
Exponential Integrators for Semilinear Problems

Borislav V. Minchev

Thesis submitted for the degree of
Philosophiae Doctor (PhD) at



Department of Informatics
University of Bergen
August 2004

EXPONENTIAL INTEGRATORS
FOR SEMILINEAR PROBLEMS

By
Borislav V. Minchev

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
PHILOSOPHIAE DOCTOR
AT
UNIVERSITY OF BERGEN
THORMHLENSGATE 55, 5020 BERGEN
AUGUST 2004

© Copyright by Borislav V. Minchev, 2004

UNIVERSITY OF BERGEN

Date: **August 2004**

Author: **Borislav V. Minchev**

Title: **Exponential Integrators for Semilinear Problems**

Department: **Informatics**

Degree: **Ph.D.** Convocation: **October** Year: **2004**

Permission is herewith granted to University of Bergen to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Author

THE AUTHOR RESERVES OTHER PUBLICATION RIGHTS, AND NEITHER THE THESIS NOR EXTENSIVE EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT THE AUTHOR'S WRITTEN PERMISSION.

THE AUTHOR ATTESTS THAT PERMISSION HAS BEEN OBTAINED FOR THE USE OF ANY COPYRIGHTED MATERIAL APPEARING IN THIS THESIS (OTHER THAN BRIEF EXCERPTS REQUIRING ONLY PROPER ACKNOWLEDGEMENT IN SCHOLARLY WRITING) AND THAT ALL SUCH USE IS CLEARLY ACKNOWLEDGED.

To My Family.

Table of Contents

Table of Contents	v
List of Tables	vii
List of Figures	viii
Abstract	ix
Acknowledgements	x
Introduction and motivation	1
1 Exponential Integrators	3
1.1 Exponential linear multistep methods	3
1.1.1 Integrating Factor methods	4
1.1.2 Exponential Time Differencing methods	5
1.2 Exponential Runge–Kutta methods	8
1.2.1 Formulation of the methods	9
1.2.2 Nonstiff order conditions	11
1.2.3 IF RK and ETD RK as special cases	21
1.2.4 More examples of ETD RK methods	26
1.2.5 Exponential RK methods for parabolic PDEs	28
1.3 General linear methods and exponential integrators	36
1.3.1 Formulation of the methods	36
1.3.2 Construction of practical GLMs	40
1.3.3 Exponential general linear methods	42
2 Exponential Integrators and Lie Group Methods	44
2.1 Background theory	44
2.2 Lie group integrators	48

2.3	The choice of action	52
2.4	Lie group integrators for semilinear problems	55
2.4.1	Crouch–Grossman based methods	56
2.4.2	RKMK based methods	56
2.4.3	Methods based on commutator free Lie group methods	64
3	Implementation Issues and Numerical Experiments	67
3.1	Decomposition methods	67
3.1.1	Block Schur–Parlett algorithm	68
3.1.2	Tridiagonal reduction	70
3.2	Krylov subspace approximations	73
3.3	Cauchy integral approach	75
3.4	Numerical experiments	78
	Conclusions	83
	Summary	83
	Contributions	83
	Future work	84
	Bibliography	86
	Article 1:	
	Lie group integrators with nonautonomous frozen vector fields.	91
	Submitted to <i>J. Appl. Num. Math.</i>	91
	Article 2:	
	Some algorithms for solving special tridiagonal block Teoplitz linear systems.	107
	<i>J. Comp. and Appl. Math., vol.156(1) (2003), 179-200.</i>	107
	Article 3:	
	A method for solving Hermitian pentadiagonal block circulant cystems of linear equations.	133
	With I. Ivanov. <i>Springer Lect. Notes Comput. Sci.</i> , 2907 (2004), 481-488.	133

List of Tables

1.1	Coefficients of ETD Adams–Bashforth methods.	8
1.2	Coefficients of ETD Adams–Moulton methods.	8
1.3	General formulation of an IF Runge–Kutta method.	9
1.4	The number of rooted trees in $2T^*$ for all orders up to ten.	12
1.5	Standard decomposition for all trees up to order 3 and their corresponding differential equations.	16
1.6	Relation between elementary differentials and elementary weights.	21
1.7	Coloured trees as linear combinations of black trees for the IF RK methods.	25
1.8	A fourth order ETD Runge–Kutta method.	26
1.9	Third order method of Cox–Matthews.	27
1.10	Fourth order method of Cox–Matthews.	27
1.11	Fourth order method of Krogstad.	27
1.12	Third order ETD2RK3 method.	28
1.13	Third order ETD2CF3 method.	28
1.14	Stiff order conditions for explicit exponential Runge–Kutta methods for $\alpha = 0$	34
1.15	Fourth order method of Hochbruck–Ostermann with five stages.	35
2.1	Actions and homeomorphisms on the manifold.	49
2.2	Third order Crouch–Grossman method.	56
2.3	Fourth order RKMK method with exact Exp.	58
2.4	Fourth order GIF1/RK method	59
2.5	Fourth order GIF2/RK method	61
2.6	Fourth order CF method.	65

List of Figures

1.1	Domain of validity for condition (1.2.18) (shaded).	30
1.2	The contour Γ from Definition 1.2.7.	31
3.1	Step size versus relative error for ETD RK (left) and GIF/RK (right) methods for the Kuramoto-Sivashinsky equation	80
3.2	Step size versus relative error for ETD RK (left) and GIF/RK (right) methods for the Allen-Cahn equation	81
3.3	Step size versus relative error for ETD RK (left) and GIF/RK (right) methods for the Korteweg de Vries equation	81

Abstract

In the present work, exponential integrators for time integration of semilinear problems are studied. These integrators, as their name suggests, use the exponential and often functions which are closely related to the exponential function inside the numerical method. Three main classes of exponential integrators, exponential linear multistep (multivalue), exponential Runge–Kutta (multistage) and exponential general linear methods, are discussed. A general formulation of exponential integrators, which includes, as special cases, all known methods, is proposed. The nonstiff order theory for exponential multistage methods, along with a non-recursive rule for generating each order condition from its corresponding rooted tree, is derived.

The natural connection between exponential integrators and Lie group methods with affine algebra action is also studied. A new approach for deriving Generalized Integrating Factor Runge–Kutta methods, which allows the nonlinear part of the problem to be approximated by trigonometric polynomials, is proposed. The crucial role of the algebra action in the overall performance of any Lie group method is discussed. A new algebra action arising from the solutions of differential equations with nonautonomous frozen vector fields is proposed. The corresponding exponential integrators based on this action are derived.

Different methods for numerically stable computation of the most commonly used functions which appear in the format of an exponential integrator are considered. A generalization of the method based on the tridiagonal reduction is proposed. The new approach allows to compute all functions included in the format of an exponential integrator in the case when the arguments are symmetric (Hermitian) matrices. Some practical issues regarding variable step size implementations as well as the main advantages and disadvantages of the considered numerical techniques are discussed. New effective methods and their modifications for solving special three and five diagonal block systems of linear equations, based on a modified LU factorization are proposed. For illustrating the theoretical results, several numerical experiments are presented.

Acknowledgements

I would like to thank Prof. Hans Munthe-Kaas, my supervisor, for his constant support during this research. I am also thankful to Ass. Prof. Ivan Ivanov for his guidance through the early years of my work.

Dr. William Wright has expressed his interest to my work at a crucial stage. The time spend together discussing many of the ideas presented in this thesis, have provided me with better understanding and with the precious second point of view. I am also grateful for his valuable comments and suggestions made while reading the manuscript of this thesis.

During my participation in the *Special Year in Geometric Integration*, Oslo, 2002- 2003, I had the opportunity to meet many outstanding people from the international GI community. I am also grateful to all the members of the Bergen–Cambridge–Trondheim research groups. They all have played an important role in my professional live.

I would like to thank the Norwegian Research Council, Meltzer Høyskolefond and the Department of Informatics, University of Bergen for their financial support during the last three years. Without it my research stays at *Center for Advanced Studies, Norwegian Academy of Sciences* and *Section de mathématiques, Université de Genève* will not be possible.

Finally, but actually on the first place, I would like to thank to my family, my parents, and specially to my wife Stela, for their *love* and for always believing in me.

Bergen, Norway
August, 2004

Borislav V. Minchev

Introduction and motivation

Many practical problems arising in the real life applications can be modeled by time dependent partial differential equations (PDEs). In most cases it is extremely difficult, if not impossible, to obtain exact solutions of these problems, therefore numerical methods are used to provide accurate approximations. An often used technique in the construction of numerical integrators for PDEs, is first to semidiscretize the equation in space. There exists many efficient and accurate methods for spatial discretization. If the original vector field can be represented like a sum of two part, linear and nonlinear, the spatial discretization leads to the following semilinear initial value problem

$$u' = Lu + N(u, t), \quad u(t_0) = u_0,$$

where $u : \mathbb{R} \rightarrow \mathbb{R}^d$, $L \in \mathbb{R}^{d \times d}$, $N : \mathbb{R}^d \rightarrow \mathbb{R}^d$ and d is a discretization parameter equal to the number of spatial grid points. Many interesting equations can be brought exactly in this form. Examples are *Allen-Cahn*, *Burgers*, *Cahn-Hilliard*, *Kuramoto-Sivashinsky*, *Navier-Stokes*, *Swift-Hohenberg*, *nonlinear Schrödinger*, *convection-diffusion* equations. Typically, after the spatial discretization, the linear part of the problem will be stiff and the nonlinear part will be nonstiff. Whenever equations model several processes with vastly different rates of evolution, that is the processes proceed at significantly different time scales, one should also expect stiffness. Implicit methods, which overcome the problem of stiffness are regarded as too expensive for implementation. On other hand explicit methods require such small time steps that they become inefficient. Therefore, it is a very important and challenging task to construct effective numerical integrators for solving stiff problems.

The first exponential integrators were introduced in 1960 as an alternative approach to overcome the phenomenon of stiffness. The main idea behind these methods is to integrate exactly the linear part of the problem, which is primarily responsible for the stiffness, and then to use an appropriate approximation of the nonlinear part. Thus the exponential function, and often functions which are closely related to the exponential function, appear in the format of the methods. This was exactly the reason why, until recently, these methods have not been regarded as practical. The latest achievements in the field of computing approximations to the product of a matrix exponential function and some related functions

with a vector, have provided a new interest in the construction and implementation of exponential integrators.

The aim of this thesis is to study different classes of exponential integrators for time integration of semilinear problems and to propose an unified framework in which this methods can be analyzed.

The thesis is organized as follows: In Chapter 1 we present the philosophy behind exponential integrators applied to semilinear problems and discuss the three main classes of exponential integrators: exponential linear multistep (multivalued), exponential Runge–Kutta (multistage) and exponential general linear methods. Next, in Chapter 2 we study the connection between exponential integrators and Lie group methods based on the affine algebra action. Finally, in Chapter 3 we discuss different techniques for a fast and numerically stable computation of the most commonly used functions which enter in the format of the methods, and present some numerical experiments.

Chapter 1

Exponential Integrators

In the first chapter of this thesis we present the philosophy behind exponential integrators applied to semilinear problems and discuss how the three main classes of numerical integrators, linear multistep (multivalued), Runge–Kutta (multistage) and general linear methods, are extended to the exponential setting. In the presentation we assume the reader has basic knowledge of numerical methods for ordinary differential equations and functional analysis as can be found in the monographs [8, 24, 25, 70]. All numerical schemes presented in this chapter are in accordance with the notations introduced in Section 1.3.

The chapter is organized as follows: First, in Section 1.1 we consider exponential linear multistep methods based on Integrating Factor (IF) and Exponential Time Differencing (ETD) approach. Next, in Section 1.2 we present the theory of exponential Runge–Kutta methods for both nonstiff and stiff cases. Finally, in Section 1.3 we explain how the idea of general linear methods can also be extended to the exponential setting.

1.1 Exponential linear multistep methods

Our field of interest is the following semi-discretized semilinear initial value problem

$$u' = Lu + N(u, t), \quad u(t_0) = u_0, \quad (1.1.1)$$

where $u : \mathbb{R} \rightarrow \mathbb{R}^d$, $L \in \mathbb{R}^{d \times d}$, $N : \mathbb{R}^d \rightarrow \mathbb{R}^d$ and d is a discretization parameter equal to the number of spatial grid points. Several interesting problems can be brought to this form by discretizing in space. Examples are Allen–Cahn, Burgers, Cahn–Hilliard, Kuramoto–Sivashinsky, Navier–Stokes, Swift–Hohenberg, nonlinear Schrödinger equations. Typically the linear part of the problem will be stiff and the nonlinear part will be nonstiff. There exist two principal approaches in the derivation of exponential linear multistep methods for solving (1.1.1). These will be introduced in the next two subsections.

1.1.1 Integrating Factor methods

The idea behind Integrating Factor (IF) methods is an old one and goes back to the work of Lawson [37]. He proposes to ameliorate the effect of the stiff linear part of equation (1.1.1) by using change of variables (also known as Lawson transformation),

$$v(t) = e^{-tL}u(t).$$

The initial value problem (1.1.1) written in the new variable is

$$v'(t) = e^{-tL}N(e^{tL}v(t), t) = \mathbf{g}(v, t), \quad v(t_0) = v_0, \quad (1.1.2)$$

where $v_0 = e^{-t_0L}u_0$. The same result can be obtained by premultiplying the original problem (1.1.1) by the so called integrating factor, e^{-tL} , that is

$$\begin{aligned} e^{-tL}u' &= e^{-tL}Lu + e^{-tL}N(u, t), \\ (e^{-tL}u)' &= e^{-tL}N(u, t). \end{aligned} \quad (1.1.3)$$

To see way the formulation (1.1.2) is preferable to the original formulation, we calculate the Jacobian of the new problem. Since

$$\frac{\partial \mathbf{g}}{\partial v} = e^{-tL} \frac{\partial N}{\partial u} e^{tL},$$

and $e^{-tL} = (e^{tL})^{-1}$, it follows that the eigenvalues of $\partial \mathbf{g} / \partial v$ are those of $\partial N / \partial u$. Therefore we should expect that numerical methods applied to (1.1.2) will not suffer from the same stability restrictions like if they are used directly on the problem (1.1.1). The idea now is to apply any numerical integrator (in the case of Lawson, a Runge–Kutta method) on the transformed equation and then to transform back the result into the original variable. Using Euler method, we obtain the following numerical scheme

$$u_n = e^{hL}u_{n-1} + e^{hL}hN_{n-1},$$

where h represents the stepsize of the method and $N_{n-1} = N(u_{n-1}, t_{n-1})$. This is known as IF Euler method. The integrating factor implicit Euler method is

$$u_n = e^{hL}u_{n-1} + e^{hL}hN_n.$$

This approach can be easily extended to the class of linear multistep Adams methods. In general, k -step IF Adams methods are defined as

$$u_n = e^{hL}u_{n-1} + \sum_{i=0}^k \beta_i e^{ihL}hN_{n-i},$$

where β_i are the coefficients of the Adams method and $N_{n-i} = N(u_{n-i}, t_{n-i})$ for $i = 0, 1, 2, \dots, k$. Similarly IF methods based on backward differentiation formulae (BDF) methods are defined as

$$u_n = \sum_{i=1}^k \alpha_i e^{ihL}u_{n-i} + \beta_0 hN_n,$$

where β_0 and α_i are the coefficients of the underlying BDF method. As an example, the second order IF Adams–Bashforth method considered in [14] is

$$u_n = e^{hL}u_{n-1} + \frac{3}{2}e^{hL}hN_{n-1} - \frac{1}{2}e^{2hL}hN_{n-2}.$$

1.1.2 Exponential Time Differencing methods

According to our knowledge, the first paper on Exponential Time Differencing (ETD) methods is the paper of Certaine [13], where an order two and order three methods, reducing to second and third order Adams–Moulton methods, are proposed. Arbitrarily high order A-stable ETD methods, reducing to the Adams–Bashforth methods when $L = 0$, are derived in [53]. Since then, this methods has been rediscovered several times in different fields and under different names.

The main idea behind the ETD method is to construct, via the variation of constants formulae, integrators of multistep type. To derive the variation of constants formulae we integrate the equation (1.1.3) between t_{n-1} and $t_n = t_{n-1} + h$. We obtain

$$\begin{aligned} u(t_{n-1} + h) &= e^{hL}u_{n-1} + e^{t_n L} \int_{t_{n-1}}^{t_{n-1}+h} e^{-\tau L} N(u(\tau), \tau) d\tau \\ &= e^{hL}u_{n-1} + \int_0^h e^{(h-\tau)L} N(u(t_{n-1} + \tau), t_{n-1} + \tau) d\tau. \end{aligned} \quad (1.1.4)$$

Note that only exact calculations are used in the derivation of (1.1.4). Therefore (1.1.4) is the exact solution of (1.1.1) with initial condition $u(t_{n-1}) = u_{n-1}$. The approach now is to replace the nonlinear term N by a Newton interpolation polynomial and then solve the resulting integral exactly. The simplest case is when we approximate $N(u(t_{n-1} + \tau), t_{n-1} + \tau)$ by a constant N_{n-1} . This gives

$$\begin{aligned} u_n &= e^{hL}u_{n-1} + \int_0^h e^{(h-\tau)L} N_{n-1} d\tau \\ &= e^{hL}u_{n-1} + \phi^{[1]}(hL)hN_{n-1}, \end{aligned}$$

where $\phi^{[1]}(z)$ is

$$\phi^{[1]}(z) = \frac{e^z - 1}{z}.$$

The above method is known as the exponential time differencing Euler method as it reduces to the classical Euler method when $L = 0$. The exponential time differencing implicit Euler method is defined as

$$u_n = e^{hL}u_{n-1} + \phi^{[1]}(hL)hN_n.$$

In general, we obtain the following explicit form of ETD Adams–Bashforth methods by approximating the nonlinear term in (1.1.4) by higher order polynomial using information

from the past (see [14, 53])

$$u_n = e^{hL}u_{n-1} + h \sum_{i=0}^{k-1} g_i(hL) \nabla^i N_{n-1}, \quad (1.1.5)$$

where $\nabla^i N_{n-1}$ denotes the i -th backward difference,

$$\nabla^0 N_{n-1} = N_{n-1}, \quad \nabla^{i+1} N_{n-1} = \nabla^i N_{n-1} - \nabla^i N_{n-2}, \quad \text{for } i = 0, 1, 2, \dots,$$

and the function $g_i(z)$ is given by the recurrence relations

$$\begin{aligned} z g_0(z) &= e^z - 1, \\ z g_{i+1}(z) + 1 &= g_i(z) + \frac{1}{2} g_{i-1}(z) + \frac{1}{3} g_{i-2}(z) + \dots + \frac{1}{i+1} g_0(z). \end{aligned}$$

Similarly, it is also possible to construct ETD Adams–Moulton methods. Next we give an other derivation of both explicit and implicit exponential Adams methods, as proposed in [4]. It is based on the following particular representation of the exact solution.

Lemma 1.1.1. *The exact solution of the initial value problem*

$$u' = Lu + N(u, t), \quad u(t_{n-1}) = u_{n-1},$$

can be expressed in the form

$$u(t_{n-1} + h) = e^{hL}u_{n-1} + \sum_{i=0}^{\infty} h^{i+1} \phi^{[i+1]}(hL) N_{n-1}^{(i)},$$

where

$$N_{n-1}^{(i)} = \left. \frac{d^i}{dt^i} \right|_{t=t_{n-1}} N(u(t), t),$$

and $\phi^{[i]}(z)$ are recursively defined as

$$\phi^{[0]}(z) = e^z, \quad \phi^{[i+1]}(z) = \frac{\phi^{[i]}(z) - \frac{1}{i!}}{z}, \quad \text{for } i = 0, 1, 2, \dots \quad (1.1.6)$$

Proof. Expand $u(t_{n-1} + h)$ in Taylor series around the point t_{n-1} to obtain

$$u(t_{n-1} + h) = u_{n-1} + \sum_{k=1}^{\infty} \frac{h^k}{k!} u_{n-1}^{(k)}, \quad (1.1.7)$$

where

$$u_{n-1}^{(k)} = \left. \frac{d^k}{dt^k} \right|_{t=t_{n-1}} u(t).$$

Because of the given differential equation, we have the following relation

$$u_{n-1}^{(k)} = L^k u_{n-1} + \sum_{i=0}^{k-1} L^{k-1-i} N_{n-1}^{(i)}, \quad \text{for } k = 1, 2, \dots$$

Substituting $u_{n-1}^{(k)}$ into (1.1.7), we obtain

$$\begin{aligned} u(t_{n-1} + h) &= \sum_{k=0}^{\infty} \frac{h^k}{k!} L^k u_{n-1} + \sum_{k=1}^{\infty} \sum_{i=0}^{k-1} \frac{h^k}{k!} L^{k-1-i} N_{n-1}^{(i)} \\ &= e^{hL} u_{n-1} + \sum_{k=0}^{\infty} \sum_{i=0}^k \frac{h^{k+1}}{(k+1)!} L^{k-i} N_{n-1}^{(i)}. \end{aligned}$$

Changing the order of the summation in the second term we obtain

$$\begin{aligned} \sum_{k=0}^{\infty} \sum_{i=0}^k \frac{h^{k+1}}{(k+1)!} L^{k-i} N_{n-1}^{(i)} &= \sum_{i=0}^{\infty} \left(\sum_{k=i}^{\infty} \frac{h^{k+1}}{(k+1)!} L^{k-i} \right) N_{n-1}^{(i)} \\ &= \sum_{i=0}^{\infty} h^{i+1} \left((hL)^{-(i+1)} \sum_{k=i+1}^{\infty} \frac{(hL)^k}{k!} \right) N_{n-1}^{(i)} \\ &= \sum_{i=0}^{\infty} h^{i+1} \phi^{[i+1]}(hL) N_{n-1}^{(i)}. \end{aligned} \tag{1.1.8}$$

□

Lemma 1.1.1 gives an alternative formulation of the exact solution and motivates the following ansatz for its numerical approximation

$$u_n = e^{hL} u_{n-1} + h \sum_{l=0}^k \beta_l N_{n-l},$$

where $\beta_0, \beta_1, \dots, \beta_k$ are coefficients to be computed in order to obtain the desired order. This is done by expanding the nonlinear terms in Taylor series around t_{n-1} . Since

$$h \sum_{l=0}^k \beta_l N_{n-l} = \sum_{i=0}^{\infty} \frac{h^{i+1}}{i!} \left(\sum_{l=0}^k \beta_l (1-l)^i \right) N_{n-1}^{(i)}, \tag{1.1.9}$$

the order conditions for the ETD Adams methods are obtained by comparing (1.1.8) and (1.1.9), resulting in

$$\frac{1}{i!} \sum_{l=j}^k \beta_l (1-l)^i = \phi^{[i+1]}(hL), \quad \text{for } i = 0, 1, 2, \dots, k-j.$$

For $j = 0$ we obtain the ETD Adams–Moulton methods, whereas for $j = 1$ we recover the ETD Adams–Bashforth methods (1.1.5). In Tables 1.1 and 1.2 we give the the first few coefficients for the ETD Adams–Bashforth and ETD Adams–Moulton methods respectively.

Finally, we mention that it is not possible to construct ETD BDF methods (see [46]).

Next, we explain how the IF and ETD approach can be further extended to the class of exponential multistage methods.

k	β_1	β_2	β_3	β_4
1	$\phi^{[1]}$	0	0	0
2	$\phi^{[1]} + \phi^{[2]}$	$-\phi^{[2]}$	0	0
3	$\phi^{[1]} + \frac{3}{2}\phi^{[2]} + \phi^{[3]}$	$-2(\phi^{[2]} + \phi^{[3]})$	$\frac{1}{2}\phi^{[2]} + \phi^{[3]}$	0
4	$\phi^{[1]} + \frac{11}{6}\phi^{[2]} + 2\phi^{[3]} + \phi^{[4]}$	$-3\phi^{[2]} - 5\phi^{[3]} - 3\phi^{[4]}$	$\frac{3}{2}\phi^{[2]} + 4\phi^{[3]} + 3\phi^{[4]}$	X

where $X = -\frac{1}{3}\phi^{[2]} - \phi^{[3]} - \phi^{[4]}$

Table 1.1: Coefficients of ETD Adams–Bashforth methods.

k	β_0	β_1	β_2	β_3
0	$\phi^{[1]}$	0	0	0
1	$\phi^{[2]}$	$\phi^{[1]} - \phi^{[2]}$	0	0
2	$\frac{1}{2}\phi^{[2]} + \phi^{[3]}$	$\phi^{[1]} - 2\phi^{[3]}$	$-\frac{1}{2}\phi^{[2]} + \phi^{[3]}$	0
3	$\frac{1}{3}\phi^{[2]} + \phi^{[3]} + \phi^{[4]}$	$\phi^{[1]} + \frac{1}{2}\phi^{[2]} - 2\phi^{[3]} - 3\phi^{[4]}$	$-\phi^{[2]} + \phi^{[3]} + 3\phi^{[4]}$	$\frac{1}{6}\phi^{[2]} - \phi^{[4]}$

Table 1.2: Coefficients of ETD Adams–Moulton methods.

1.2 Exponential Runge–Kutta methods

As already mentioned in the previous section, the first exponential Runge–Kutta method was constructed by Lawson in [37]. In fact, once we have transformed the original problem (1.1.1) into the form (1.1.2), it is easy to derive an Integrating Factor Runge–Kutta (IF RK) method. In analogy to the multistep case, the idea now is to apply an arbitrary s -stage Runge–Kutta method to the transformed equation (1.1.2) and then to transform the result back into the original variable. If $\mathcal{A} = (\alpha_{ij})$, $b = (\beta_i)$ and $c = (c_i)$ are the coefficients of the underlying multistage method then the general formulation of an s -stage IF Runge–Kutta method, written in accordance with the notations introduced in Section 1.3, is given in Table 1.3. This approach requires that the c vector has nondecreasing coefficients, which is usually the case for the most commonly used numerical schemes.

Probably the first successful attempt to construct Exponential Time Differencing Runge–Kutta (ETD RK) methods is due to the work of Friedli [20]. Later Strehmel and Weiner [61] constructed the adaptive Runge–Kutta methods and studied their B -convergence [62]. This later class of methods is an extension of the W-method of Streihaug and Wolfbrandt [60], but is also very closely related to the methods of Friedli. The only difference between these methods and the ETD Runge–Kutta methods, presented in Section 1.2.3, is that the later methods compute the ETD $\phi^{[i]}$ functions (1.1.6) exactly, while the W-methods and the adaptive Runge–Kutta methods generally use Padé approximations to these functions.

0	0	...	0	0	$e^{c_1 hL}$
$\alpha_{21}e^{c_2 hL}$	0	...	0	0	$e^{c_2 hL}$
$\alpha_{31}e^{c_3 hL}$	$\alpha_{32}e^{(c_3-c_2)hL}$...	0	0	$e^{c_3 hL}$
\vdots	\vdots		\vdots	\vdots	
$\alpha_{s1}e^{c_s hL}$	$\alpha_{s2}e^{(c_s-c_2)hL}$...	$\alpha_{ss-1}e^{(c_s-c_{s-1})hL}$	0	$e^{c_s hL}$
$\beta_1 e^{hL}$	$\beta_2 e^{(1-c_2)hL}$...	$\beta_{s-1}e^{(1-c_{s-1})hL}$	$\beta_s e^{(1-c_s)hL}$	e^{hL}

Table 1.3: General formulation of an IF Runge–Kutta method.

The first modern exponential Runge–Kutta methods proposed in the literature are the methods of Hochbruck and Lubich [27]. In that paper, the authors constructed Rosenbrock-like exponential integrators which use the first ETD function $\phi^{[1]}$. Another class of methods known as Runge–Kutta Munthe-Kaas (RKMK) methods was first proposed in [48]. These transform the original differential equation to a new differential equation which evolves on a Lie algebra, and then take advantage of the fact that the Lie algebra is a linear space. In [50], in order to construct integrators for the differential equation (1.1.1), Munthe-Kaas proposed to combine the use of RKMK methods with the *affine action* (see Section 2.3). The affine action was also used in [12] to construct integrators based on the Commutator Free (CF) Lie group methods. As we shall see in Chapter 2, all of these methods can be regarded as “pure” exponential integrators which use other than the traditional IF and ETD $\phi^{[i]}$ functions (see Section 1.2.3). Motivated by this observation, in the next section we give a slightly more general formulation of the exponential Runge–Kutta methods proposed by Friedli, which allows us to include as special cases all multistage methods mentioned above.

1.2.1 Formulation of the methods

For simplicity of the presentation, we choose to represent the initial value problem (1.1.1) in autonomous form

$$u' = Lu + N(u(t)), \quad u(t_0) = u_0. \quad (1.2.1)$$

The aim is to construct a class of exponential integrators which overcome the stiffness in the differential equation (1.2.1) by using a set of precomputed functions along with evaluations of the nonlinear part of the differential equation. The precomputed functions, denoted by $\phi^{[l]}$, can however be a large overhead depending on the dimensionality of the problem, the spatial discretization of the differential equation and the structure of the matrix L . Thus, such methods are likely to be competitive, when the $\phi^{[l]}$ functions can be evaluated efficiently.

Let for $l \in \mathbb{N}$ and for all $\lambda \in \mathbb{R}$ the operators $\phi^{[l]}(\lambda) : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}^{d \times d}$ be analytic, map the spectrum of L into a bounded region in \mathbb{C} and can be computed exactly or up to arbitrary

high order cheaply, then

$$\phi^{[l]}(\lambda)(hL) = \sum_{j \geq 0} \phi_j^{[l]}(\lambda)(hL)^j. \quad (1.2.2)$$

The most common choices of $\phi^{[l]}$ are those associated with the IF RK and ETD RK methods. The exact representation of $\phi^{[l]}$ for these special cases can be found in Section 1.2.3. However, we mention that other choices are also possible (see Chapter 2). The exact structure of $\phi^{[l]}$, which leads to methods is still unclear. If h represents the stepsize and U_i denotes the internal stage approximation of the exact solution for $i = 1, 2, \dots, s$ then the computations performed in step number n are related by the equations

$$\begin{aligned} U_i &= \sum_{j=1}^s \sum_{l=1}^m \alpha_{ij}^{[l]} \phi^{[l]}(c_i)(hL) hN(U_j) + e^{c_i hL} u_{n-1}, \\ u_n &= \sum_{j=1}^s \sum_{l=1}^m \beta_j^{[l]} \phi^{[l]}(1)(hL) hN(U_j) + e^{hL} u_{n-1}, \end{aligned} \quad (1.2.3)$$

where m puts a limit on the number of $\phi^{[l]}$ functions which can be computed. For the method to be explicit $\alpha_{ij}^{[l]} = 0$ for all $j \geq i$. To keep the number of computations of the $\phi^{[l]}$ functions down we choose $\alpha_{ij}^{[l]} = 0$ for all $l \geq i$, this ensures that $m \leq s$, and the number of $\phi^{[l]}$ functions used to compute the internal stage approximations increases as the number of stages increase. There is a certain balance between the initial computational cost of evaluating the $\phi^{[l]}$ functions and the benefit they provide from their availability. The best combination is likely to be problem dependent. However, we believe that initially this is a reasonable choice for locating competitive methods. It is also recognized that the term

$$\sum_{l=1}^m \alpha_{ij}^{[l]} \phi^{[l]}(\lambda)(hL),$$

could be simplified due to the general nature of the $\phi^{[l]}$ functions. However, this simplification is not made, since it may be more difficult to evaluate the resulting $\phi^{[l]}$ functions. The fourth order commutator free Lie group method (see Subsection 2.4.3) is an example.

It is also possible to represent the computations of the method in a matrix notation. Introduce the set of constant strictly lower triangular $s \times s$ matrices $\alpha^{[l]}$, with coefficients $\alpha_{ij}^{[l]}$, the first l rows of $\alpha^{[l]}$ are chosen to be zero, this is equivalent to the requirements placed on $\alpha_{ij}^{[l]}$ already. The strictly block lower triangular $ds \times ds$ matrix A which describes the internal stage computations, is

$$A = \sum_{l=1}^m \Phi^{[l]}(c)(hL) (\alpha^{[l]} \otimes I_d),$$

where \otimes is the Kronecker product, I_d is the $d \times d$ identity matrix and the block diagonal $ds \times ds$ matrix $\Phi^{[l]}$, is given by

$$\Phi^{[l]}(c)(hL) = \text{diag} \left(\phi^{[l]}(c_1)(hL), \dots, \phi^{[l]}(c_s)(hL) \right).$$

The $d \times ds$ matrix b^T is defined in a similar fashion to A , except there are now no restrictions on the vectors $\beta^{[l]^T}$ as there was on the matrices $\alpha^{[l]}$, therefore

$$b^T = \sum_{i=1}^m \phi^{[l]}(\lambda)(hL)(\beta^{[l]^T} \otimes I_d).$$

We are now in a position to formulate the method. For ease of notation, we introduce the vectors U , $N(U)$ and e^{chL} , as

$$U = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_s \end{bmatrix}, \quad N(U) = \begin{bmatrix} N(U_1) \\ N(U_2) \\ \vdots \\ N(U_s) \end{bmatrix}, \quad e^{chL} = \begin{bmatrix} e^{c_1 hL} \\ e^{c_2 hL} \\ \vdots \\ e^{c_s hL} \end{bmatrix}.$$

If h represents the stepsize then the computation performed in step number n are related by the equations

$$\begin{aligned} U &= AhN(U) + e^{chL}u_{n-1}, \\ u_n &= b^T hN(U) + e^{hL}u_{n-1}. \end{aligned} \tag{1.2.4}$$

Alternatively the computations performed in step number n can be represented in a more Runge–Kutta type formulation as follows

$$\begin{array}{c|c|c|c|c} c & \alpha^{[1]} & \alpha^{[2]} & \dots & \alpha^{[m]} \\ \hline & \beta^{[1]^T} & \beta^{[2]^T} & \dots & \beta^{[m]^T} \end{array},$$

where each element in row number i of $\alpha^{[l]}$ is multiplied by $\phi^{[l]}(c_i)(hL)$ and where each element in $\beta^{[l]^T}$ is multiplied by $\phi^{[l]}(1)(hL)$. The resulting matrices are then added in a component by component sense to give the matrices A and b^T .

We next derive the nonstiff order theory for the exponential Runge–Kutta methods (1.2.4). It turns out that the order conditions for this methods are the same as the order conditions for the adaptive Runge–Kutta methods developed in [5]. An advantage of our approach is that it provides a non-recursive rule for generating each order condition from its corresponding rooted tree. In addition it is easy to generalize the same approach in the case of exponential general linear methods.

1.2.2 Nonstiff order conditions

As for many numerical methods the tools used here to develop the nonstiff order conditions are the rooted trees and the B-series. For those not familiar with these concepts we suggest the monographs [8, 23] for a complete treatment. Let $2T^*$ denote the set of all bi-coloured

(black and white) rooted trees with the requirement that the valency of the white nodes is always one. This means that any white node will have at most one upward branch leaving that node, which leads to a node which can be coloured either white or black. The fact that the white node has valency one is because the first term on the right hand side of (1.2.1) is linear. All the bi-coloured trees in the set $2T^*$ we call $2T^*$ -trees. As is the case for many numerical methods the number of conditions necessary to obtain a certain order is exactly equivalent to the number of certain rooted trees up to the required order. It is therefore, necessary to have a formula for the number of trees of each order in the set $2T^*$. Let θ_n be the number of rooted trees in $2T^*$ with n vertices. Using Algorithm 1.2.1 from Subsection 1.2.2.1, it is possible count the number of rooted trees in $2T^*$ for each $n \in \mathbb{N}$. In Table 1.4 we list the number of rooted trees for each order up to ten.

n	1	2	3	4	5	6	7	8	9	10
θ_n	2	4	11	34	117	421	1589	6162	24507	99268
$\Theta = \sum_i^n \theta_n$	2	6	17	51	168	589	2178	8340	32847	132115

Table 1.4: The number of rooted trees in $2T^*$ for all orders up to ten.

We now define several operations on trees which are required for constructing the order conditions. Let τ represent an arbitrary rooted tree in $2T^*$. The order of the tree τ , denoted by $|\tau|$, is defined as the number of vertices in the tree τ . Each tree τ can be decomposed as $\tau = [\text{tp}(\tau); \tau_1, \dots, \tau_\ell]$, where $\text{tp}(\tau) = \{\circ, \bullet\}$ represents the colour of the root node and τ_1, \dots, τ_ℓ is the forest remaining after the root node has been removed. Let \emptyset represent the empty set which remains if the root of \bullet or \circ is removed. Clearly $\bullet = [\text{tp}(\bullet); \emptyset]$ and $\circ = [\text{tp}(\circ); \emptyset]$. The symmetry σ is defined in the same way as for Runge–Kutta methods, $\sigma(\circ) = \sigma(\bullet) = 1$ and

$$\sigma(\tau) = \sigma(\tau_1) \dots \sigma(\tau_\ell) \mu_1! \mu_2! \dots,$$

where the integers μ_1, μ_2, \dots count the equal trees among $\tau_1 \dots \tau_\ell \in 2T^*$. This implies that $\sigma(\bigvee^\circ) \neq \sigma(\bigvee^\bullet)$ because the colouring of the nodes are different. The density γ of the tree is the same as for the single coloured trees and it is defined as the product over all vertices of the order of the subtree rooted at that vertex. A one to one correspondence between the rooted bi-coloured trees and the elementary differentials means that the later can be defined recursively as a function of the former, as follows

$$F(\tau)(u) = \begin{cases} Lu, & \text{if } \tau = \circ, \\ N(u), & \text{if } \tau = \bullet, \\ LF(\tau_1)(u), & \text{if } \text{tp}(\tau) = \circ, \\ N^{(\ell)}(u)(F(\tau_1)(u), \dots, F(\tau_\ell)(u)), & \text{if } \text{tp}(\tau) = \bullet. \end{cases} \quad (1.2.5)$$

Before we continue with expressing the exact and the numerical solution in terms of the above defined elementary differentials we need to insure that they can be regarded as independent. In order to do this, it is convenient to introduce an order relation in the set $2T^*$. In the next subsection we present a constructive way how to generate and order the elements in $2T^*$.

