memory requirements and enables the solution of larger-scale instances than the corresponding full-rank implementation. In the next chapter, we will test and compare the algorithms previously introduced, to solve the matrix equation (2.12), arising from the discretization of a non-autonomous ODE system reformulated via $\star$-algebra, using experimental data coming from an NMR procedure.

Chapter 4

Numerical Results for the NMR Model

In this chapter, we test and compare the different numerical methods introduced previously. In particular, we test them on real data coming from a nuclear magnetic resonance (NMR) spectroscopy simulation, a technique used in several industrial and chemical applications, for example. More details on the NMR model will be explained in Section 4.1, where we also detail how the differential equation modeling the NMR simulation is transformed into a matrix equation. In this section, we refer to [14], where a more detailed analysis of the NMR characteristics and the various calculations involved in the model description have been carried out. In Section 4.2, we present a comparison of the methods used to solve the derived matrix equation of the form (2.12) arising from the $\star$-approach, for different values of the parameters and of the dimension of the problem, with the aim of understanding whether low-rank versions of the algorithms provide a real advantage compared to the Krylov methods already implemented in MATLAB. We will compare them in terms of both efficiency (computational time, memory usage) and accuracy (errors, residuals).

4.1 Simulation for NMR

We examine the problem of understanding the spin dynamics of groups of particles such as proteins, membrane systems or macromolecular complexes, following the description made in [14]. We consider Nuclear Magnetic Resonance (NMR) spectroscopy, a powerful and widely used method that studies how atomic nuclei respond to magnetic fields. As explained in more detail in [15] and [16], this technique not only shows how particles move, but also helps reveal their chemical properties and the three-dimensional shape of molecules. In particular, we focus on solid-state NMR spectroscopy with magic-

angle spinning (MAS) [16]. In this technique, the sample is quickly rotated around an axis tilted by the so-called *magic angle*, $\theta_m = \cos^{-1}(1/\sqrt{3}) \approx 54.7°$, relative to the external magnetic field. It is referred to as "magic" because rapidly spinning the sample at this specific angle averages out the spatial components of the motion that depend on its orientation with respect to the magnetic field. A schematic illustration is shown in Figure 4.1. The evolution of the spin system in a MAS NMR experiment can be modeled

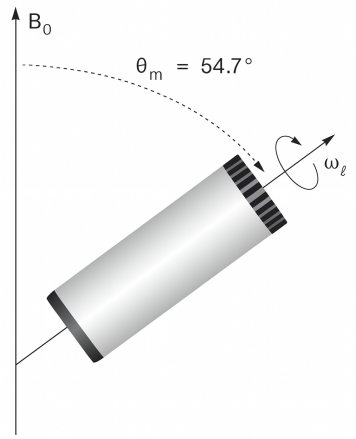


Figure 4.1: Schematic drawing of an MAS rotor during NMR (adapted from [17]).

using a Schrödinger equation

$$\begin{cases} \dfrac{d}{dt}\,\psi(t) = -i\,\mathcal{H}(t)\,\psi(t), \quad t \geq 0, \\ \psi(0) = \psi_0, \end{cases} \tag{4.1}$$

where $\mathcal{H}(t)$ represents the Hamiltonian operator of the system, the wave function $\psi(t)$ represents the state of the system at time t and i is the imaginary unit. We note that the model does not include any spatial dependence, since we are dealing with a MAS experiment, as mentioned above, and that the Hamiltonian depends on time. Therefore, we can state that the Schrödinger equation (4.1) represents a system of non-autonomous ODEs. Moreover, the dimension of the problem is 2^{N_s}, where N_s is the number of spins involved in the system; hence, even a small number of spins leads to a very large system of ODEs to be solved.

In order to describe the Hamiltonian in problem (4.1), we first recall the definition of Pauli matrices. We set

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

These matrices are Hermitian and satisfy the same characteristic equation, namely $\lambda^2 - 1 = 0$. Moreover, it can be easily verified that, together with the 2×2 identity matrix

I_2, they form a basis for the space of 2×2 complex matrices. They are commonly used to represent the spin components. More precisely, for a system consisting of N_s spin-1/2 particles, we define, for each spin $k = 1, \ldots, N_s$,

$$I_{k_x} := \frac{1}{2}\, I_{2^{(k-1)}} \otimes \sigma_x \otimes I_{2^{(N_s-k)}},$$

Similarly, we define I_{k_y} and I_{k_z}. Observe that all these matrices have size $2^{N_s} \times 2^{N_s}$. As an example, we consider a homonuclear system, namely a system composed of atoms of the same chemical element, described by a simplified yet still descriptive form given by the contributions of isotropic chemical shifts and dipolar interactions:

$$\mathcal{H}(t) = \mathcal{H}_{ICS} + \mathcal{H}_{DD}(t). \tag{4.2}$$

In this formulation, we have omitted anisotropic chemical shifts, radio-frequency pulses, quadrupolar interactions, and indirect spin–spin couplings. The terms included here are defined according to [18], where a more detailed expression of $\mathcal{H}$ can also be found. As shown in [14], after some computations, we can write the time-dependent dipole-dipole coupling component of the Hamiltonian as