1.2.2.1 Construction of $2T^*$ -trees

When using applications like Maple or Matlab, to implement any formulas involving trees, it is crucial to have a compact way of generating all the trees up to some order in a recursive manner. It is also important to be able to identify all the elements in $2T^*$ with the set of positive integers. This requires an order relation in $2T^*$. The idea is to represent each $2T^*$ -tree as a pair of two $2T^*$ -trees of lower order. Here we follow the approach based on the standard decomposition of N -trees proposed in [52]. First we need to define the product between two $2T^*$ -trees. Let $\tau = [\text{tp}(\tau); \tau_1, \dots, \tau_k]$ and π are two $2T^*$ -trees.

Definition 1.2.1. [52, Definition 5] The product $\tau \cdot \pi \in 2T^*$ is defined as follows:

$$[\text{tp}(\tau); \emptyset] \cdot \pi = [\text{tp}(\tau); \pi], \quad [\text{tp}(\tau); \tau_1, \dots, \tau_k] \cdot \pi = [\text{tp}(\tau); \tau_1, \dots, \tau_k, \pi].$$

Obviously $\text{tp}(\tau \cdot \pi) = \text{tp}(\tau)$ and $(\tau \cdot \pi_1) \cdot \pi_2 = (\tau \cdot \pi_2) \cdot \pi_1$ for $\tau, \pi_1, \pi_2 \in 2T^*$. Now we define an order relation in the set $2T^*$ in terms of the standard decomposition of trees.

Definition 1.2.2. [52, Definition 6] Given $\tau_1, \tau_2 \in 2T^*$, then $\tau_1 < \tau_2$ if one of the four conditions is fulfilled:

- $\tau_1 = \bullet$ and $\tau_2 = \circ$,
- $|\tau_1| < |\tau_2|$,
- $|\tau_1| = |\tau_2|$ and $\text{dec}_1(\tau_1) < \text{dec}_1(\tau_2)$,
- $|\tau_1| = |\tau_2|$, $\text{dec}_1(\tau_1) = \text{dec}_1(\tau_2)$ and $\text{dec}_2(\tau_1) < \text{dec}_2(\tau_2)$.

The standard decomposition is given by

Definition 1.2.3. [52, Definition 6] The standard decomposition $\text{dec}(\tau)$ of a $2T^*$ -tree τ of order ≥ 2 is the pair $\text{dec}(\tau) = (\text{dec}_1(\tau), \text{dec}_2(\tau)) \in 2T^* \times 2T^*$ such that

$$\tau = \text{dec}_1(\tau) \cdot \text{dec}_2(\tau)$$

and $\text{dec}_2(\tau)$ is maximal.

Clearly the following two relations hold

$$|\tau| = |\text{dec}_1(\tau)| + |\text{dec}_2(\tau)|, \quad \text{tp}(\tau) = \text{tp}(\text{dec}_1(\tau)).$$

The above definitions give us one to one correspondence between the rooted bi-coloured $2T^*$ -trees and the set of all positive integers, together with a convenient way of representing each tree in a unique way as a pointer of two positive integers. All $2T^*$ -trees have an additional restriction to the form of the first trees $\text{dec}_1(\tau)$ in the standard decomposition. The fact that the valency of the white nodes is always one implies that the only tree with white root in $\text{dec}_1(\tau)$ could be \circ . With this in mind together with the fact that it is sufficient to store for each p the first rooted tree $\text{first}(p)$ of order p we propose the following algorithm, based on the algorithm given in [52], for computing the standard decomposition $\text{dec}(\tau) = (\text{dec}_1(\tau), \text{dec}_2(\tau))$ for all $2T^*$ -trees of order up to certain order p_{\max} .

Algorithm 1.2.1.

```

 $\text{dec}_1(1) = 1; \text{dec}_1(2) = 2; \text{tp}(1) = 1; r(1) = 1;$ 
 $\text{dec}_2(1) = 0; \text{dec}_2(2) = 0; \text{tp}(2) = 2; r(2) = 1;$ 
 $\tau = 2;$ 
for  $p = 2, \dots, p_{\max}$  do
    % Index of the 1st tree of order  $p$ 
     $\text{first}(p) = \tau + 1;$ 
    % Construct all the  $2T^*$ -trees  $\tau$  ( $\text{dec}(\tau) = (\tau_1, \tau_2)$ ) of order  $p$ 
    for  $q = 1, \dots, p - 1$  do
        % All  $\tau_1$  of order  $q$ 
        for  $\tau_1 = \text{first}(q), \dots, \text{first}(q + 1) - 1$  do
             $l = \max(\text{first}(p - q), \text{dec}_2(\tau_1));$ 
            % All the  $\tau_2 \geq \text{dec}_2(\tau_1)$  of order  $p - q$ 
            for  $\tau_2 = l, \dots, \text{first}(p - q + 1) - 1$  do
                if  $[\text{tp}(\tau_1) = 1]$  or  $[(\text{tp}(\tau_1) = 2) \text{ and } (r(\tau_1) = 1)]$  then
                     $\tau = \tau + 1; r(\tau) = p; \text{tp}(\tau) = \text{tp}(\tau_1);$ 
                     $\text{dec}_1(\tau) = \tau_1; \text{dec}_2(\tau) = \tau_2;$ 
                end
            end
        end
    end
end

```

Note that $r(\tau) = |\tau|$ and that the last value of τ in Algorithm 1.2.1 is exactly equal to Θ (see Table 1.4). Thus Algorithm 1.2.1 provides us a way for computing the number of $2T^*$ -trees up to an arbitrary order p_{\max} . The values of $\text{dec}(\tau) = (\text{dec}_1(\tau), \text{dec}_2(\tau))$ for all $2T^*$ -trees of order up to three are given in Table 1.5. Now we are in a position to discuss the independence of the elementary differentials given by (1.2.5).

1.2.2.2 Independence of the elementary differentials

The independence of the elementary differentials insures tha the exact and the numerical solution can be uniquely expand in B-series. In this subsection we introduce a set of differential equations, similar to the set proposed in [8, Chapter 3], for which any finite number of elementary differentials evaluate to independent vectors.

Let \mathcal{U} is the set of all $2T^*$ -trees up to order p and Θ is the number of the elements in \mathcal{U} . Assume that an order relation in \mathcal{U} is introduced and that all the pure white trees are numbered with $\delta_1, \dots, \delta_p$. If the standard decomposition, described in Subsection 1.2.2.1, is used the first three δ_ς are 2, 6 and 14. Let any tree $\tau_i \in \mathcal{U}$ be decomposed as

$$\tau_i = [\text{tp}(\tau_i); \tau_{i_1}^{m_1}, \tau_{i_2}^{m_2}, \dots, \tau_{i_k}^{m_k}] \quad \text{for } i = 1, 2, \dots, \Theta, \quad (1.2.6)$$

where all the equal trees are collected together and the power notation to indicate the number of repetitions is used. This implies that all the trees τ_{i_j} for $j = 1, 2, \dots, k$ are distinct. Now for every tree $\tau_i \in \mathcal{U}$ we consider the following differential equations

$$u'_i = \nu u_{f(i_1)} + (1 - \nu) \prod_{j=1}^k \frac{1}{m_j!} u_{g(i_j)}^{m_j}, \quad \text{for } i = 1, 2, \dots, \Theta, \quad (1.2.7)$$

where

$$u_{f(i_1)} = \begin{cases} u_{\delta_p}, & \text{if } \tau_{i_1} = \emptyset, \\ u_{i_1}, & \text{else,} \end{cases}, \quad u_{g(i_j)} = \begin{cases} 1, & \text{if } \tau_{i_j} = \emptyset, \\ u_{\delta_{\varsigma+1}}, & \text{if } i_j = \delta_\varsigma, \\ u_{i_j}, & \text{else,} \end{cases}$$

and

$$\nu = \begin{cases} 0, & \text{if } \text{tp}(\tau_i) = \bullet, \\ 1, & \text{if } \text{tp}(\tau_i) = \circ. \end{cases}$$

The initial values are supposed to be $u_i(t_0) = 0$ for $i = 1, \dots, \Theta$; $i \neq \delta_1$ and $u_{\delta_1}(t_0) = 1$. If $e_1, e_2, \dots, e_\Theta$ denote the natural basis in \mathbb{R}^Θ then the resulting Θ dimensional system of differential equations obey the following property:

Theorem 1.2.1. *The values of the elementary differentials for the differential equations (1.2.7), evaluated at the initial value are given by*

$$F(\tau_i)(u(t_0)) = \begin{cases} e_{\delta_1}, & \text{if } i = \delta_p, \\ e_{\delta_{\varsigma+1}}, & \text{if } i = \delta_\varsigma, \\ e_i, & \text{else,} \end{cases} \quad \text{for } i = 1, 2, \dots, \Theta.$$

Proof. The Proof follows by induction on the number of the elements Θ in the set \mathcal{U} and from the definition of the differential equations (1.2.7). \square

i	τ_i	$\text{dec}_1(\tau_i)$	$\text{dec}_2(\tau_i)$		$u' = Lu + N$
1	\bullet	1	0	$[\bullet; \emptyset]$	$u'_1 = 0 + 1,$
2	\circ	2	0	$[\circ; \emptyset]$	$u'_2 = u_{14} + 0,$
3	$\bullet \vdots \bullet$	1	1	$[\bullet; \bullet]$	$u'_3 = 0 + u_1,$
4	$\circ \vdots \bullet$	1	2	$[\bullet; \circ]$	$u'_4 = 0 + u_6,$
5	$\bullet \vdots \circ$	2	1	$[\circ; \bullet]$	$u'_5 = u_1 + 0,$
6	$\circ \vdots \circ$	2	2	$[\circ; \circ]$	$u'_6 = u_2 + 0,$
7	$\bullet \vdots \bullet \vdots \bullet$	1	3	$[\bullet; \bullet \vdots \bullet]$	$u'_7 = 0 + u_3,$
8	$\circ \vdots \bullet \vdots \bullet$	1	4	$[\bullet; \circ \vdots \bullet]$	$u'_8 = 0 + u_4,$
9	$\bullet \vdots \circ \vdots \bullet$	1	5	$[\bullet; \bullet \vdots \circ]$	$u'_9 = 0 + u_5,$
10	$\circ \vdots \circ \vdots \bullet$	1	6	$[\bullet; \circ \vdots \circ]$	$u'_{10} = 0 + u_{14},$
11	$\bullet \vdots \bullet \vdots \circ$	2	3	$[\circ; \bullet \vdots \bullet]$	$u'_{11} = u_3 + 0,$
12	$\circ \vdots \bullet \vdots \circ$	2	4	$[\circ; \circ \vdots \bullet]$	$u'_{12} = u_4 + 0,$
13	$\bullet \vdots \circ \vdots \circ$	2	5	$[\circ; \bullet \vdots \circ]$	$u'_{13} = u_5 + 0,$
14	$\circ \vdots \circ \vdots \circ$	2	6	$[\circ; \circ \vdots \circ]$	$u'_{14} = u_6 + 0,$
15	$\bullet \vee \bullet$	3	1	$[\bullet; \bullet^2]$	$u'_{15} = 0 + \frac{1}{2}u_1^2,$
16	$\bullet \vee \circ$	3	2	$[\bullet; \bullet, \circ]$	$u'_{16} = 0 + u_1 u_6,$
17	$\circ \vee \circ$	4	2	$[\bullet; \circ^2]$	$u'_{17} = 0 + \frac{1}{2}u_6^2.$

Table 1.5: Standard decomposition for all trees up to order 3 and their corresponding differential equations.

The independence of the elementary differentials follows from the independence of the natural basis $e_1, e_2, \dots, e_\Theta$. In the case when $p = 3$ ($\Theta = 17$) and the order relation defined in Subsection 1.2.2.1 is introduced, the standard decomposition for all 2T*-trees, together with their recursive definitions (1.2.6) and the corresponding differential equations (1.2.7)

are given in Table 1.5.

1.2.2.3 Expansions of the exact and the numerical solutions

To compare the exact solution and the numerical solution as Taylor series expansions it is convenient to use B-series. For an elementary weight function $a : 2T^* \rightarrow \mathbb{R}$ the B-series is defined as

$$B(a(\tau), u) = a(\emptyset)u + \sum_{\tau \in 2T^*} h^{|\tau|} \frac{a(\tau)}{\sigma(\tau)} F(\tau)(u).$$

The exact solution of (1.2.1) can be represented by the following B-series

$$u(t+h) = B(\gamma(\tau)^{-1}, u).$$

Note that this is exactly the same as for Runge–Kutta methods. This is due to the definition of γ . We are now interested in finding the elementary weight function which describes the operations of the numerical method.

Before we do this, it is convenient to substitute the expansions for the functions $\phi^{[l]}$ defined in (1.2.2) into the computations of the method (1.2.3) and simplify, which gives

$$\begin{aligned} U_i &= \sum_{j=1}^s \sum_{l \geq 0} a_{ij}^{[l]} (hL)^l hN(U_j) + e^{c_i hL} u_{n-1}, \\ u_n &= \sum_{j=1}^s \sum_{l \geq 0} b_j^{[l]} (hL)^l hN(U_j) + e^{hL} u_{n-1}, \end{aligned}$$

where the coefficients of the method are now

$$\begin{aligned} a_{ij}^{[l]} &= \sum_{k=1}^m \alpha_{ij}^{[k]} \phi_l^{[k]}(c_i), \\ b_j^{[l]} &= \sum_{k=1}^m \beta_j^{[k]} \phi_l^{[k]}(1). \end{aligned} \tag{1.2.8}$$

To obtain B-series expansions of the numerical solution we need the following three results. The first result shows that the substitution of a B-series into the nonlinear part of the differential equation is again a B-series. The second result shows that a B-series operated on by powers of hL , is again a B-series. The final result shows that a B-series operated on by a power series of hL , is again a B-series.

Lemma 1.2.2. *Let $a : 2T^* \rightarrow \mathbb{R}$ be a mapping satisfying $a(\emptyset) = 1$, then*

$$hN(B(a(\tau), u)) = B(a'(\tau), u),$$

where the derivative of the elementary weight function satisfies, $a'(\emptyset) = 0$ and

$$a'(\tau) = \begin{cases} 0, & \text{if } \tau = [\circ; \tau_1], \\ a(\tau_1) \dots a(\tau_\ell), & \text{if } \tau = [\bullet; \tau_1, \dots, \tau_\ell]. \end{cases}$$

Proof. We refer to [8, 23] for very similar proofs. However, we mention that the elementary weight function depends on the colour of the trees root because the B-series is substituted into N which is represented by a black node. Therefore the elementary weight will be non-zero only for trees which have a black root node. \square

Lemma 1.2.3. *Let $a : 2T^* \rightarrow \mathbb{R}$ be a mapping, then*

$$(hL)^l B(a(\tau), u) = B((\mathcal{L}^l a)(\tau), u),$$

where the elementary weight function satisfies, $(\mathcal{L}^l a)(\emptyset) = 0$, and

$$(\mathcal{L}^l a)(\tau) = \begin{cases} (\mathcal{L}^{l-1} a)(\tau_1), & \text{if } \tau = [\circ; \tau_1], \\ 0, & \text{if } \tau = [\bullet; \tau_1, \dots, \tau_\ell]. \end{cases}$$

Proof. Consider first the case when $l = 1$, which gives

$$\begin{aligned} hL B(a(\tau), u) &= hLu + \sum_{\tau \in 2T^*} h^{|\tau|+1} \frac{a(\tau)}{\sigma(\tau)} LF(\tau)(u) \\ &= \sum_{\tau \in 2T^*} h^{|\tau|} \frac{(\mathcal{L}a)(\tau)}{\sigma(\tau)} F(\tau)(u), \end{aligned}$$

where the elementary weight function must be zero for all trees with a black root node. The general l follows by recursively applying the result for $l = 1$. \square

Corollary 1.2.4. *Let $\psi_x(z)$ be a power series*

$$\psi_x(z) = \sum_{l \geq 0} x^{[l]} z^l,$$

and let $a : 2T^* \rightarrow \mathbb{R}$ be a mapping, then

$$\psi_x(hL) B(a(\tau), u) = B((\psi_x(\mathcal{L})a)(\tau), u)$$

where the elementary weight function satisfies, $(\psi_x(\mathcal{L})a)(\emptyset) = x^{[0]}a(\emptyset)$, and

$$(\psi_x(\mathcal{L})a)(\tau) = \sum_{l \geq 0} x^{[l]} (\mathcal{L}^l a)(\tau).$$

Proof. This corollary is proved by repeated uses of Lemma 1.2.3, as follows

$$\begin{aligned}
\psi_x(hL)B(a(\tau), u) &= \sum_{l \geq 0} x^{[l]}(hL)^l B(a(\tau), u) \\
&= \sum_{l \geq 0} x^{[l]} B(\mathcal{L}^l a(\tau), u) \\
&= B\left(\sum_{l \geq 0} x^{[l]} \mathcal{L}^l a(\tau), u\right).
\end{aligned}$$

□

We now have everything we need to represent the numerical method using B-series. Using Corollary 1.2.4, it follows that

$$e^{hL}u_{n-1} = B((e^{\mathcal{L}}\mathbf{1})(\tau), u_{n-1}),$$

since $u_{n-1} = B(\mathbf{1}(\tau), u_{n-1})$, where the elementary weight function $\mathbf{1}(\tau)$ is non-zero only for the empty set. Also from Lemma 1.2.2 and Corollary 1.2.4, we see that

$$\begin{aligned}
\sum_{l \geq 0} a_{ij}^{[l]}(hL)^l hN(U_j) &= \psi_{a_{ij}}(hL)B(\xi'_j(\tau), u_{n-1}) \\
&= B((\psi_{a_{ij}}(\mathcal{L})\xi'_j)(\tau), u_{n-1}).
\end{aligned}$$

Putting this all together and translating in terms of trees gives the generating functions

$$\begin{aligned}
\xi_i(\tau) &= \sum_{j=1}^s (\psi_{a_{ij}}(\mathcal{L})\xi'_j)(\tau) + (e^{c_i \mathcal{L}}\mathbf{1})(\tau), \\
\alpha(\tau) &= \sum_{j=1}^s (\psi_{b_j}(\mathcal{L})\xi'_j)(\tau) + (e^{\mathcal{L}}\mathbf{1})(\tau).
\end{aligned}$$

To represent these elementary weight functions in matrix form we introduce certain matrices. Let for $l = 0, 1, 2, \dots$ and $k = 1, \dots, m$, $\phi_l^{[k]}$, be the $s \times s$ diagonal matrix defined by

$$\phi_l^{[k]}(c) = \text{diag}\left(\phi_l^{[k]}(c_1), \dots, \phi_l^{[k]}(c_s)\right).$$

Also for $l = 0, 1, 2, \dots$ the matrices $A^{[l]}$ and the vectors $b^{[l]T}$ to be defined by

$$\begin{aligned}
A^{[l]} &= \sum_{k=1}^m \phi_l^{[k]}(c) \alpha^{[k]}, \\
b^{[l]T} &= \sum_{k=1}^m \phi_l^{[k]}(1) \beta^{[k]T}.
\end{aligned} \tag{1.2.9}$$

Note that (1.2.9) are the matrix versions of (1.2.8). The matrix representation of the elementary weight functions interpreted in the natural way are therefore

$$\begin{aligned}\xi(\tau) &= (\psi_A(\mathcal{L})\xi')(\tau) + (e^{c\mathcal{L}}\mathbf{1})(\tau), \\ \alpha(\tau) &= (\psi_{b^T}(\mathcal{L})\xi')(\tau) + (e^{\mathcal{L}}\mathbf{1})(\tau).\end{aligned}$$

Given that we have B-series expansions for both the exact and the numerical solutions, we can now define order in the same way as for Runge–Kutta methods.

Definition 1.2.4. An exponential Runge–Kutta method with elementary weight function $a : 2T^* \rightarrow \mathbb{R}$, has order p , if for all $\tau \in 2T^*$ such that $|\tau| \leq p$,

$$a(\tau) = \frac{1}{\gamma(\tau)}.$$

This shows that it is not possible in general to obtain order by simply using a Runge–Kutta method for the nonlinear part N of the problem. There are coupling conditions between the nonlinear and linear parts of the problem despite the fact that the linear part has been solved exactly. Note that if $L = 0$, then $A^{[l]} = 0$ and $b^{[l]^T} = 0$ for all $l = 1, 2, \dots$. Thus, all order conditions simply reduce to the order conditions corresponding to the black trees. In this case $A = A^{[0]} \otimes I_d$ and $b^T = b^{[0]^T} \otimes I_d$. Therefore, the method (1.2.4) is equivalent to a Runge–Kutta method for the nonlinear part N . This method is known as the underlying Runge–Kutta method.

It is clear that Definition 1.2.4 is automatically satisfied for all purely white trees. We now define the matrices

$$C^{[j]} = \frac{1}{(j+1)!} C^{j+1},$$

where $C = \text{diag}(c_1, \dots, c_s)$. This strange choice of index is so that the non-recursive rule is straight forward. Now for all remaining trees the elementary weight function a of the numerical solution can equivalently be computed using the following rule:

- Attach $b^{[j]^T}$ to the root black node.
- Attach $A^{[j]}$ to all remaining nonterminal black nodes.
- Attach $A^{[j]}e$ to all terminal black nodes.
- Attach $C^{[j]}e$ to all terminal white nodes.
- Attach I to all remaining white nodes.

The value j is the number of white nodes directly below the corresponding node. Now for each tree multiply from the root to the leaf as in the case for Runge–Kutta methods, then multiply these expressions in a component by component sense.

Table 1.6 gives the elementary differentials for all non-pure white trees of order three and less and the corresponding order, density and elementary weight function of the tree.















τ	$ \tau $	$\gamma(\tau)$	$F(\tau)$	$\alpha(\tau)$	$\alpha(\tau)$
	1	1	N	$\sum_i b_i^{[0]}$	$b^{[0]T} e$
	2	2	$N'N$	$\sum_{ij} b_i^{[0]} a_{ij}^{[0]}$	$b^{[0]T} A^{[0]} e$
	2	2	$N'L$	$\sum_i b_i^{[0]} c_i$	$b^{[0]T} C^{[0]} e$
	2	2	LN	$\sum_i b_i^{[1]}$	$b^{[1]T} e$
	3	6	$N'N'N$	$\sum_{ijk} b_i^{[0]} a_{ij}^{[0]} a_{jk}^{[0]}$	$b^{[0]T} A^{[0]} A^{[0]} e$
	3	6	$N'N'L$	$\sum_{ij} b_i^{[0]} a_{ij}^{[0]} c_j$	$b^{[0]T} A^{[0]} C^{[0]} e$
	3	6	$N'LN$	$\sum_{ij} b_i^{[0]} a_{ij}^{[1]}$	$b^{[0]T} A^{[1]} e$
	3	6	$N'LL$	$\frac{1}{2} \sum_i b_i^{[0]} c_i^2$	$b^{[0]T} C^{[1]} e$
	3	6	$LN'N$	$\sum_{ij} b_i^{[1]} a_{ij}^{[0]}$	$b^{[1]T} A^{[0]} e$
	3	6	$LN'L$	$\sum_{ij} b_i^{[1]} c_i$	$b^{[1]T} C^{[0]} e$
	3	6	LLN	$\sum_i b_i^{[2]}$	$b^{[2]T} e$
	3	3	$N''(N, N)$	$\sum_{ijk} b_i^{[0]} a_{ij}^{[0]} a_{ik}^{[0]}$	$b^{[0]T} (A^{[0]} e)(A^{[0]} e)$
	3	3	$N''(N, L)$	$\sum_{ij} b_i^{[0]} a_{ij}^{[0]} c_i$	$b^{[0]T} (A^{[0]} e)(C^{[0]} e)$
	3	3	$N''(L, L)$	$\sum_i b_i^{[0]} c_i^2$	$b^{[0]T} (C^{[0]} e)(C^{[0]} e)$

Table 1.6: Relation between elementary differentials and elementary weights.

1.2.3 IF RK and ETD RK as special cases

In this subsection we show that the IF RK and ETD RK methods, introduced in the beginning of Section 1.2, are special cases of the exponential Runge–Kutta methods (1.2.4). Therefore, the presented nonstiff order theory governs these special cases. For each of these special cases we first need to identify the structure of the $\phi^{[l]}$ functions. One possible approach is to use the variation of constants formulae (1.1.4). It is preferable to replace the nonlinear part N with an approximation, which will make it possible to solve the integral

in (1.1.4) exactly.

Consider approximations of the nonlinear term N which leads to the IF RK methods. Let $N(u(t_{n-1} + \tau), t_{n-1} + \tau) \approx \delta_i e^{L(t_{n-1} + \tau)}$, where δ_i is a constant chosen in such a way that the approximation matches $N(u(t_{n-1} + \tau), t_{n-1} + \tau)$ for $\tau = c_i h$. In this case for $l = 1, \dots, s$ we get

$$\phi^{[l]}(\lambda)(hL) = e^{(\lambda - c_l)hL}. \quad (1.2.10)$$

Every IF Runge–Kutta method (see Table 1.3) can be represented in the form (1.2.4) with $\phi^{[l]}$ functions given by (1.2.10) and with a special choice of the coefficient matrices $\alpha^{[l]}$ and the coefficient vectors $\beta^{[l]^T}$. This choice reduces the set of all order conditions to a set which consists only of the order conditions corresponding to the black trees. In the proof of this fact we will need the following lemma.

Lemma 1.2.5. *Let $t \in \mathbb{R} \setminus \{0, -1, -2, \dots\}$, then for $j = 0, 1, 2, \dots$*

$$\sum_{k=0}^j \frac{(-1)^k}{k!(j-k)!} \frac{1}{(k+t)} = \frac{1}{t(t+1) \cdots (t+j)}.$$

Proof. We prove this statement by induction on j . For $j = 0$ the equality is obviously true since $\frac{1}{t} = \frac{1}{t}$. Assume now that the statement is true for some j . Consider

$$\begin{aligned} (t+j+1) \sum_{k=0}^{j+1} \frac{(-1)^k}{k!(j+1-k)!} \frac{1}{(k+t)} &= \sum_{k=0}^{j+1} \frac{(-1)^k}{k!(j+1-k)!} \frac{(t+k) + (j+1-k)}{(t+k)} \\ &= \sum_{k=0}^{j+1} \frac{(-1)^k}{k!(j+1-k)!} + \sum_{k=0}^{j+1} \frac{(-1)^k}{k!(j+1-k)!} \frac{(j+1-k)}{(k+t)} \\ &= \sum_{k=0}^j \frac{(-1)^k}{k!(j-k)!} \frac{1}{(k+t)} = \frac{1}{t(t+1) \cdots (t+j)}. \end{aligned}$$

□

The following theorem defines the structure of the matrices $\alpha^{[l]}$ and the vectors $\beta^{[l]^T}$ for the IF RK methods. With this structure of the coefficients, to achieve certain order, it is sufficient to satisfy only the black trees. This implies that the transformed differential equation (1.1.2) is solved using a Runge–Kutta method.

Theorem 1.2.6. *Let all the non-zero coefficients of an exponential Runge–Kutta method (1.2.4), with $\phi^{[l]}$ functions given by (1.2.10) be located in column number l of the matrix $\alpha^{[l]}$ and in position number l of the vector $\beta^{[l]^T}$ for $l = 1, 2, \dots, s$. The method has order p iff all order conditions corresponding to the black trees are satisfied.*

Proof. From Definition 1.2.4, it follows directly that if the exponential Runge–Kutta method has order p then all order conditions corresponding to the black trees are satisfied. Let us assume that all the order conditions corresponding to the black trees are satisfied. We need

to prove that all the remaining order conditions are also satisfied. From the definition of the $\phi^{[l]}$ functions (1.2.10), it follows that for $j = 0, 1, 2, \dots$,

$$\phi_j^{[l]}(1) = \frac{(1 - c_l)^j}{j!}, \quad \phi_j^{[l]}(c) = \frac{1}{j!} \text{diag}((c_1 - c_l)^j, \dots, (c_s - c_l)^j). \quad (1.2.11)$$

Since all order conditions corresponding to the black trees involve only the coefficients $A^{[0]}$, $b^{[0]T}$ and c , we need to express every other order condition in terms of these coefficients. Having in mind the special structure of the matrices $\alpha^{[l]}$ and the vectors $\beta^{[l]T}$, after substituting (1.2.11) into (1.2.9), we obtain for $j = 1, 2, \dots$,

$$\begin{aligned} A^{[j]} &= \sum_{k=0}^j \frac{(-1)^k}{k!(j-k)!} C^{[0]^{j-k}} A^{[0]} C^{[0]^k}, \\ b^{[j]T} &= \sum_{k=0}^j \frac{(-1)^k}{k!(j-k)!} b^{[0]T} C^{[0]^k}. \end{aligned} \quad (1.2.12)$$

From the fact that all order conditions corresponding to the black trees are satisfied, it follows that $A^{[0]}$, $b^{[0]T}$ and c form a Runge–Kutta method. Therefore,

$$\begin{aligned} C^{[0]}e &= A^{[0]}e, \\ C^{[0]}\zeta &= (A^{[0]}e)(\zeta), \\ C^{[0]^k}\zeta &= (A^{[0]}e) \dots (A^{[0]}e)(\zeta), \end{aligned} \quad (1.2.13)$$

where ζ is an arbitrary vector and the multiplications in the second and third expressions are in a component by component sense.

Now we are in a position to define a procedure which transforms every coloured tree τ into a linear combination of black trees of order at most $|\tau|$. Each tree τ can be decomposed as $\tau = (\tau_b, \tau_j, \tau_t)$, where τ_b is a coloured tree on the bottom with less number of white nodes than τ ; τ_j is tall white tree with $j \geq 1$ white nodes and τ_t is black tree on the top. First applying formula (1.2.12) and then (1.2.13), for the order condition corresponding to a tree τ , we obtain the following three representations in terms of black trees or trees with less white vertices.

If $\tau_t = \emptyset$ then τ reduces to

$$\tau = \begin{array}{c} \circ \\ \vdots \\ \circ \\ \bullet \\ \text{---} \tau_b \end{array} = \frac{1}{j!} \begin{array}{c} \bullet \cdots \bullet \\ \diagup \quad \diagdown \\ \bullet \\ \text{---} \tau_b \end{array}.$$

If $\tau_b = \emptyset$ then τ reduces to

$$\tau = \begin{array}{c} \tau_t \\ \bullet \\ \vdots \\ \circ \end{array} = \delta_0 \begin{array}{c} \tau_t \\ \bullet \end{array} + \dots + \delta_k \begin{array}{c} \bullet \cdots \bullet \\ \diagup \quad \diagdown \\ \bullet \\ \text{---} \tau_t \end{array} + \dots + \delta_j \begin{array}{c} \bullet \cdots \bullet \\ \diagup \quad \diagdown \\ \bullet \\ \text{---} \tau_t \end{array},$$

where $\delta_k = \frac{(-1)^k}{k!(j-k)!}$ for $k = 0, 1, 2, \dots, j$. In the general case when $\tau_{\{t,b\}} \neq \emptyset$, then τ can be

represented as

$$\tau = \begin{array}{c} \circ \\ \vdots \\ \circ \end{array} \begin{array}{c} \tau_t \\ \vdots \\ \tau_b \end{array} = \delta_0 \begin{array}{c} \circ \cdots \circ \\ \vdots \\ \circ \end{array} \begin{array}{c} \tau_t \\ \vdots \\ \tau_b \end{array} + \cdots + \delta_k \begin{array}{c} \circ \cdots \circ \\ \vdots \\ \circ \end{array} \begin{array}{c} \tau_t \\ \vdots \\ \tau_b \end{array} + \cdots + \delta_j \begin{array}{c} \circ \cdots \circ \\ \vdots \\ \circ \end{array} \begin{array}{c} \tau_t \\ \vdots \\ \tau_b \end{array} . \quad (1.2.14)$$

For each of the trees in the linear combination we apply the same procedure. Thus, after a finite number of steps all the trees in the combination will be black. From (1.2.9) it is clear that the order of every single black tree cannot exceed the order of coloured tree. See Table 1.7 for the representations of all trees of order three and less. To complete the proof we need to show that $a(\tau) = 1/\gamma(\tau)$ for all coloured trees τ , where $|\tau| \leq p$. We prove this by induction on the number of steps θ in the transformation process. Let $\theta = 1$. Every coloured tree τ has representation $\tau = \sum_{k=0}^j \delta_k \tau_k$, where all τ_k are black trees. If $\gamma(\tau_t) = x_1|\tau_t|x_2$ then $a(\tau_k) = \frac{1}{\gamma(\tau_k)} = \frac{1}{x_1(|\tau_t|+k)x_2}$ and by Lemma 1.2.5 for $t = |\tau_t|$ it follows that

$$\begin{aligned} a(\tau) &= \sum_{k=0}^j \frac{(-1)^k}{k!(j-k)!} a(\tau_k) \\ &= \sum_{k=0}^j \frac{(-1)^k}{k!(j-k)!} \frac{1}{\gamma(\tau_k)} \\ &= \sum_{k=0}^j \frac{(-1)^k}{k!(j-k)!} \frac{1}{x_1(|\tau_t|+k)x_2} \\ &= \frac{1}{x_1|\tau_t|(|\tau_t|+1) \cdots (|\tau_t|+j)x_2} = \frac{1}{\gamma(\tau)}. \end{aligned}$$

Assume that $a(\tau) = 1/\gamma(\tau)$ for all coloured trees τ with θ steps in the transformation process. Let τ be a tree with $\theta + 1$ steps in the transformation process. From (1.2.14) it follows that $\tau = \sum_{k=0}^j \delta_k \tau_k$, where τ_k are coloured trees with θ steps in the transformation process and hence $a(\tau_k) = 1/\gamma(\tau_k)$. If $\gamma(\tau_t) = x_1|\tau_t|x_2$ then $a(\tau_k) = \frac{1}{\gamma(\tau_k)} = \frac{1}{x_1(|\tau_t|+k)x_2}$ and by Lemma 1.2.5 for $t = |\tau_t|$ it again follows that $a(\tau) = 1/\gamma(\tau)$. \square

The following result shows that the non-zero coefficients of an IF RK method are those of the underlying Runge–Kutta method.

Corollary 1.2.7. *Under the assumptions of Theorem 1.2.6 it follows that all non-zero coefficients of an exponential Runge–Kutta method (1.2.4) are exactly equal to the coefficients of the underlying Runge–Kutta method.*

Proof. Since $\phi_0^{[l]}(1) = 1$ and $\phi_0^{[l]}(c) = I$, it follows from (1.2.9), that

$$\begin{aligned} A^{[0]} &= \alpha^{[1]} + \alpha^{[2]} + \cdots + \alpha^{[s-1]}, \\ b^{[0]T} &= \beta^{[1]T} + \beta^{[2]T} + \cdots + \beta^{[s]T}. \end{aligned}$$

The result now follows from the special structure of $\alpha^{[l]}$ and $\beta^{[l]T}$. \square

Table 1.7 gives all the trees up to order 3 with at least one white node in terms of linear combinations of trees with only black nodes. This represents how the order conditions for the IF RK methods are automatically satisfied by the order conditions of the underlying Runge–Kutta method.

τ		τ
	=	
	=	
	=	-
	=	$\frac{1}{2}$ - $\frac{1}{2}$
	=	

τ		τ
	=	\bullet -
	=	-
	=	$\frac{1}{2}$ \bullet - + $\frac{1}{2}$
	=	-
	=	

Table 1.7: Coloured trees as linear combinations of black trees for the IF RK methods.

We now consider approximations of the nonlinear term N which leads to the ETD RK methods. Let $N(u(t_{n-1} + \tau), t_{n-1} + \tau) \approx p_{n-1}(\tau)$, where $p_{n-1}(\tau)$ is interpolation polynomial of degree $s - 1$ that matches $N(u(t_{n-1} + \tau), t_{n-1} + \tau)$ at the points $\tau = c_1 h, c_2 h, \dots, c_s h$. This approximation leads exactly to the $\phi^{[i]}$ functions (1.1.6) from Lemma 1.1.1. We call this functions ETD $\phi^{[i]}$ functions. The explicit form of the first few of them is

$$\begin{aligned}
\phi^{[1]}(\lambda)(hL) &= \phi^{[1]}(\lambda hL) = 1I_m + \frac{\lambda}{2!}hL + \frac{\lambda^2}{3!}(hL)^2 + \frac{\lambda^3}{4!}(hL)^3 + \dots, \\
\phi^{[2]}(\lambda)(hL) &= \phi^{[2]}(\lambda hL) = \frac{1}{2!}I_m + \frac{\lambda}{3!}(hL) + \frac{\lambda^2}{4!}(hL)^2 + \frac{\lambda^3}{5!}(hL)^3 + \dots, \\
\phi^{[3]}(\lambda)(hL) &= \phi^{[3]}(\lambda hL) = \frac{1}{3!}I_m + \frac{\lambda}{4!}(hL) + \frac{\lambda^2}{5!}(hL)^2 + \frac{\lambda^3}{6!}(hL)^3 + \dots.
\end{aligned} \tag{1.2.15}$$

As an example of an ETD Runge–Kutta method, in Table 1.8, we write out an exponential integrator, which reduces to the classical fourth order Runge–Kutta method (1.3.5) for $L = 0$. The coefficients of the method are chosen in such a way that all nonstiff order conditions up to order four as well as half of the conditions of order five are satisfied.

As we mentioned before, other choices for the $\phi^{[l]}$ functions (1.2.2), rather than the IF (1.2.10) and the ETD (1.1.6) functions, are also possible. In Chapter 2 we give examples of functions which originate from the framework of Lie group methods. The question how to find the best set of functions (1.2.2) is open and needs further investigation. It seems that the most commonly used functions in the literature are the ETD $\phi^{[i]}$ functions. In the next subsection we present some third and fourth order ETD RK methods, which recently have been studied by several authors.

$$\left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & I \\ \frac{1}{2}\phi^{[1]} & 0 & 0 & 0 & e^{\frac{1}{2}hL} \\ \frac{21}{50}\phi^{[1]} - \frac{6}{25}\phi^{[2]} & \frac{2}{25}\phi^{[1]} + \frac{6}{25}\phi^{[2]} & 0 & 0 & e^{\frac{1}{2}hL} \\ \frac{19}{20}\phi^{[1]} - \frac{9}{10}\phi^{[2]} - 3\phi^{[3]} & \frac{21}{5}\phi^{[2]} - \frac{108}{5}\phi^{[3]} & \frac{1}{20}\phi^{[1]} - \frac{33}{10}\phi^{[2]} + \frac{123}{5}\phi^{[3]} & 0 & e^{hL} \\ \hline b_1(hL) & b_2(hL) & b_3(hL) & b_4(hL) & e^{hL} \end{array} \right],$$

where

$$\begin{aligned} b_1(hL) &= \frac{31}{30}\phi^{[1]} - \frac{17}{5}\phi^{[2]} + 6\phi^{[3]} - 4\phi^{[4]}, \\ b_2(hL) &= -\frac{1}{10}\phi^{[1]} + \frac{1}{5}\phi^{[2]} - 4\phi^{[3]} + 12\phi^{[4]}, \\ b_3(hL) &= \frac{1}{30}\phi^{[1]} + \frac{23}{5}\phi^{[2]} - 8\phi^{[3]} - 4\phi^{[4]}, \\ b_4(hL) &= \frac{1}{30}\phi^{[1]} - \frac{7}{5}\phi^{[2]} + 6\phi^{[3]} - 4\phi^{[4]}. \end{aligned}$$

Table 1.8: A fourth order ETD Runge–Kutta method.

1.2.4 More examples of ETD RK methods

All methods presented in this subsection are based on in some sense “mysterious” observations, which do not explicitly involve the order theory from Subsection 1.2.2 or any other order theory. We include this methods as examples of exponential integrators, which during the last few years have been studied by several authors [14, 29, 30, 35, 36]. Thus these methods have obtained an important place on their own. We try as much as reasonably possible, for each method, to present the authors main motivation used in the derivation process. In the presentation of each method we use the notations introduced in Section 1.3. We mention also that all the methods satisfy the classical (nonstiff) order conditions from Subsection 1.2.2 up to the order which their authors predict, but most of them suffer from order reduction in the stiff case. We will comment more on this in Subsection 1.2.5.2.

In Tables 1.9 and 1.10 we give the two multistage methods proposed in [14]. When the linear part L is equal to zero, they reduce to the classical third and fourth order Runge–Kutta method respectively. Regarding the derivation of the fourth order method (Table 1.10), in [14] is written: “The computer algebra package Maple was used to confirm that this method is indeed fourth order”.

A surprising connection between the fourth order method of Cox–Matthews (Table 1.10) and the fourth order commutator free Lie group method (Table 2.6), based on the affine algebra action (see Section 2.3), was found in [36, Article 5]. Note that the internal stages

0	0	0	I
$\frac{1}{2}\phi^{[1]}$	0	0	$e^{\frac{1}{2}hL}$
$-\phi^{[1]}$	$2\phi^{[1]}$	0	e^{hL}
$\phi^{[1]} - 3\phi^{[2]} + 4\phi^{[3]}$			e^{hL}

Table 1.9: Third order method of Cox–Matthews.

0	0	0	0	I
$\frac{1}{2}\phi^{[1]}$	0	0	0	$e^{\frac{1}{2}hL}$
0	$\frac{1}{2}\phi^{[1]}$	0	0	$e^{\frac{1}{2}hL}$
$\frac{1}{2}\phi^{[1]}\left(\frac{hL}{2}\right)\left(e^{\frac{hL}{2}} - I\right)$	0	$\phi^{[1]}\left(\frac{hL}{2}\right)$	0	e^{hL}
$\phi^{[1]} - 3\phi^{[2]} + 4\phi^{[3]}$				e^{hL}

Table 1.10: Fourth order method of Cox–Matthews.

of this two methods are the same. It is shown in [36, Article 5] that the main step of Cox–Matthews method can be reproduced based on the techniques of *continuous* Runge–Kutta methods [24, Chapter II.6]. Motivated from the same idea, but also applied to the internal stages of the method, a new fourth order method is derived in [36, Article 5]. This method, which is also based on the classical fourth order Runge–Kutta method (1.3.5), is given in Table 1.11.

0	0	0	0	I
$\frac{1}{2}\phi^{[1]}$	0	0	0	$e^{\frac{1}{2}hL}$
$\frac{1}{2}\phi^{[1]} - \phi^{[2]}$	$\phi^{[2]}$	0	0	$e^{\frac{1}{2}hL}$
$\phi^{[1]} - 2\phi^{[2]}$	0	$2\phi^{[2]}$	0	e^{hL}
$\phi^{[1]} - 3\phi^{[2]} + 4\phi^{[3]}$				e^{hL}

Table 1.11: Fourth order method of Krogstad.

Similar techniques are used in the construction of ETD2RK3 (Table 1.12) and ETD2CF3 (Table 1.13) methods proposed in [45]. The first of this methods also reduces, for $L = 0$, to

$$\left[\begin{array}{ccc|c} 0 & 0 & 0 & I \\ \frac{1}{2}\phi^{[1]} & 0 & 0 & e^{\frac{1}{2}hL} \\ \phi^{[1]} - 4\phi^{[2]} & 4\phi^{[2]} & 0 & e^{hL} \\ \hline \phi^{[1]} - 3\phi^{[2]} + 4\phi^{[3]} & 4\phi^{[2]} - 8\phi^{[3]} & -\phi^{[2]} + 4\phi^{[3]} & e^{hL} \end{array} \right]$$

Table 1.12: Third order ETD2RK3 method.

$$\left[\begin{array}{ccc|c} 0 & 0 & 0 & I \\ \frac{1}{3}\phi^{[1]} & 0 & 0 & e^{\frac{1}{3}hL} \\ \frac{2}{3}\phi^{[1]} - \frac{4}{3}\phi^{[2]} & \frac{4}{3}\phi^{[2]} & 0 & e^{\frac{2}{3}hL} \\ \hline \phi^{[1]} - \frac{9}{2}\phi^{[2]} + 9\phi^{[3]} & 6\phi^{[2]} - 18\phi^{[3]} & -\frac{3}{2}\phi^{[2]} + 9\phi^{[3]} & e^{hL} \end{array} \right]$$

Table 1.13: Third order ETD2CF3 method.

the classical third order Runge–Kutta method. It is simply a modification of the third order method of Cox–Matthews, which adapts the continuous idea for the internal stages as well. The second method is based on the third order commutator free Lie group method proposed in [12]. However, instead of using the affine algebra action (see Section 2.3), the internal stages of ETD2CF3 are derived based on the algebra action proposed in [Article 1, p.91]. Splitting of the main step is again avoided by using the continuous technique suggested in [36, Article 5].

1.2.5 Exponential RK methods for parabolic PDEs

The order theory presented in Section 1.2.2 highly relies on the assumption that L is bounded linear operator on a d -dimensional Euclidian space \mathbb{R}^d . In this case the meaning of the exponential operator e^{tL} is given by the well known formula

$$e^{tL} = \sum_{k=0}^{\infty} \frac{t^k L^k}{k!}, \quad t \in \mathbb{R}.$$

The related ETD $\phi^{[i]}$ functions are also defined in a similar way (see 1.2.15).

To analyze the order of an exponential integrator applied to a Partial Differential Equation (PDE), we need a framework, which allows us to define the exponential and the related functions in the case when L is an unbounded linear operator. In the sequel, we follow the presentation in [29, 30] and restrict our considerations to the framework of sectorial operators and analytic semigroups. It is well known that parabolic problems like reaction-diffusion equations, incompressible Navier-Stokes equations in two and three space dimensions as well

as displacement of a shock fit into this framework (see [26, Chapter 3], [42, Section 7.3] and [54, Lecture 5]).

The key idea, used in [29, 30], is to analyze exponential integrators of Runge–Kutta type for parabolic PDEs based on the observation that any parabolic PDE can be viewed as an abstract Ordinary Differential Equation (ODE). We illustrate this concept in the next example.

Consider the following one dimensional reaction-diffusion equations

$$\begin{aligned}\frac{\partial U}{\partial t}(x, t) &= \frac{\partial}{\partial x} \left(a(x) \frac{\partial U}{\partial x}(x, t) \right) + \varphi(x, t, U(x, t)), \\ U(x, 0) &= U_0(x) \quad 0 < x < 1, \\ U(0, t) &= U(1, t) = 0 \quad t > 0.\end{aligned}\tag{1.2.16}$$

We define the functions

$$\begin{aligned}u(t) &= [x \rightarrow U(x, t)] \in L^2(0, 1), \\ u_0 &= [x \rightarrow U(x, 0)] \in L^2(0, 1), \\ N(u, t) &= [x \rightarrow \varphi(x, t, u(x))] \in L^2(0, 1)\end{aligned}$$

and the linear operator

$$Lu = \left[x \rightarrow \frac{\partial}{\partial x} \left(a(x) \frac{\partial U}{\partial x}(x, t) \right) \right].$$

Thus, the equation (1.2.16) is equivalent to the following abstract semilinear problem on the Banach space $L^2(0, 1)$

$$u'(t) = Lu + N(u, t), \quad u(t_0) = u_0,\tag{1.2.17}$$

where now the linear operator L is unbounded. An appropriate setting where the exponential and the related operators have a precise meaning, for unbounded linear operator L in a Banach space X , is the framework of sectorial operators and analytic semigroups. We next introduce the main concepts and some error bounds which are crucial for the convergence analysis presented in the recent papers [29, 30].

1.2.5.1 The framework

Let $(X, \|\cdot\|_X)$ be a real or complex Banach space and let L be a linear operator in X with dense domain $\mathcal{D}(L) = \{x \in X : Lx \in X\}$ equipped with the graph norm $\|x\|_{\mathcal{D}(L)} = \|Lx\|_X$. In order to define the exponential operator e^{tL} we need the notion of *sectorial* operators.

Definition 1.2.5. ([26, Definition 1.3.1], [42, Definition 2.0.1]) A densely defined and closed linear operator L in X is called *sectorial* if there are constants $a \in \mathbb{R}$, $\pi/2 < \varphi < \pi$ and $M > 1$ such that the following resolvent condition is satisfied

$$\|(\lambda I - L)^{-1}\|_{X \leftarrow X} \leq \frac{M}{|\lambda - a|}\tag{1.2.18}$$

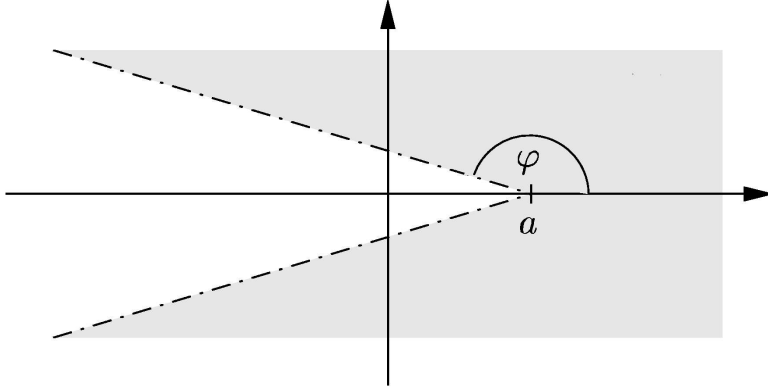


Figure 1.1: Domain of validity for condition (1.2.18) (shaded).

on the sector $\{\lambda \in \mathbb{C} : \lambda \neq a, |\arg(\lambda - a)| < \varphi\}$.

The sector from Definition 1.2.5 is plotted on Figure 1.1. Next we give the definition of an *analytic semigroup*.

Definition 1.2.6. ([26, Definition 1.3.3], [42, Section 2]) The family $(S(t))_{t \geq 0}$ of linear operators on X forms a *semigroup* if the relations $S(0) = I$ and

$$S(s)S(t) = S(s + t), \quad s, t \geq 0,$$

are satisfied. The semigroup is called *analytic* if the map $t \rightarrow S(t)$ is analytic.

Assuming that the linear operator L is sectorial, we can define the exponential operator e^{tL} by the means of the Cauchy integral formula.

Definition 1.2.7. ([26, Definition 1.3.4], [42, Definition 2.0.2]) Let L be a sectorial operator in X , and, for some radius $r > 0$ and an angle $\pi/2 < \eta < \varphi$, let the counterclockwise oriented contour Γ consists of the half lines γ_+ and γ_- given by

$$\gamma_{\pm} = a + \{\lambda \in \mathbb{C} : |\lambda| \geq r, \arg(\lambda) = \pm\eta\},$$

and the arc $\gamma_r = a + \{\lambda \in \mathbb{C} : |\lambda| = r, \arg(\lambda) \leq \eta\}$, see Figure 1.2. Then, the family $(e^{tL})_{t \geq 0}$ defined through

$$e^{tL} = \frac{1}{2\pi i} \int_{\Gamma} e^{\lambda t} (\lambda I - L)^{-1} d\lambda, \quad t > 0, \quad e^{0L} = I, \quad (1.2.19)$$

is an analytic semigroup generated by L in X .

The operator $e^{tL} : X \rightarrow X$ is linear and bounded in X for each $t \geq 0$. This follows directly from the resolvent bound (1.2.18). For the rest of the presentation in this section

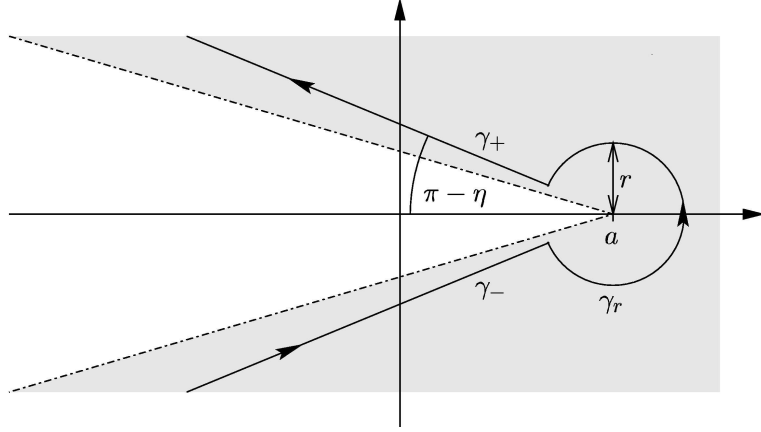


Figure 1.2: The contour Γ from Definition 1.2.7.

we assume that the linear operator L is sectorial and thus the operator e^{tL} is well-defined and bounded in X .

When the parameter a is negative, from Definition 1.2.5, follows that the operator $-L$ has a bounded inverse. Motivated by the definition of the gamma function, for any $\alpha > 0$, we can define fractional powers of $-L$ through the following expression

$$\int_0^\infty e^{tL} t^{\alpha-1} dt = (-L)^{-\alpha} \int_0^\infty e^{-x} x^{\alpha-1} dx = (-L)^{-\alpha} \Gamma(\alpha).$$

Therefore

$$(-L)^{-\alpha} = \frac{1}{\Gamma(\alpha)} \int_0^\infty e^{tL} t^{\alpha-1} dt.$$

The fractional powers of $-L$, for any $\alpha > 0$, are now defined by

$$(-L)^\alpha : \mathcal{D}((-L)^\alpha) \rightarrow X : x \rightarrow (-L)^\alpha x := ((-L)^{-\alpha})^{-1} x,$$

where $\mathcal{D}((-L)^\alpha) = \text{range}((-L)^{-\alpha})$.

If $a > 0$, we consider the shifted operator $\tilde{L} = \omega I - L$, with $\omega > a$. Fractional powers of \tilde{L} are defined in the similar way.

From the Cauchy's integral formula (1.2.19) and the resolvent bound (1.2.18) the following uniform stability bounds are derived in [26, 42, 54]

$$\begin{aligned} \|e^{tL}\|_{X \leftarrow X} &\leq C_0 e^{at}, & \text{for } 0 \leq t \leq T, \\ \|\tilde{L}^\alpha e^{tL}\|_{X \leftarrow X} &\leq C_{\alpha,\omega} e^{\omega t} t^{-\alpha}, & \text{for } 0 < t \leq T, \quad \omega > a. \end{aligned} \quad (1.2.20)$$

We next discuss the main assumptions which the nonlinearity N has to satisfy. As in the ODE case, the minimal requirement on N is to be Lipschitz-continuous in X . However,

for abstract ODEs, the above condition is hardly satisfied in practice. For example, if $X = L^2(0, 1)$ and $N(u, t) = u^2$, it is not even guaranteed that $u^2 \in L^2(0, 1)$. Therefore, we need to consider N like an operator defined on a smaller space V , such that $\mathcal{D}(L) \subset V \subset X$. Thus, the main assumption imposed on the nonlinearity $N : V \times [0, T] \rightarrow X$ is to be *locally Lipschitz-continuous* in a strip along the exact solution u . Therefore, there exist a real number $L(R, T)$, such that

$$\|N(v, t) - N(w, t)\|_X \leq L\|v - w\|_V$$

for all $t \in [0, T]$ and $\max(\|v - u(t)\|_V, \|w - u(t)\|_V) \leq R$, where R is the radius of the strip.

In [29, 30], the space V is chosen to be the fractional power space $X_\alpha = \mathcal{D}(\tilde{L}^\alpha)$, $0 \leq \alpha < 1$. Thus V is linear Banach space with norm $\|v\|_V = \|\tilde{L}^\alpha v\|_X$. The value of α depends from the underlying space X and from the type and dimensionality of the problem. It can be determined by the following Lemma, which relies on the classical Sobolev embedding theorem (see[54]).

Lemma 1.2.8. *Let $1 \leq p < \infty$ and $\Omega \subset \mathbb{R}^d$ be an open and bounded set with smooth boundary. Let L be sectorial in $L^p(\Omega)$ with domain $\mathcal{D}(L) \subset W^{m,p}(\Omega)$ for some $m \geq 1$. Then for $0 \leq \alpha \leq 1$*

$$V = X_\alpha \subset C^\nu(\bar{\Omega}) \subset L^\infty, \quad 0 \leq \nu < m\alpha - \frac{d}{p}$$

with continuous embeddings.

From Lemma 1.2.8, it follows that for second-order parabolic problems on $L^2(\Omega)$ i.e. $m = 2$ and $p = 2$, $V \subset L^\infty$ for $d/4 < \alpha < 1$. Thus it is clear that for three dimensional problems, we have to work with stronger norms than in the one and two dimensional cases.

We next observe that the definition of the space $V = X_\alpha$ is related with the stability bounds (1.2.20) for the exponential and the related operators. To illustrate this connection, we represent the exact solution of (1.2.17) by the variation of constants formulae

$$\underbrace{u(t)}_V = e^{tL}u_0 + \int_0^t \underbrace{e^{(t-\tau)L}}_{V \leftarrow X} \underbrace{N(u(\tau), \tau)}_X d\tau. \quad (1.2.21)$$

Since $u(t) \in V$, it is clear that we need to consider e^{tL} like an operator from X to V . From (1.2.20) and the definition of the operator norm, for $0 < t \leq T$, we have

$$\begin{aligned} \|e^{tL}\|_{V \leftarrow X} &= \sup_{\substack{w \in X \\ w \neq 0}} \frac{\|e^{tL}w\|_V}{\|w\|_X} = \sup_{\substack{w \in X \\ w \neq 0}} \frac{\|\tilde{L}^\alpha e^{tL}w\|_X}{\|w\|_X} \\ &\leq \sup_{\substack{w \in X \\ w \neq 0}} \frac{\|\tilde{L}^\alpha e^{tL}\|_{X \leftarrow X} \|w\|_X}{\|w\|_X} \leq Ct^{-\alpha}. \end{aligned} \quad (1.2.22)$$

Let us now define the operators $\phi^{[i]}$ for $i = 1, 2, \dots$ by the formula

$$\phi^{[i]}(tL) = t^{-i} \int_0^t e^{(t-\tau)L} \frac{\tau^{i-1}}{(i-1)!} d\tau. \quad (1.2.23)$$

Note that $\phi^{[i]}$ are bounded operators in X and that when L is a matrix, the above formulas reproduce exactly the ETD $\phi^{[i]}$ functions given in (1.1.6).

The stability estimate (1.2.22) can be easily extended to all operators $\phi^{[i]}$

$$\begin{aligned} \|\phi^{[i]}(tL)\|_{V \leftarrow X} &\leq t^{-i} \int_0^t \|e^{(t-\tau)L}\|_{V \leftarrow X} \frac{\tau^{i-1}}{(i-1)!} d\tau \\ &\leq Ct^{-i} \int_0^t (t-\tau)^{-\alpha} \tau^{i-1} d\tau \leq Ct^{-\alpha}. \end{aligned} \quad (1.2.24)$$

The last assumption which we impose on the problem (1.2.17) is that it possesses a sufficiently smooth solution $u : [0, T] \rightarrow V$ with derivatives in V . In addition we suppose that $N : V \times [0, T] \rightarrow X$ is sufficiently often Fréchet differentiable in a strip along the exact solution. The above assumption, insures that the composition map

$$F : [0, T] \rightarrow X : t \rightarrow F(t) = N(u(t), t)$$

is a smooth map and therefore F admits Taylor series expansion.

In the next two subsections, we consider respectively explicit and implicit exponential integrators for solving semilinear parabolic PDEs.

1.2.5.2 Explicit methods

Under the assumptions given in the previous subsection, the convergence properties of the explicit exponential Runge–Kutta methods presented in Subsection 1.2.1 are studied in [29].

The main idea behind the analysis presented there is to derive bounds for the error recursion between the exact solution represented by the variation of constants formulae (1.2.21) and the numerical solution given by the scheme (1.2.4). Because of the α -dependence in (1.2.22) and (1.2.24), the error bounds depend from the size of the parameter α . Therefore, the number of order conditions for a method to be of a given order p also depends from α . The simplest possible case is when $\alpha = 0$ ($V = X$). In Table 1.14 we list the stiff order conditions derived in [29], which guarantee that an s -stage explicit exponential Runge–Kutta method (1.2.4) is of order p (for $p \leq 4$), in the case when $\alpha = 0$ and the ETD $\phi^{[i]}$ functions (1.2.23) are used. The operators J and K , which appear in conditions number 5,7,8 and 9 are arbitrary bounded operators in X .

The order conditions from Table 1.14 allow us to analyze the stiff order of the explicit exponential Runge–Kutta methods presented so far. It is shown in [29], that the third order method of Cox–Matthews [14] as well as the method ETD2RK3 (see Subsection 1.2.4) suffer from order reduction down to order two in the worst case. The method ETD2CF3 given in Subsection 1.2.4 has a full stiff order three, in the case when $\alpha = 0$. Similarly, the fourth

No.	order	order condition
1	1	$\sum_{i=1}^s b_i(hL) = \phi^{[1]}(hL)$
2	2	$\sum_{i=2}^s b_i(hL)c_i = \phi^{[2]}(hL)$
3	2	$\sum_{j=1}^{i-1} a_{ij}(hL) = c_i \phi^{[1]}(c_i hL)$
4	3	$\sum_{i=2}^s b_i(hL)c_i^2 = 2\phi^{[3]}(hL)$
5	3	$\sum_{i=2}^s b_i(hL)J\left(\phi^{[2]}(c_i hL)c_i^2 - \sum_{j=2}^{i-1} a_{ij}(hL)c_j\right) = 0$
6	4	$\sum_{i=2}^s b_i(hL)c_i^3 = 6\phi^{[4]}(hL)$
7	4	$\sum_{i=2}^s b_i(hL)J\left(\phi^{[3]}(c_i hL)c_i^3 - \frac{1}{2}\sum_{j=2}^{i-1} a_{ij}(hL)c_j^2\right) = 0$
8	4	$\sum_{i=2}^s b_i(hL)J\sum_{j=2}^{i-1} a_{ij}(hL)J\left(\phi^{[2]}(c_j hL)c_j^2 - \sum_{k=2}^{j-1} a_{jk}(hL)c_k\right) = 0$
9	4	$\sum_{i=2}^s b_i(hL)c_i K\left(\phi^{[2]}(c_i hL)c_i^2 - \sum_{j=2}^{i-1} a_{ij}(hL)c_j\right) = 0$

Table 1.14: Stiff order conditions for explicit exponential Runge–Kutta methods for $\alpha = 0$.

order method of Cox–Matthews [14] has only stiff order two and the fourth order method of Krogstad (see Subsection 1.2.4) has a stiff order three in the worst case. For each of the above methods, higher order (up to the classical nonstiff order) is possible, if additional smoothness conditions of the problem are satisfied. In general, it can be shown that it is not possible to construct stiff fourth order exponential Runge–Kutta method with only four stages [29]. In Table 1.15 we present the fourth order, five stages method of Hochbruck–Ostermann [29], which satisfies all the order conditions from Table 1.14 except the condition number 7. It is satisfied only in a weak form, that is with $b_i(0)$ instead of $b_i(hL)$.

All exponential integrators which we have considered so far were based on the explicit idea. In some sense, it seems unnecessary to consider implicit exponential integrator, since the hope is to overcome the stiffness by using the exponential and the related functions $\phi^{[i]}$ in the format of the method. However, we should keep in mind that for implicit exponential

$$\left[\begin{array}{ccccc|c} 0 & 0 & 0 & 0 & 0 & I \\ \frac{1}{2}\phi^{[1]} & 0 & 0 & 0 & 0 & e^{\frac{1}{2}hL} \\ \frac{1}{2}\phi^{[1]} - \phi^{[2]} & \phi^{[2]} & 0 & 0 & 0 & e^{\frac{1}{2}hL} \\ \phi^{[1]} - 2\phi^{[2]} & \phi^{[2]} & \phi^{[2]} & 0 & 0 & e^{hL} \\ \frac{1}{2}\phi^{[1]} - \frac{1}{4}\phi^{[2]} - a_{52}\left(\frac{hL}{2}\right) & a_{52}\left(\frac{hL}{2}\right) & a_{52}\left(\frac{hL}{2}\right) & \frac{1}{4}\phi^{[2]} - a_{52}\left(\frac{hL}{2}\right) & 0 & e^{\frac{1}{2}hL} \\ \hline \phi^{[1]} - 3\phi^{[2]} + 4\phi^{[3]} & 0 & 0 & -\phi^{[2]} + 4\phi^{[3]} & 4\phi^{[2]} - 8\phi^{[3]} & e^{hL} \end{array} \right],$$

where $a_{52}\left(\frac{hL}{2}\right) = \frac{1}{2}\phi^{[2]}\left(\frac{hL}{2}\right) - \phi^{[3]}(hL) + \frac{1}{4}\phi^{[2]}(hL) - \frac{1}{2}\phi^{[3]}\left(\frac{hL}{2}\right)$

Table 1.15: Fourth order method of Hochbruck–Ostermann with five stages.

integrators, we can solve the arising nonlinear equations by using fixed-point iteration. What we gain by using implicit multistage integrators is the ability to increase the stage order of the method. Motivated from this observation, a special class of implicit exponential integrators of collocation type, for solving parabolic problems, is studied in [30]. Next, we consider this class of methods.

1.2.5.3 Implicit collocation methods

The main idea behind the exponential collocation integrators [30], is to replace the function $N(u, t)$ in the variation of constants formulae (1.1.4)

$$u(t_{n-1} + h) = e^{hL}u_{n-1} + \int_0^h e^{(h-\tau)L}N(u(t_{n-1} + \tau), t_{n-1} + \tau)d\tau$$

by polynomial approximation $p_{n-1}(\tau)$ of collocation type.

Let c_1, c_2, \dots, c_s be non-confluent collocation nodes in the interval $[0, 1]$ and $U_i \approx u(t_{n-1} + c_i h)$. If $p_{n-1}(\tau)$ is the unique collocation polynomial of degree $s - 1$, which satisfies the conditions $p_{n-1}(c_i h) = N(U_i, t_{n-1} + c_i h)$ for $i = 1, 2, \dots, s$, then we can define the numerical solution u_n of (1.2.17) at time $t_n = t_{n-1} + h$ by the formulas

$$\begin{aligned} u_n &= e^{hL}u_{n-1} + \int_0^h e^{(h-\tau)L}p_{n-1}(\tau)d\tau, \\ U_i &= e^{c_i hL}u_{n-1} + \int_0^{c_i h} e^{(c_i h-\tau)L}p_{n-1}(\tau)d\tau. \end{aligned} \tag{1.2.25}$$

Since $p_{n-1}(\tau)$ is a polynomial of τ with coefficients that are linear combinations of the values $N(U_i, t_{n-1} + c_i h)$, calculating exactly the integrals form (1.2.25), we obtain

$$\int_0^{c_i h} e^{(c_i h-\tau)L}p_{n-1}(\tau)d\tau = h \sum_{j=1}^s a_{ij}(c_i hL)N(U_j, t_{n-1} + c_j h),$$

$$\int_0^h e^{(h-\tau)L} p_{n-1}(\tau) d\tau = h \sum_{i=1}^s b_i(hL) N(U_i, t_{n-1} + c_i h),$$

where the coefficients a_{ij} and b_i are linear combinations of the first s , $\phi^{[i]}$ functions (1.2.23). Thus, it is clear that the general format of an exponential integrator of collocation type is just a special case of the exponential Runge–Kutta methods introduced in Subsection 1.2.1. The advantage of the above approach is that it does not require any order theory because of the high stage order. Therefore the construction of new integrators is easy, since there is no need to solve complicated order conditions. Of course all this comes on the price of working with a very restrictive class of methods. The main advantage of the exponential integrators of collocation type comes from the fact that they are implicit, which allows to build up the stage order of the method, and at the same time the nonlinear equations which arise in the format of the method can be simply solved using fixed-point iterations.

In [30], the convergence properties of the exponential integrators of collocation type, applied to semilinear parabolic PDEs, are analyzed. It is shown there, that the methods converge at least with their stage order. Higher and even fractional order of convergence is possible if additional temporal and spatial regularity are required. For problems with periodic boundary conditions classical (nonstiff) order can be obtained.

The above result suggests that the ideal integrators for solving stiff problems will be explicit exponential integrators with higher stage order. Thus, we naturally arrive to the idea of using *general linear methods* [8, 69] like underlying methods in the construction of exponential integrators. In the next section we introduce the main concept behind general linear methods and show how they can be extended to the exponential setting.

1.3 General linear methods and exponential integrators

In the previous two sections we have considered exponential integrators based on the two main classes for solving ordinary differential equations: linear multivalued (Section 1.1) and multistage (Section 1.2) methods. General linear methods (GLMs) were introduced in [7] as a unifying framework for the traditional methods to study the properties of consistency, stability and convergence. The extremely general nature of this methods allows new methods with clear advantages over the traditional methods to be constructed. We next present the main idea behind general linear methods and introduce some notations, which are used in the representation of every exponential integrator included in this thesis.

1.3.1 Formulation of the methods

For simplicity of the presentation, we again represent the original semilinear problem (1.1.1) in the following autonomous form

$$u' = Lu + N(u(t)) = f(u(t)), \quad u(t_0) = u_0, \quad f(u(t)) : \mathbb{R}^d \rightarrow \mathbb{R}^d. \quad (1.3.1)$$

Following [8, Chapter 5], for the numerical solution of (1.3.1), we consider a method in which a collection of vectors forms the input at the beginning of a step, and a similar collection is passed on as output from the current step and as input into the next step. Thus the method is a multivalue method. In addition, as for Runge–Kutta methods, it is assumed that during the computations that constitute the step of the method, s approximations to the solution at points near the current time step are evaluated. Therefore the method is also a multistage method. Methods which are both multistage and multivalued are known as general linear methods.

Assume that at the beginning of step number n , r quantities

$$u_1^{[n-1]}, u_2^{[n-1]}, \dots, u_r^{[n-1]},$$

are available from approximations computed in the previous step number $n - 1$. The corresponding quantities evaluated during the step number n are denoted by

$$u_1^{[n]}, u_2^{[n]}, \dots, u_r^{[n]}.$$

Let the internal stage values of step number n are denoted by

$$U_1, U_2, \dots, U_s$$

and the derivatives evaluated at this step are denoted by

$$f(U_1), f(U_2), \dots, f(U_s).$$

If h represents the stepsize, then the quantities imported into and evaluated in step number n are related by the equations

$$\begin{aligned} U_i &= \sum_{j=1}^s a_{ij} h f(U_j) + \sum_{j=1}^r d_{ij} u_j^{[n-1]}, \quad i = 1, 2, \dots, s, \\ u_i^{[n]} &= \sum_{j=1}^s b_{ij} h f(U_j) + \sum_{j=1}^r v_{ij} u_j^{[n-1]}, \quad i = 1, 2, \dots, r, \end{aligned} \tag{1.3.2}$$

where $A = (a_{ij})$, $B = (b_{ij})$, $D = (d_{ij})$, $V = (v_{ij})$ are coefficients of the method. Introducing the vector notations

$$U = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_s \end{bmatrix}, \quad f(U) = \begin{bmatrix} f(U_1) \\ f(U_2) \\ \vdots \\ f(U_s) \end{bmatrix}, \quad u^{[n-1]} = \begin{bmatrix} u_1^{[n-1]} \\ u_2^{[n-1]} \\ \vdots \\ u_r^{[n-1]} \end{bmatrix}, \quad u^{[n]} = \begin{bmatrix} u_1^{[n]} \\ u_2^{[n]} \\ \vdots \\ u_r^{[n]} \end{bmatrix},$$

allows us to rewrite the method (1.3.2) in the following more compact form

$$\begin{bmatrix} U \\ u^{[n]} \end{bmatrix} = \begin{bmatrix} A \otimes I_d & D \otimes I_d \\ B \otimes I_d & V \otimes I_d \end{bmatrix} \begin{bmatrix} h f(U) \\ u^{[n-1]} \end{bmatrix},$$

where \otimes is the Kronecker product and I_d is the $d \times d$ identity matrix. Usually, in order to simplify the notations, the Kronecker product with I_d is omitted. Thus the coefficients of every general linear method, given by the matrices A, B, D, V , can be written together as the following partitioned $(s + r) \times (s + r)$ matrix

$$M = \left[\begin{array}{c|c} A & D \\ \hline B & V \end{array} \right]. \quad (1.3.3)$$

Note that the matrix A from (1.3.3) is similar to the \mathcal{A} matrix in the Runge–Kutta methods and determines the implementation cost of the method. The quantities $u^{[n]}$, which are passed from step to step, can have a very general nature. Common choices in linear multistep methods, are approximations to the solution and the derivatives at various previous points, backward difference approximations or approximations to the Nordsieck vector. Like for Runge–Kutta methods, the stage values U of GLMs are approximations to the solution at points near the current time step. That is $U_i \approx u(t_n + c_i h)$, where usually $0 \leq c_i \leq 1$, for $i = 1, 2, \dots, s$. The vector

$$c = [c_1, c_2, \dots, c_s]^T,$$

is called the vector of abscissae.

Examples of GLMs

Next, we give few examples showing how some well know traditional methods can be alternatively represented like general linear methods. Let us first consider k -step linear multistep methods of Adams type

$$u_n = u_{n-1} + h \sum_{i=0}^k \beta_i f(u_{n-i}). \quad (1.3.4)$$

Natural way of rewriting (1.3.4) as GLM is to choose the number of the internal stages to be only one, corresponding to approximation of $u(t_n)$, and the $k + 1$ quantities passed from step to step to be of the form

$$u^{[n-1]} = \begin{bmatrix} u_{n-1} \\ hf(u_{n-1}) \\ hf(u_{n-2}) \\ \vdots \\ hf(u_{n-k}) \end{bmatrix}.$$

Thus, (1.3.4) can be represented in the following general linear form

$$\begin{bmatrix} U_1 \\ \hline u_n \\ hf(U_1) \\ hf(u_{n-1}) \\ \vdots \\ hf(u_{n-k-1}) \end{bmatrix} = \begin{bmatrix} \beta_0 & 1 & \beta_1 & \cdots & \beta_{k-1} & \beta_k \\ \hline \beta_0 & 1 & \beta_1 & \cdots & \beta_{k-1} & \beta_k \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} hf(U_1) \\ \hline u_{n-1} \\ hf(u_{n-1}) \\ hf(u_{n-2}) \\ \vdots \\ hf(u_{n-k}) \end{bmatrix}.$$

Similarly, backward differentiation formulae (BDF) methods

$$u_n = \sum_{i=1}^k \alpha_i u_{n-i} + h\beta_0 f(u_n),$$

are represented as

$$\begin{bmatrix} U_1 \\ \hline u_n \\ u_{n-1} \\ u_{n-2} \\ \vdots \\ u_{n-k-1} \end{bmatrix} = \begin{bmatrix} \beta_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{k-1} & \alpha_k \\ \hline \beta_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{k-1} & \alpha_k \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} hf(U_1) \\ \hline u_{n-1} \\ u_{n-2} \\ u_{n-3} \\ \vdots \\ u_{n-k} \end{bmatrix}.$$

As far as Runge–Kutta methods are concern, it is usually convenient to represent them as general linear methods which pass only one quantity from step to step. That is an approximation to $u(t_{n-1})$. In this way, we can identify the matrices A and B form (1.3.3) with the A matrix and the row vector b^T of the Runge–Kutta method respectively. Choosing $D = e$ and $V = 1$, we can easily see that the classical fourth order Runge–Kutta method

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}, \quad (1.3.5)$$

can be written as

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \hline U_4 \\ u_n \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & 0 & 1 \\ 0 & \frac{1}{2} & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ \hline \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} & 1 \end{bmatrix} \begin{bmatrix} hf(U_1) \\ hf(U_2) \\ hf(U_3) \\ hf(U_4) \\ \hline u_{n-1} \end{bmatrix}.$$

We mention also that it is not always appropriate to represent a Runge–Kutta method like a general liner method with $r = 1$. Example is the Lobatto IIIA method

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{5}{24} & \frac{1}{3} & -\frac{1}{24} \\ \frac{1}{2} & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}.$$

It is a method which has the so called *FSAL property*. This means that the *First* stage of the current step is the *Same As* the *Last* stage of the previous step. This special property allows us to eliminate the first stage of the method by passing one more quantity from step to step. Thus the method can be represented in the following general linear form

$$\begin{bmatrix} U_1 \\ U_2 \\ \hline u_n \\ hf(U_2) \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & -\frac{1}{24} & 1 & \frac{5}{24} \\ \frac{2}{3} & \frac{1}{6} & 1 & \frac{1}{6} \\ \hline \frac{2}{3} & \frac{1}{6} & 1 & \frac{1}{6} \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} hf(U_1) \\ hf(U_2) \\ \hline u_{n-1} \\ hf(u_{n-1}) \end{bmatrix}.$$

1.3.2 Construction of practical GLMs

We first introduce the concepts of pre-consistency, consistency, stability and convergence of general linear methods. The pre-consistency and consistency conditions for a GLM insure that the method can solve exactly the trivial differential equations $u' = 0$ and $u' = 1$. If the pre-consistency vector \bar{u} and the consistency vector \bar{v} are determined by

$$\begin{aligned} u^{[n-1]} &= \bar{u}u(t_{n-1}) + \bar{v}hu'(t_{n-1}) + \mathcal{O}(h^2), \\ u^{[n]} &= \bar{u}u(t_n) + \bar{v}hu'(t_n) + \mathcal{O}(h^2), \end{aligned}$$

then the pre-consistency conditions are

$$D\bar{u} = e, \quad V\bar{u} = \bar{u},$$

and the consistency conditions are

$$Ae + D\bar{v} = c, \quad Be + V\bar{v} = \bar{u} + \bar{v},$$

where $e = [1, 1, \dots, 1]^T \in \mathbb{R}^s$. Stability of GLM concerns the boundedness of the numerical solution when the method is applied to the trivial differential equation $u' = 0$. Thus the general linear method M , defined by (1.3.3), is *stable* if there exist a constant C such that for all $n = 1, 2, \dots$, $\|V^n\| \leq C$. The method M is *convergent* if there exist a non-zero vector $\bar{u} \in \mathbb{R}^r$ such that if $u^{[0]} = \bar{u}u(t_0) + \mathcal{O}(h)$, then $u^{[n]} = \bar{u}u(t_0 + nh) + \mathcal{O}(h)$ for all n , given that nh is bounded. The fundamental result that consistency and stability are necessary and sufficient conditions for convergence of a general linear method is proven in [7].