$$H_{DD}(t) = \delta \sum_{k=1}^{N_s-1} \sum_{q=k+1}^{N_s} \frac{1}{r_{kq}^3} \left[\sqrt{2}\, \sin(2\beta_{kq}) \cos(\gamma_{kq} + \omega_r t) - \sin^2(\beta_{kq}) \cos(2\gamma_{kq} + 2\omega_r t)\right] M_{kq},$$

where

$$M_{kq} = 2I_{k_z} I_{q_z} - \big(I_{k_x} I_{q_x} + I_{k_y} I_{q_y}\big), \tag{4.3}$$

and δ is a physical constant, β_{kq} and γ_{kq} are Euler angles, ω_r denotes the MAS frequency, r_{kq} is the distance between spins k and q, and I_{k_α} represent the spin operators introduced above. We now turn to the time-independent part of the Hamiltonian that is not modulated by the MAS rotation. It can be defined as follows:

$$\mathcal{H}_{ICS} = \sum_{k=1}^{N_s} \Omega_k\, I_{k_z}, \tag{4.4}$$

where Ω_k are values representing chemical shift differences (dependent on the magnetic field) and are determined by the physical properties of the sample.

As outlined at the beginning of subsection 2.2.1, we can rewrite the coefficient matrix of system (4.1) as

$$\tilde{A}(t) = -i\,\mathcal{H}(t) = -i\big(\mathcal{H}_{\mathrm{ICS}} + \mathcal{H}_{\mathrm{DD}}(t)\big) = -i\big(C + M_1 g_1(t) + M_2 g_2(t) + M_3 g_3(t) + M_4 g_4(t)\big),$$

where the constant part matrix C, the matrices M_i, and the time-dependent functions $g_i(t)$ are obtained by carrying out the calculations in equations (4.3) and (4.4). In particular,

$$C = \mathcal{H}_{ICS} = \sum_{k=1}^{N_s} \Omega_k\, I_{k_z},$$

and

$$M_1 = \sum_{k=1}^{N_s-1} \sum_{q=k+1}^{N_s} \frac{\sin(2\beta_{kq})\cos(\gamma_{kq})}{r_{kq}^3} M_{kq}, \qquad M_2 = \sum_{k=1}^{N_s-1} \sum_{q=k+1}^{N_s} \frac{\sin(2\beta_{kq})\sin(\gamma_{kq})}{r_{kq}^3} M_{kq},$$

$$M_3 = \sum_{k=1}^{N_s-1} \sum_{q=k+1}^{N_s} \frac{\sin^2(\beta_{kq})\cos(2\gamma_{kq})}{r_{kq}^3} M_{kq}, \qquad M_4 = \sum_{k=1}^{N_s-1} \sum_{q=k+1}^{N_s} \frac{\sin^2(\beta_{kq})\sin(2\gamma_{kq})}{r_{kq}^3} M_{kq},$$

so

$$H_{DD}(t) = \delta\sqrt{2}\, M_1 \cos(\omega_r t) - \delta\sqrt{2}\, M_2 \sin(\omega_r t) - \delta\, M_3 \cos(2\omega_r t) + \delta\, M_4 \sin(2\omega_r t).$$

Finally, the time-dependent functions are

$$g_1(t) = \delta\sqrt{2}\, \cos(\omega_r t), \qquad g_2(t) = -\delta\sqrt{2}\, \sin(\omega_r t),$$

$$g_3(t) = -\delta\, \cos(2\omega_r t), \qquad g_4(t) = \delta\, \sin(2\omega_r t).$$

Hence, the problem, before the discretization via Legendre polynomials, rewrites as:

$$\begin{cases} \dfrac{d}{dt}\psi(t) = \tilde{A}(t)\psi(t) = -i\left(C + \sum_{i=1}^{4} M_i g_i(t)\right)\psi(t), \quad t \geq 0, \\ \psi(0) = \psi_0, \end{cases} \tag{4.5}$$

We can also rewrite the time-dependent part of the problem in an equivalent way using complex exponentials; this representation allows for a more compact and algebraically convenient expression. Let us define

$$M_a = M_1 + \mathrm{i}M_2, \qquad M_b = M_3 + \mathrm{i}M_4,$$

and denote by M_a^* and M_b^* their conjugate transpose. The Hamiltonian of our system will be given by:

$$\tilde{A}(t) = -\mathrm{i}\Bigg(C + \frac{\sqrt{2}}{2}\,\delta\, e^{\mathrm{i}\omega_r t}\, M_a + \frac{\sqrt{2}}{2}\,\delta\, e^{-\mathrm{i}\omega_r t}\, M_a^* - \frac{1}{2}\,\delta\, e^{2\mathrm{i}\omega_r t}\, M_b - \frac{1}{2}\,\delta\, e^{-2\mathrm{i}\omega_r t}\, M_b^* \Bigg).$$

After defining $f_1(t) = \frac{\sqrt{2}}{2}\,\delta\, e^{\mathrm{i}\omega_r t}$, $f_2(t) = \frac{\sqrt{2}}{2}\,\delta\, e^{-\mathrm{i}\omega_r t}$, $f_3(t) = \frac{1}{2}\,\delta\, e^{2\mathrm{i}\omega_r t}$, $f_4(t) = \frac{1}{2}\,\delta\, e^{-2\mathrm{i}\omega_r t}$ and $A_1 = -iM_a$, $A_2 = -iM_a^*$, $A_3 = iM_b$, $A_4 = iM_b^*$, and introducing the constant part of the system through $f_5(t) = 1$, $A_5 = -iC$, the problem can be rewritten as:

$$\frac{d}{dt}\psi(t) = \left(\sum_{i=1}^{5} A_i f_i(t)\right)\psi(t), \quad \psi(0) = \psi_0.$$

Now, following the description made in the previous chapters, via the $\star$-approach we can obtain a $\star$-linear system that can be discretized using Legendre polynomials, obtaining

$$(I_{MN} - \mathcal{A}_M)x = \psi_0 \otimes \varphi_M(0),$$

with $\mathcal{A}_M = \sum_{k=1}^{5} A_k \otimes F_M^{(k)}$, where $F_M^{(k)}$ are the Legendre coefficient matrices of our function basis $\{f_k\}$ and $\varphi_M(0)$ is the vector containing the Legendre polynomial basis evaluated in zero. We know from Section 2.2.1 that this linear system can be rewritten in an equivalent way as a matrix equation:

$$X - \sum_{k=1}^{5} F_M^{(k)} X A_k^\top = \varphi_M(0)\psi_0^\top, \tag{4.6}$$

and the final solution to our problem will be given by multiplying X by the Legendre coefficient matrix of the Heaviside function Θ, as underlined before. For further details concerning the calculations and the parameters involved, we refer the reader to [14].

4.2 Numerical Experiments

We now want to compare our low-rank methods with other well-known and widely used approaches for solving the multiterm linear matrix equation (4.6), obtained from the $\star$-discretization of the non-autonomous system of ODEs (4.5), introduced in the previous sections. The comparison will be carried out in terms of computational time, errors, and residuals, in order to assess whether the low-rank algorithms can be competitive and potentially converge faster, or require less computational effort, than standard Krylov ones when solving matrix equations of increasing size. In this section, we compare some Krylov methods, namely GMRES, BiCG, and BiCGSTAB in their standard form, as introduced in Section 3.2. These solvers are pre-implemented MATLAB routines and are applied to the vectorized linear system associated with (4.6). In our implementation, the coefficient matrix of the vectorized problem is never assembled explicitly: the required matrix-vector products are computed using the linear matrix operator, by exploiting the underlying Kronecker structure, as underlined in Section 3.1. In addition, we consider the low-rank variants of BiCG and BiCGSTAB, described in Section 3.3 in Algorithms 13 and 9, respectively.

To establish a connection with a realistic physical setting, we make use of the raw molecular coordinates of the cationic tin oxo-cluster. Specifically, we extract from this data set the first N_s protons, so as to construct a system composed of N_s spins. The Cartesian coordinates of these nuclei are precisely computed in the accompanying code. However, since the dimension of the Hilbert space grows exponentially as 2^{N_s}, increasing

N_s quickly leads to computational limitations: either the available memory becomes insufficient or the algorithms become too slow to be efficiently executed. The challenge, therefore, is to develop faster and more efficient algorithms, taking advantage of different techniques to obtain a solution to the model even for higher values of N_s. Accurate values of the nuclear chemical shifts are, in general, difficult to determine in solid-state NMR. In our numerical experiments, the chemical shifts are not randomly generated. Instead, we assign them deterministically as N_s equally spaced points in the interval $[-2, 2]$ ppm, and then convert them into Hz by multiplying by the proton Larmor frequency, fixed at 500 MHz. This choice provides a simple, reproducible, and controlled distribution of chemical-shift offsets across the spins. The initial condition is constructed starting from a randomly generated vector, which is then transformed by the operator defining the initial excitation of the system. The resulting vector is finally normalized, in order to obtain a unit-norm initial state consistent with the chosen physical configuration. Throughout the simulations, we set the spinning frequency to $\omega_r = v_r \cdot 2\pi 10^3$, with $v_r = 150$ and we study the Schrödinger equation with Hamiltonian given in eq. (4.2) over different time intervals $t \in \left[0, k\frac{4\pi}{\omega_r}\right]$, with $k \in \mathbb{N}$. We set the stopping tolerance for the convergence criteria of all the tested iterative methods to $\texttt{tol} = 10^{-6}$. To compute a reference solution, we use MATLAB's built-in Runge-Kutta solver `ode89` with a lower tolerance, $\texttt{tol\_ref} = 10^{-10}$, so that we can compare, in terms of error, the solutions produced by our algorithms against the reference solution. The experiments are conducted by varying several parameters, including m, the size of the discretization matrix. The initial value is set to $m = 200$, and then m is varied according to the parameter k as $m = 200\,k$. Another important parameter is the number of spins N_s, since the dimension of the problem grows rapidly as 2^{N_s}. We therefore tested $N_s = 10, 14, 18$ and 19.

We start by testing the methods with $N_s = 10$ and $k = 1, 2, 4$; therefore, the dimension of the problem is equal to $N = 2^{10} = 1024$. Varying k we are doubling or quadruplicating the dimension of the discretization matrix and the time interval, hence we also need to increase the maximum rank parameter r_{max} to ensure convergence. The value of r_max is of fundamental importance for the low-rank variants, as it can be interpreted as the maximum amount of memory that can be allocated. If it is chosen too small, convergence will not be achieved because the truncations become too aggressive and remove too much information. If it is chosen too large, one risks retaining all the information but ending up with matrices that are not that small, which prevents any computational cost savings and makes the methods non-competitive. Therefore, selecting r_max amounts to a trade-off between the accuracy and efficiency of the algorithm. The results are reported in Table 4.1.