The order of accuracy of a general linear method is defined relative to a starting procedure S . It is used to produce the initial vector $u^{[0]}$ from the given initial value u_0 . In general the starting procedure can be represented by

$$\begin{aligned}\bar{U} &= S_{11}hf(\bar{U}) + S_{12}u_0, \\ u^{[0]} &= S_{21}hf(\bar{U}) + S_{22}u_0,\end{aligned}$$

where the vectors \bar{U} and $f(\bar{U})$ are

$$\bar{U} = \begin{bmatrix} \bar{U}_1 \\ \bar{U}_2 \\ \vdots \\ \bar{U}_{\bar{s}} \end{bmatrix}, \quad f(\bar{U}) = \begin{bmatrix} f(\bar{U}_1) \\ f(\bar{U}_2) \\ \vdots \\ f(\bar{U}_{\bar{s}}) \end{bmatrix},$$

and $f(\bar{U}_1), f(\bar{U}_2), \dots, f(\bar{U}_{\bar{s}})$ are the derivatives at the internal stages $\bar{U}_1, \bar{U}_2, \dots, \bar{U}_{\bar{s}}$. Note that the number of stages \bar{s} , required to compute the r components of $u^{[0]}$, can be different from the number of the stages s used in the method. From the pre-consistency conditions follows, that $S_{12} = e$ and $S_{22} = \bar{u}$. Similarly to (1.3.3), the starting procedure can be also represented by a partitioned $(\bar{s} + r) \times (\bar{s} + 1)$ matrix

$$S = \left[\begin{array}{c|c} S_{11} & S_{12} \\ \hline S_{21} & S_{22} \end{array} \right].$$

If E represents the shift operator, that shifts the exact solution from t_{n-1} to t_n , then the general linear method M has order of accuracy p if $M \circ S - S \circ E = \mathcal{O}(h^{p+1})$, where $M \circ S$ denotes the combined effect of applying the starting procedure S followed by a step of the method M . The meaning of $S \circ E$ is defined in a similar way.

Until recently, very few general linear methods which are significantly different from the traditional methods have been developed. This is mainly due to the very general structure of these methods, which results in a complicated order theory. A reasonable way to advance in the construction of new competitive methods is to introduce some initial assumptions on the structure of the methods. In this way the originally large class of GLMs is limited to a smaller subclass where practical methods are likely to exist.

Like we saw in Subsection 1.2.5.3, it is important to base the construction of exponential integrators on methods which have higher stage order. Thus a desirable property of the

underlying method is to have stage order equal to the overall order p of the method. Imposing this assumption simplifies the order construction of GLMs and provides asymptotically correct error estimates, which are useful in variable stepsize, variable order implementations. To overcome difficulties with changing the stepsize, it is also convenient to require the quantities passed from step to step to be approximations of the Nordsieck vector i.e.

$$u^{[n]} \approx \begin{bmatrix} u(t_n) \\ hu'(t_n) \\ \vdots \\ \frac{h^p}{p!}u^{(p)}(t_n) \end{bmatrix},$$

with $r = p+1$. Under the above assumptions the order conditions for general linear methods become much simpler. It can be shown (see [8, Chapter 5], [69, Chapter 3]) that necessary conditions for a method to have stage order and order p are

$$\begin{aligned} D &= C - ACK, \\ V &= E - BCK, \end{aligned}$$

where the matrices $C \in \mathbb{R}^{s \times p+1}$, $K \in \mathbb{R}^{p+1 \times p+1}$ and $E = \exp(K)$ are given by

$$C = \begin{bmatrix} e & c & \frac{c^2}{2!} & \cdots & \frac{c^{p-1}}{(p-1)!} & \frac{c^p}{p!} \end{bmatrix}, \quad K = \begin{bmatrix} 0 & e_1 & e_2 & \cdots & e_{(p-1)} & e_p \end{bmatrix}.$$

Other important property which we would like a GLM to have is that its stability region should be identical to the stability region of the corresponding Runge–Kutta methods. Sufficient conditions for a general linear method to have Runge–Kutta stability are

$$\begin{aligned} BA &\equiv XB, \\ BD &\equiv XV - VX, \end{aligned}$$

where \equiv means two matrices are equivalent except for their first rows, X is a doubly companion matrix and the spectrum of V is $\{0, 1\}$ (see [69, Chapter 3]). General linear methods satisfying the above conditions are known to have a property called Inherent Runge–Kutta Stability (IRKS). Recently practical general linear methods with IRKS have been constructed in [9, 10, 69].

1.3.3 Exponential general linear methods

Let us now consider how general linear methods can also be extended to the exponential setting. Motivated from (1.2.3) and (1.3.2), for the solution of the semilinear problem

(1.1.1), we consider the following unified format of exponential integrators

$$\begin{aligned} U_i &= \sum_{j=1}^s \sum_{l=1}^m \alpha_{ij}^{[l]} \phi^{[l]}(c_i)(hL) hN(U_j) + \sum_{j=1}^r \sum_{l=1}^m \delta_{ij}^{[l]} \phi^{[l]}(c_i)(hL) u_j^{[n-1]}, \\ u_i^{[n]} &= \sum_{j=1}^s \sum_{l=1}^m \beta_{ij}^{[l]} \phi^{[l]}(1)(hL) hN(U_j) + \sum_{j=1}^r \sum_{l=1}^m \nu_{ij}^{[l]} \phi^{[l]}(1)(hL) u_j^{[n-1]}. \end{aligned} \quad (1.3.6)$$

Similarly, to the traditional GLMs, we can represent (1.3.6) in the following matrix form

$$\begin{bmatrix} U \\ u^{[n]} \end{bmatrix} = \begin{bmatrix} A(\phi) & D(\phi) \\ B(\phi) & V(\phi) \end{bmatrix} \begin{bmatrix} hN(U) \\ u^{[n-1]} \end{bmatrix},$$

where each of the coefficient matrices $A(\phi), B(\phi), D(\phi), V(\phi)$ has entries which are linear combinations of the $\phi^{[l]}$ functions. Therefore, the matrix

$$M(\phi) = \begin{bmatrix} A(\phi) & D(\phi) \\ B(\phi) & V(\phi) \end{bmatrix}, \quad (1.3.7)$$

has also entries which are linear combinations of the $\phi^{[l]}$ functions. All exponential integrators presented in this thesis are given exactly in the form (1.3.7). In addition the following convention is used: The argument of $\phi^{[l]}$ is always dropped when it is evaluated at the corresponding abscissae value.

According to our knowledge, the first exponential integrators which fit in the format (1.3.6), were constructed in [36, Article 5]. We discuss these methods in Subsection 2.4.2. However, there are several interesting properties regarding the *Generalized Integrating Factor Runge–Kutta* (GIF/RK) methods of Krogstad which we would like to point out here. First of all they are exponential general linear methods which pass the quantities $u_n, hN_{n-1}, hN_{n-2}, \dots$ from step to step. Secondly they are examples of methods, where other than the traditional IF and ETD $\phi^{[i]}$ functions are used (see Table 2.4 and Table 2.5). The improved accuracy of GIF/RK methods is due to their higher stage order (see [46]). However, as it was pointed out in [36, Article 5], it seems that the accuracy comes at the price of stability. Thus, extending the idea of general linear methods with IRKS property to the exponential settings, might be a reasonable way to overcome this difficulty. Finally, we mark that the generalized integrating factor Runge–Kutta methods are also Lie group methods on the manifold. We consider the framework of Lie group methods and how it is related with the construction of exponential integrators in the next chapter.

Chapter 2

Exponential Integrators and Lie Group Methods

In this chapter we discuss special types of exponential integrators, which arise from the framework of Lie group methods on manifolds. The main idea behind Lie group methods and *Geometric Integration* in general, is to construct numerical integrators which preserve certain qualitatively properties of the exact flow of a differential equation. A classical example is a differential equation evolving on a sphere. In many cases it is important to construct a numerical integrator which produces an approximation to the exact solution which also belongs to the sphere.

When the differential equation evolves on a linear space, it is easy to construct an integrator which stays on the linear space. The challenge now is how to define the basic motions on the manifold in such a way that they provide a good approximation to the flow of the original vector field. In this case the theory of the Lie group integrators provides us, through the freedom in choice of a *group action*, a suitable framework to work with. All the exponential integrators presented in this chapter are motivated from the above observation.

The chapter is organized as follows: we survey the basic theory involved in Section 2.1. In Section 2.2 we present different Lie group integrators and comment on their numerical implementation. Next, in Section 2.3 we discuss the importance of the Lie group action, and present different ways how to define it (see also [Article 1, p.91]). Finally in Section 2.4, based on the main Lie group methods given in Section 2.2, we derive the corresponding exponential integrators for solving the semilinear problem (1.1.1).

2.1 Background theory

Most of the theory presented in this section follows the expositions in [32, 36, 50] and gives the necessary basic background needed to construct Lie group methods which evolve on a manifold. We refer to the monographs [1, 44] for further information concerning manifolds, Lie groups and Lie algebras and to the monograph [23] for numerical treatment

and introduction to geometric integration in general.

We start with introducing the concept of a manifold acted upon by Lie group, which provides us with abstract definition of the domain where the differential equation evolves. Let

$$u' = f(u(t)), \quad u(t_0) = u_0, \quad (2.1.1)$$

be a differential equation defined on d -dimensional topological space \mathcal{M} which in a small neighborhood of any point “looks like” \mathbb{R}^d but globally, typically has a different geometry. We will call such a space \mathcal{M} a *d-dimensional manifold*. Any d -dimensional manifold can be represented as a d -dimensional surface embedded in \mathbb{R}^N for some $N \geq d$. In other words \mathcal{M} is defined as

$$\mathcal{M} = \{x \in \mathbb{R}^N : g(x) = 0\},$$

where $g : \mathbb{R}^N \rightarrow \mathbb{R}^{N-d}$ is differentiable and $g'(x)$ has a full rank for $x \in \mathcal{M}$. We note that this is a very concrete definition of a manifold, but is however sufficient for our purposes. In fact all the exponential integrators presented in Section 2.4 simply evolve on a manifold $\mathcal{M} \equiv \mathbb{R}^d$. A more general definition of a manifold, based on *coordinate charts*, can also be given (see [1, 36, 50]).

We now give a definition of a Lie group, its tangent space, and discuss different maps related with them.

Definition 2.1.1. A *Lie group* is a differential manifold \mathcal{G} equipped with a group product $\star : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ satisfying

$$\begin{aligned} g \star (k \star l) &= (g \star k) \star l \quad \forall g, k, l \in \mathcal{G}, & (\text{associativity}) \\ \exists e \in \mathcal{G} \text{ such that } e \star g &= g \star e = g, & (\text{identity element}) \\ \forall g \in \mathcal{G} \exists g^{-1} \in \mathcal{G} \text{ such that } g^{-1}g &= e, & (\text{inverse}) \\ \text{the maps } (g, k) \rightarrow g \star k \text{ and } g \rightarrow g^{-1} & & (\text{smoothness}) \\ & \text{are smooth functions.} \end{aligned}$$

Definition 2.1.2. A *Lie algebra* is a vector space \mathfrak{g} equipped with a bilinear bracket $[\cdot, \cdot] : \mathfrak{g} \times \mathfrak{g} \rightarrow \mathfrak{g}$ satisfying

$$\begin{aligned} [\Theta_1, \Theta_2] &= -[\Theta_2, \Theta_1], & (\text{skew-symmetry}) \\ [\Theta_1, [\Theta_2, \Theta_3]] + [\Theta_2, [\Theta_3, \Theta_1]] + [\Theta_3, [\Theta_1, \Theta_2]] &= 0, & (\text{Jacobi identity}) \end{aligned}$$

for all $\Theta_1, \Theta_2, \Theta_3 \in \mathfrak{g}$.

If we fix the first argument in the bracket $[\cdot, \cdot]$ to be the point $\hat{\Theta} \in \mathfrak{g}$ then the map $\text{ad}_{\hat{\Theta}} : \mathfrak{g} \rightarrow \mathfrak{g}$ given by $\text{ad}_{\hat{\Theta}}(\Theta) = [\hat{\Theta}, \Theta]$ is linear. Powers of $\text{ad}_{\hat{\Theta}}$ are defined recursively as

$$\begin{aligned} \text{ad}_{\hat{\Theta}}^0(\Theta) &= \Theta, \\ \text{ad}_{\hat{\Theta}}^1(\Theta) &= [\hat{\Theta}, \Theta], \\ \text{ad}_{\hat{\Theta}}^k(\Theta) &= \text{ad}_{\hat{\Theta}}(\text{ad}_{\hat{\Theta}}^{k-1}\Theta) = [\hat{\Theta}, [\dots, [\hat{\Theta}, \Theta]]], \quad \text{for } k > 1. \end{aligned}$$

Let us denote the set of all right invariant vector fields on \mathcal{G} by $\mathfrak{X}(\mathcal{G})$. It can be shown that $\mathfrak{X}(\mathcal{G})$ has a structure of a Lie algebra with a bracket given by the (minus) *Jacobi bracket* (see [36, 50]). Let $T_e\mathcal{G}$ be the tangent space of \mathcal{G} at the identity element e . We define the product $\odot : T_e\mathcal{G} \times \mathcal{G} \rightarrow T\mathcal{G}$ by

$$\Theta \odot g = \left. \frac{d}{dt} \right|_{t=0} \gamma(t) \star g,$$

where $\gamma(t)$ is a smooth curve in \mathcal{G} such that $\gamma(0) = e$ and $\gamma'(0) = \Theta$. If we fix the first argument $\Theta \in T_e\mathcal{G}$ in the product $\Theta \odot g$, we get a vector field $\mathbf{f}_\Theta(\cdot) = \Theta \odot (\cdot)$ on \mathcal{G} . In this way the map $\Theta \rightarrow \mathbf{f}_\Theta$ is a homomorphism between $T_e\mathcal{G}$ and $\mathfrak{X}(\mathcal{G})$. This automatically implies that the set $T_e\mathcal{G}$ also has the structure of a Lie algebra. From now on we use the symbol \mathfrak{g} to denote the Lie algebra $T_e\mathcal{G}$ and say that \mathfrak{g} is a Lie algebra of a Lie group \mathcal{G} . The bracket in this case is defined by

$$[\Theta_1, \Theta_2] = \left. \frac{\partial^2}{\partial s \partial t} \right|_{t=s=0} \gamma_1(s) \star \gamma_2(t) \star \gamma_1(s)^{-1},$$

where $\gamma_1(s)$ and $\gamma_2(t)$ are smooth curves in \mathcal{G} such that $\gamma_1(0) = \gamma_2(0) = e$ and $\gamma_1'(0) = \Theta_1$, $\gamma_2'(0) = \Theta_2$. In the case of a matrix Lie algebra \mathfrak{g} , the bracket $[\Theta_1, \Theta_2]$ is simply the matrix commutator $\Theta_1\Theta_2 - \Theta_2\Theta_1$.

Now we define the *exponential* map which gives an important connection between a Lie group and its Lie algebra. It is a local diffeomorphism (for finite dimensions) from a neighborhood of $0 \in \mathfrak{g}$ onto a neighborhood of $e \in \mathcal{G}$.

Definition 2.1.3. Let \mathcal{G} be a Lie group and \mathfrak{g} its Lie algebra. The *exponential* map $\text{Exp} : \mathfrak{g} \rightarrow \mathcal{G}$ is defined as $\text{Exp}(\Theta) = \gamma(1)$, where $\gamma(t) \in \mathcal{G}$ satisfies the differential equation

$$\gamma(t)' = \mathbf{f}_\Theta(\gamma(t)), \quad \gamma(0) = e.$$

In the case of a matrix Lie algebra \mathfrak{g} , the exponential map is simply the matrix exponential

$$e^\Theta = \sum_{k=0}^{\infty} \frac{1}{k!} \Theta^k.$$

Definition 2.1.4. The *differential* of the exponential map is defined as the *right trivialized tangent* of the exponential map, that is, a map $d\text{Exp} : \mathfrak{g} \times \mathfrak{g} \rightarrow \mathfrak{g}$ defined through the relation

$$d\text{Exp}(\hat{\Theta}, \Theta) \text{Exp}(\hat{\Theta}) = \left. \frac{d}{dt} \right|_{t=0} \text{Exp}(\hat{\Theta} + t\Theta).$$

If we fix the first argument $\hat{\Theta} \in \mathfrak{g}$ then the map $d\text{Exp}_{\hat{\Theta}} : \mathfrak{g} \rightarrow \mathfrak{g}$ given by $d\text{Exp}_{\hat{\Theta}}(\Theta) = d\text{Exp}(\hat{\Theta}, \Theta)$ is linear and satisfies the following theorem.

Theorem 2.1.1. [50, Theorem 3] The differential of the exponential map and its inverse are given by the following formulas

$$\begin{aligned} dExp_{\hat{\Theta}} &= \left. \frac{e^z - 1}{z} \right|_{z=ad_{\hat{\Theta}}} = \sum_{k=0}^{\infty} \frac{1}{(k+1)!} ad_{\hat{\Theta}}^k, \\ dExp_{\hat{\Theta}}^{-1} &= \left. \frac{z}{e^z - 1} \right|_{z=ad_{\hat{\Theta}}} = \sum_{k=0}^{\infty} \frac{B_k}{k!} ad_{\hat{\Theta}}^k, \end{aligned}$$

where the coefficients B_k are the Bernoulli numbers $\{1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots\}$.

A crucial point in the theory of Lie group integrator is to define the basic motions on the manifold \mathcal{M} . Typically they are given by the Lie group \mathcal{G} and its action \cdot on \mathcal{M} . All numerical schemes presented in Section 2.2, advance by following flows defined with respect to the basic motions on \mathcal{M} in such a way that, it is guaranteed that we stay on the manifold.

Definition 2.1.5. An *action* of a Lie group \mathcal{G} on a manifold \mathcal{M} is a smooth map $\cdot : \mathcal{G} \times \mathcal{M} \rightarrow \mathcal{M}$ satisfying

$$\begin{aligned} e \cdot p &= p \quad \forall p \in \mathcal{M}, \\ g \cdot (k \cdot p) &= (g \star k) \cdot p \quad \forall g, k \in \mathcal{G}, p \in \mathcal{M}. \end{aligned}$$

We say that the group action is *transitive* if for any two points $p_1, p_2 \in \mathcal{M}$, there exists at least one element $g \in \mathcal{G}$ such that $g \cdot p_1 = p_2$. This allows us to move from any point $p_1 \in \mathcal{M}$ to any other point $p_2 \in \mathcal{M}$, by letting the element $g \in \mathcal{G}$, act on the first point p_1 . A manifold \mathcal{M} which is acted upon by a Lie group via a transitive action is called a *homogeneous space*.

Via the exponential map, every group action naturally defines an *algebra action* $* : \mathfrak{g} \times \mathcal{M} \rightarrow \mathcal{M}$, by the identity

$$\Theta * p = \text{Exp}(\Theta) \cdot p. \quad (2.1.2)$$

Note that the algebra action is not uniquely determined by the group action. Every diffeomorphism

$$\Psi : \mathfrak{g} \rightarrow \mathcal{G}, \quad (2.1.3)$$

such that $\Psi(0) = e$ and $\Psi'(0) = I$, where I is the identity of the algebra, defines an algebra action by the formula $\Theta * p = \Psi(\Theta) \cdot p$.

Let $\mathfrak{X}(\mathcal{M})$ denote the set of all vector fields on \mathcal{M} . There is a one to one correspondence between the elements in the Lie algebra \mathfrak{g} and the space $\mathfrak{X}(\mathcal{M})$. This can be shown in the same way as the correspondence between \mathfrak{g} and $\mathfrak{X}(\mathcal{G})$. If we define the product $\otimes : \mathfrak{g} \times \mathcal{M} \rightarrow T\mathcal{M}$ by

$$\Theta \otimes p = \left. \frac{d}{dt} \right|_{t=0} \gamma(t) \cdot p,$$

where $\gamma(t)$ is a smooth curve in \mathcal{G} such that $\gamma(0) = e$ and $\gamma'(0) = \Theta$, then for a fix $\Theta \in \mathfrak{g}$ the product $\Theta \circledast p$ gives a vector field $\mathcal{F}_\Theta(\cdot) = \Theta \circledast (\cdot)$ on \mathcal{M} . In this way, the map $\Theta \rightarrow \mathcal{F}_\Theta$ is an algebra homomorphism between \mathfrak{g} and $\mathfrak{X}(\mathcal{M})$. The vector field \mathcal{F}_Θ is called a *frozen* vector field. The following relation between the frozen vector field and the exponential map holds.

Theorem 2.1.2. *For every $\Theta \in \mathfrak{g}$ and every $p \in \mathcal{M}$*

$$\mathcal{F}_\Theta(p) = \Theta \circledast p = \left. \frac{d}{dt} \right|_{t=0} \text{Exp}(t\Theta) \cdot p.$$

Proof. By definition

$$\begin{aligned} \Theta \circledast p &= \left. \frac{d}{dt} \right|_{t=0} \gamma(t) \cdot p = \left. \frac{d}{dt} \right|_{t=0} (e + t\Theta + \mathcal{O}(t^2)) \cdot p \\ &= \left. \frac{d}{dt} \right|_{t=0} t\Theta \cdot p = \left. \frac{d}{dt} \right|_{t=0} \text{Exp}(t\Theta) \cdot p. \end{aligned}$$

□

The last Theorem is still valid if we replace the exponential map with the diffeomorphism $\Psi : \mathfrak{g} \rightarrow \mathcal{G}$ defined above. This shows that the algebra homomorphism between \mathfrak{g} and $\mathfrak{X}(\mathcal{M})$ is independent of the choice of Ψ . We will use this fact later in the construction of our exponential integrators.

Finally, we give the following fundamental result, which is a starting point for every Lie group integrator.

Theorem 2.1.3. *[36] Every differential equation evolving on a homogeneous space \mathcal{M} can always be written as*

$$u'(t) = F(u) \circledast u, \quad u(t_0) = u_0, \quad (2.1.4)$$

where $F : \mathcal{M} \rightarrow \mathfrak{g}$.

Before we continue with the presentation of the main Lie group integrators, we summarize in Table 2.1 all the actions and homeomorphisms which has been defined so far. We note that in the literature authors often use the symbol “.” to denote each of the operations given in Table 2.1 and let its meaning to be determinate by the arguments. However, we find this approach a bit confusing, so we have chosen different notations for each operation.

2.2 Lie group integrators

All algorithms presented in this section are stated in terms of *actions* and reduce to traditional Runge–Kutta methods when the manifold \mathcal{M} is the vector space \mathbb{R}^d . The numerical

notation	diagram	name
\star	$\mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$	group operation
\cdot	$\mathcal{G} \times \mathcal{M} \rightarrow \mathcal{M}$	group action
$*$	$\mathfrak{g} \times \mathcal{M} \rightarrow \mathcal{M}$	algebra action
\odot	$\mathfrak{g} \times \mathcal{G} \rightarrow T\mathcal{G}$	homeomorphism between \mathfrak{g} and $\mathfrak{X}(\mathcal{G})$
\otimes	$\mathfrak{g} \times \mathcal{M} \rightarrow T\mathcal{M}$	homeomorphism between \mathfrak{g} and $\mathfrak{X}(\mathcal{M})$

Table 2.1: Actions and homeomorphisms on the manifold.

schemes advance from the point u_n to the point u_{n+1} with a time step, of size h . The same results can also be formulated in terms of *rigid frames* (see [12, 15, 55]).

The first Lie group integrators we consider are the methods of Crouch and Grossman [15]. The main idea behind these methods is to compose the flows of frozen vector fields at every stage. An s -stage Crouch and Grossman method is formulated by the following algorithm

Algorithm 2.2.1. (Crouch–Grossman method)

for $i = 1, \dots, s$ **do**

$$U_i = \text{Exp}(h\alpha_{is}F_s) \cdots \text{Exp}(h\alpha_{i1}F_1) \cdot u_n$$

$$F_i = F(U_i)$$

end

$$u_{n+1} = \text{Exp}(h\beta_sF_s) \cdots \text{Exp}(h\beta_1F_1) \cdot u_n$$

The function F gives the *generic presentation* (2.1.4) of the differential equation. The coefficients α_{ij}, β_j are parameters of the method. If $\alpha_{ij} = 0$ for $i \leq j$ the method is *explicit* and is *implicit* otherwise. The order of the method is defined like for the traditional Runge–Kutta methods. The parameters α_{ij}, β_j are chosen to satisfy certain order conditions which, because of the non-commutativity of the basic flows, are different from the order conditions for Runge–Kutta methods. In general one can not hope that an order p Runge–Kutta method will lead to an order p Crouch–Grossman method. A complete order theory for this methods involving *ordered rooted trees* is developed in [55]. There the authors propose the following explicit method of order three

$$\begin{array}{c|cc}
0 & & \\
\frac{3}{4} & \frac{3}{4} & \\
\frac{17}{24} & \frac{119}{216} & \frac{17}{108} \\
\hline
& \frac{13}{51} & -\frac{2}{3} \quad \frac{24}{17}
\end{array} \quad . \quad (2.2.1)$$

By careful investigation of the order conditions, the authors in [55] found that it is not possible to obtain explicit methods of order four with only four stages. Further, they

derive a family of fourth order methods with five stages. Higher order methods of Crouch–Grossman type are derived in [33]. An exponential integrator based on the scheme (2.2.1), for the semilinear problem (1.1.1), is given in Section 2.4.

We next consider a generalization of the methods first proposed in [48] and later named as Runge–Kutta Munthe-Kaas (RKMK) methods. In their original formulation these methods achieve only second order on a general Lie group. To construct higher order RKMK methods, a correction function was introduced in [49]. Here we follow the formulation of the methods on homogeneous manifolds given in [50].

The main idea behind these methods is to transform the original equation evolving on a manifold into a corresponding equation on a Lie algebra. Since the Lie algebra is a linear space and every Runge–Kutta method in general preserves *linear invariants*, one can apply any traditional Runge–Kutta method to the transformed equation. Thus, it is guaranteed that the numerical approximation will again be in the Lie algebra. To obtain the desired approximation on the manifold one just needs to apply the inverse transformation. An s -stage RKMK method is formulated by the following algorithm

Algorithm 2.2.2. (Runge–Kutta Munthe-Kaas method)

for $i = 1, \dots, s$ **do**

$$\Theta_i = h \sum_{j=1}^s \alpha_{ij} K_j$$

$$F_i = F(\Psi(\Theta_i) \cdot u_n)$$

$$K_i = d\Psi^{-1}(F_i)$$

end

$$u_{n+1} = \Psi(h \sum_{i=1}^s \beta_i K_i) \cdot u_n$$

Here F gives the generic presentation (2.1.4), Ψ is a diffeomorphism (2.1.3) and $d\Psi$ is defined as in Definition 2.1.4. The coefficients α_{ij} , β_j are the classical Runge–Kutta coefficients. Note that each of the quantities $\Theta_i, F_i, K_i \in \mathfrak{g}$ for $i = 1, \dots, s$. The values of the internal stages, which lie on the manifold, can be computed by letting the elements $\Theta_i \in \mathfrak{g}$, act on the point $u_n \in \mathcal{M}$. In other words $U_i = \Theta_i * u_n$, for $i = 1, \dots, s$. A natural choice for the diffeomorphism Ψ is the Exp map, but other choices are also possible. For example choosing Ψ to be an approximation to the Exp map can lead to a significant reduction in the overall computational cost of the algorithm. In Section 2.4 we present different choices of Ψ and relate them with some of the exponential integrators given in [36, Articles 4 and 5].

The main challenge in the implementation of RKMK methods comes from the necessity of computing the $d\Psi^{-1}$ map. This in general can be a very costly task, especially in the case when $\Psi \equiv \text{Exp map}$. Thus, the approach proposed in [50] is to replace $d\text{Exp}^{-1}$ with its truncated approximation of order higher than the order of the method. This results in introducing commutators into the format of the method.

The last methods under consideration are the Commutator Free (CF) Lie group methods proposed in [12]. They are based on the idea of operating on the elements in the algebra

\mathfrak{g} , like for RKMK methods, and then composing the corresponding resulting flows of frozen vector fields, in a Crouch–Grossman like manner. In this way it is possible to construct an integrator which does not involve any commutators and still uses less Exp evaluations than in the method of Crouch–Grossman. A general format of an s -stage *commutator free* Lie group methods is given in the following algorithm

Algorithm 2.2.3. (Commutator-free Lie group method)

for $i = 1, \dots, s$ **do**

$$U_i = \text{Exp}(h \sum_{k=1}^s \alpha_{iJ}^k F_k) \cdots \text{Exp}(h \sum_{k=1}^s \alpha_{i1}^k F_k) \cdot u_n$$

$$F_i = F(U_i)$$

end

$$u_{n+1} = \text{Exp}(h \sum_{k=1}^s \beta_J^k F_k) \cdots \text{Exp}(h \sum_{k=1}^s \beta_1^k F_k) \cdot u_n$$

Here again F gives the generic presentation (2.1.4) and the coefficients α_{ij}^k, β_j^k are parameters of the method. They are determined based on the order theory which can be adapted from the order theory presented in [55]. In general the method is *implicit* unless $\alpha_{ij}^k = 0$ for $i \leq k$ then it is *explicit*. The parameter J counts the number of Exp evaluations for each stage. It is preferable to keep it as low as possible in order to obtain competitive methods. Another way of reducing the computational cost is by reusing the flow calculation. The following explicit fourth order method, based on the classical fourth order Runge–Kutta method (1.3.5), which effectively uses only 5 Exp per step is given in [12]

$$\begin{array}{c|cccc}
 0 & & & & \\
 \frac{1}{2} & \frac{1}{2} & & & \\
 \frac{1}{2} & 0 & \frac{1}{2} & & \\
 \frac{1}{2} & \frac{1}{2} & 0 & 0 & \\
 \frac{1}{2} & -\frac{1}{2} & 0 & 1 &
 \end{array} \left. \vphantom{\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & \\ \frac{1}{2} & -\frac{1}{2} & 0 & 1 & \end{array}} \right\} \quad . \quad (2.2.2)$$

$$\begin{array}{c|cccc}
 & \frac{1}{4} & \frac{1}{6} & \frac{1}{6} & -\frac{1}{12} \\
 & \frac{1}{12} & \frac{1}{6} & \frac{1}{6} & \frac{1}{4}
 \end{array} \left. \vphantom{\begin{array}{c|cccc} & \frac{1}{4} & \frac{1}{6} & \frac{1}{6} & -\frac{1}{12} \\ & \frac{1}{12} & \frac{1}{6} & \frac{1}{6} & \frac{1}{4} \end{array}} \right\}$$

We use the symbol $\}$ to denote all the substages included in a stage with $J > 1$.

We close this section with the observation that in order to implement Algorithm 2.2.1 and Algorithm 2.2.3 the only two things which we need to know are the generic presentation (2.1.4) of the differential equation and the algebra action (2.1.2) on the manifold \mathcal{M} . To implement Algorithm 2.2.2, in addition, we need to know either the $d\Psi^{-1}$ map or how the commutators between two elements in \mathfrak{g} are defined. This observation allows us to construct Lie group integrators without even knowing what is the exact structure of the Lie group \mathcal{G} and how it acts on \mathcal{M} . This provides us the freedom to define the Lie algebra \mathfrak{g} independently from the Lie group \mathcal{G} . For example, like a Lie algebra of vector fields on

\mathcal{M} (see [51]). This approach is very useful for applications concerning PDEs, where the Lie algebra is infinite dimensional and establishing the structure of a Lie group $\mathcal{G} = \text{Exp}(\mathfrak{g})$ is technically more demanding than in the finite dimensional case (see [32]). A way of circumventing this problem is to semidiscretize the PDE in space and thus obtain a finite-dimensional system. In the next section we utilize this approach and discuss the choice of action as well as the structure of \mathcal{G} , in the case when $\mathcal{M} \equiv \mathbb{R}^d$.

2.3 The choice of action

When the differential equation evolves on \mathbb{R}^d constructing an integrator which stays in \mathbb{R}^d is a trivial task and does not require a special theory. However, using the framework of Lie group methods, one can construct useful methods specially designed to solve the equation (2.1.1). In this section we pay particular attention to the choice of group action \cdot and how it is related to the actual performance of the method.

We first consider the case when the Lie group $\mathcal{G} \equiv \mathbb{R}^d$ and its action upon \mathcal{M} is given by the vector addition. In other words

$$g \cdot p = g + p, \quad \text{for all } g \in \mathcal{G} \text{ and } p \in \mathcal{M}. \quad (2.3.1)$$

From Definition 2.1.1, it follows that the group operation \star is also given by vector addition and that the identity element e is the zero vector in \mathbb{R}^d . The Lie algebra $\mathfrak{g} = T_e \mathcal{G}$ can be identified with \mathbb{R}^d and the algebra homeomorphism between \mathfrak{g} and $\mathfrak{X}(\mathcal{G})$ defined by

$$\Theta \odot g = \left. \frac{d}{dt} \right|_{t=0} \gamma(t) \star g = \left. \frac{d}{dt} \right|_{t=0} g + t\Theta + \mathcal{O}(t^2) = \Theta,$$

is simply the identity map. This implies that the exponential map and the homeomorphism between \mathfrak{g} and $\mathfrak{X}(\mathcal{M})$ are also given by the identity map. The generic presentation (2.1.4) of the differential equation is

$$u' = F(u) \otimes u = F(u).$$

If $b \in \mathbb{R}^d$ is a fixed element in \mathfrak{g} then its corresponding frozen vector field $\mathcal{F}_b(p) = b$ for all points p on the manifold. The Lie group integrators presented in Section 2.2 use the flow of a frozen vector field, as the basic motions on \mathcal{M} . This means that the solution of the differential equation

$$u' = b, \quad u(t_0) = u_0,$$

which is given by translation $u(t_0 + h) = u_0 + hb$, defines an algebra action $hb * u_0$. Since translations commute, we obtain the trivial bracket $[a, b] = 0$, for all $a, b \in \mathfrak{g}$. Note that via the relation (2.1.2) the above algebra action reproduces the group action (2.3.1). This choice simply reduces all the methods presented in Section 2.2 to the traditional Runge–Kutta methods.

Like it was mentioned in the previous section, in order to construct Lie group integrators it is enough to know the algebra action and the generic presentation of the differential

equation. In the subsequent, discussions which concern the structure of the Lie group and its action on \mathcal{M} are kept with the scope of completeness.