As can be observed in Table 4.1, low-rank variants perform very well for a relatively

Table 4.1: Comparison of iterative methods for $N_s = 10$ with varying k and $r_{\max}$.

Method	k	$r_{\max}$	Time (s)	Iterations	Residual
BiCGSTAB	1	–	0.438	6	8.9498e-07
BiCG	1	–	0.881	6	2.4732e-07
GMRES	1	–	0.745	6	1.9434e-07
BiCGSTAB low-rank	1	25	0.357	4	1.1157e-07
BiCG low-rank	1	25	0.379	6	5.0372e-07
BiCGSTAB	2	–	1.075	7	6.8178e-07
BiCG	2	–	1.981	7	2.5561e-07
GMRES	2	–	1.475	7	1.5735e-07
BiCGSTAB low-rank	2	50	0.752	4	5.4366e-07
BiCG low-rank	2	50	0.926	7	4.6165e-07
BiCGSTAB	4	–	3.726	9	6.8608e-08
BiCG	4	–	6.824	8	6.5250e-07
GMRES	4	–	3.957	8	2.7469e-07
BiCGSTAB low-rank	4	53	1.417	5	1.3313e-07
BiCG low-rank	4	53	1.815	9	7.4538e-07

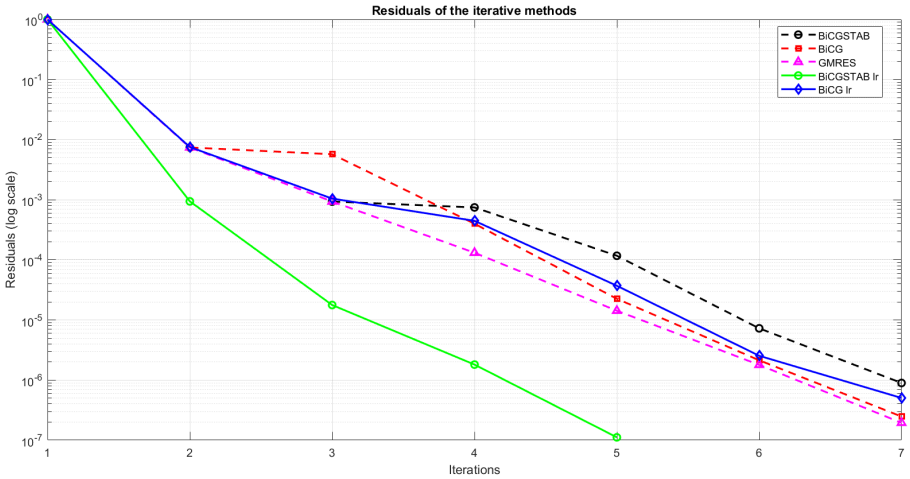
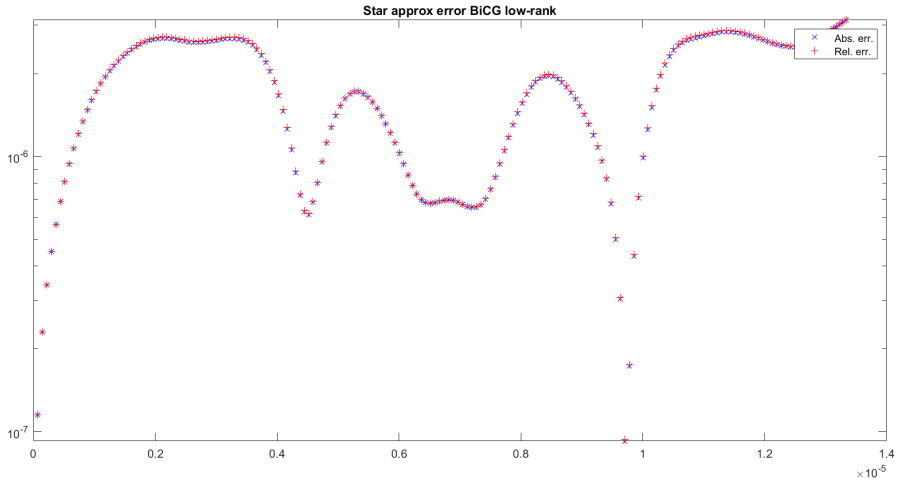


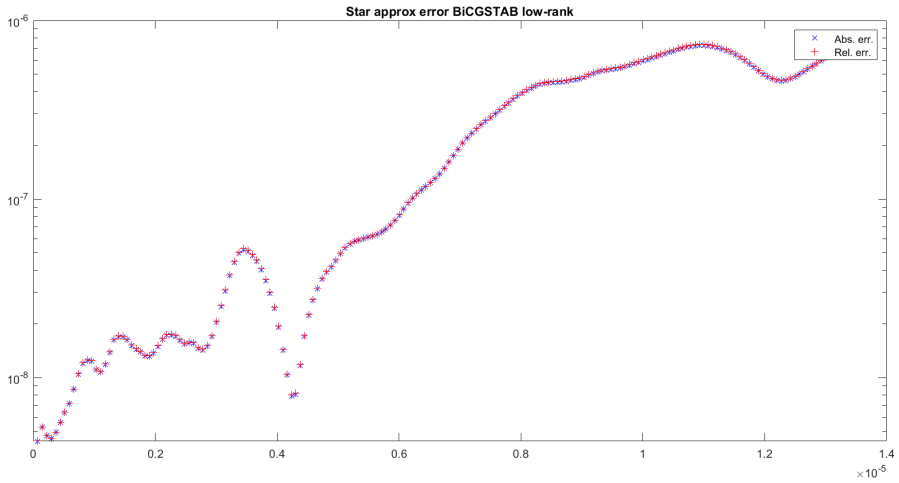
Figure 4.2: Residual histories for the different methods ($N_s = 10$ and $k = 1$). The non-dashed lines correspond to the low-rank variants.