A more interesting choice for the algebra action, rather than translations, is to move from the point u_0 to another point by following the flow of a linear vector field

$$u' = Au, \quad u(t_0) = u_0,$$

where $A \in \mathbb{R}^{d \times d}$ is a constant matrix. In this case the Lie group \mathcal{G} is the general linear group $GL(d) = \{G \in \mathbb{R}^{d \times d} : \det G \neq 0\}$, its Lie algebra $\mathfrak{gl}(d)$ is the set of all $d \times d$ matrices and the bracket is given by the matrix commutator $[A, B] = AB - BA$. The only problem here is that the action is not transitive. This is due to the fact that there is no way to advance from the point $u_0 = 0$. A way to avoid this inconvenience is to consider the action arising from the solution of the following differential equation

$$u' = Au + b, \quad u(t_0) = u_0, \quad (2.3.2)$$

where $A \in \mathbb{R}^{d \times d}$ and $b \in \mathbb{R}^d$ are constants. This idea was first introduced in [50] and then further investigated for the heat equation in [12, 39, 63], for stiff PDEs in [36, Articles 4 and 5] and for the Schrödinger equation in [3]. The action arising from (2.3.2) is often referred to as the *affine* action. The Lie group \mathcal{G} in this case can be identified with the set of all pairs $(G, g) \in GL(d) \times \mathbb{R}^d$ which we denote by the *semidirect product* $GL(d) \rtimes \mathbb{R}^d$ (see [67]). Its action on the manifold is defined by

$$(G, g) \cdot p = Gp + g \quad \text{for all } (G, g) \in \mathcal{G} \text{ and } p \in \mathcal{M}.$$

From Definition 2.1.1, it follows that the group operation \star is given by

$$(G_1, g_1) \star (G_2, g_2) = (G_1 G_2, G_1 g_2 + g_1)$$

and that the identity element $e = (I, \mathbf{o})$, where I is the $d \times d$ identity matrix and \mathbf{o} is the zero vector in \mathbb{R}^d . The Lie algebra $\mathfrak{g} = T_e \mathcal{G}$ is the set of all pairs $(A, b) \in \mathfrak{gl}(d) \rtimes \mathbb{R}^d$ and the bracket is defined as

$$[(A_1, b_1), (A_2, b_2)] = (A_1 A_2 - A_2 A_1, A_1 b_2 - A_2 b_1). \quad (2.3.3)$$

The algebra homeomorphism between \mathfrak{g} and $\mathfrak{X}(\mathcal{G})$ is given by $(A, b) \rightarrow f_{(A, b)}((\cdot, \cdot))$, where

$$f_{(A, b)}((G, g)) = (A, b) \odot (G, g) = \frac{d}{dt} \Big|_{t=0} \gamma(t) \star (G, g) = (AG, Ag + b).$$

If $\gamma(t)$ is a smooth curve in \mathcal{G} such that $\gamma(0) = e$ and $\gamma'(0) = (A, b)$, then the solution of the differential equation

$$\gamma'(t) = f_{(A, b)}(\gamma(t)), \quad \gamma(0) = e,$$

is given by $\gamma(t) = (e^{tA}, \phi^{[1]}(tA)tb)$, where the function $\phi^{[1]}$ is the first ETD function from (1.1.6). This implies that the exponential map is defined as

$$\text{Exp}(A, b) = (e^A, \phi^{[1]}(A)b). \quad (2.3.4)$$

Therefore, the algebra action is

$$h(A, b) * u_0 = \text{Exp}(h(A, b)) \cdot u_0 = e^{hA} u_0 + \phi^{[1]}(hA) hb, \quad (2.3.5)$$

which is exactly the solution of the differential equation (2.3.2) at the time $t_0 + h$. Finally, the frozen vector field is given by

$$\mathcal{F}_{(A,b)}(u) = (A, b) \otimes u = Au + b.$$

An important observation concerning the affine action is that the generic presentation (2.1.4) of a differential equation (2.1.1) is not uniquely defined. For example, if we choose $F(u) = (0, f(u))$ then we will recover exactly the classical settings considered in the beginning of this section. Another possible choice is $F(u) = (J, f(u) - Ju)$, where J is the Jacobian of f at the point u . In this case the RKMK method based on forward Euler is given by

$$u_{n+1} = u_n + J^{-1}(e^{hJ} - I)f(u).$$

It is a second order method which is often referred to as the exponential fitted Euler method [28].

When the vector field of the differential equation is given in nonautonomous form, like in (1.1.1), we need to transform it to *autonomous* form by appending t to the dependent variables. This increases the dimension of the manifold and allow us to include the time variable to the arguments of the function F . Such an approach is very useful, especially if we want to construct time dependent frozen vector fields (see [Article 1, p.91]). A natural choice for F , which reflects the semilinear structure of (1.1.1), is

$$F(u, t) = (L, N(u, t)). \quad (2.3.6)$$

Methods based on this generic presentation for the equation (1.1.1) are given in the next section.

More complicated differential equations, which hopefully approximate the original vector field in a better way, can also be used to define the basic motions on \mathcal{M} . The main requirement is that we should be able to solve them exactly and thus define the algebra action like the flow of some differential equation. A way how to construct Lie group integrators based on the action arising from the solution of the following differential equation

$$u' = Au + b + ct, \quad u(t_0) = u_0, \quad (2.3.7)$$

where $A \in \mathbb{R}^{d \times d}$, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}^d$ are constants, is proposed in [Article 1, p.91]. There the corresponding exponential integrator for solving the semilinear problem (1.1.1), based on the fourth order commutator free Lie group method (2.2.2), is also derived. The main idea is to rewrite the above equations in *autonomous* form and then apply the affine action to the transformed equation. This approach can be further generalized by including second and higher order terms of t in the frozen vector field. However, one should keep in mind

that there is a certain balance between the cost of finding the exponential of (2.3.7) and the benefit provided from choosing a better approximation. We remark that in any case the corresponding algebra homeomorphism \otimes will always be the same as in the case of the affine action. This means that for a given problem by choosing the right algebra action (for example, the tangent to the real flow) one can improve the actual performance of the method and even increase its order like in the case of the exponentially fitted Euler method. More discussions about the role of the algebra action are also included in the next section.

2.4 Lie group integrators for semilinear problems

Here we present different exponential integrators for solving the semilinear problem (1.1.1), which are based on the methods considered in Section 2.2 and used the affine algebra action introduced in Section 2.3. Most of the methods included in this section are derived in [36, Articles 4 and 5]. We give their equivalent formulations, which are in accordance with the presentation followed in this thesis, and discuss the connection between the *Generalized Integrating Factor Runge–Kutta* (GIF/RK) methods introduced in [36, Articles 5] and the RKMK methods. In addition, we propose a new approach regarding the derivation of GIF/RK methods which is applicable to problems where the nonlinear part of (1.1.1) is better approximated by trigonometric polynomials.

Before we consider the first numerical schemes, we make the following assumptions. Let h be a constant step size and u_1, u_2, \dots, u_n are approximations of the solution of (1.1.1) at the moments $t_j = t_0 + jh$ for $j = 1, \dots, n$. We are looking for an approximation to the solution at time $t_{n+1} = t_0 + (n+1)h$. In other words we consider the following problem

$$u' = Lu + N(u, t_n + t) = f(u, t_n + t), \quad u(t_n) = u_n. \quad (2.4.1)$$

Note that, in general, we allow our numerical schemes to have incoming data consisting not only from the quantity u_n but also from some of the quantities available from the previous steps. Thus, the resulting methods are *exponential general linear methods* introduced in Section 1.3.

All methods presented in this section are based on the pure Runge–Kutta idea and use only u_n as the incoming data at the beginning of step number $n+1$. The only exceptions are the GIF/RK methods, which are exponential general linear methods. Other examples of Lie group methods which are also exponential general linear methods are the methods based on the action presented in [Article 1, p.91]. It is rather surprising that with the framework of Lie group integrators, which we saw in Section 2.2 are entirely based on the Runge–Kutta idea, one can get more general methods like general linear methods. This can be explained by the crucial role of the algebra action in the overall performance of the method.

In the subsequent work we have adopted the notations from Section 1.3. In addition, we introduce the following convention: Unless it is not specified the arguments in the row

number i of the functions $\phi^{[j]}$ (from Lemma 1.1.1) and $\widehat{\phi}^{[j]}$ (to be defined below) are $c_i hL$, for $i = 1, 2, \dots, s$ and $j = 1, 2, \dots$

2.4.1 Crouch–Grossman based methods

Keeping in mind (2.3.5) and (2.3.6) the derivation of an exponential integrator based on Algorithm 2.2.1 is a straight forward process. Our first example is a method which corresponds to the scheme (2.2.1).

$$\left[\begin{array}{ccc|c} 0 & 0 & 0 & \mathbf{I} \\ \frac{3}{4}\phi^{[1]} & 0 & 0 & e^{\frac{3}{4}hL} \\ \frac{119}{216}e^{\frac{17}{108}hL}\phi^{[1]}(\frac{119}{216}hL) & \frac{17}{108}\phi^{[1]}(\frac{17}{108}hL) & 0 & e^{\frac{17}{24}hL} \\ \hline \frac{13}{51}e^{\frac{38}{51}hL}\phi^{[1]}(\frac{13}{51}hL) & -\frac{2}{3}e^{\frac{24}{17}hL}\phi^{[1]}(-\frac{2}{3}hL) & \frac{24}{17}\phi^{[1]}(\frac{24}{17}hL) & e^{hL} \end{array} \right]$$

Table 2.2: Third order Crouch–Grossman method.

2.4.2 RKMK based methods

Every method based on Algorithm 2.2.2 requires either the exact $d\Psi^{-1}$ map or its approximation which involves commutators between two elements in \mathfrak{g} . In the case of the semilinear problem (2.4.1) the structure of the Lie algebra \mathfrak{g} is simpler than the general structure presented in Section 2.3. In fact, since the linear part L stays constant, the Lie algebra \mathfrak{g} can be identified with all pairs $(\lambda L, b)$, where $\lambda \in \mathbb{R}$ and $b \in \mathbb{R}^d$. This allows us to compute the $d\Psi^{-1}$ map exactly and relatively cheaply. The bracket (2.3.3) in this case is given by

$$[(\widehat{\lambda}L, \widehat{b}), (\lambda L, b)] = (\mathbf{O}, L(\widehat{\lambda}b - \lambda\widehat{b})),$$

where \mathbf{O} denotes the zero matrix in $\mathbb{R}^{d \times d}$.

RKMK methods with exact Exp map

Let us first consider the case when $\Psi \equiv \text{Exp}$ map. From (2.3.4), it follows that the exponential map in this case is

$$\text{Exp}(\lambda L, b) = (e^{\lambda L}, \phi^{[1]}(\lambda L)b).$$

It is easy to check that

$$\text{ad}_{(\widehat{\lambda}L, \widehat{b})}^k(\lambda L, b) = (\mathbf{O}, \widehat{\lambda}^{k-1}L^k(\widehat{\lambda}b - \lambda\widehat{b})), \quad \text{for } k \geq 1$$

and thus the expression for dExp^{-1} has the form

$$\begin{aligned} \text{dExp}_{(\hat{\lambda}L, \hat{b})}^{-1}(\lambda L, b) &= \sum_{k=0}^{\infty} \frac{B_k}{k!} \text{ad}_{(\hat{\lambda}L, \hat{b})}^k(\lambda L, b) \\ &= \left(\lambda L, \phi^{[1]^{-1}}(\hat{\lambda}L) \left(b - \frac{\lambda}{\hat{\lambda}} \hat{b} \right) + \frac{\lambda}{\hat{\lambda}} \hat{b} \right). \end{aligned} \quad (2.4.2)$$

When $\hat{\lambda} = 0$ the dExp^{-1} is equal to the identity map in \mathfrak{g} .

We know that every RKMK method is equivalent to a traditional Runge–Kutta method applied to the corresponding differential equation in the Lie algebra \mathfrak{g} , followed by the inverse transformation of the approximate solution back to the manifold (see Section 2.2). In order to find the differential equation in \mathfrak{g} we choose $\Theta(t) = (tL, z(t))$ to be a curve in \mathfrak{g} such that $\Theta(0) = (\mathbf{O}, \mathbf{o})$ and the solution of (2.4.1) is given by

$$u(t_n + t) = \text{Exp}(\Theta(t)) \cdot u_n = e^{tL} u_n + t \phi^{[1]}(tL) z(t). \quad (2.4.3)$$

By differentiating (2.4.3) and taking into account (2.4.2) and the generic presentation (2.3.6), we obtain the following corresponding differential equation for $z(t)$

$$z'(t) = \phi^{[1]^{-1}}(tL) (N(\text{Exp}(tL, z) \cdot u_n, t_n + t) - z/t) + z/t, \quad z(0) = \mathbf{o}. \quad (2.4.4)$$

Thus we see that a single step of a RKMK method for the equation (2.4.1) is equivalent to one step of a traditional Runge–Kutta (RK) method for the equation (2.4.4), followed by computing $u(t_n + t)$ from the approximation $z(t)$ via (2.4.3). In Table 2.3, we give the RKMK method based on the classical fourth order Runge–Kutta method (1.3.5) with exact Exp map.

The above approach reminds us to the technique used in the construction of the Integrating Factor Runge–Kutta (IF RK) methods presented in Section 1.2. As a matter of fact there exists a connection between the IF RK and the RKMK methods in a sense that every IF RK method is simply a RKMK method with a special choice for the diffeomorphism Ψ . This fact was first observed in [36, Article 4]. Based on a similar idea a class of methods, which can be reasonably named as *Generalized Integrating Factor Runge–Kutta* (GIF/RK) methods, was derived in [36, Article 5]. Next we discuss this class of methods and show that they also can be brought into the framework of RKMK methods.

RKMK methods with approximations to the Exp map

Let us chose the diffeomorphism Ψ to be the following approximation to the Exp map

$$\Psi(\lambda L, b) = (e^{\lambda L}, e^{\lambda L} b). \quad (2.4.5)$$

In this case, by direct computations, it is possible to find the following closed form for the $\text{d}\Psi^{-1}$ map (see [36, Article 4])

$$\text{d}\Psi_{(\hat{\lambda}L, \hat{b})}^{-1}(\lambda L, b) = (\lambda L, e^{-\hat{\lambda}L} b).$$

$$\left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & I \\ \frac{1}{2}\phi^{[1]} & 0 & 0 & 0 & e^{\frac{1}{2}hL} \\ \frac{1}{2}\phi^{[1]} - \frac{1}{2}I & \frac{1}{2}I & 0 & 0 & e^{\frac{1}{2}hL} \\ \widehat{\phi}^{[3]^2}\left(\frac{hL}{2}\right) & -\widehat{\phi}^{[2]}\widehat{\phi}^{[3]}\left(\frac{hL}{2}\right) & \widehat{\phi}^{[2]} & 0 & e^{hL} \\ \hline b_1(hL) & b_2(hL) & b_3(hL) & \frac{1}{6}I & e^{hL} \end{array} \right],$$

where

$$\begin{aligned} \widehat{\phi}^{[2]}(z) &= \phi^{[1]}(z)\phi^{[1]-1}\left(\frac{z}{2}\right) = \frac{e^{\frac{z}{2}} + I}{2}, \\ \widehat{\phi}^{[3]}(z) &= \phi^{[1]-1}(z) - I = \frac{e^z - z - I}{I - e^z}, \\ b_1(hL) &= \frac{1}{6}\phi^{[1]} - \frac{1}{3}\phi^{[1]}\widehat{\phi}^{[3]}\left(\frac{hL}{2}\right) - \frac{1}{6}\phi^{[1]}\left(\widehat{\phi}^{[3]} - 2I\right)\widehat{\phi}^{[3]^2}\left(\frac{hL}{2}\right), \\ b_2(hL) &= \frac{1}{3}\widehat{\phi}^{[2]} + \frac{1}{6}\widehat{\phi}^{[2]}\left(\widehat{\phi}^{[3]} - 2I\right)\widehat{\phi}^{[3]}\left(\frac{hL}{2}\right), \\ b_3(hL) &= \frac{1}{3}\widehat{\phi}^{[2]} - \frac{1}{6}\widehat{\phi}^{[2]}\widehat{\phi}^{[3]}. \end{aligned}$$

Table 2.3: Fourth order RKMK method with exact Exp.

Thus, again, if $\Theta(t) = (tL, z(t))$ is a curve in \mathfrak{g} such that $\Theta(0) = (\mathbf{O}, \mathbf{o})$ and

$$u(t_n + t) = \Psi(\Theta(t)) \cdot u_n = e^{tL}(u_n + z(t)), \quad (2.4.6)$$

then the corresponding differential equation for $z(t)$ is

$$z'(t) = e^{-tL}N(\Psi(tL, z) \cdot u_n, t_n + t), \quad z(0) = \mathbf{o}.$$

If we put $v(t) = u_n + z(t)$ then (2.4.6) is simply the Lawson transformation (see Subsection 1.1.1) and the corresponding differential equation for $v(t)$ has the familiar form

$$v'(t) = e^{-tL}N(e^{tL}v, t_n + t), \quad v(0) = u_n.$$

In this way we see that IF RK methods are simply RKMK methods with Ψ given by (2.4.5). General formulation of an IF RK method is given in Table 1.3.

Let us now define the diffeomorphism Ψ to be

$$\Psi(\lambda L, b) = (e^{\lambda L}, e^{\lambda L}(b - \lambda N_n) + \lambda \phi^{[1]}(\lambda L)N_n),$$

where $N_n = N(u_n, t_n)$. Note that at the beginning of step number $n + 1$ the value u_n is already known and therefore we can easily compute N_n . In this case

$$d\Psi_{(\hat{\lambda}L, \hat{b})}^{-1}(\lambda L, b) = (\lambda L, e^{-\hat{\lambda}L}(b - \lambda N_n) + \lambda N_n),$$

and by setting

$$u(t_n + t) = \Psi(\Theta(t)) \cdot u_n = e^{tL}(u_n + z(t) - tN_n) + t\phi^{[1]}(tL)N_n, \quad (2.4.7)$$

we obtain the following differential equation for $z(t)$

$$z'(t) = e^{-tL}(N(\Psi(tL, z) \cdot u_n, t_n + t) - N_n) + N_n, \quad z(0) = \mathbf{o}. \quad (2.4.8)$$

If we substitute in this case $v(t) = u_n + z(t) - tN_n$ then (2.4.7) has the form

$$u(t_n + t) = e^{tL}v(t) + t\phi^{[1]}(tL)N_n, \quad (2.4.9)$$

and the corresponding equation for $v(t)$ is

$$v'(t) = e^{-tL}(N(u(t_n + t), t_n + t) - N_n), \quad v(0) = u_n. \quad (2.4.10)$$

Thus, applying any traditional Runge–Kutta method to the equation (2.4.8) or (2.4.10) and then transforming back the result into the manifold by (2.4.7) or (2.4.9) respectively, is a RKMK method which we denote by GIF1/RK. This in fact is the first of the generalized integrating factor Runge–Kutta methods proposed in [36, Article 5]. In Table 2.4, we give the general formulation of a fourth order GIF1/RK method which corresponds to a RK method with coefficients $\mathcal{A} = (\alpha_{ij})$, $b = (\beta_i)$, $c = (c_i)$.

$$\left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & I \\ c_2\phi^{[1]} & 0 & 0 & 0 & e^{c_2hL} \\ c_3\phi^{[1]} - \alpha_{32}e^{(c_3-c_2)hL} & \alpha_{32}e^{(c_3-c_2)hL} & 0 & 0 & e^{c_3hL} \\ c_4\phi^{[1]} - \alpha_{42}e^{(c_4-c_2)hL} - \alpha_{43}e^{(c_4-c_3)hL} & \alpha_{42}e^{(c_4-c_2)hL} & \alpha_{43}e^{(c_4-c_3)hL} & 0 & e^{c_4hL} \\ \hline b_1(hL) & \beta_2e^{(1-c_2)hL} & \beta_3e^{(1-c_3)hL} & \beta_4e^{(1-c_4)hL} & e^{hL} \end{array} \right],$$

where $b_1(hL) = \phi^{[1]} - \beta_2e^{(1-c_2)hL} - \beta_3e^{(1-c_3)hL} - \beta_4e^{(1-c_4)hL}$.

Table 2.4: Fourth order GIF1/RK method

If we allow our self to choose an approximation to the Exp map which at the beginning of step number $n + 1$ uses not only the value N_n , but also and the value of N_{n-1} from the

end of the previous step with size h , we can recover the second of the generalized integrating factor Runge–Kutta methods from [36, Article 5]. Indeed if

$$\Psi(\lambda L, b) = \left(e^{\lambda L}, e^{\lambda L} \left(b - \lambda N_n - \frac{\lambda^2}{2} \frac{N_n - N_{n-1}}{h} \right) + \lambda \phi^{[1]}(\lambda L) N_n + \lambda^2 \phi^{[2]}(\lambda L) \frac{N_n - N_{n-1}}{h} \right),$$

then

$$\mathrm{d}\Psi_{(\hat{\lambda}L, \hat{b})}^{-1}(\lambda L, b) = \left(\lambda L, e^{-\hat{\lambda}L} \left(b - \lambda \left(N_n - \hat{\lambda} \frac{N_n - N_{n-1}}{h} \right) \right) + \lambda \left(N_n - \hat{\lambda} \frac{N_n - N_{n-1}}{h} \right) \right).$$

Substituting

$$u(t_n + t) = \Psi(\Theta(t)) \cdot u_n = e^{tL} v(t) + t \phi^{[1]}(tL) N_n + t^2 \phi^{[2]}(tL) \frac{N_n - N_{n-1}}{h}, \quad (2.4.11)$$

with $v(t) = u_n + z(t) - t N_n - \frac{t^2}{2} \frac{N_n - N_{n-1}}{h}$, we obtain the following equation for $v(t)$

$$v'(t) = e^{-tL} \left(N(u(t_n + t), t_n + t) - N_n - t \frac{N_n - N_{n-1}}{h} \right), \quad v(0) = u_n.$$

Its numerical solution by a traditional RK method, followed by the transformation (2.4.11), gives the second of the GIF/RK methods which we denote by GIF2/RK. In Table 2.5, we give the general formulation of a fourth order GIF2/RK method, which again corresponds to a RK method with coefficients $\mathcal{A} = (\alpha_{ij})$, $b = (\beta_i)$, $c = (c_i)$. Note that GIF2/RK methods are example of exponential general linear methods (see Section 1.3).

This process can be further continued and in general one can show that the GIF/RK methods constructed in [36, Article 5] are also RKMK methods. The prove of this fact is a straight forward extension of the above process.

We note that for the semilinear problem (2.4.1) all RKMK methods, based on the affine group action, are simply RK methods applied to the transformed equation after some change of the variables and then the approximate solution is transformed back into the original variables. In this way, it is quite likely for any method based on this idea to be a RKMK method with appropriate choice for the diffeomorphism Ψ .

The above presented choices for the diffeomorphism Ψ might looks some how strange and unfounded, but all becomes clear when we observe that the Lawson transformation (see Subsection 1.1.1) together with the transformations (2.4.9) and (2.4.11) fits into the form

$$u(t_n + t) = \phi_{t, \hat{F}}(v(t)), \quad (2.4.12)$$

where $\phi_{t, \hat{F}}$ is the flow of the differential equation

$$u' = \hat{F}(u, t), \quad u(0) = u_n. \quad (2.4.13)$$

The vector field $\hat{F}(u, t)$ approximates the original vector field $f(u, t_n + t)$ of (2.4.1) around the point u_n and $\hat{F}(u_n, 0) = f(u_n, t_n) = Lu_n + N(u_n, t_n)$. The corresponding differential equation for the v variable is easily obtained by differentiating (2.4.12) and it has the form

$$v'(t) = \left(\frac{\partial u}{\partial v} \right)^{-1} \left(f(u, t_n + t) - \hat{F}(u, t) \right), \quad v(0) = u_n. \quad (2.4.14)$$

$$\left[\begin{array}{cccc|cc} 0 & 0 & 0 & 0 & I & 0 \\ a_{21}(hL) & 0 & 0 & 0 & e^{c_2 hL} & d_{22}(hL) \\ a_{31}(hL) & \alpha_{32}e^{(c_3-c_2)hL} & 0 & 0 & e^{c_3 hL} & d_{32}(hL) \\ a_{41}(hL) & \alpha_{42}e^{(c_4-c_2)hL} & \alpha_{43}e^{(c_4-c_3)hL} & 0 & e^{c_4 hL} & d_{42}(hL) \\ \hline b_1(hL) & \beta_2e^{(1-c_2)hL} & \beta_3e^{(1-c_3)hL} & \beta_4e^{(1-c_4)hL} & e^{hL} & v_{12}(hL) \\ I & 0 & 0 & 0 & 0 & 0 \end{array} \right],$$

where

$$\begin{aligned} a_{21}(hL) &= c_2\phi^{[1]} + c_2^2\phi^{[2]}, \\ a_{31}(hL) &= c_3\phi^{[1]} + c_3^2\phi^{[2]} - \alpha_{32}(1+c_2)e^{(c_3-c_2)hL}, \\ a_{41}(hL) &= c_4\phi^{[1]} + c_4^2\phi^{[2]} - \alpha_{42}(1+c_2)e^{(c_4-c_2)hL} - \alpha_{43}(1+c_3)e^{(c_4-c_3)hL}, \\ b_1(hL) &= \phi^{[1]} + \phi^{[2]} - \beta_2(1+c_2)e^{(1-c_2)hL} - \beta_3(1+c_3)e^{(1-c_3)hL} - \beta_4(1+c_4)e^{(1-c_4)hL}, \\ d_{22}(hL) &= -c_2^2\phi^{[2]}, \\ d_{32}(hL) &= -c_3^2\phi^{[2]} + c_2\alpha_{32}e^{(c_3-c_2)hL}, \\ d_{42}(hL) &= -c_4^2\phi^{[2]} + c_2\alpha_{42}e^{(c_4-c_2)hL} + c_3\alpha_{43}e^{(c_4-c_3)hL}, \\ v_{12}(hL) &= -\phi^{[2]} + c_2\beta_2e^{(1-c_2)hL} + c_3\beta_3e^{(1-c_3)hL} + c_4\beta_4e^{(1-c_4)hL}. \end{aligned}$$

Table 2.5: Fourth order GIF2/RK method

Now a single step with any RK method applied to the equation (2.4.14), followed by the transformation (2.4.12), will lead to a new method for the corresponding differential equation (2.4.1). This, in fact, is the original formulation of the generalized integrating factor Runge–Kutta methods given in [36, Article 5]. Note that we do not really need to know what is the differential equation in the z variable. It was introduced in the above presentation with the sole purpose of showing the exact correspondence between GIF/RK and RKMK methods.

An important requirement for the vector field \widehat{F} , which follows from the formulation of the method, is that it should not only approximate the original vector field, but it should also have a flow $\phi_{t,\widehat{F}}$ which is easy to compute exactly. Similar methods, based on the idea of approximating the flow $\phi_{t,\widehat{F}}$ by some numerical scheme, are derived in [43].

For the semilinear problem (2.4.1), the choice of \widehat{F} proposed in [36, Article 5] is $\widehat{F}(u, t) = Lu + L_k(N, t)$, where $L_k(N, t)$ is the Lagrange interpolating polynomial of degree $k-1$ for the function $N(u(t_n+t), t_n+t)$ which passes through the k points $N_n, N_{n-1}, \dots, N_{n-k+1}$. Thus, the choice $\widehat{F}(u, t) = Lu$ reproduces the IF RK methods and by letting $\widehat{F}(u, t)$ be the k -th Lagrange interpolating polynomial, we obtain the k -th generalized integrating factor

Runge–Kutta method GIF k /RK. The first three polynomials $L_k(N, t)$ for a constant step size h , which lead to GIF1/RK, GIF2/RK and GIF3/RK, are respectively

$$\begin{aligned} L_1(N, t) &= N_n, \\ L_2(N, t) &= N_n + t \left(\frac{N_n - N_{n-1}}{h} \right), \\ L_3(N, t) &= N_n + t \left(\frac{\frac{1}{2}N_{n-2} - 2N_{n-1} + \frac{3}{2}N_n}{h} \right) + \frac{t^2}{2} \left(\frac{N_{n-2} - 2N_{n-1} + N_n}{h^2} \right). \end{aligned}$$

The exact flow $\phi_{t, \hat{F}}$ in this case is given by Lemma 1.1.1 and by observing that $\frac{\partial \phi_{t, \hat{F}}(v)}{\partial v} = e^{tL}$ the equation (2.4.14) becomes

$$v'(t) = e^{-tL} (N(u(t_n + t), t_n + t) - L_k(N, t)) \quad v(0) = u_n.$$

We pay attention to the fact that in order to implement any of the GIF k /RK methods for $k > 1$, at the beginning of each step, we need to know the value of N and therefore of u not only at the grid point t_n but also at the $k - 1$ previous grid points. Thus, in general, this methods require a starting procedure (see Subsection 1.3.2), which produces the needed values of u and N , before the integration process begins. Such a procedure can be given for example, by the first few steps of any of the other methods presented in this section. Methods which pass more than one quantity from step to step and still use internal stages approximations are examples of general linear methods (see Section 1.3). Thus, GIF k /RK methods for $k > 1$, are examples of *exponential general linear methods*.

Another possible choice for the vector field \hat{F} , which also satisfies the main requirements for constructing a GIF/RK methods is $\hat{F}(u, t) = Lu + T_k(N, t)$, where $T_k(N, t)$ is a trigonometric polynomial approximating the function $N(u(t_n + t), t_n + t)$. This choice of \hat{F} , might be preferable when the nonlinear part N of the original differential equation is periodic and square-integrable. Next we briefly discuss methods based on this idea.

GIF/RK methods for semilinear problems with periodic nonlinear part

We start by giving the exact formula for the flow of a vector field $\hat{F}(u, t) = Lu + T_k(N, t)$, where

$$T_k(N, t) = b + \sum_{\alpha=1}^k (c_\alpha \sin(\alpha t) + d_\alpha \cos(\alpha t)); \quad k \in \mathbb{N}, \quad c_\alpha, d_\alpha \in \mathbb{R}.$$

Lemma 2.4.1. *The solution of the differential equation*

$$u' = \hat{F}(u, t) = Lu + b + \sum_{\alpha=1}^k (c_\alpha \sin(\alpha t) + d_\alpha \cos(\alpha t)), \quad u(t_n) = u_n$$

at the moment $t_n + h$ is given by

$$u(t_n + h) = e^{hL} u_n + hb\phi^{[1]}(hL) + \sum_{\alpha=1}^k \left(c_\alpha \phi_{\sin}^{[\alpha]}(t_n, h, L) + d_\alpha \phi_{\cos}^{[\alpha]}(t_n, h, L) \right), \quad (2.4.15)$$

where

$$\begin{aligned}\phi_{\sin}^{[\alpha]}(t_n, h, L) &= \frac{e^{hL}(L \sin(\alpha t_n) + \alpha \cos(\alpha t_n)) - L \sin(\alpha(t_n + h)) - \alpha \cos(\alpha(t_n + h))I}{\alpha^2 I + L^2}, \\ \phi_{\cos}^{[\alpha]}(t_n, h, L) &= \frac{e^{hL}(L \cos(\alpha t_n) - \alpha \sin(\alpha t_n)) - L \cos(\alpha(t_n + h)) + \alpha \sin(\alpha(t_n + h))I}{\alpha^2 I + L^2}.\end{aligned}$$

Here the meaning of the operator in the denominator is simply given by applying its inverse to the numerator.

Proof. From the variation of constants formulae, it follows that

$$u(t_n + h) = e^{hL}u_n + e^{(t_n+h)L} \int_{t_n}^{t_n+h} e^{-\tau L} \left(b + \sum_{\alpha=1}^k (c_\alpha \sin(\alpha\tau) + d_\alpha \cos(\alpha\tau)) \right) d\tau.$$

The solution of the first integral gives the second term in (2.4.15). It is the same as in the case when the approximation is based on Lagrange interpolating polynomial. Let us now consider the following integral

$$X = \int_{t_n}^{t_n+h} e^{-\tau L} \sin(\alpha\tau) d\tau.$$

Integrating by parts twice leads to an equation for X . Multiplying its solution with $e^{(t_n+h)L}$ gives the relation

$$e^{(t_n+h)L} \int_{t_n}^{t_n+h} e^{-\tau L} \sin(\alpha\tau) d\tau = \phi_{\sin}^{[\alpha]}(t_n, h, L).$$

In the same way it can be shown that

$$e^{(t_n+h)L} \int_{t_n}^{t_n+h} e^{-\tau L} \cos(\alpha\tau) d\tau = \phi_{\cos}^{[\alpha]}(t_n, h, L).$$

□

From the above Lemma, it follows that the flow $\phi_{t, \hat{F}}$ of (2.4.13) is simply given by (2.4.15) with $t_n = 0$ in the expressions for $\phi_{\sin}^{[\alpha]}$ and $\phi_{\cos}^{[\alpha]}$. The corresponding differential equation for $v(t)$ is

$$v'(t) = e^{-tL} (N(u(t_n + t), t_n + t) - T_k(N, t)), \quad v(0) = u_n. \quad (2.4.16)$$

Now the construction of the method follows the same pattern as before. For example if $N(u(t_n + t), t_n + t) \approx N_n + \frac{N_n - N_{n-1}}{\sin h} \sin t$, then we substitute

$$u(t_n + t) = e^{tL}v(t) + t\phi^{[1]}(tL)N_n + \phi_{\sin}^{[1]}(0, t, L) \frac{N_n - N_{n-1}}{\sin h},$$

where $\phi_{\sin}^{[\alpha]}(0, t, L) = \frac{e^{tL} - L \sin t - \cos I}{\alpha^2 I + L^2}$. Applying a traditional RK method to (2.4.16) and then transforming the result into the u variable leads to a new method which is again given by Table 2.5, but with the following new coefficients

$$\begin{aligned}
a_{21}(hL) &= c_2 \phi_{\sin}^{[1]} + \widehat{\phi}_{\sin}^{[1]}, \\
a_{31}(hL) &= c_3 \phi_{\sin}^{[1]} + \widehat{\phi}_{\sin}^{[1]} - \alpha_{32} (1 + \widehat{c}_2) e^{(c_3 - c_2)hL}, \\
a_{41}(hL) &= c_4 \phi_{\sin}^{[1]} + \widehat{\phi}_{\sin}^{[1]} - \alpha_{42} (1 + \widehat{c}_2) e^{(c_4 - c_2)hL} - \alpha_{43} (1 + \widehat{c}_3) e^{(c_4 - c_3)hL}, \\
b_1(hL) &= \phi_{\sin}^{[1]} + \widehat{\phi}_{\sin}^{[1]} - \beta_2 (1 + \widehat{c}_2) e^{(1 - c_2)hL} - \beta_3 (1 + \widehat{c}_3) e^{(1 - c_3)hL} - \beta_4 (1 + \widehat{c}_4) e^{(1 - c_4)hL}, \\
d_{22}(hL) &= -\widehat{\phi}_{\sin}^{[1]}, \\
d_{32}(hL) &= -\widehat{\phi}_{\sin}^{[1]} + \widehat{c}_2 \alpha_{32} e^{(c_3 - c_2)hL}, \\
d_{42}(hL) &= -\widehat{\phi}_{\sin}^{[1]} + \widehat{c}_2 \alpha_{42} e^{(c_4 - c_2)hL} + \widehat{c}_3 \alpha_{43} e^{(c_4 - c_3)hL}, \\
v_{12}(hL) &= -\widehat{\phi}_{\sin}^{[1]} + \widehat{c}_2 \beta_2 e^{(1 - c_2)hL} + \widehat{c}_3 \beta_3 e^{(1 - c_3)hL} + \widehat{c}_4 \beta_4 e^{(1 - c_4)hL},
\end{aligned}$$

where $\widehat{c}_j = \frac{\sin(c_j h)}{\sin h}$, for $j = 1, \dots, 4$, and $\widehat{\phi}_{\sin}^{[1]}(c_j hL) = \frac{1}{h \sin h} \phi_{\sin}^{[1]}(0, c_j h, L)$. Note that in this case we have used again the convention from the beginning of this section.

In Lemma 2.4.1 we give a more general formulation of the functions $\phi_{\sin}^{[\alpha]}$ and $\phi_{\cos}^{[\alpha]}$ than the one needed in the format of the method. The reason why is to show how the initial grid point t_n enters into the functions $\phi_{\sin}^{[\alpha]}$ and $\phi_{\cos}^{[\alpha]}$. Thus, it is clear that if one wants to construct methods based on the idea proposed in [Article 1, p.91] with algebra action arising from the solution of the differential equation $u' = \widehat{F}(u, t)$, then the values of $\phi_{\sin}^{[\alpha]}$ and $\phi_{\cos}^{[\alpha]}$ should be recomputed at the beginning of each step. This increases the computational cost of such methods and they are likely to be competitive with the other methods presented in this section, only if a variable step size strategy is used (see also Chapter 3).

Before we consider exponential integrators based on commutator free Lie group methods, we point out that the idea of generalized integrating factor methods can be further exploited. Instead of applying a Runge–Kutta method to the transformed equation (2.4.14) we can alternatively use a linear multistep method. Thus, it is possible to greatly increase the class of exponential linear multistep methods by constructing Generalized Integrating Factor Linear Multistep (GIF/LM) methods (see [46]). It is worth mentioning that integrating factor Adams methods, ETD Adams–Bashforth and ETD Adams–Moulton methods, considered in Section 1.1, are all special cases of this general class of methods.

2.4.3 Methods based on commutator free Lie group methods

Finally, we consider methods based on Algorithm 2.2.3. Using the formula (2.3.5) for the algebra action and the definition (2.3.6) of the generic presentation of the differential equation (2.4.1), it is easy to observe that the corresponding exponential integrator is obtained from the underlying commutator free Lie group method by simply multiplying each of the

coefficients of the method with the function $\phi^{[1]}$. For example, the method based on the scheme (2.2.2) is given by

$$\left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & I \\ \frac{1}{2}\phi^{[1]} & 0 & 0 & 0 & e^{\frac{hL}{2}} \\ 0 & \frac{1}{2}\phi^{[1]} & 0 & 0 & e^{\frac{hL}{2}} \\ \frac{1}{2}\phi^{[1]} & 0 & 0 & 0 & e^{\frac{hL}{2}} \\ -\frac{1}{2}\phi^{[1]} & 0 & \phi^{[1]} & 0 & e^{\frac{hL}{2}} \end{array} \right\}, \quad (2.4.17)$$

$$\left[\begin{array}{cccc|c} \frac{1}{4}\phi^{[1]} & \frac{1}{6}\phi^{[1]} & \frac{1}{6}\phi^{[1]} & -\frac{1}{12}\phi^{[1]} & e^{\frac{hL}{2}} \\ -\frac{1}{12}\phi^{[1]} & \frac{1}{6}\phi^{[1]} & \frac{1}{6}\phi^{[1]} & \frac{1}{4}\phi^{[1]} & e^{\frac{hL}{2}} \end{array} \right\}$$

where again we use the symbol $\}$ to denote all the substages included in a stage with $J > 1$.