small N_s, both in terms of iteration count and computational cost, reducing execution time with respect to their standard counterparts. For example, we show the history of the relative residuals of the different methods for $k = 1$ in Figure 4.2. Regarding the error with respect to the reference solution (for example, for $k = 1$, in Figure 4.3), the low-rank methods always achieve errors close to, or below, 10^{-5}, which is satisfactory for

our purposes. In addition, the overall shape of the solution is preserved in both the real and imaginary parts, as shown, for example, for $k = 1$, in Figure 4.4.



(a) BiCG low-rank.



(b) BiCGSTAB low-rank.

Figure 4.3: Errors of the low-rank methods compared to the reference solution obtained with `ode89` using a tight tolerance ($N_s = 10$ and $k = 1$).

We now test the methods for $N_s = 14$, so that the problem size becomes $N = 2^{14} = 16384$. As stated previously, when the problem size increases, the rank of the solution also grows, but much more slowly than the size, and it typically remains relatively small. However, we need to update the maximum rank allowed in the truncations in order to ensure convergence and to obtain good results. As shown in Table 4.2 the low-rank version of BiCG and BiCGSTAB requires essentially the same number of iterations as the full method, but with a lower runtime. The gain in computational time and cost increases as the time interval increases for $k = 4$. For completeness, we include the

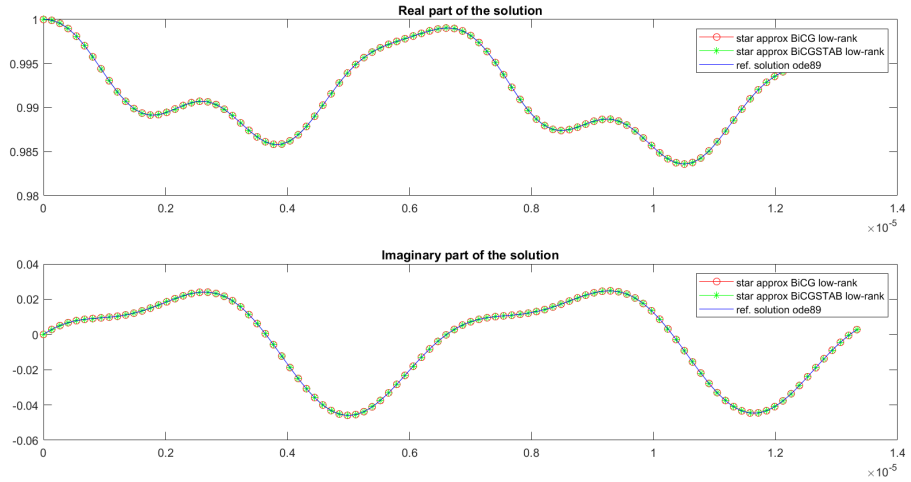


Figure 4.4: Comparison between the reference solution and the solutions obtained with the low-rank methods ($N_s = 10$ and $k = 1$).

errors plot for $k = 2$ in Figure 4.6, where, as we can observe, they remain small for both methods, together with the solution plot in Figure 4.7 and the history of the relative residuals in Figure 4.5.

Table 4.2: Comparison of iterative methods for $N_s = 14$ with varying k and $r_{\max}$.

Method	k	$r_{\max}$	Time (s)	Iterations	Residual
BiCGSTAB	1	–	5.508	7	5.5817e-07
BiCG	1	–	10.472	7	4.8996e-07
GMRES	1	–	8.382	7	1.6133e-07
BiCGSTAB low-rank	1	28	4.198	4	8.5596e-07
BiCG low-rank	1	28	7.871	7	3.0162e-07
BiCGSTAB	2	–	16.081	8	9.0454e-07
BiCG	2	–	44.471	8	3.0574e-07
GMRES	2	–	21.352	8	1.8181e-07
BiCGSTAB low-rank	2	55	11.993	4	9.0204e-07
BiCG low-rank	2	55	13.454	8	3.2231e-07
BiCGSTAB	4	–	76.708	9	7.2437e-07
BiCG	4	–	110.480	9	7.1683e-07
GMRES	4	–	68.659	9	5.6573e-07
BiCGSTAB low-rank	4	70	22.642	6	7.9533e-07
BiCG low-rank	4	70	22.834	9	8.0153e-07

Now, if we increase, for example, the number of spins to $N_s = 18, 19$, the dimension becomes $N = 2^{18} = 262144$ and $N = 2^{19} = 524288$, and the standard MATLAB