It is possible to rewrite (2.4.17) in an equivalent form which does not involve a splitting of the internal stages. In Table 2.6, we give such a presentation which is rather unpractical, since it is more expensive to implement. Note that in (2.4.17) the first substage of the fourth internal stage is the same as the second internal stage. This form of the method makes it particularly suitable for numerical implementation. We have included the other formulation as well in order to show that this method also fits into the framework of the “pure” RK methods.

$$\left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & I \\ \frac{1}{2}\phi^{[1]} & 0 & 0 & 0 & e^{\frac{1}{2}hL} \\ 0 & \frac{1}{2}\phi^{[1]} & 0 & 0 & e^{\frac{1}{2}hL} \\ \frac{1}{2}\phi^{[1]}\left(\frac{hL}{2}\right)\left(e^{\frac{hL}{2}} - I\right) & 0 & \phi^{[1]}\left(\frac{hL}{2}\right) & 0 & e^{hL} \\ \hline \frac{1}{2}\phi^{[1]} - \frac{1}{3}\phi^{[1]}\left(\frac{hL}{2}\right) & \frac{1}{3}\phi^{[1]} & \frac{1}{3}\phi^{[1]} & -\frac{1}{6}\phi^{[1]} + \frac{1}{3}\phi^{[1]}\left(\frac{hL}{2}\right) & e^{hL} \end{array} \right]$$

Table 2.6: Fourth order CF method.

In conclusion, we briefly discuss accuracy, stability and minimizing the number of the function evaluations, for all of the above presented methods.

For *stiff* semilinear problems (1.1.1) the RKMK methods with exact Exp suffer from a step size restriction. This is due to the small convergence radius of the $d\text{Exp}^{-1}$ map. Therefore this class of methods are unpractical for such problems. The GIF/RK methods have several orders in magnitude improvement in accuracy over the standard IF RK methods (see the numerical experiments Section 3.4), but also suffer from stability restrictions, especially for problems with eigenvalues of the linearization of the nonlinear part lying on the

imaginary axis (see [36, Article 5]). The most expensive methods are the Crouch–Grossman methods. Thus, combining the lower computational cost with the relatively good stability properties and accuracy for the commutator free Lie group methods, suggests that for stiff problems they are the best exponential integrators arising from the framework of Lie group methods.

When the equation (1.1.1) comes from the discretization of a highly-oscillatory problems, for example the nonlinear Schrödinger equation, then the RKMK methods does not suffer so dramatically from numerical instability like in the previous case. The best Lie group method in this case seems to be the IF RK methods [3].

Chapter 3

Implementation Issues and Numerical Experiments

In this chapter we address some practical issues regarding the implementation of the exponential integrators considered in the previous two chapters. We also present results from numerical experiments, with some of the discussed methods, for several numerical examples.

The main computational challenge in the implementation of every exponential integrator comes from the necessity of computing the exponential and the related $\phi^{[l]}$ functions (1.2.2) by some fast and numerically stable algorithm. A great variety of methods for computing the matrix exponential operator are available in the literature. We refer to [47] and the references within for a comprehensive review of these methods. In this chapter we focus on different numerical techniques for computing the most commonly used ETD $\phi^{[i]}$ functions (1.1.6). We also discuss the main advantages and disadvantages of the considered methods as well as some practical issues concerning variable step size implementations.

The chapter is organized as follows: We first consider, in Section 3.1, methods based on Schur decomposition followed by higher order rational Chebyshev or Padé approximations to the function $\phi^{[i]}$. Next, in Section 3.2 we present a technique, which allows us to approximate the action of each $\phi^{[i]}$ function on a given vector by means of a projection process onto a small Krylov subspace. In Section 3.3 we discuss methods based on the Cauchy integral formula and comment its implementation in some special cases. Finally, in Section 3.4 we compare different exponential integrators in several numerical examples.

3.1 Decomposition methods

Let us recall the explicit form of the ETD $\phi^{[i]}$ functions, which appear in the formulation of most of the exponential integrators presented so far

$$\phi^{[1]}(z) = \frac{e^z - 1}{z}, \quad \phi^{[i+1]}(z) = \frac{\phi^{[i]}(z) - \frac{1}{i!}}{z}, \quad \text{for } i = 2, 3, \dots \quad (3.1.1)$$

Like we saw in the previous two chapters, other functions can also enter in the format of an exponential integrator. Examples are integrating factor RK methods and methods based on the the Lie group theory. However, in most of the known cases, the new functions are closely related with functions from (3.1.1). Therefore, here we restrict our considerations only to the ETD $\phi^{[i]}$ functions.

A straightforward implementation of (3.1.1) suffers, for small z , from cancellation errors. An illustration of this phenomena, for the first function $\phi^{[1]}$, is given in [35, Table 2.1]. The cancellation errors are even more severe for the second and the subsequent $\phi^{[i]}$ functions. In practice, the arguments of the functions in (3.1.1) are matrices of the form $A = \gamma h L$, where $\gamma \in \mathbb{R}$, h is the step size and L is the discretized linear operator. The size d of the matrix L depends of the spatial discretization and the dimensionality of the problem. Usually, d is very large and thus we need fast, reliable and numerically stable algorithm for computing the functions (3.1.1).

At the heart of all decomposition methods is the similarity transformation

$$A = SBS^{-1}, \quad (3.1.2)$$

where B is such that $\phi^{[i]}(B)$ is easy to compute. Then

$$\phi^{[i]}(A) = S\phi^{[i]}(B)S^{-1}. \quad (3.1.3)$$

The choice of the matrices S and B in (3.1.2) usually involves two conflicting tasks: Make B close to diagonal so that $\phi^{[i]}(B)$ is easy to compute and make S well conditioned so that errors in evaluating $\phi^{[i]}(B)$ are not magnified. Thus, it is natural to consider methods based on the Schur decomposition so that S is unitary and therefore well conditioned. Next we present a general algorithm, first proposed in [16], which employs the block form of the Parlett recurrence and does not impose any restrictions on the argument matrix A .

3.1.1 Block Schur–Parlett algorithm

The first step of the algorithm is to compute the Schur decomposition $A = QTQ^*$, where Q is unitary and T is upper triangular. This can be achieved by the standard QR algorithm [22]. We now need to calculate the matrices $F^{[i]} = \phi^{[i]}(T)$. Assuming that the diagonal elements $f_{kk}^{[i]}$ of $F^{[i]}$ are already known, we can compute the whole $F^{[i]}$ using the scalar Parlett recurrence [56]. This is exactly how the MATLAB 6.5 (R13) built in function `funm` works. The problem with this approach is that it fails to produce correct results, in floating point arithmetic, when the eigenvalues of T are equal or close to each other.

To avoid this problem, the authors in [16] propose to reorder T into a block upper triangular matrix \tilde{T} . The diagonal blocks \tilde{T}_{kk} of \tilde{T} are constructed in such a way that the eigenvalues of each diagonal block are “close” to each other and the distinct diagonal blocks are with “sufficiently distinct” eigenvalues. To define the meaning of “close” and “sufficiently distinct”, the authors introduce a special parameter in the format of the method. Since \tilde{T} is block upper diagonal, so is $\tilde{F}^{[i]} = \phi^{[i]}(\tilde{T})$. From the commutativity relation

$\tilde{F}^{[i]}\tilde{T} = \tilde{T}\tilde{F}^{[i]}$ (see the explicit form of the ETD functions (1.2.15)), it follows the following block form of the Parlett recurrence

$$\tilde{T}_{kk}\tilde{F}_{kl}^{[i]} - \tilde{F}_{kl}^{[i]}\tilde{T}_{ll} = \tilde{F}_{kk}^{[i]}\tilde{T}_{kl} - \tilde{T}_{kl}\tilde{F}_{ll}^{[i]} + \sum_{j=k+1}^{l-1} \left(\tilde{F}_{kj}^{[i]}\tilde{T}_{jl} - \tilde{T}_{kj}\tilde{F}_{jl}^{[i]} \right), \quad (3.1.4)$$

where $\tilde{T} = (\tilde{T}_{kl})$ and $\tilde{F}^{[i]} = (\tilde{F}_{kl}^{[i]})$. If the diagonal blocks $\tilde{F}_{kk}^{[i]}$ are already evaluated, the above relation allows us to compute the whole matrix $\tilde{F}^{[i]}$ by solving the *Sylvester equation* (3.1.4) with respect to $\tilde{F}_{kl}^{[i]}$. The imposed requirements for the eigenvalues of \tilde{T}_{kk} guarantee that the equation (3.1.4) is nonsingular and well conditioned [16]. Thus, what we still need to specify is how to compute the diagonal blocks $\tilde{F}_{kk}^{[i]}$. In [16], the authors suggest to use a suitable Taylor series truncation or Padé approximation. These are local approximations which are very accurate near the origin, but may suffer in accuracy away from it. In addition, the accuracy of such approaches depends of the norm of $\tilde{F}_{kk}^{[i]}$ and usually, when the norm is large, they require a scaling and squaring strategy [47]. For the first function $\phi^{[1]}$ an analogous formula, similar to the well-known fundamental property of the exponential function, is proposed in [28]. It is based on the equalities $\phi^{[1]}(2z) = (e^z + 1)\phi^{[1]}(z)/2$ and $e^{2z} = e^ze^z$. In general, for $i > 1$, we do not have a simple generalization of the above formulas and thus an alternative approach for computing $\tilde{F}_{kk}^{[i]}$ can be to use higher order Chebyshev (uniform) rational approximations [68]. We discuss this type of approximations in detail in Subsection 3.1.2.

Next we summarize the overall block Schur–Parlett algorithm for computing $\phi^{[i]}(A)$ for a general matrix A and $i = 1, 2, \dots$

Algorithm 3.1.1. (Block Schur–Parlett algorithm)

- Compute the Schur decomposition $A = QTQ^*$.
- Reorder T into a block upper triangular matrix \tilde{T} .
- Compute $\phi^{[i]}(\tilde{T}_{kk})$ for all diagonal blocks \tilde{T}_{kk} .
- Find $\phi^{[i]}(\tilde{T})$ by solving the Sylvester equation (3.1.4).
- Compute $\phi^{[i]}(A) = Q\phi^{[i]}(\tilde{T})Q^*$.

The cost of Algorithm 3.1.1 depends from the eigenvalue distribution of A , and it is between $28d^3$ and $d^4/3$ flops (see [16]) for each i , where d is the dimensionality of A .

The advantages of this method are that it is numerically stable and works without any restrictions on the structure of the matrix A . The main disadvantage is its higher computational cost, which makes it applicable only if integrators with fixed stepsize are used. In this case all $\phi^{[i]}$ functions can be computed only once, at the beginning of integration process. Thus, the above method provides a benchmark for the computational cost of a method.

In the case when A is real symmetric (or complex Hermitian) a cheaper method, also based on the Schur decomposition, can be constructed. In the next subsection we generalize the approach proposed in [40] for computing the matrix exponential operator and later extended in [41] for computing the first function $\phi^{[1]}$, to a general algorithm for computing all the functions from (3.1.1).

3.1.2 Tridiagonal reduction

A common first step in the computation of the eigenvalues and the eigenvectors of a *symmetric* matrix A is to use a tridiagonal reduction of the form $A = QTQ^T$, where Q is orthogonal and T is symmetric tridiagonal. Such a representation of A can be obtained by using Householder reflections [22]. Thus, according to (3.1.3), the computation of $\phi^{[i]}(A)$ requires the value $\phi^{[i]}(T)$. In [40, 41] efficient numerical algorithms for computing the exponential and the $\phi^{[1]}$ function, based on the above tridiagonal reduction followed by Chebyshev (uniform) rational approximation, are developed. Here we utilize this approach and propose how it can be generalized, so that each of the functions from (3.1.1) can be efficiently computed.

We first consider, how to find Chebyshev rational approximations to e^T , for a given tridiagonal matrix T . The key idea is to compute the largest eigenvalue λ_1 of T (for example by bisection method) and then make use of the following obvious equality $e^T = e^{\lambda_1} e^{T - \lambda_1 I}$, where I is $d \times d$ identity matrix. The eigenvalues of $T - \lambda_1 I$ are always located in the interval $(-\infty, 0]$. This allows us to approximate $e^{T - \lambda_1 I}$ by a rational function $R_p(z) = N_p(z)/D_p(z)$ (where N_p and D_p are polynomials of z of degree p), which minimizes the maximum error for approximating e^z on $(-\infty, 0]$. An approximation to e^T is then obtained by

$$e^T \approx e^{\lambda_1} R_p(T - \lambda_1 I). \quad (3.1.5)$$

What we gain in this way is that, regardless of the spectrum of T , we can always use a rational Chebyshev approximation to $e^{T - \lambda_1 I}$ in the interval $(-\infty, 0]$, which has the same coefficients. If we choose to represent R_p via its partial fraction expansion (see [21])

$$R_p(z) = \alpha_0^{(p)} + \sum_{j=1}^p \frac{\alpha_j^{(p)}}{z - \theta_j^{(p)}}, \quad (3.1.6)$$

the coefficients $\alpha_j^{(p)}$ and the poles $\theta_j^{(p)}$ of R_p can be computed once and for all. To achieve standard double precision (64-bit floating point numbers), it is sufficient to choose $p = 14$. Since the coefficients and the poles appears in complex conjugate pairs, for even p , it is enough to add just the first $p/2$ terms in the sum in (3.1.6), and then double the real part of the result. This in fact leads to significant computational savings, since it halves the number of matrix inversions in the corresponding formula

$$R_p(T - \lambda_1 I) = \alpha_0^{(p)} I + \sum_{j=1}^p \alpha_j^{(p)} \left[T - (\lambda_1 + \theta_j^{(p)}) I \right]^{-1}. \quad (3.1.7)$$

The values of $\alpha_j^{(p)}$ and $\theta_j^{(p)}$, for $p = 14$ and 16 , are listed in [41, Table 2].

Here we have chosen a *diagonal* rational approximation R_p , that is, the numerator N_p has the same degree as the denominator D_p . We note however, that alternative strategies (e.g. using L-stable approximations) might also be considered without altering the principle of the method.

Before we consider how the approximation (3.1.5) can be used to approximate any of the functions $\phi^{[i]}$ from (3.1.1), we mention that, once we have the largest eigenvalue λ_1 of \mathbf{T} , we can always calculate a Chebyshev rational approximation to $\phi^{[i]}(z)$ on the interval $(-\infty, \lambda_1]$ and then used it to approximate $\phi^{[i]}(\mathbf{T})$. However, this approach is rather unpractical since it highly depends on the value λ_1 . Therefore we need to recompute the coefficients of the approximation for every different λ_1 , which is quite a costly task.

An alternative approach for computing $\phi^{[i]}(\mathbf{T})$, in the case where the largest eigenvalue of \mathbf{T} belongs to the interval $(-\infty, 0]$, is to replace $e^{\mathbf{T}}$ in the definition of each of the functions $\phi^{[i]}$ by its Chebyshev rational approximation (3.1.5). Thus, for example, the first function $\phi^{[1]}$ can be approximated by

$$\phi^{[1]}(\mathbf{T}) \approx \mathbf{T}^{-1} \left[\left(e^{\lambda_1} \alpha_0^{(p)} - 1 \right) I + e^{\lambda_1} \sum_{j=1}^p \alpha_j^{(p)} \left[\mathbf{T} - (\lambda_1 + \theta_j^{(p)}) I \right]^{-1} \right].$$

The above formula should not be used when λ_1 is positive. The problem in this case is that for each of the functions $\phi^{[i]}$ we can not just shift the argument by $\lambda_1 I$, since the relation between $\phi^{[i]}(\mathbf{T})$ and $\phi^{[i]}(\mathbf{T} - \lambda_1 I)$ is not that simple as it is for the exponential function.

A different approach for approximating $\phi^{[1]}(\mathbf{T})$, in the case when $\lambda_1 > 0$, is proposed in [41]. It is based on the observation that for

$$B = \begin{bmatrix} \mathbf{T} & I \\ 0 & 0 \end{bmatrix},$$

the exponential of B is given by

$$e^B = \begin{bmatrix} e^{\mathbf{T}} & \phi^{[1]}(\mathbf{T}) \\ 0 & I \end{bmatrix}. \quad (3.1.8)$$

Therefore, applying the approximation (3.1.5) with R_p given by (3.1.7) and $\mathbf{T} = B$, we obtain

$$e^B = e^{\lambda_1} e^{B - \lambda_1 I} \approx e^{\lambda_1} \left\{ \alpha_0^{(p)} I + \sum_{j=1}^p \alpha_j^{(p)} \left[B - (\lambda_1 + \theta_j^{(p)}) I \right]^{-1} \right\}. \quad (3.1.9)$$

Equating the entries in positions (1,2) of (3.1.8) and (3.1.9) leads to the following approximation for $\phi^{[1]}(\mathbf{T})$

$$\phi^{[1]}(\mathbf{T}) \approx e^{\lambda_1} \sum_{j=1}^p \frac{\alpha_j^{(p)}}{\lambda_1 + \theta_j^{(p)}} \left[\mathbf{T} - (\lambda_1 + \theta_j^{(p)}) I \right]^{-1}.$$

We have found that, for λ_1 positive, the same idea can be easily generalized to the case when approximations to $\phi^{[i]}(\mathbf{T})$ for $i > 1$ are needed. To approximate the function $\phi^{[2]}(\mathbf{T})$, we take the matrix B to be of the form

$$B = \begin{bmatrix} \mathbf{T} & 0 & I \\ 0 & 0 & 0 \\ 0 & I & 0 \end{bmatrix}.$$

It is easy to see, for example by direct computation, that its exponential is given by

$$e^B = \begin{bmatrix} e^{\mathbf{T}} & \phi^{[2]}(\mathbf{T}) & \phi^{[1]}(\mathbf{T}) \\ 0 & I & 0 \\ 0 & I & I \end{bmatrix}.$$

As before, based on formulas (3.1.5) and (3.1.7), we obtain the following approximation

$$\phi^{[2]}(\mathbf{T}) \approx e^{\lambda_1} \sum_{j=1}^p \frac{\alpha_j^{(p)}}{(\lambda_1 + \theta_j^{(p)})^2} \left[\mathbf{T} - (\lambda_1 + \theta_j^{(p)})I \right]^{-1}.$$

Approximations for the rest of the functions $\phi^{[i]}$, for $\lambda_1 > 0$, can be computed by a straight forward generalization of the above process.

In the next algorithm we summarize the method based on the tridiagonal reduction for computing any of the functions $\phi^{[i]}(A)$ for a symmetric matrix A and $i = 1, 2, \dots$

Algorithm 3.1.2. (Tridiagonal Reduction)

- Calculate a symmetric tridiagonal reduction $A = Q\mathbf{T}Q^T$.
- Find the largest eigenvalue λ_1 of \mathbf{T} .
- Compute $\phi^{[i]}(\mathbf{T})$ by a Chebyshev rational approximation with respect to the value of λ_1 .
- Calculate $\phi^{[i]}(A)$ by $\phi^{[i]}(A) = Q\phi^{[i]}(\mathbf{T})Q^T$.

The cost of the algorithm, for each i , is $\mathcal{O}(\frac{4}{3}d^3 + d + d^2 + 2d^3)$, where d is the dimensionality of A . Therefore, the total number of operations for computing each of the functions $\phi^{[i]}$ is $\mathcal{O}(\frac{10}{3}d^3)$. In the case of constant step size, once we have computed the first function $\phi^{[1]}$, we can reduce the work for computing the other $\phi^{[i]}$ functions by reusing the tridiagonal decomposition.

If the matrix A is symmetric and tridiagonal, the above algorithm requires only $\mathcal{O}(d^2)$ operations, since its first and last step are not needed. In this case instead of computing all the $\phi^{[i]}$ functions once, before the integration begins, and then apply them at every step to a different vectors v , we can repeatedly compute the action of each $\phi^{[i]}$ on its corresponding

vector v . The total number of operations in this case still does not exceed the work required to compute the matrix-vector product $\phi^{[i]}$ times v , especially when $d \gg p$. This is because the inverse matrices in the third step of the algorithm are replaced by solutions of linear systems with tridiagonal coefficient matrices. In general, algorithms designed to solve such systems require $\mathcal{O}(d)$ operations, which leads to $\mathcal{O}(pd)$ flops needed to compute the action $\phi^{[i]}v$. This is to be compared with $\mathcal{O}(d^2)$ operations needed for a matrix-vector product. In [Article 2 p.107], we present several direct methods for solving linear systems of equations, some of which can be easily adopted to the tridiagonal case considered here. We will comment more on this in Section 3.3.

When A is tridiagonal, the above approach allows to preserve the total number of operations, needed to implement an exponential integrator, approximately the same even when a variable step size strategy is used. The advantage of the Algorithm 3.1.2 is that it is less expensive than Algorithm 3.1.1. Its disadvantages are that it is applicable only in the case when the matrix A is symmetric (Hermitian) and that it cannot be used with variable step size strategy, unless in the case when A is tridiagonal.

We next consider a method which is entirely based on the idea of approximating the action of each of the functions $\phi^{[i]}$ on a given state vector v . This approach is useful for implementations employing a variable step size strategy.

3.2 Krylov subspace approximations

Since the mid-eighties, the idea of using the Krylov subspace approximations to the action of the evolution operators has been studied by many authors. A short and definitely incomplete list of publications on this topic includes [17, 18, 21, 27, 28, 57, 58]. The convergence properties of the action of the matrix exponential operator are investigated in [17, 21, 58]. Later in [27] sharper error estimates are derived. It is also shown that, unless a good preconditioner is not available, the Krylov approximation to $e^A v$ converges faster than its corresponding approximation to the solution of the linear system $(I - A)x = v$. An approximation to the action of the first function $\phi^{[1]}$, based on the Krylov subspace approximation techniques, was considered in [58] and later studied in [27]. It was found that it obeys the same error bounds as the approximation to the matrix exponential operator. This provided the initial motivation for developing a Rosenbrock-like exponential integrators [28].

The main idea of the Krylov subspace technique is to approximately project the action of the evolution operator $\phi^{[i]}(A)$ on a state vector $v \in \mathbb{C}^d$, to a small Krylov subspace

$$K_m \equiv \text{span}\{v, Av, \dots, A^{m-1}v\}.$$

Usually, even for a relatively small $m \ll d$, an accurate approximation can be obtained [58]. Thus, the approach is to approximate the action of $\phi^{[i]}(A)$ by the action of $\phi^{[i]}$ applied to the projection of A on the smaller subspace K_m .

It is convenient to choose an orthogonal basis $V_m = [v_1, v_2, \dots, v_m]$ of K_m . It can be generated by the Arnoldi algorithm, with $v_1 = v/\|v\|_2$ as an initial vector.

Algorithm 3.2.1. (Arnoldi)

```

Compute  $v_1 = v/\|v\|_2$ .
for  $j = 1, 2, \dots, m$  do
  for  $i = 1, 2, \dots, j$  do
     $h_{i,j} = (Av_j, v_i)$ ,
  end
   $w = Av_j - \sum_{i=1}^j h_{i,j}v_i$ ,
   $h_{j+1,j} = \|w\|_2$ ,  $v_{j+1} = w/h_{j+1,j}$ ,
end

```

Alternatively, the Lanczos algorithm for generating a *biorthogonal* basis on the subspace K_m , can also be used (see [58]).

Let H_m be the $m \times m$ upper Hessenberg matrix consisting of the coefficients $h_{i,j}$. Since the Algorithm 3.2.1 is just a modified Gram-Schmidt process, the following relation holds

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T, \quad (3.2.1)$$

where e_i denotes the i^{th} unit vector in \mathbb{R}^m . Using the fact that V_m is orthogonal, from the above relation, it follows that $V_m^T AV_m = H_m$. Therefore, H_m represents the orthogonal projection of A to the subspace K_m , with respect to the basis V_m . Similarly, $V_m V_m^T \phi^{[i]}(A)v$ is the projection of $\phi^{[i]}(A)v$ on K_m , that is the closest approximation to $\phi^{[i]}(A)v$ from K_m . If $\beta \equiv \|v\|_2$ then $v = \beta v_1$ and since $v_1 = V_m e_1$, we have

$$\begin{aligned} \phi^{[i]}(A)v &\approx V_m V_m^T \phi^{[i]}(A)v = \beta V_m V_m^T \phi^{[i]}(A)v_1 \\ &= \beta V_m V_m^T \phi^{[i]}(A)V_m e_1. \end{aligned} \quad (3.2.2)$$

From the computational point of view, the above formula is not useful, since it still involves operations with the big matrix A . To avoid this, the idea is to replace the term $V_m^T \phi^{[i]}(A)V_m$ in (3.2.2) by suitable approximation. Note that $V_m^T \phi^{[i]}(A)V_m$ is an $m \times m$ matrix. By induction, from (3.2.1), one can prove [58, Lemma 3.1] that

$$p_{m-1}(A)v = \beta V_m p_{m-1}(H_m)e_1, \text{ for all polynomials } p_{m-1} \text{ of degree } \leq m-1.$$

Therefore, it is natural to approximate $V_m^T \phi^{[i]}(A)V_m$ by $\phi^{[i]}(H_m)$. From (3.2.2) we obtain

$$\phi^{[i]}(A)v \approx \beta V_m \phi^{[i]}(H_m)e_1. \quad (3.2.3)$$

The approximation (3.2.3) can also be derived from the standard Krylov approximation to the solution of linear systems of equations [57] and the Cauchy integral formula (see [27]).

The advantage of using (3.2.3) is that instead of working with the original large matrix A we use its orthogonal approximation H_m , which has much smaller dimension. The action $\phi^{[i]}(A)v$ is then computed in $\mathcal{O}(md)$ operations by using only matrix-vector multiplications between elements with the original large size d . Thus, when $m \ll d$ the cost of computing the expression $\beta V_m \phi^{[i]}(H_m)e_1$ is usually much less than the cost needed to compute $\phi^{[i]}(A)v$. In addition, when A is Hermitian, it is possible to speed up the whole process by applying the Arnoldi algorithm to a shifted and inverted matrix [66]. It is chosen in such a way that it emphasizes the eigenvalues of importance, so that the algorithm can quickly find them. This approach relies on the fact that efficient preconditioned iterative solver is used to find the action of the inverse matrix.

The computation of $\phi^{[i]}(H_m)e_1$ requires $\mathcal{O}(m^3)$ operations in general and only $\mathcal{O}(m^2)$ if the matrix H_m is tridiagonal (e.g. A is symmetric). It can be done by using a Chebyshev rational approximation, evaluated by partial fraction expansion (see Subsection 3.1.2).

The main computational cost of an exponential integrator, using Krylov subspace approximation technique, comes from the repeated application of Algorithm 3.2.1. At every step we need to construct several bases of Krylov subspaces with respect to the same matrix A and different vectors v . In general Algorithm 3.2.1 requires $\mathcal{O}(md^2)$ operations. We mention also that it assumes exact arithmetic is used. In practice, round off errors and cancellations might cause a loss of the orthogonality between the vectors v_i . A significant improvement in the performance can be achieved by using double orthogonalization [38]. In addition, a convergence criterion is needed to determine the value of m that gives a sufficiently accurate approximation. Thus, we conclude that the above approach is preferable in the case when the number of Krylov bases needed can be significantly reduced. Efficient exponential integrators based on this *reduced* idea are developed in [28]. The advantage of the Krylov subspace approximation technique is that, implementations based on a variable stepsize strategy, do not increase the total computational cost of the integrator.

3.3 Cauchy integral approach

The last approach for computing the functions $\phi^{[i]}$ or their action on a given vector v , which we consider, is introduced in [35]. It is based on the Cauchy integral formula

$$\phi^{[i]}(A) = \frac{1}{2\pi i} \int_{\Gamma_A} \phi^{[i]}(\lambda)(\lambda I - A)^{-1} d\lambda, \quad (3.3.1)$$

where Γ_A is a contour in the complex plane that encloses the eigenvalue of A , and it is also well separated from 0. It is practical to choose the contour Γ_A to be a circle centered on the real axis. In this way when A is real, based on the symmetry, one can evaluate the integral only on the upper half of the circle and then double the real part of the result. To approximate the integral in (3.3.1), the authors in [35] propose to use the trapezoid rule,

which converges exponentially [65]. Therefore, we obtain the following approximation

$$\phi^{[i]}(A) \approx \frac{1}{k} \sum_{j=1}^k \lambda_j \phi^{[i]}(\lambda_j) (\lambda_j I - A)^{-1}, \quad (3.3.2)$$

where k is the number of the equally spaced points λ_j along the contour Γ_A . Usually, values of $k = 32$ or $k = 64$, are sufficient to insure correct computations.

For problems with diagonal matrix A it is beneficial to choose the contour Γ_A to be, in addition, a circle centered at A . Thus, (3.3.2) simply reduces to the **mean** of $\phi^{[i]}$ over the equally spaced points along Γ_A (or again just half of them for a real A). When the matrix A is non-diagonal, the cost for computing an approximation to the functions $\phi^{[i]}$ increases. This is due to the number of the matrix inverses involved in (3.3.2). In the case of constant stepsize, the total impact on the computational time is still small, since all the $\phi^{[i]}$ functions can be evaluated only once before the integration begins. However, in the case of variable step size, direct application of (3.3.2) leads to significant increase in the computational work. To gain more insight into how the formula (3.3.1) can be effectively used, in the case of variable step size, we recall that the matrix $A = \gamma h L$, where $\gamma \in \mathbb{R}$, h is the step size and L is the discretized linear operator. Note that every time when we need to change the step size h , it is enough to change only the parameter γ .

The idea now is to represent $\phi^{[i]}(\gamma h L)$ in such a way that the number of the matrix inverses used in (3.3.1), respectively in (3.3.2), is independent of γ . If we can choose a suitable contour Γ , such that for all different values of γ , which appear in the integration process, it encloses the eigenvalues of $\gamma h L$ and $\gamma \Gamma$ is well separated from 0 then we can compute each of the functions $\phi^{[i]}$ by the following formula

$$\phi^{[i]}(A) = \phi^{[i]}(\gamma h L) = \frac{1}{2\pi i} \int_{\Gamma} \phi^{[i]}(\gamma \lambda) (\lambda I - h L)^{-1} d\lambda. \quad (3.3.3)$$

As before, approximating the integral in (3.3.3) by the trapezoid rule, we get

$$\phi^{[i]}(A) \approx \frac{1}{k} \sum_{j=1}^k \lambda_j \phi^{[i]}(\gamma \lambda_j) (\lambda_j I - h L)^{-1}, \quad (3.3.4)$$

where now λ_j are the equally spaced points along the contour Γ . The above formula allows to reduce the computational work needed to evaluate the functions $\phi^{[i]}$ or their action on a given vector v , in the case when a variable step size is used. The main advantage comes from the fact that the inverse matrices in (3.3.4) no longer depend of γ . However, in any case we have to compute k (or $k/2$) matrix inverses.

In the case when the matrix L arises from a finite difference approximation to a second order partial differential operator, we can benefit from its sparse block structure. Similar to the idea presented in Subsection 3.1.2, in this case, we can also evaluate the action of the function $\phi^{[i]}$ on a given vector v . If the number of operations required does not exceed $\mathcal{O}(d^2)$ the cost of a matrix-vector product then we obtain a competitive method. Note that

in general the matrix $\phi^{[i]}(\gamma h L)$ does not retain the sparse block structure of L . Thus, what we need is an efficient method for solving special sparse block linear systems of equations. When the matrix $-L$ is symmetric (Hermitian) and positive definite the most favourable methods are *preconditioned conjugate gradient* and *multigrid methods* [34]. They are based on the idea of approximating the solution by an iterative procedure and usually require $\mathcal{O}(d)$ operations. The problem with these methods is that they require a good preconditioner or a rather complex algorithm with considerable overhead to organize the computations. In addition, we note that the matrices in (3.3.4) are symmetric (Hermitian) only if $\lambda_j \in \mathbb{R}$.

Alternatively, direct methods for solving linear systems of equations can be used. In order to be competitive, such methods are specially design to take advantage from the sparse block structure of the coefficient matrix. In general, the structure of L depends on the dimensionality of the problem, the type of the approximation and the boundary conditions imposed. When L is block tridiagonal, two methods are of practical importance: the Buneman variant of *cyclic reduction* [6] and the decomposition method based on the *fast Fourier transform* (FFT) [11]. Both of these methods compute the solution in $\mathcal{O}(d \log_2 n)$ operations, where n is the block size of L . Combination of the above two methods known as Fourier analysis-cyclic reduction (FACR) is proposed in [31]. The asymptotic operation count for this method is reduced to $\mathcal{O}(d \log_2 \log_2 n)$ (see [64]). A restriction for all of these methods is that n should be power of two or a composite of small primes.

In [Article 2 p.107], we present a method for solving tridiagonal block Teoplitz linear systems of equations, which does not impose any restrictions on n . The method is based on a modified LU factorization and it is fully applicable to all the matrices in (3.3.4). In [Article 3 p.133], the same idea is generalized to the case when the coefficient matrix has pentadiagonal block circulant structure. The complexity of this methods is $\mathcal{O}(d^2)$. Significant computational savings can be achieved by using the techniques of solving linear systems with multiple right hand sides. Since the methods are based on the LU decomposition idea, we can factorize the k ($k/2$) coefficient matrices arising from (3.3.4) once and for all and then use only the back-substitution formulas to find their action on a given vector v . Thus, for $k \ll d$ we obtain methods which can be implemented with a variable step size, without to increase the total computational work.

Before we consider some numerical experiments, we summarize the main advantages and disadvantages of the different methods for computing the functions $\phi^{[i]}$ or their actions on a vector, presented in this chapter.

- Methods based on Algorithm 3.1.1 are expensive and not suitable for implementations using variable step size. Their main advantage is that they do not place any restrictions on the coefficient matrix.
- Methods using Krylov subspace approximation techniques are suitable for implementations involving variable step size, but are applicable only in the case when the number of the needed Krylov bases can be significantly reduced.
- Methods based on the Cauchy integral formula are suitable for both constant and

variable stepsize implementations. Particularly cheap methods, in the case of variable stepsize, can be obtained if the coefficient matrix has a sparse block structure. Alternatively, for tridiagonal matrices, Algorithm 3.1.2 can be used.

3.4 Numerical experiments

In this section, we present numerical experiments with some of the exponential integrators studied in the thesis. The examples under consideration are Kuramoto-Sivashinsky, Allen-Cahn and Korteweg de Vries (KdV) equations. Since all of the tested exponential integrators, except the method of Hochbruck–Ostermann given in Table 1.15, are based on the nonstiff order conditions, to avoid possible order reduction, we consider examples where the nonlinear term has sufficient spatial regularity. In general, for applications concerning PDEs, the classical order of convergence is not always obtained (see Subsection 1.2.5). Order reduction, due to the lack of sufficient temporal and spatial smoothness, should be expected. For parabolic problems with periodic boundary conditions, full order of convergence can be observed [29, 30]. In our numerical experiments, for the Kuramoto-Sivashinsky and the KdV equations, we also impose periodic boundary conditions. Thus, the linear term L has a diagonal structure and the computation of the $\phi^{[i]}$ functions by the Cauchy integral formula (see Section 3.3) simply reduce to the **mean** of $\phi^{[i]}$ over the contour Γ .

For all numerical experiments, a constant step size is used. Computation of the $\phi^{[i]}$ functions, in the non-diagonal example (Allen-Cahn equation), is also done by the Cauchy integral approach. In this case, the total computational cost of a method is still small, since all the matrix inverses in (3.3.4) are evaluated only once before the time-stepping begins.

Extensive numerical tests between *linearly implicit* methods [2], *split step* methods [59] and exponential integrators based on the linear multistep and multistage idea are presented in [14, 35]. There the authors conclude that the Exponential Time Differencing Runge–Kutta (ETD RK) methods, presented in Subsection 1.2.3, consistently out perform all the other methods.

Comparison between the stability regions for different exponential integrators based on Lie group methods, applied to semi-discretized stiff problems, is given in [36]. There the author concludes that, for such kind of problems, the RKMK methods (see Section 2.2) suffer from numerical instability. It is due to the small convergence radius of the dExp^{-1} map.

Taking into account the above arguments, we have chosen to implement the following exponential integrators:

- **ETD RK3(CM)** The third order method of Cox–Matthews given in Table 1.9;
- **ETD2RK3** The third order method from Table 1.12;
- **ETD2CF3** The third order method from Table 1.13;
- **ETD RK4(CM)** The fourth order method of Cox–Matthews given in Table 1.10;

- **ETD RK4(Kr)** The fourth order method of Krogstad given in Table 1.11;
- **ETD RK4(Min)** The fourth order method from Table 1.8;
- **ETD RK4(HO)** The fourth order method of Hochbruck–Ostermann given in Table 1.15;
- **IF RK4** The fourth order Integrating Factor Runge–Kutta method based on the classical fourth order Runge–Kutta method (1.3.5);
- **GIF1/RK4** The fourth order Generalized Integrating Factor Runge–Kutta method from Table 2.4 with the classical fourth order Runge–Kutta coefficients (1.3.5);
- **GIF2/RK4** The fourth order Generalized Integrating Factor Runge–Kutta method from Table 2.5 with the classical fourth order Runge–Kutta coefficients (1.3.5);
- **GIF3/RK4** The fourth order Generalized Integrating Factor Runge–Kutta method with third order polynomial approximation to the nonlinear part N , also with the classical fourth order Runge–Kutta coefficients (1.3.5);
- **CF4** The fourth order Commutator Free Lie group method given in Table 2.6.

For each of the three test problems we compare accuracy against step size. The results are given in different figures for each example.