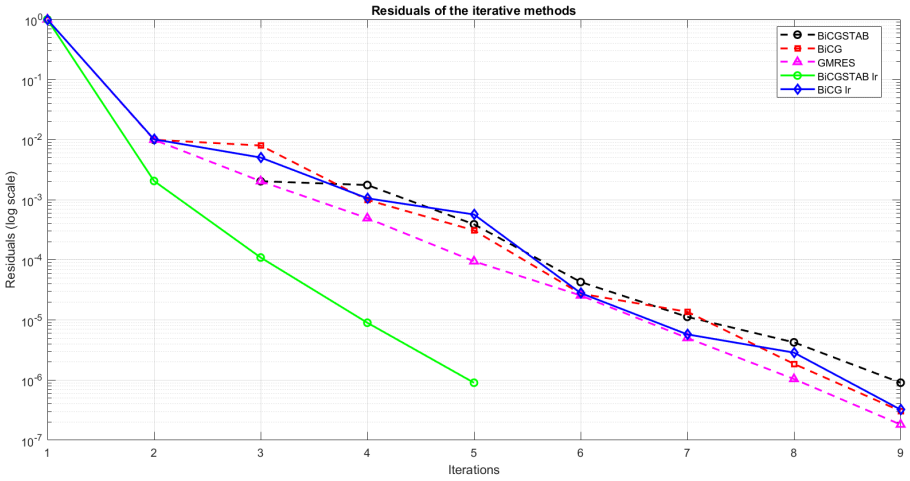
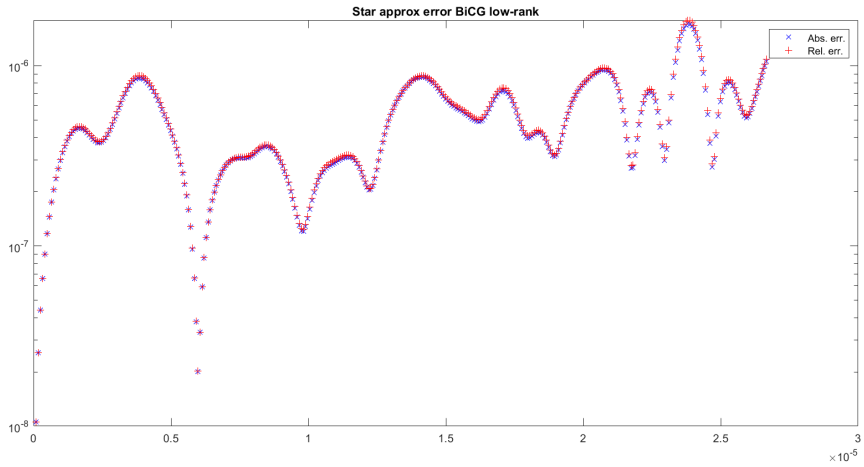
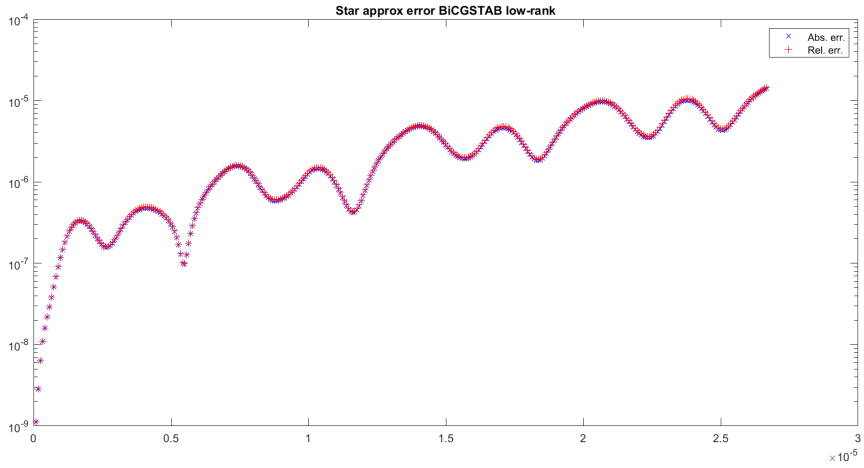


Figure 4.5: Residual histories for the different methods ($N_s = 14$ and $k = 2$).



(a) BiCG low-rank.



(b) BiCGSTAB low-rank.

Figure 4.6: Errors of the low-rank methods compared to the reference solution obtained with `ode89` using a tight tolerance ($N_s = 14$ and $k = 2$).