The Kuramoto-Sivashinsky equation

The first example is the Kuramoto-Sivashinsky equation

$$u_t = -uu_x - u_{xx} - u_{xxx}, \quad x \in [0, 32\pi]$$

with periodic boundary conditions and with initial condition borrowed from [35]

$$u(x, 0) = \cos\left(\frac{x}{16}\right) \left(1 + \sin\left(\frac{x}{16}\right)\right).$$

A 128-point Fourier spectral discretization in space is used. Since the boundary conditions are periodic the transformed equation in the Fourier space can be represented in the form (1.1.1), the linear and nonlinear parts are defined as

$$(L\hat{u})(k) = (k^2 - k^4)\hat{u}(k), \quad N(\hat{u}) = -\frac{ik}{2}(F((F^{-1}(\hat{u}))^2)),$$

where F denotes the discrete Fourier transform. The integration in time is done entirely in the Fourier space until $t = 65$. The results for the different numerical schemes are plotted in Figure 3.1.

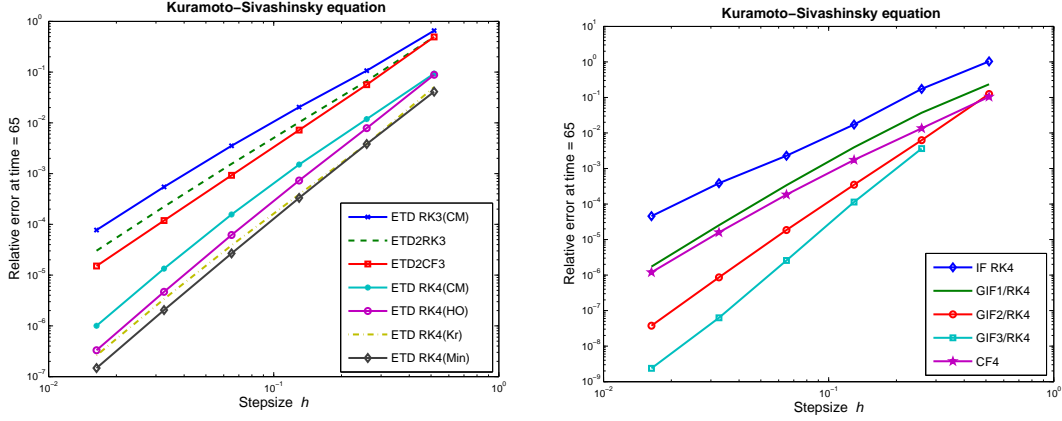


Figure 3.1: Step size versus relative error for ETD RK (left) and GIF/RK (right) methods for the Kuramoto-Sivashinsky equation

The Allen-Cahn equation

The second example is the Allen-Cahn equation written in the form

$$u_t = \varepsilon u_{xx} + u - u^3, \quad x \in [-1, 1],$$

where $\varepsilon = 0.01$ and with boundary and initial conditions also borrowed from [35]

$$u(-1, t) = -1, \quad u(1, t) = 1, \quad u(x, 0) = 0.53x + 0.47 \sin(-1.5\pi x).$$

After discretization in space based on the Chebyshev grid points we can rewrite the equation in the form (1.1.1), with

$$L = \varepsilon D^2, \quad N(u) = u - u^3,$$

where D is the Chebyshev differentiation matrix [65]. Therefore, the matrix L is full. The integration in time is until $t = 31$. In Figure 3.2, we have plotted the results for the different numerical schemes under consideration.

The Korteweg de Vries equation

The final example is the KdV equation considered in [36, Article 5]

$$u_t = -u_{xxx} - uu_x, \quad x \in [-\pi, \pi],$$

with periodic boundary conditions and with initial condition

$$u(x, 0) = 3C / \cosh^2(\sqrt{C}x/2),$$

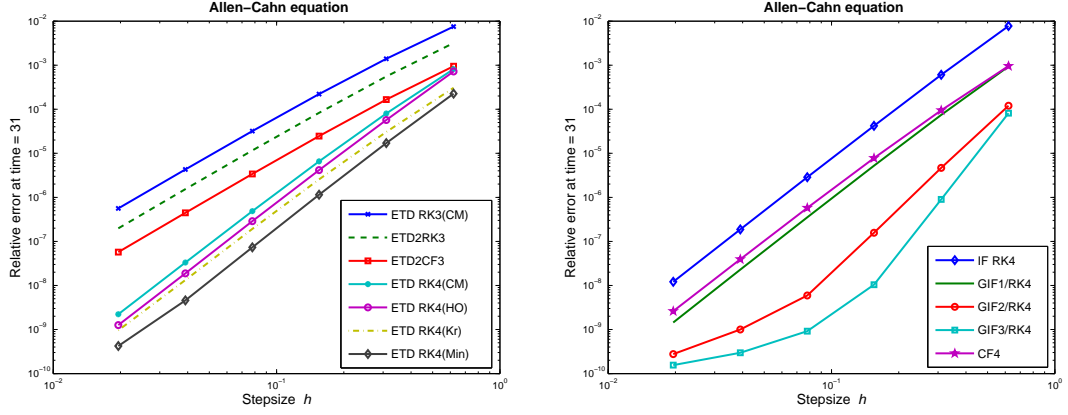


Figure 3.2: Step size versus relative error for ETD RK (left) and GIF/RK (right) methods for the Allen-Cahn equation

where $C = 625$. The exact solution is $2\pi/C$ periodic and is given by $u(x, t) = u(x - Ct, 0)$. We use a 256-point Fourier spectral discretization in space. In this case the matrix L is again diagonal. The integration in time is done for one period. The results for the different numerical schemes are plotted in Figure 3.3.

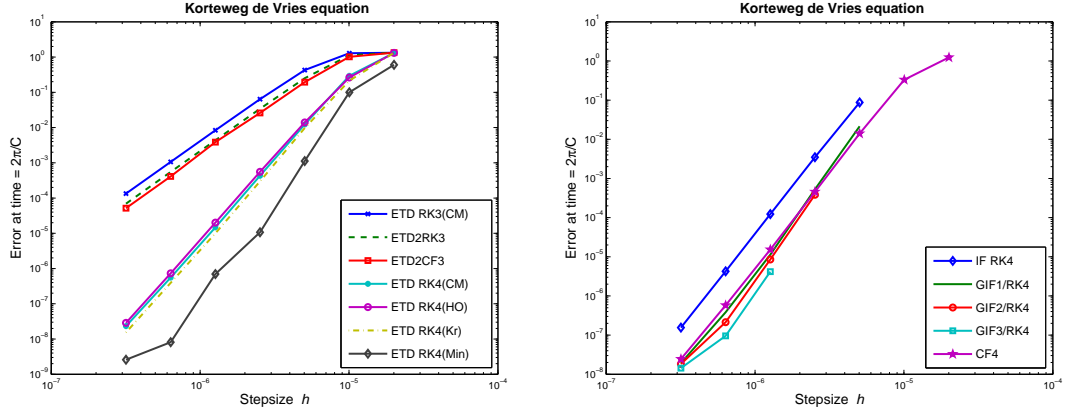


Figure 3.3: Step size versus relative error for ETD RK (left) and GIF/RK (right) methods for the Korteweg de Vries equation

As seen from Figure 3.1–3.3, all methods exhibit the predicted nonstiff order of convergence under the chosen initial and boundary conditions. For all examples, the scheme ETD RK4(Min) is more efficient than the other ETD RK schemes. This can be explained with

the fact that it was constructed based on the nonstiff order theory presented in Subsection 1.2.2, so that the error coefficients for the elementary differentials of higher order are minimized. As expected, the ETD RK methods perform better than the IF RK methods even based on the same underlying Runge–Kutta method. This is due to the fact that the local error of IF RK methods is bigger than the local error of ETD RK methods (see [46]). The local error for the GIF/RK methods reduces with increasing the degree of the polynomial approximation to the nonlinear part of the problem. This results in several orders in magnitude improvement in accuracy over the standard IF RK methods. However, as it was pointed from Krogstad in [36, Article 5], the improved accuracy comes to some extent at the price of stability. This fact, in particular, is well illustrated in the numerical experiments for the KdV equation, where the GIF/RK methods fail to produce results for large stepsizes. The reason for this is the reduced stability regions for GIF/RK methods [36, Article 5].

Conclusions

The aim of this thesis was to study different classes of exponential integrators for time integration of semilinear problems and to propose an unified framework in which this methods can be analyzed. Below, we give a summary of the main topics treated in the thesis.

Summary

In Chapter 1 we presented the philosophy behind exponential integrators applied to semilinear problems and discussed the three main classes of exponential integrators: exponential linear multistep (multivalue), exponential Runge–Kutta (multistage) and exponential general linear methods. Next, in Chapter 2 we studied the connection between exponential integrators and Lie group methods based on the affine algebra action. The freedom in the choice of the algebra action, allows all exponential integrators presented in Chapter 2 as well as the methods proposed in [Article 1,91] to be applied to nonautonomous and quasilinear problems of the the form

$$u' = L(u, t)u + N(u, t), \quad u(t_0) = u_0.$$

The only difference in this case is in the definition of the generic function F , which provides the representation of the differential equation on the manifold. Because of the time dependence of the linear part, the resulting methods require at the beginning of each step to recompute the exponential and the related $\phi^{[l]}$ functions included in the format of the method. In Chapter 3 we discussed different algorithms for fast and numerically stable computation of the most commonly used ETD $\phi^{[i]}$ functions. Finally, we tested some of the exponential integrators studied in the thesis, on several well known examples.

Next, we summarize the main achievements of this thesis.

Contributions

We proposed in Subsection 1.2.1 a general class of exponential Runge–Kutta methods specifically designed for time integration of semilinear problems. This class of methods include as special cases all known exponential Runge–Kutta methods. The nonstiff order theory developed in Subsection 1.2.2, reduces to the the order theory for the adaptive Runge–Kutta

methods proposed in [5]. An advantage of our approach is that it provides a non-recursive rule for generating each order condition from its corresponding rooted tree. In addition, the same technique can also be applied to derive the nonstiff order conditions for the exponential general linear methods considered in Subsection 1.3.3. We believe that locating good methods based on the nonstiff order conditions, is the first step for developing competitive methods for realistic stiff problems.

We also studied the natural connection between exponential integrators and Lie group methods with affine algebra action. In particular, in Subsection 2.4.2, we showed that the Generalized Integrating Factor Runge–Kutta (GIF/RK) methods introduced in [36, Article 5] are examples of RKMK methods. In addition, we proposed a new approach in the derivation of GIF/RK methods, which allows the nonlinear part of the problem (1.1.1) to be approximated by trigonometric polynomials. The choice of the algebra action plays a crucial role in the overall performance of any Lie group method. In [Article 1, p.91], we suggested a way how to construct exponential integrators based on the framework of Lie group methods with algebra action arising from the solutions of differential equations with nonautonomous frozen vector fields.

The main computational challenge in the implementation of every exponential integrator comes from the necessity of computing the exponential and the related functions by some fast and numerically stable algorithm. In Subsection 3.1.2, we generalized the tridiagonal reduction approach, proposed in [41], to a general algorithm for computing all ETD $\phi^{[i]}$ functions. We also commented on the main advantages and disadvantages of other numerical techniques, which can also be used for computing the ETD $\phi^{[i]}$ functions. Usually, a variable stepsize implementation of an exponential integrator requires efficient linear solver. In the case when the linear part of the problem arises from a finite difference approximation, we can benefit from its sparse block structure. In [Article 2 p.107], we presented a method for solving tridiagonal block Teoplitz linear systems of equations. The method is a modification of the LU factorization proposed in [19]. In [Article 3 p.133], the same approach was generalized to the case when the coefficient matrix has pentadiagonal block circulant structure.

Future work

Many questions have arisen during the work on this thesis, which we intend to consider in future work. For example, the choice of the $\phi^{[l]}$ functions plays a significant role in the actual performance of any exponential integrator. Notice that an ETD method will generally perform better than an IF method, even based on the same underlying method. The possible choices of $\phi^{[l]}$ functions which lead to methods are still unclear. Thus, determining the set of functions which provides in some sense optimal methods, is a crucial question which requires further investigation.

Developing the nonstiff and respectively the stiff order conditions for exponential general linear methods is a field of ongoing research. Such an order theory, will allow other

exponential general linear methods, different from the GIF/RK methods and the methods introduced in [Article 1, p.91], to be constructed. The improved accuracy of GIF/RK methods, is due to their higher stage order (see [46]). The main difficulty with these methods, as Krogstad pointed out in [36, Article 5], is that the improved accuracy comes to some extent at the price of stability. A way to overcome this difficulty is to extend the class of general linear methods with inherent Runge–Kutta stability to the exponential setting.

Probably the main question regarding the exponential integrators, which still needs to be answered is: Are these methods fully competitive with the existing numerical techniques for solving stiff problems. The answer of this question requires extensive numerical experiments with various implementations, from fixed stepsize and fixed order to variable stepsize and variable order codes. Thus, it is clear that the choice of reliable and numerically stable algorithms for computing the exponential and the related functions is an important direction for future investigations.

Differential algebraic equations (DAEs) are ordinary differential equations subject to algebraic constraints. Many problems which arise in the real applications can be represented by DAEs. Examples are the discretized incompressible Navier–Stokes equations, where the incompressibility condition is the algebraic constraint. Extending the class of exponential integrators to the set of DAEs is an other interesting topic for future research.

Bibliography

- [1] R. Abraham, J. Marsden, and T. Ratiu, *Manifolds, Tensor Analysis and Applications*, Springer, Second edition, 1988.
- [2] U. Ascher, S. Ruuth, and B. Wetton, *Implicit-explicit methods for time-dependent partial differential equations*, SIAM J. Numer. Anal. **32** (1995), 797–823.
- [3] H. Berland and B. Owren, *Fourth order exponential time integrators for the nonlinear Schrödinger equation*, private conversation, April, 2004.
- [4] G. Beylkin, J. Keiser, and L. Vozovoi, *A new class of time discretization schemes for the solution of nonlinear PDEs*, J. Comput. Phys. **147** (1998), 362–387.
- [5] J. Bruder, K. Strehmel, and R. Weiner, *Partitioned adaptive Runge–Kutta methods for the solution of nonstiff and stiff systems*, Numer. Math. **52** (1988), 621–638.
- [6] O. Buneman, *A compact non-iterative Poisson solver*, Rep. 294, Stanford University Institute for Plasma Research, 1969.
- [7] J. C. Butcher, *On the convergence of numerical solutions to ordinary differential equations*, Math. Comp. **20** (1966), 1–10.
- [8] ———, *Numerical methods for ordinary differential equations*, John Wiley & Sons, 2003.
- [9] J. C. Butcher and Z. Jackiewicz, *Construction of general linear methods with runge-kutta stability properties*, Numer. Algorithms **36** (2004), 63–72.
- [10] J. C. Butcher and W. Wright, *The construction of practical general linear methods*, BIT **43** (2003), 695–721.
- [11] B. Buzbee, G. Golub, and C. Nielson, *On direct methods for solving Poisson’s equation*, SIAM J. Numer. Anal. **7** (1970), 627–656.
- [12] E. Celledoni, A. Marthinsen, and B. Owren, *Commutator-free Lie group methods*, FGCS **19(3)** (2003), 341–352.

- [13] J. Certaine, *The solution of ordinary differential equations with large time constant*, Math. methods for digital comput. (1960), 128–132.
- [14] P. M. Cox and P.C. Matthews, *Exponential time differencing for stiff systems*, J. Comput. Phys. **176** (2002), 430–455.
- [15] P. Crouch and R. Grossman, *Numerical integration of ordinary differential equations on manifolds*, J. Nonlinear. Sci. **3** (1993), 1–33.
- [16] P. Davies and N. Higham, *A Schur-Parlett algorithm for computing matrix functions*, SIAM J. Matrix Anal. Appl. **25(2)** (2003), 464–485.
- [17] V. L. Druskin and L. A. Knizhnerman, *Error bounds in the simple Lanczos procedure for computing functions of symmetric matrices and eigenvalues*, Comput. Maths. Math. Phys. **7** (1991), 20–30.
- [18] ———, *Krylov subspace approximations of eigenpairs and matrix functions in exact an computer arithmetic*, Numer. Lin. Alg. Appl. **2** (1995), 205–217.
- [19] S.M. El-Sayed, *Study of special matrices and numerical methods for special matrix equations*, Ph.D. thesis, Bulgarian Academy of Sciences, 1996, in Bulgarian.
- [20] A. Friedli, *Verallgemeinerte Runge–Kutta verfahren zur lösung steifer differentialgleichungssysteme*, Lect. Notes Math. **631** (1978).
- [21] E. Gallopoulos and Y. Saad, *Efficient solution of parabolic equations by Krylov approximation methods*, SIAM J. Sci. Statist. Comput. **13** (1992), 1236–1264.
- [22] G. Golub and C. Van Loan, *Matrix computations*, Johns Hopkins University Press, Baltimore, 1996, 3td ed.
- [23] E. Hairer, C. Lubich, and G. Wanner, *Geometric numerical integration*, Springer, 2003, Number 31 in Springer Series in Computational Mathematics.
- [24] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I, Non-stiff Problems*, Springer, 1993, Number 8 in Springer Series in Computational Mathematics, 2nd edition.
- [25] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II, Stiff and Differential–Algebraic Problems*, Springer, 1996, Number 14 in Springer Series in Computational Mathematics, 2nd edition.
- [26] D. Henry, *Geometric theory of semilinear parabolic equations*, Lecture Notes in Mathematics 840, Springer, Berlin, 1981.
- [27] M. Hochbruck and C. Lubich, *On Krylov subspace approximations to the matrix exponential operator*, SIAM J. Numer. Anal. **34** (1997), 1911–1925.

- [28] M. Hochbruck, C. Lubich, and H. Selhofer, *Exponential integrators for large systems of differential equations*, SIAM J. Sci. Comput. **19**(5) (1998), 1552–1574.
- [29] M. Hochbruck and A. Osterman, *Explicit exponential Runge–Kutta methods for semi-linear parabolic problems*, Submitted to *SIAM J. Numer. Anal.*, 2004.
- [30] ———, *Exponential Runge–Kutta methods for parabolic problems*, To appear in *Appl. Numer. Math.*, 2004.
- [31] R. Hockney, *The potential calculation and some applications*, Methods of Computational Physics **7** (1969), 136–211.
- [32] A. Iserles, H. Munthe-Kaas, S.P. Nørsett, and A. Zanna, *Lie-group methods*, Acta Numerica **9** (2000), 215–365.
- [33] Z. Jackiewicz, A. Marthinsenand, and B. Owren, *Construction of Runge–Kutta methods of Crouch–Grossman type of high order*, Adv. Comput. Math. **13**(4) (2000), 405–415.
- [34] C. Johnson, *Numerical solution of partial differential equations by the finite element methods*, Cambridge University press, 1987.
- [35] A.-K. Kassam and L.N. Trefethen, *Fourth order time stepping for stiff PDEs*, To appear in: *SIAM J. Sci. Comp.*, 2003.
- [36] S. Krogstad, *Topics in numerical Lie group integration*, Ph.D. thesis, University of Bergen, 2003.
- [37] J. Lawson, *Generalized Runge–Kutta processes for stable systems with large Lipschitz constants*, SIAM J. Numer. Anal. **4** (1969), 372–390.
- [38] R. Lehoucq, D. Sorensen, and C. Yang, *ARPACK user’s guide*, SIAM, 1998.
- [39] E. Lodden, *Geometric integration of the heat equation*, Master’s thesis, University of Bergen, 2000.
- [40] Y. Y. Lu, *Exponentials of symmetric matrices through tridiagonal reductions*, Linear Algebra Appl. **279** (1998), 317–324.
- [41] ———, *Computing a matrix function for exponential integrators*, J. Comp. and Appl. Math. **161**(1) (2003), 203–216.
- [42] A. Lunardi, *Analytic semigroups and optimal regularity in parabolic problems*, Birkhäuser, Basel, 1995.
- [43] Y. Maday, A. Patera, and E. Rønquist, *An operator-integration-factor splitting method for time-dependent problems: application to incompressible fluid flow*, J. Sci. Comp. **5**(4) (1990), 263–292.

- [44] J. Marsden and T. Ratiu, *Introduction to Mechanics and Symmetry*, Springer, Second edition, 1999, Number 17 in Text in Applied Mathematics.
- [45] B. Minchev, *Exponential time differencing and Lie-group methods for stiff problems*, Talk given at the International Conference on Scientific Computation and Differential Equations, SciCADE 2003, Trondheim, Norway, June 30 - July 4, 2003.
- [46] B. Minchev and W. Wright, *A review of exponential integrators*, in preparation, University of Bergen, 2004.
- [47] C. Moler and C. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Review **45**(1) (2003), 3–49.
- [48] H. Munthe-Kaas, *Lie-Butcher theory for Runge–Kutta methods*, BIT **35** (1995), 572–587.
- [49] ———, *Runge–Kutta methods on Lie groups*, BIT **38** (1998), 92–111.
- [50] ———, *High order Runge–Kutta methods on manifolds*, Appl. Num. Math. **29** (1999), 115–127.
- [51] H. Munthe-Kaas and W. Wright, *A hitchhikers guide to Lie-Butcher theory*, in preparation, University of Bergen, 2004.
- [52] A. Murua, *Formal series and numerical integrators, part I: Systems of ODEs and symplectic integrators*, Appl. Numer. Math. **29** (1999), 221–251.
- [53] S. Nørsett, *An A-stable modification of the Adams-Bashforth methods*, Lect. Notes Math. **109** (1969), 214–219.
- [54] A. Ostermann, *Numerical solutions of abstract ODEs*, Lecture notes from Dobbiaco Summer School on Numerical Analysis, 2004.
- [55] B. Owren and A. Marthinsen, *Runge–Kutta methods adapted to manifolds and based on rigid frames*, BIT **39**(1) (1999), 116–142.
- [56] B.N. Parlett, *A recurrence among the elements of functions of triangular matrices*, Linear Algebra Appl. **14** (1976), 117–121.
- [57] Y. Saad, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comp. **37** (1981), 105–126.
- [58] ———, *Analysis of some Krylov subspace approximations to the matrix exponential operator*, SIAM J. Numer. Anal. **29**(1) (1992), 209–228.
- [59] J. Sanz-Serna and M. Calvo, *Numerical Hamiltonian problems*, Chapman & Hall, London, 1994.

- [60] T. Steihaug and A. Wolfbrand, *An attempt to avoid exact jacobian and nonlinear equations in the numerical solution of stiff differential equations*, Math. Comp. **33** (1979), 521–534.
- [61] K. Strehmel and R. Weiner, *Behandlung steifer anfangswertprobleme gewöhnlicher differentialgleichungen mit adaptiven Runge–Kutta methoden*, Computing **29** (1982), 153–165.
- [62] ———, *B-convergence results for linearly implicit one step methods*, BIT **27** (1987), 264–281.
- [63] A. Suslowicz, *Application of numerical Lie group integrators to parabolic PDEs*, Technical report **219**, University of Bergen, 2001.
- [64] P. Swarztrauber, *The methods of cyclic reduction, Fourier analysis and the FACR algorithms for the discrete solution of the Poisson’s equation on a rectangle*, SIAM Review **19** (1977), 490–501.
- [65] L. N. Trefethen, *Spectral methods in MATLAB*, SIAM, 2000.
- [66] J. van den Eshof and M. Hochbruck, *Preconditioning Lanczos approximations to the matrix exponential*, tech. rep. Heinrich-Heine Universität Düsseldorf, 2004.
- [67] V. Varadarajan, *Lie groups, Lie algebras, and their representation*, Springer, New York, 1984.
- [68] R. S. Varga, *On higher order stable implicit methods for solving parabolic differential equations*, J. Math. and Phys **XL** (1961), 220–231.
- [69] W. Wright, *General linear methods with inherent Runge–Kutta stability*, Ph.D. thesis, The University of Aucland, New Zealand, 2002.
- [70] K. Yosida, *Functional analysis*, Springer, Berlin, 1980, Grundlehren der mathematischen Wissenschaften, vol. 123.

Article 1

Lie group integrators with nonautonomous frozen vector fields

Submitted to
Appl. Numer. Math.

Lie group integrators with nonautonomous frozen vector fields

Borislav V. Minchev

*Department of Computer Science, University of Bergen,
Thormhøllensgate 55, N-5020 Bergen, Norway*

Abstract

Lie group methods for nonautonomous semi-discretized in space, partial differential equations are considered. The choice of frozen vector field and its corresponding algebra action on the manifold for such problems is discussed. A new exponential integrator for semilinear problems, based on commutator free Lie group methods with algebra action arising from the solutions of differential equations with nonautonomous frozen vector fields is derived. The proposed new scheme is then compared with some existing methods in several numerical experiments.

Key words: Lie group methods, algebra action, exponential integrators, stiff systems

1991 MSC: 65L06, 65M29, 35G10, 58F39

1 Introduction

Recently, a lot of Lie group integrators for solving semi-discretized partial differential equations (PDEs) has been derived in the literature. The original idea was first introduced in [16] and then further investigated for the heat equation in [4,12,17], stiff PDEs in [10,11,14], convection diffusion problems in [3] and for the Schrödinger equation in [1]. What is common between all this methods is that to advance from one point to another they all use an algebra action arising from the solution of an *autonomous* differential equation, which does not depend explicitly on the time t .

In this paper we propose a way how to construct Lie group integrators for nonautonomous problems based on an algebra action arising from the solution

Email address: Borko.Minchev@ii.uib.no (Borislav V. Minchev).

URL: <http://www.ii.uib.no/~borko> (Borislav V. Minchev).

of a differential equation which can depend explicitly on t . The idea is a natural extension of the autonomous case. The approach is to rewrite the differential equation in its equivalent autonomous form and then to apply the *affine* action [16] to the transformed equation. Thus, what we obtain are time dependent *frozen* vector fields. This provides us some extra freedom in the construction process, which can be used to choose the algebra action to be a better approximation of the flow of the original vector field.

The paper is organized as follows: We start in Section 2 with introducing some notation and the basic theory involved. Next we consider the framework for nonautonomous problems and discuss how it is related with the choice of the algebra action. In Section 3 we propose a new time dependent frozen vector field and its corresponding algebra action. In addition, we discuss some further generalizations. In Section 4 we derive a new exponential integrator for semilinear problems based on the fourth order commutator free Lie group method [4]. Finally in Section 5 we compare the proposed new exponential integrator with some existing methods and discuss the advantages of the new approach.

2 Background theory and notations

The framework which we use in this paper is given by Lie groups and their action upon a homogeneous manifold [8,15,16]. We take advantage of the fact that, in order to construct a Lie group integrator, we do not really need to know what the structure of the Lie group is and how it acts on the manifold. It is enough to specify the *generic presentation* of the differential equation and the algebra action on the manifold (see [14]). For simplicity, we do not include a discussion on the structure of the underlying Lie group and how it acts on a manifold.

Let us first consider the following differential equation defined on a $d + p$ dimensional manifold $\mathcal{M} \equiv \mathbb{R}^{d+p}$.

$$y' = f(y(t)), \quad y(t_0) = y_0. \quad (2.1)$$

The very first question in the construction of a Lie group integrator is how to define the basic motions on \mathcal{M} . Since \mathbb{R}^{d+p} is a linear space, it is easy to construct integrator which stays on the manifold. The challenge in this case is how to choose the basic motions in such a way that they provide a good approximation to the flow of the original vector field. In this paper we define the basic movements on \mathcal{M} to be given by the solution of a simpler differential equation

$$y' = \mathcal{F}_\Theta(y), \quad y(t_0) = y_0, \quad (2.2)$$

which locally approximates (2.1). Thus, the Lie algebra \mathfrak{g} is generated from

the set of all coefficients Θ of the *frozen* vector fields \mathcal{F}_Θ and the Lie algebra action $* : \mathfrak{g} \times \mathcal{M} \rightarrow \mathcal{M}$ on the manifold is simply given by the solution of (2.2). In other words, if $\Theta \in \mathfrak{g}$, its action upon the point $y_0 \in \mathcal{M}$, which we denote by $h\Theta * y_0$, is given by the solution of (2.2) at time $t_0 + h$. Every frozen vector field can be represented in the form $\mathcal{F}_\Theta(y) = \Theta \circledast y$, where $\circledast : \mathfrak{g} \times \mathcal{M} \rightarrow T\mathcal{M}$ and according to (2.2), it satisfies

$$\Theta \circledast y = \left. \frac{d}{dt} \right|_{t=0} t\Theta * y.$$

Note that the map $\Theta \rightarrow \mathcal{F}_\Theta$ is an algebra homomorphism between \mathfrak{g} and the set of all vector fields on \mathcal{M} . If the algebra action $*$ is *transitive* i.e. starting from a point $y_0 \in \mathcal{M}$ we can reach any other point $y_1 \in \mathcal{M}$ by letting some element $\Theta \in \mathfrak{g}$ act on y_0 , the differential equation (2.1) can be rewritten in the form

$$y' = F(y) \circledast y, \quad y(t_0) = y_0, \quad (2.3)$$

where $F : \mathcal{M} \rightarrow \mathfrak{g}$. The above formulation is called the *generic presentation* of the differential equation on the manifold and plays an important role in the theory of the Lie group integrators (see [16]).

The choice of the frozen vector field \mathcal{F}_Θ and its corresponding algebra action, very much depends of the actual structure of $\mathbf{f}(y)$. The simplest possible case is when $\mathfrak{g} = \{\mathbf{b} \in \mathbb{R}^{d+p}\}$, $F(y_0) = \mathbf{f}(y_0) = \mathbf{b}$ and $\mathcal{F}_\mathbf{b}(y) = \mathbf{b} \circledast y = \mathbf{b}$ then the algebra action on \mathcal{M} is given by translations and we recover the traditional integration schemes. In the case when $\mathbf{f}(y) = \mathbf{L}(y)y + \mathbf{N}(y)$, one can define the Lie algebra $\mathfrak{g} = \{(\mathbf{A}, \mathbf{b}) \in \mathbb{R}^{(d+p) \times (d+p)} \rtimes \mathbb{R}^{d+p}\}$, the function $F(y_0) = (L(y_0), N(y_0)) = (\mathbf{A}, \mathbf{b})$ and the frozen vector field $\mathcal{F}_{(\mathbf{A}, \mathbf{b})}(y) = (\mathbf{A}, \mathbf{b}) \circledast y = \mathbf{A}y + \mathbf{b}$. This is exactly the affine algebra action proposed in [16]. Note that such a representation of $\mathbf{f}(y)$ is always possible, for example by letting $\mathbf{L}(y)$ be the Jacobian of \mathbf{f} at the point y and $\mathbf{N}(y) = \mathbf{f}(y) - \mathbf{L}(y)y$. Other choices are also possible see for example [12,17].

In this paper we are interesting in the construction of Lie group methods for the following nonautonomous problem defined on \mathbb{R}^d

$$u' = f(u, t), \quad u(t_0) = u_0. \quad (2.4)$$

Formally it does not fit in the above presented framework, but by adding the trivial differential equation $t' = 1$ to the system (2.4), we can rewrite it in the form (2.1) with $p = 1$ and

$$\mathbf{f} = \begin{bmatrix} f(u, t) \\ 1 \end{bmatrix}, \quad y = \begin{bmatrix} u \\ t \end{bmatrix}.$$

This of course is a very well known idea in the theory of ODEs, however its application to Lie group methods, if done carefully, can lead to some extra

freedom which we would like to exploit. Note that now the time variable t goes in the definition of the manifold \mathcal{M} and therefore it also appears like one of the arguments of the generic function F , the frozen vector field \mathcal{F} and its corresponding algebra action.

From a computational point of view it might look some what unreasonable to increase the dimensionality of the problem, but we keep in mind that the solution of (2.4) is given by the first d components of the solution of (2.1). Thus, the approach is to apply a Lie group method to equation (2.1) and then to restate it in \mathbb{R}^d .

The simplest nonautonomous case is when the Lie algebra $\mathfrak{g} = \{\mathbf{b} \in \mathbb{R}^{d+1}\}$ or equivalently $\mathfrak{g} = \{(b^{[0]}, \lambda) : b^{[0]} \in \mathbb{R}^d, \lambda \in \mathbb{R}\}$ then the generic function is given by $F\left(\begin{bmatrix} u_0 \\ t_0 \end{bmatrix}\right) = (f(u_0, t_0), 1) = (b^{[0]}, 1)$, the frozen vector field is $\mathcal{F}_{(b^{[0]}, \lambda)}\left(\begin{bmatrix} u \\ t \end{bmatrix}\right) = \begin{bmatrix} b^{[0]} \\ \lambda \end{bmatrix}$ and the algebra action is $h(b^{[0]}, \lambda) * \begin{bmatrix} u_0 \\ t_0 \end{bmatrix} = \begin{bmatrix} u_0 + hb^{[0]} \\ t_0 + h\lambda \end{bmatrix}$. When $f(u, t) = L(u, t)u + N(u, t)$ then the Lie algebra $\mathfrak{g} = \{(A, \mathbf{b}) \in \mathbb{R}^{d+1 \times d+1} \rtimes \mathbb{R}^{d+1}\}$. It can also be represented like the set of all triples $(A, b^{[0]}, \lambda)$ closed under linear combinations and *commutators* between the elements (see section 3), where $A \in \mathbb{R}^{d \times d}$, $b^{[0]} \in \mathbb{R}^d$, $\lambda \in \mathbb{R}$. In this case the generic function is defined like $F\left(\begin{bmatrix} u_0 \\ t_0 \end{bmatrix}\right) = (L(u_0, t_0), N(u_0, t_0), 1) = (A, b^{[0]}, 1)$, the frozen vector field is $\mathcal{F}_{(A, b^{[0]}, \lambda)}\left(\begin{bmatrix} u \\ t \end{bmatrix}\right) = \begin{bmatrix} Au + b^{[0]} \\ \lambda \end{bmatrix}$ and its corresponding algebra action is given by $h(A, b^{[0]}, \lambda) * \begin{bmatrix} u_0 \\ t_0 \end{bmatrix} = \begin{bmatrix} e^{hA}u_0 + hb^{[0]}\phi^{[1]}(hA) \\ t_0 + h\lambda \end{bmatrix}$, where e^{hA} denotes the matrix exponential and the function $\phi^{[1]}$ is given in Lemma 2 (see section 3).

If we consider just the first d components of the algebra action: in the first case we simply obtain translations like basic motions on \mathbb{R}^d ; in the second case we again recover the affine action. Thus, we conclude that for the two cases the only difference between autonomous and nonautonomous systems is in the definition of the generic function F , which for nonautonomous systems depends also from the time variable. We remark that in the above two cases the frozen vector field does not really depend of t . This explains the observed similarities between autonomous and nonautonomous systems.

A more interesting situation arises when the function $f(u, t)$ has the form $L(u, t)u + N^{[0]}(u, t) + tN^{[1]}(u, t)$. A natural question now is how to choose the frozen vector field in this case. In the next section, we propose a time dependent frozen vector field and its corresponding algebra action which reflects the structure of f . In addition, we discuss how this idea can be further generalized if second and higher powers of t are included.

3 Nonautonomous frozen vector fields

We first consider the case when the vector field of (2.4) has the form

$$f(u, t) = L(u, t)u + N^{[0]}(u, t) + tN^{[1]}(u, t).$$

If we treat the nonlinear part as a single function $N = N^{[0]} + tN^{[1]}$ then we do not gain anything in comparison with the affine case presented in the previous section. A more demanding task is to allow our frozen vector field to be time dependent. It is desirable in this case to approximate the nonlinear part of $f(u, t)$ by a linear polynomial of t .

The only way to include t in the definition of the frozen vector field is to append it to the dependent variables. Thus, by adding the trivial differential equation $v' = 1$, $v(t_0) = t_0$ to the system (2.4), we obtain

$$y' = \mathbb{L}y + \mathbb{N}, \quad y(t_0) = y_0, \quad (3.1)$$

where

$$\mathbb{L} = \begin{bmatrix} L(u, t) & N^{[1]}(u, t) \\ 0 & 0 \end{bmatrix}, \quad \mathbb{N} = \begin{bmatrix} N^{[0]}(u, t) \\ 1 \end{bmatrix}, \quad y = \begin{bmatrix} u \\ v \end{bmatrix}, \quad y_0 = \begin{bmatrix} u_0 \\ t_0 \end{bmatrix}.$$

The advantage of rewriting (2.4) in the above form is that now we can easily see how to define the Lie algebra \mathfrak{g} , the generic function F , the frozen vector field \mathcal{F} and its corresponding algebra action (see section 2). The Lie algebra in this case is $\mathfrak{g} = \{(\mathbf{A}, \mathbf{b}) \in \mathbb{R}^{d+1 \times d+1} \rtimes \mathbb{R}^{d+1}\}$, with

$$\mathbf{A} = \begin{bmatrix} A & b^{[1]} \\ 0 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b^{[0]} \\ \lambda \end{bmatrix}, \quad (3.2)$$

where $A \in \mathbb{R}^{d \times d}$, $b^{[1]}, b^{[0]} \in \mathbb{R}^d$ and $\lambda \in \mathbb{R}$. Equivalently we can represent \mathfrak{g} like the quadruplet $(A, b^{[1]}, b^{[0]}, \lambda)$ closed under linear combinations and commutators. The function F which provides the generic presentation is given by

$$F\left(\begin{bmatrix} u_0 \\ t_0 \end{bmatrix}\right) = (L(u_0, t_0), N^{[1]}(u_0, t_0), N^{[0]}(u_0, t_0), 1) = (A, b^{[1]}, b^{[0]}, 1). \quad (3.3)$$

To obtain an explicit form of the frozen vector field we use the following result.