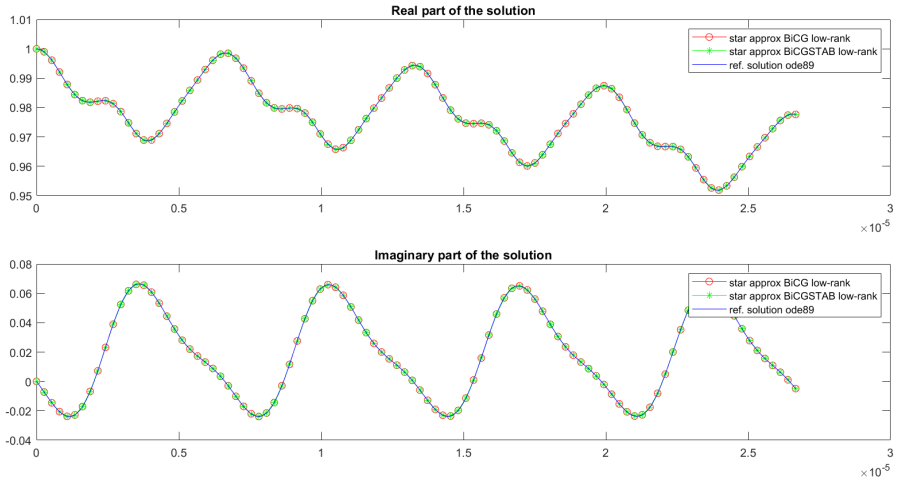


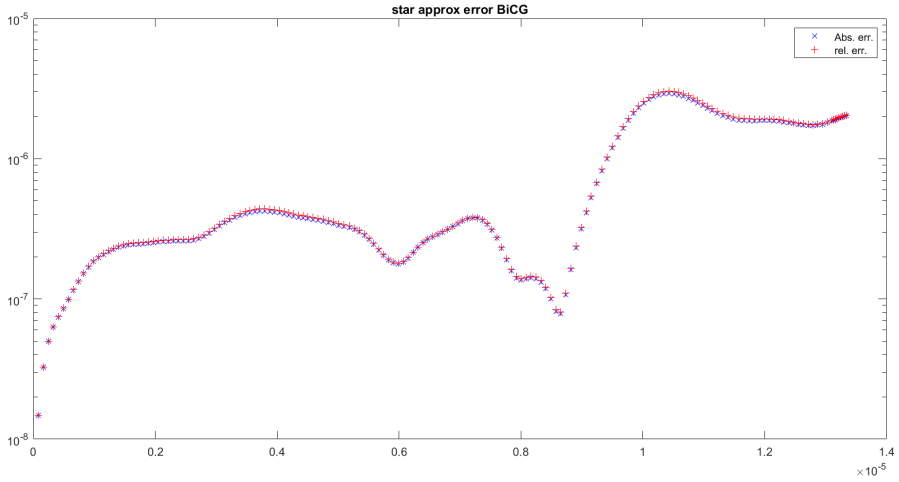
Figure 4.7: Comparison between the reference solution and the solutions obtained with the low-rank methods ($N_s = 14$ and $k = 2$).

Krylov methods (GMRES, BiCG and BiCGSTAB) can no longer be used, since there is not enough memory to store the vectorized input required to solve the linear system. However, we can use our low-rank versions of the algorithms, which produce the results reported in Table 4.3.

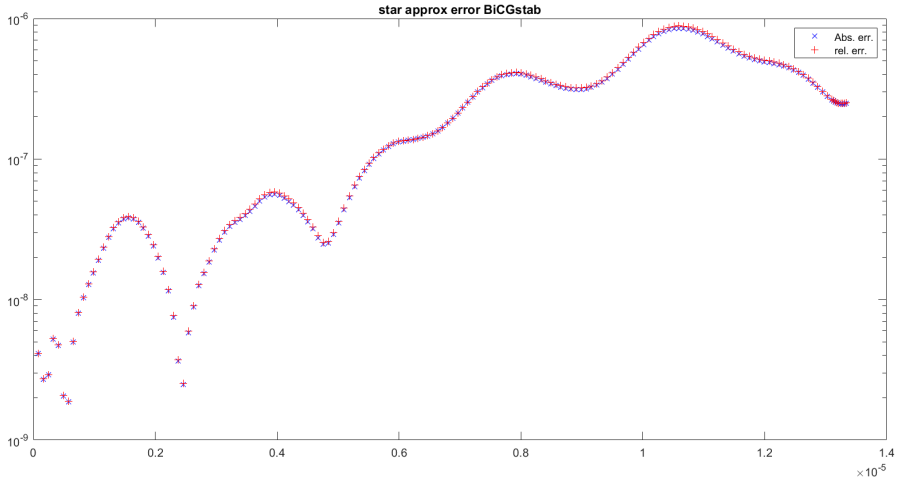
Table 4.3: Comparison of low-rank iterative methods for $N_s = 18$ and $N_s = 19$ with varying k and $r_{\max}$.

Method	N_s	k	$r_{\max}$	Time (s)	Iterations	Residual
BiCGSTAB low-rank	18	1	32	110.37	4	9.0868e-08
BiCG low-rank	18	1	32	136.62	7	8.3446e-07
BiCGSTAB low-rank	18	2	58	217.34	4	7.5252e-07
BiCG low-rank	18	2	58	351.54	8	6.2062e-07
BiCGSTAB low-rank	19	1	35	298.92	4	8.8071e-08
BiCG low-rank	19	1	35	384.24	7	3.4009e-07
BiCGSTAB low-rank	19	2	65	1132.1	4	7.5433e-07
BiCG low-rank	19	2	65	2062.8	8	8.872e-07

As we can observe, the computational times are now larger; however, we can still compute the solution while keeping the error with respect to the reference solution below 10^{-5} (e.g., Figure 4.8). This is possible because the maximum rank remains small, as its growth is marginal compared with that of the problem dimensions.



(a) BiCG low-rank.



(b) BiCGSTAB low-rank.

Figure 4.8: Errors of the low-rank methods compared to the reference solution obtained with `ode89` using a tight tolerance ($N_s = 19$ and $k = 1$).

Chapter 5

Conclusions

The results presented in this thesis show that the strategy based on the $\star$-algebra framework, together with the corresponding algebraic reformulation of the problem, leads to a very promising approach for solving the multiterm matrix equation arising from non-autonomous ODE systems. In particular, the use of low-rank Krylov methods makes it possible to significantly reduce computational costs and runtimes compared with the corresponding full variants, while keeping the error small and achieving competitive numerical performance. A key aspect is the adoption of low-rank compression and truncation strategies; indeed, using the numerical low-rank structure of the solution, the rank growth throughout the iterations can be controlled, thus reducing memory requirements. This allows the computation of solutions even for larger values of the discretization parameters and for higher-dimensional instances (e.g., increasing the number of spins) without running into prohibitive storage limitations, which typically arise when standard full approaches are used. The main objective of this thesis was to identify efficient methods for the solution of multiterm matrix equations taking into account the low-rank structure, obtaining algorithms that are competitive with classical methods while providing a concrete gain in terms of memory and computational cost. The reported experiments indicate that this goal has been achieved: the combination of the $\star$-based reformulation with low-rank Krylov solvers provides a solid basis, both from a theoretical and from a numerical point of view.

A more ambitious goal is to show that this strategy is not only promising, but can perform better than classical solvers for non-autonomous ODEs commonly used in practice (e.g., pre-implemented Runge-Kutta methods such as `ode45`, `ode89` in MATLAB). In this direction, the work carried out here represents a natural starting point. There is clear room for improvement both on the algorithmic side, for instance, by incorporating randomized techniques (such as subspace embedding) and on the discretization side, by investigating alternative bases to the one adopted here (e.g., Fourier-type bases). In sum-

mary, this thesis shows that consistently using the low-rank structure within the $\star$-based framework can lead to methods that scale better for large problems. Moreover, it points to a research direction that is currently very active, with several possible extensions and the long-term goal of developing solvers for non-autonomous ODEs that are competitive with, and possibly better than, the standard methods commonly used today.

Bibliography

[1] Stefano Pozza and Niel Van Buggenhout. A new matrix equation expression for the solution of non-autonomous linear systems of ODEs. *Proceedings in Applied Mathematics and Mechanics*, 22(1):e202200117, 2023.

[2] Stefano Pozza and Niel Van Buggenhout. A new Legendre polynomial-based approach for non-autonomous linear ODEs. *Electronic Transactions on Numerical Analysis*, 60:292–326, 2024.

[3] Pierre-Louis Giscard and Stefano Pozza. Lanczos-like algorithm for the time-ordered exponential: The $*$-inverse problem. *Applications of Mathematics*, 65(6):807–827, 2020.

[4] Stefano Pozza. Decay phenomenon and quadratures in matrix function approximation. Habilitation thesis, Department of Numerical Mathematics, Charles University, 2024.

[5] Stefano Pozza. A new closed-form expression for the solution of ODEs in a ring of distributions and its connection with the matrix algebra. *Linear and Multilinear Algebra*, 73(9):2025–2035, 2025.

[6] Stefano Pozza and Niel Van Buggenhout. The $\star$-product approach for linear ODEs: A numerical study of the scalar case. In *Programs and Algorithms of Numerical Mathematics*, pages 187–198, Prague, 2023. Institute of Mathematics of the Czech Academy of Sciences.

[7] Pulin Kumar Bhattacharyya. *Distributions: Generalized Functions with Applications in Sobolev Spaces*. De Gruyter, Berlin and Boston, 2012.

[8] Manon Ryckebusch, Abderrahman Bouhamidi, and Pierre-Louis Giscard. A Fréchet Lie group on distributions. *Journal of Mathematical Analysis and Applications*, 546(1):129195, 2025.

[9] Nikolai N. Lebedev. *Special Functions and Their Applications*. Dover Publications, New York, 1972.

[10] Valeria Simoncini. Computational methods for linear matrix equations. *SIAM Review*, 58(3):377–441, 2016.

[11] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2nd edition, 2003.

[12] Peter Lancaster. Explicit solutions of linear matrix equations. *SIAM Review*, 12(4):544–566, 1970.

[13] Mihail Konstantinov, D. Wei Gu, Volker Mehrmann, and Petko Petkov. *Perturbation Theory for Matrix Equations*, volume 9 of *Studies in Computational Mathematics*. Elsevier, Amsterdam, 2003.

[14] Lorenzo Lazzarino. Numerical approximation of the time-ordered exponential for spin dynamic simulation. Master's thesis, Charles University, Faculty of Mathematics and Physics, Prague, 2023.

[15] Alia Alia, Swapna Ganapathy, and Huub J. M. de Groot. Magic angle spinning (MAS) NMR: a new tool to study the spatial and electronic structure of photosynthetic complexes. *Photosynthesis Research*, 102(2-3):415–425, 2009.

[16] Bernd Reif, Sharon E. Ashbrook, Lyndon Emsley, and Mei Hong. Solid-state NMR spectroscopy. *Nature Reviews Methods Primers*, 1(1):2, 2021.

[17] Andreas Detken, Matthias Ernst, and Beat H. Meier. Towards biomolecular structure determination by high-resolution solid-state NMR: Assignment of solid peptides. *Chimia*, 55:844–851, 2001.

[18] Mads Bak, Jimmy T. Rasmussen, and Niels Chr. Nielsen. SIMPSON: A general simulation program for solid-state NMR spectroscopy. *Journal of Magnetic Resonance*, 147(2):296–330, 2000.