Lemma 1 *The solution of the differential equation $y' = \mathbf{A}y + \mathbf{b}$, $y(t_0) = y_0$, at the time $t_0 + h$ is given by*

$$y(t_0 + h) = \begin{bmatrix} u(t_0 + h) \\ t_0 + h\lambda \end{bmatrix},$$

where $u(t_0 + h)$ is the solution of

$$u' = Au + b^{[0]} + t_0(1 - \lambda)b^{[1]} + t\lambda b^{[1]}, \quad u(t_0) = u_0.$$

Proof: The proof follows directly by substituting (3.2) in the differential equation $y' = Ay + b$ and then solving it with respect to the last variable. \square

Thus, we have obtained the following time dependent frozen vector field

$$\mathcal{F}_{(A, b^{[1]}, b^{[0]}, \lambda)} \left(\begin{bmatrix} u \\ t \end{bmatrix} \right) = (A, b^{[1]}, b^{[0]}, \lambda) \otimes \begin{bmatrix} u \\ t \end{bmatrix} = \begin{bmatrix} Au + c_0 + tc_1 \\ \lambda \end{bmatrix}, \quad (3.4)$$

where $c_0 = b^{[0]} + t_0(1 - \lambda)b^{[1]}$ and $c_1 = \lambda b^{[1]}$. Note that for $\lambda = 1$ we have $c_0 = b^{[0]}$, $c_1 = b^{[1]}$ and therefore the generic presentation $F\left(\begin{bmatrix} u \\ t \end{bmatrix}\right) \otimes \begin{bmatrix} u \\ t \end{bmatrix} = \begin{bmatrix} f(u, t) \\ 1 \end{bmatrix}$ is satisfied.

The last thing which we need to define is the algebra action corresponding to the frozen vector field (3.4). In the next Lemma we give a general formula for the flow of the frozen vector field, which approximates the nonlinear part of f by a polynomial of t of degree p .

Lemma 2 *The solution of the differential equation*

$$u' = Au + \sum_{j=0}^p t^j c_j, \quad u(t_0) = u_0,$$

where $p \in \mathbb{N}$, $A \in \mathbb{R}^{d \times d}$ and $c_j \in \mathbb{R}^d$ at the time $t_0 + h$ is given by

$$u(t_0 + h) = e^{hA}u_0 + \sum_{k=0}^p h^{k+1} \delta_k \phi^{[k+1]}(hA),$$

where $\delta_k = \sum_{j=k}^p \frac{j!}{(j-k)!} t_0^{j-k} c_j$, $\phi^{[1]}(z) = \frac{e^z - 1}{z}$ and $\phi^{[k+1]}(z) = \frac{\phi^{[k]}(z) - \phi^{[k]}(0)}{z}$.

Proof: From the variation of constants formulae it follows that

$$\begin{aligned} u(t_0 + h) &= e^{hA}u_0 + e^{hA} \int_0^h e^{-\tau A} \left(\sum_{j=0}^p (t_0 + \tau)^j c_j \right) d\tau \\ &= e^{hA}u_0 + \sum_{j=0}^p \left(\delta_j e^{hA} \frac{1}{j!} \int_0^h e^{-\tau A} \tau^j d\tau \right). \end{aligned}$$

Multiple applications of integration by parts complete the proof. \square

Combining the results of Lemma 1 and Lemma 2 leads to the following explicit form for the algebra action corresponding to the vector field (3.4)

$$h(A, b^{[1]}, b^{[0]}, \lambda) * \begin{bmatrix} u_0 \\ t_0 \end{bmatrix} = \begin{bmatrix} e^{hA}u_0 + h(b^{[0]} + t_0b^{[1]})\phi^{[1]}(hA) + h^2\lambda b^{[1]}\phi^{[2]}(hA) \\ t_0 + h\lambda \end{bmatrix} \quad (3.5)$$

Once we have defined the generic presentation, the Lie algebra \mathfrak{g} and its action on \mathcal{M} we can use any Lie group method to find the solution of (2.4). The solution of (3.1) is simply given by its first d components.

In the case when a Runge-Kutta Munthe-Kaas method with exact Exp map is used the format requires the inverse of the dExp map (see [16]). Computationally it might be very expensive to find exactly the dExp^{-1} map and thus the approach proposed in [16] is to replace it with polynomial approximation of order higher than the order of the method. This imposes the necessity of using *commutators* between the elements of \mathfrak{g} . In this case if $\Theta_i = (A_i, b_i^{[1]}, b_i^{[0]}, \lambda_i)$ for $i = 1, 2$ are two elements from \mathfrak{g} then their commutator is given by

$$[\Theta_1, \Theta_2] = \left([A_1, A_2], A_1b_2^{[1]} - A_2b_1^{[1]}, A_1b_2^{[0]} - A_2b_1^{[0]} + \lambda_2b_1^{[1]} - \lambda_1b_2^{[1]}, 0 \right),$$

where $[A_1, A_2] = A_1A_2 - A_2A_1$ is the matrix commutator.

The above approach can be easily generalized when the function $f(u, t) = L(u, t)u + \sum_{k=0}^p t^k N^{[k]}(u, t)$. In this case we append p trivial differential equations corresponding to t, t^2, \dots, t^p to the system (2.4). Thus, the dimension of the manifold is $d + p$, but we keep in mind that we are only interested in its first d components. The Lie algebra is $\mathfrak{g} = \{(A, b^{[p]}, \dots, b^{[0]}, \lambda) : A \in \mathbb{R}^{d \times d}, \lambda \in \mathbb{R}, b^{[k]} \in \mathbb{R}^d\}$ and its action upon the manifold is given by Lemma 2. The coefficients c_j can be found in the same way as in Lemma 1. For $p = 2$ they are

$$\begin{aligned} c_0 &= b^{[0]} + (1 - \lambda)t_0b^{[1]} + (1 - \lambda)^2t_0^2b^{[2]}, \\ c_1 &= \lambda b^{[1]} + 2\lambda(1 - \lambda)t_0b^{[2]}, \\ c_3 &= \lambda^2b^{[2]}. \end{aligned}$$

We conclude this section with the observation that based on the same idea, methods with approximations of the nonlinear part of f by trigonometric polynomials can also be derived. In this case, the exact flow of the frozen vector field can be computed in the similar manner (see [14]).

4 Exponential integrator for semilinear problems

In this section we derive an exponential integrator based on the frozen vector field (3.4) and its corresponding algebra action (3.5) for the semilinear problem

$$u' = Lu + N(u, t), \quad u(t_0) = u_0, \quad (4.1)$$

where L is a constant linear term and N is a nonlinear term. Such systems often arise after the spatial discretization of certain PDEs. Comparisons between the stability regions for different Lie group methods applied to semi-discretized stiff PDEs is given in [11]. There the author suggests that for this type of problem the best methods are likely to be the *commutator free* Lie group methods [4]. This provides our motivation in the choice of the Lie group method.

Next we give an equivalent formulation of the method proposed in [4]. This formulation allows us to construct methods without knowing what the exact structure of the Lie group acting on the manifold is, or how the Exp map between the Lie algebra and the Lie group is defined. The general format of an s stage commutator free Lie group method advancing from point y_n to point y_{n+1} with a time step of size h is given by the following algorithm.

Algorithm 1 (*Commutator-free Lie group method*)

```

for  $i = 1, \dots, s$  do
     $U_i = (h \sum_{k=1}^s \alpha_{ij}^k F_k) * \dots * (h \sum_{k=1}^s \alpha_{i1}^k F_k) * y_n$ 
     $F_i = F(U_i)$ 
end
 $u_{n+1} = (h \sum_{k=1}^s \beta_j^k F_k) * \dots * (h \sum_{k=1}^s \beta_1^k F_k) * y_n$ 

```

Here the function F gives the generic presentation (2.3), the coefficients α_{ij}^k, β_j^k are parameters of the method and the value J counts the number of flow calculations required at each stage. In [4], the following fourth order method based on the classical fourth order method of Kutta is proposed.

$$\begin{array}{c|cccc}
 0 & & & & \\
 \frac{1}{2} & \frac{1}{2} & & & \\
 \frac{1}{2} & 0 & \frac{1}{2} & & \\
 \frac{1}{2} & \frac{1}{2} & 0 & 0 & \\
 \frac{1}{2} & -\frac{1}{2} & 0 & 1 & \} \\
 \hline
 & \frac{1}{4} & \frac{1}{6} & \frac{1}{6} & -\frac{1}{12} \\
 & \frac{1}{12} & \frac{1}{6} & \frac{1}{6} & \frac{1}{4} \}
 \end{array} \quad (4.2)$$

We use the symbol $\}$ to denote all the substages included in a stage with

$J > 1$. Note that in (4.2) the frozen vector field corresponding to the second stage is the same as for the first substage of the fourth stage. This reduces the cost of the method.

In order to use the frozen vector field (3.4) from the previous section we rewrite the nonlinear part of (4.1) in the form

$$N(u, t) = N_n + t \frac{N(u, t) - N_n}{t} = N^{[0]} + t N^{[1]}, \quad (4.3)$$

where $N_n = N(u_n, t_n)$ is the value of the nonlinear part at the beginning of the step number n . Keeping in mind (3.3) and (3.5), based on (4.2), we have found a new fourth order exponential integrator, which written in the original u variable is given by

$$\begin{aligned} U_1 &= u_n, \\ U_2 &= e^{\frac{hL}{2}} u_n + h \frac{1}{2} \phi^{[1]} N_n, \\ U_3 &= e^{\frac{hL}{2}} u_n + h \left[\frac{1}{2} \phi^{[1]} N_n + \left(\frac{t_n}{2} \phi^{[1]} + \frac{h}{4} \phi^{[2]} \right) N_2^{[1]} \right], \\ U_4 &= e^{\frac{hL}{2}} U_2 + h \left[\frac{1}{2} \phi^{[1]} N_n + \left(t_n \phi^{[1]} + \frac{h}{2} \phi^{[1]} + \frac{h}{2} \phi^{[2]} \right) N_3^{[1]} \right], \\ \hat{U} &= e^{\frac{hL}{2}} u_n + h \left[\frac{1}{2} \phi^{[1]} N_n + \left(t_n \phi^{[1]} + \frac{h}{2} \phi^{[2]} \right) \left(\frac{N_2^{[1]}}{6} + \frac{N_3^{[1]}}{6} - \frac{N_4^{[1]}}{12} \right) \right], \\ u_{n+1} &= e^{\frac{hL}{2}} \hat{U} + h \left[\frac{1}{2} \phi^{[1]} N_n + \left(t_n \phi^{[1]} + \frac{h}{2} \phi^{[1]} + \frac{h}{2} \phi^{[2]} \right) \left(\frac{N_2^{[1]}}{6} + \frac{N_3^{[1]}}{6} + \frac{N_4^{[1]}}{4} \right) \right], \end{aligned} \quad (4.4)$$

where $N_j^{[1]} = \frac{N(U_j, t_n + c_j h) - N_n}{t_n + c_j h}$ for $j = 1, \dots, 4$ and the arguments of all the $\phi^{[j]}$ functions are $\frac{hL}{2}$.

It is possible to rewrite (4.4) in equivalent form which does not involves splitting of the internal stages (see [1,14]). Such a representation is rather useless, since its implementation is more expensive than the one proposed, but it shows that (4.4) is a method based just on the pure Runge–Kutta idea.

If we represent the nonlinear part of (4.1), as a polynomial of second degree

$$N(u, t) = N_n + t \frac{N_n - N_{n-1}}{t} + t^2 \frac{N(u, t) - 2N_n + N_{n-1}}{t^2},$$

where N_n and N_{n-1} are the values of N at the end of step number n and $n-1$ respectively, we obtain a method which fits into the framework of *general linear methods* [2]. Thus, we see that by just changing the algebra action, any Lie group method based on a pure Runge–Kutta method can result in a more general method. This is a very interesting phenomena which highlights the important role of the algebra action.

5 Numerical experiments

In this section we present results from numerical experiments on the Kuramoto-Sivashinsky and Allen-Cahn equations. For both examples we compare the following four methods:

- **IF4** The fourth order integrating factor method [5,10,18] based on the classical fourth order method of Kutta.
- **CF4** The fourth order commutator free Lie group method (4.2) with affine algebra action [4].
- **CF4A1** The method (4.4) with algebra action given by (3.5).
- **ETDRK4B** The method of Krogstad [10].

Since all of the above methods are based on the nonstiff order conditions, to avoid possible order reduction, we consider examples where the nonlinear term N has sufficient spatial regularity. In general, for applications concerning PDEs, the classical order of convergence is not always obtained. Order reduction, due to the lack of sufficient temporal and spatial smoothness, should be expected. For parabolic problems, full order of convergence can be observed, if periodic boundary conditions are imposed [6,7].

To avoid problems with numerical instability, the computation of the $\phi^{[i]}$ functions, which suffer from cancelation errors when the eigenvalues of the discretized linear operator are close to zero, we use the approach of Kassam and Trefethen [9]. The idea is to evaluate the $\phi^{[i]}$ functions by Cauchy's integral formula

$$\phi^{[i]}(\gamma h L) = \frac{1}{2\pi i} \int_{\Gamma} \phi^{[i]}(\gamma \lambda) (\lambda I - h L)^{-1} d\lambda, \quad (5.1)$$

where $\gamma \in \mathbb{R}$. The contour Γ is a closed curve in the complex plane that encloses the eigenvalue of $\gamma h L$ and such that $\gamma \Gamma$ is well separated from zero. The trapezoidal rule is then used to approximate the integral in (5.1). If the discretized linear operator L is diagonal (Kuramoto-Sivashinsky equation) then the integral reduces simply to the mean of $\phi^{[i]}$ over the contour Γ . However, for non-diagonal problems (Allen-Cahn equation), the computations become more expensive and require the computation of several matrix inverses. That is why for such problems it is important for a method to use as few ϕ function evaluations as possible. In addition, if L has a special sparse structure one can apply effective methods to find its inverse [13,14].

The Kuramoto-Sivashinsky equation

The first example is the Kuramoto-Sivashinsky equation

$$u_t = -u u_x - u_{xx} - u_{xxx}, \quad x \in [0, 32\pi]$$

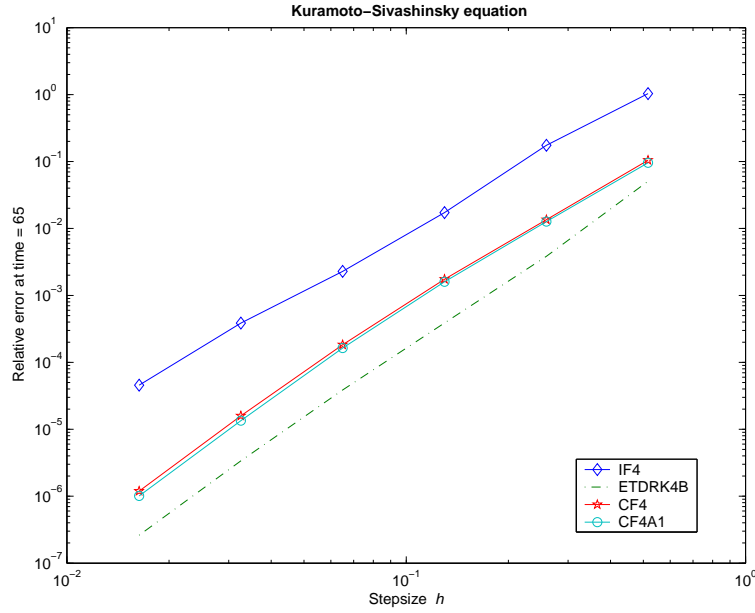


Fig. 1. Step size versus relative error for fourth order methods for the Kuramoto-Sivashinsky equation

with periodic boundary conditions and with the initial condition borrowed from [9]

$$u(x, 0) = \cos\left(\frac{x}{16}\right) \left(1 + \sin\left(\frac{x}{16}\right)\right).$$

A 128-point Fourier spectral discretization in space is used. Since the boundary conditions are periodic the transformed equation in the Fourier space can be represented in the form (4.1), the linear and nonlinear parts are defined as

$$(L\hat{u})(k) = (k^2 - k^4)\hat{u}(k), \quad N(\hat{u}) = -\frac{ik}{2}(\mathbf{F}((\mathbf{F}^{-1}(\hat{u}))^2)),$$

where \mathbf{F} denotes the discrete Fourier transform. The integration in time is done entirely in the Fourier space until $t = 65$. The results for the four different numerical schemes are plotted in Figure 1.

The Allen-Cahn equation

The second example is the Allen-Cahn equation written in the form

$$u_t = \varepsilon u_{xx} + u - u^3, \quad x \in [-1, 1],$$

where $\varepsilon = 0.01$ and with boundary and initial conditions also borrowed from [9]

$$u(-1, t) = -1, \quad u(1, t) = 1, \quad u(x, 0) = 0.53x + 0.47 \sin(-1.5\pi x).$$

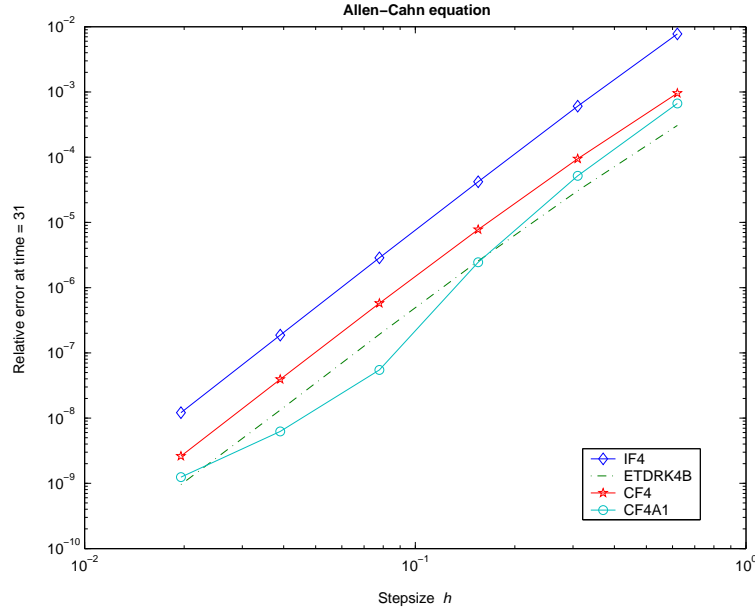


Fig. 2. Step size versus relative error for fourth order methods for the Allen-Cahn equation

After discretization in space based on the Chebyshev grid points we can rewrite the equation in the form (4.1), with

$$L = \varepsilon D^2, \quad N(u) = u - u^3,$$

where D is the Chebyshev differentiation matrix [18], this means the matrix L is full. The standard built in MATLAB function `inv` was used to find the matrix inverse in (5.1). The integration in time is until $t = 31$. In Figure 2 we have plotted the results for the four different numerical schemes.

For both examples we see that all the methods exhibit the expected fourth order, but the best with respect to the accuracy is the ETD4RK4B method. For the Kuramoto-Sivashinsky equation the improvement of using the algebra action (3.5) in CF4A1 is small in comparison with the affine action in CF4. However, for the non-diagonal example, the CF4A1 performs significantly better than CF4 and it is competitive with the ETD4RK4B method. This together with the fact that it uses only 1 exponential and 2 ϕ function evaluations per step (for comparison ETD4RK4B uses 2 exponentials and 5 ϕ function evaluations per step) suggests that CF4A1 is the best method in this case.

6 Concluding remarks

In this paper we have introduced the use of time dependent frozen vector fields in the construction of Lie group integrators for nonautonomous problems. This approach, provides extra freedom in the choice of the algebra action and allows us to choose the basic motions on the manifold to be given by the solutions of differential equations, which better approximate the flow of the original vector field. Based on this idea we have derived a new fourth order exponential integrator for semilinear problems with constant linear part. We do not claim that the proposed representation (4.3) of the nonlinear part is optimal. Other choices are worth investigating. However, the results from the numerical experiments suggest that the new method based on (4.3) is efficient in the case when the discretized linear operator is non-diagonal and a variable step size strategy is used. The content of this paper poses many questions which need to be answered. For example what are the stability regions of such methods and to what extent the spatial regularity of the problem affects the overall order of the method. An other important question is, for a given problem, how do we find a good algebra action? The goal is to choose a differential equation which is easier to solve, but still captures the key features of the original one. This is a very challenging task and it is likely to be problem dependent. The option presented here is to approximate the vector field by a higher order polynomial with constant coefficients. However, we should keep in mind that there is a certain balance between the benefit provided by increasing the order of the approximation and the computational cost of its corresponding algebra action.

Acknowledgments

The author would like to thank Hans Munthe-Kaas and William Wright for useful discussions throughout the work on this paper. This work has been partially supported by Norwegian Research Council through the GI4PDE project, contract number 142955/431.

References

- [1] H. Berland and B. Owren, *Fourth order exponential time integrators for the nonlinear Schrödinger equation*, private conversation, April, 2004.
- [2] J. C. Butcher, *Numerical methods for ordinary differential equations*, John Wiley & Sons, 2003.
- [3] E. Celledoni, *Eulerian and Semi-Lagrangian schemes based on commutator free*

- [4] E. Celledoni, A. Marthinsen, and B. Owren, *Commutator-free Lie group methods*, FGCS **19(3)** (2003), 341–352.
- [5] B. Fornberg and T.A. Driscoll, *A fast spectral algorithm for nonlinear wave equations with linear dispersion*, J.Comp. Phys. **155** (1999), 456–467.
- [6] M. Hochbruck and A. Osterman, *Explicit exponential Runge-Kutta methods for semilinear parabolic problems*, Submitted to *SIAM J. Numer. Anal.*, 2004.
- [7] ———, *Exponential Runge-Kutta methods for parabolic problems*, To appear in *Appl. Numer. Math.*, 2004.
- [8] A. Iserles, H. Munthe-Kaas, S.P. Nørsett, and A. Zanna, *Lie-group methods*, Acta Numerica **9** (2000), 215–365.
- [9] A.-K. Kassam and L.N. Trefethen, *Fourth order time stepping for stiff PDEs*, SIAM J. Sci. Comp., to appear, 2003.
- [10] S. Krogstad, *Generalized integrating factor methods for stiff PDEs*, Preprint submitted to J. Comput. Phys., 2003.
- [11] ———, *Topics in numerical Lie group integration*, Ph.D. thesis, University of Bergen, 2003.
- [12] E. Lodden, *Geometric integration of the heat equation*, Master’s thesis, University of Bergen, 2000.
- [13] B. Minchev, *Some algorithms for solving special tridiagonal block teoplitz linear systems*, J. Comp. and Appl. Math. **156(1)** (2003), 179–200.
- [14] ———, *Exponential integrators for semilinear problems*, Ph.D. thesis, University of Bergen, 2004.
- [15] H. Munthe-Kaas, *Runge-Kutta methods on Lie groups*, BIT **38** (1998), 92–111.
- [16] ———, *High order Runge-Kutta methods on manifolds*, Appl. Num. Math. **29** (1999), 115–127.
- [17] A. Suslowicz, *Application of numerical Lie group integrators to parabolic PDEs*, Technical report **219**, University of Bergen, 2001.
- [18] L. N. Trefethen, *Spectral methods in MATLAB*, SIAM, 2000.

Article 2

Some Algorithms for Solving Special Tridiagonal Block
Teoplitz Linear Systems

Published in
J. Comp. and Appl. Math.
156(1), 179-200, 2003

Some Algorithms for Solving Special Tridiagonal Block Teoplitz Linear Systems

Borislav V. Minchev

Department of Computer Science, University of Bergen, N-5020 Bergen, Norway

Abstract

This paper is focused on different methods and algorithms for solving tridiagonal block Teoplitz systems of linear equations. We consider the El-Sayed method [4] for such systems and propose several modifications that lead to different algorithms, which we discuss in detail. Our algorithms are then compared with some classical techniques as far as implementation time is concerned, number of operations and storage. Comments and conclusions for computing efficiency of the proposed new algorithms are given. Numerical experiments corroborating the theoretical results are also presented.

Key words: linear system, block Teoplitz matrix, matrix equation, Woodbury's formula

1991 MSC: 65F10

1 Introduction

Many problems arising in practice lead to the solution of linear systems of equations with special coefficient matrices. Tridiagonal block Teoplitz linear systems arise in numerical solution of ordinary and partial differential equations (ODE and PDE), interpolation problems, boundary value problems (BVP), etc. [2,3,7,13]. It is known that these systems have the form

$$M x = f, \tag{1}$$

where

Email address: Borko.Minchev@ii.uib.no (Borislav V. Minchev).

URL: <http://www.ii.uib.no/~borko> (Borislav V. Minchev).

Published in J. Comp. and Appl. Math., 156(1), 179-200, 2003

$$M = \begin{pmatrix} A & B & & & \\ B^* & A & B & & 0 \\ & B^* & A & . & \\ & & . & \ddots & \\ & & & . & \ddots \\ 0 & & & . & . & B \\ & & & & B^* & A \end{pmatrix}, \quad (2)$$

is an Hermitian tridiagonal block Teoplitz matrix with block size n . A and B are $m \times m$ matrices, x and f are column vectors with size nm .

The aim of this paper is to discuss different algorithms for solving (1) and compare them as far as time for implementation, number of operations and storage are concerned.

We organize the present paper as fallows:

1. In Section 2 we review LU factorization, Cholesky factorization [8] and adaptation the Cyclic Reduction method [1] corresponding to the special form of M . We modify the method described in [4].
2. In Section 3 we develop algorithms based on these modifications in order to optimize floating point operations, memory space and implementation.
3. Finally we verify the results in a number of numerical experiments.

2 Methods for solving special block tridiagonal Teoplitz linear system

Let us recall some classical direct methods for solving the linear system (1): LU factorization, Cholesky factorization, Cyclic Reduction [8], as well as one modification of LU factorization described in [4], which is based on the solution of a nonlinear matrix equation. For simplicity's we introduce the following notation $x = \{x_i\}_{i=1,\dots,n}$, $f = \{f_i\}_{i=1,\dots,n}$, where x_i and f_i are blocks with size $m \times 1$.

2.1 Block LU factorization

The matrix (2) admits following LU factorization

$$M = L U,$$

where

$$L = \begin{pmatrix} A_1 & & & & \\ B_1 & A_2 & & & 0 \\ & B_2 & \cdot & & \\ & & \ddots & & \\ & & & \ddots & \\ 0 & & & & B_{n-1} & A_n \end{pmatrix}, \quad U = \begin{pmatrix} F_1 & G_1 & & & \\ & F_2 & G_2 & & 0 \\ & & \cdot & \cdot & \\ & & & \ddots & \\ & 0 & & & \ddots & G_{n-1} \\ & & & & & F_n \end{pmatrix},$$

where by, the matrices A_i , B_i , F_i and G_i satisfy the relations

$$\left. \begin{aligned} A_1 F_1 &= A \\ B_1 &= B^* F_1^{-1} \\ G_1 &= A_1^{-1} B \\ A_i F_i &= A - B_{i-1} G_{i-1} \\ B_i &= B^* F_i^{-1} \\ G_i &= A_i^{-1} B \\ A_n F_n &= A - B_{n-1} G_{n-1} \end{aligned} \right\} \quad \text{for } i = 2, \dots, n-1. \quad (3)$$

The matrices A_i and F_i are lower and upper triangular respectively and are obtained by LU factorization.

Thus, solving the linear system (1) is equivalent to solving two simpler systems

$$L y = f, \quad y = \{y_i\}_{i=1, \dots, n}$$

and

$$U x = y, \quad x = \{x_i\}_{i=1, \dots, n},$$

whose solutions are

$$\begin{aligned} y_1 &= A_1^{-1} f_1 \\ y_i &= A_i^{-1} (f_i - B_{i-1} y_{i-1}), \quad \text{for } i = 2, \dots, n \end{aligned} \quad (4)$$

and

$$\begin{aligned} x_n &= F_n^{-1} y_n \\ x_i &= F_i^{-1} (y_i - G_i x_{i+1}), \quad \text{for } i = n-1, \dots, 1 \end{aligned} \quad (5)$$

respectively.

2.2 Block Cholesky factorization

When the matrix M is positive definite the following factorization

$$M = L L^*,$$

exist, where

$$L = \begin{pmatrix} A_1 & & & & \\ B_1 & A_2 & & & 0 \\ & B_2 & . & & \\ & & . & . & \\ & & & . & . \\ 0 & & & & B_{n-1} & A_n \end{pmatrix}.$$

The matrices A_i , B_i satisfy the relations

$$\left. \begin{aligned} A_1 A_1^* &= A \\ B_1 &= B^* (A_1^*)^{-1} \\ A_i A_i^* &= A - B_{i-1} B_{i-1}^* \\ B_i &= B^* (A_i^*)^{-1} \\ A_n A_n^* &= A - B_{n-1} B_{n-1}^* \end{aligned} \right\} \quad \text{for } i = 2, \dots, n-1. \quad (6)$$

It is well known that A_i and A_i^* are lower and upper triangular respectively and are obtained by Cholesky factorization.

In this manner, solving the linear system (1) is again equivalent to solving two simpler systems

$$L y = f, \quad y = \{y_i\}_{i=1, \dots, n}$$

and

$$L^* x = y, \quad x = \{x_i\}_{i=1, \dots, n}.$$

The solution of the first system can be found by (4). The solution of the second system satisfies

$$\begin{aligned} x_n &= (A_n^*)^{-1} y_n \\ x_i &= (A_i^*)^{-1} (y_i - B_i^* x_{i+1}), \quad \text{for } i = n-1, \dots, 1. \end{aligned} \quad (7)$$

2.3 Block Cyclic Reduction

In this section we adapt Bini's method [1] to the special case when the coefficient matrix is given as in (2). Let us derive explicitly the substitution formulas for computing the block coordinates of the solution x . Recall that Block Cyclic Reduction can be applied only if the block size of M is power of 2 in other words let $n = 2^p$. By performing an even-odd permutation of the block-rows and block columns in (1) we obtain

$$\begin{pmatrix} D_1^{(0)} & L^{(0)*} \\ L^{(0)} & D_2^{(0)} \end{pmatrix} \begin{pmatrix} x_+^{(0)} \\ x_-^{(0)} \end{pmatrix} = \begin{pmatrix} f_+^{(0)} \\ f_-^{(0)} \end{pmatrix}, \quad (8)$$

where

$$\begin{aligned} x_+^{(0)} &= \{x_{+k}^{(0)}\}_{k=1,\dots,2^{p-1}}, \quad x_-^{(0)} = \{x_{-k}^{(0)}\}_{k=1,\dots,2^{p-1}}, \quad f_+^{(0)} = \{f_{+k}^{(0)}\}_{k=1,\dots,2^{p-1}}, \\ f_-^{(0)} &= \{f_{-k}^{(0)}\}_{k=1,\dots,2^{p-1}} \end{aligned}$$

are column vectors, whose elements are blocks of size $m \times 1$, satisfying the relations

$$x_{+k}^{(0)} = x_{2k}, \quad x_{-k}^{(0)} = x_{2k-1}, \quad f_{+k}^{(0)} = f_{2k}, \quad f_{-k}^{(0)} = f_{2k-1} \quad \text{for } k = 1 \dots 2^{p-1}.$$

The cells

$$D_1^{(0)} = D_2^{(0)} = \begin{pmatrix} A & & \\ & \cdot & 0 \\ & & \cdot \\ 0 & & A \end{pmatrix}, \quad L^{(0)} = \begin{pmatrix} B & & \\ B^* & \cdot & 0 \\ & \cdot & \cdot \\ 0 & \cdot & \cdot \\ & & B^* & B \end{pmatrix}.$$

are matrices of block size $2^{p-1} \times 2^{p-1}$.

We apply one step of block-Gaussian elimination to (8) and obtain

$$\begin{cases} (D_2^{(0)} - L^{(0)} D_1^{(0)-1} L^{(0)*}) x_-^{(0)} = f_-^{(0)} - L^{(0)} D_1^{(0)-1} f_+^{(0)} \\ x_+^{(0)} = D_1^{(0)-1} (f_+^{(0)} - L^{(0)*} x_-^{(0)}). \end{cases} \quad (9)$$

Let

$$\begin{aligned} M^{(1)} &= D_2^{(0)} - L^{(0)} D_1^{(0)-1} L^{(0)*} \\ x^{(1)} &= x_-^{(0)} \\ f^{(1)} &= f_-^{(0)} - L^{(0)} D_1^{(0)-1} f_+^{(0)}. \end{aligned}$$

Note that now the first equation of (9) has the form

$$M^{(1)} x^{(1)} = f^{(1)}. \quad (10)$$

Observe, that the matrix $M^{(1)}$ has the form

$$M^{(1)} = \begin{pmatrix} F^{(1)} & B^{(1)} & & & \\ B^{(1)*} & A^{(1)} & & & \\ & & \cdot & \cdot & \\ & & & \cdot & \cdot \\ & & & & \cdot & B^{(1)} \\ & & & & B^{(1)*} & A^{(1)} \end{pmatrix},$$

where $F^{(1)} = A - BA^{-1}B^*$. Obviously, it is also a block tridiagonal matrix and, except for the north-western corner block $F^{(1)}$ it has a block Teoplitz

structure, with block size $2^{p-1} \times 2^{p-1}$. Applying once again an even-odd permutation of the block rows and block columns to (10), we obtain

$$\begin{pmatrix} D_1^{(1)} & L^{(1)*} \\ L^{(1)} & D_2^{(1)} \end{pmatrix} \begin{pmatrix} x_+^{(1)} \\ x_-^{(1)} \end{pmatrix} = \begin{pmatrix} f_+^{(1)} \\ f_-^{(1)} \end{pmatrix}, \quad (11)$$

where

$$x_+^{(1)} = \{x_{+k}^{(1)}\}_{k=1,\dots,2^{p-2}}, \quad x_-^{(1)} = \{x_{-k}^{(1)}\}_{k=1,\dots,2^{p-2}}, \quad f_+^{(1)} = \{f_{+k}^{(1)}\}_{k=1,\dots,2^{p-2}}, \\ f_-^{(1)} = \{f_{-k}^{(1)}\}_{k=1,\dots,2^{p-2}}$$

are column vectors, whose elements are blocks of size $m \times 1$ and satisfy the relations

$$x_{+k}^{(1)} = x_{2k}^{(1)}, \quad x_{-k}^{(1)} = x_{2k-1}^{(1)}, \quad f_{+k}^{(1)} = f_{2k}^{(1)}, \quad f_{-k}^{(1)} = f_{2k-1}^{(1)}, \quad \text{for } k = 1, \dots, 2^{p-2}.$$

Again the cells

$$D_1^{(1)} = \begin{pmatrix} A^{(1)} & & & \\ & \cdot & & 0 \\ & & \cdot & \\ 0 & & & A^{(1)} \end{pmatrix}, \quad D_2^{(1)} = \begin{pmatrix} F^{(1)} & & & \\ & A^{(1)} & & 0 \\ & & \cdot & \\ 0 & & & A^{(1)} \end{pmatrix}, \\ L^{(1)} = \begin{pmatrix} B^{(1)} & & & \\ B^{(1)*} & \cdot & & 0 \\ & \ddots & & \\ 0 & & \cdot & \cdot \\ & & & B^{(1)*} & B^{(1)} \end{pmatrix}$$

are matrices of block size $2^{p-2} \times 2^{p-2}$. We apply again one step of Gaussian elimination to (11) and obtain

$$M^{(2)}x^{(2)} = f^{(2)},$$

where $M^{(2)}$ is of the same type as $M^{(1)}$, but with block size $2^{p-2} \times 2^{p-2}$. Proceeding in a similar fashion, we obtain a sequence of linear systems of the form

$$M^{(j)}x^{(j)} = f^{(j)} \quad \text{for } j = 1, \dots, p,$$

where

$$M^{(j)} = \begin{pmatrix} F^{(j)} & B^{(j)} & & & \\ B^{(j)*} & A^{(j)} & \cdot & & \\ & & \cdot & \ddots & \\ & & & \ddots & \cdot \\ & & & & \cdot & B^{(j)} \\ & & & & & B^{(j)*} & A^{(j)} \end{pmatrix}$$

is square matrix with block of size 2^{p-j} for $j = 1, \dots, p$. When $j = p$ the cells $M^{(p)} = F^{(p)}$.

The blocks of the matrix $M^{(j)}$ obey the following relations:

$$\left. \begin{aligned} B^{(j)} &= -B^{(j-1)}A^{(j-1)^{-1}}B^{(j-1)}, \\ A^{(j)} &= A^{(j-1)} - B^{(j-1)*}A^{(j-1)^{-1}}B^{(j-1)} \\ &\quad - B^{(j-1)}A^{(j-1)^{-1}}B^{(j-1)*}, \\ F^{(j)} &= F^{(j-1)} - B^{(j-1)}A^{(j-1)^{-1}}B^{(j-1)*}, \end{aligned} \right\} \text{ for } j = 1, \dots, p, \quad (12)$$

where $A^{(0)} = A$, $B^{(0)} = B$, $F^{(0)} = A$.

For the block column vectors $f^{(j)}$ and $x^{(j)}$, we have

$$\left. \begin{aligned} f^{(j)} &= f_-^{(j-1)} - L^{(j-1)}D_1^{(j-1)}f_+^{(j-1)}, \\ f_{+k}^{(j)} &= f_{2k}^{(j)}, \\ f_{-k}^{(j)} &= f_{2k-1}^{(j)} \end{aligned} \right\} \text{ for } k = 1, \dots, 2^{p-j-1}, \quad \text{for } j = 0, \dots, p-1 \quad (13)$$

and

$$\left. \begin{aligned} x_-^{(j-1)} &= x^{(j)}, \\ x_+^{(j-1)} &= D_1^{(j-1)^{-1}}(f_+^{(j-1)} - L^{(j-1)*}x_-^{(j-1)}), \\ x_{2k}^{(j-1)} &= x_{+k}^{(j-1)}, \\ x_{2k-1}^{(j-1)} &= x_{-k}^{(j-1)} \end{aligned} \right\} \text{ for } k = 1, \dots, 2^{p-j-2}, \quad \text{for } j = p, \dots, 1, \quad (14)$$

where $f^{(0)} = f$, $x^{(p)} = F^{(p)^{-1}}f^{(p)}$, $x^{(0)} = x$.

For $j=0, \dots, p-1$ the cells

$$D_1^{(j)} = \begin{pmatrix} A^{(j)} & & 0 \\ & \cdot & \\ 0 & & A^{(j)} \end{pmatrix}, \quad L^{(j)} = \begin{pmatrix} B^{(j)} & & 0 \\ B^{(j)*} & \cdot & \\ 0 & \cdot & B^{(j)*} B^{(j)} \end{pmatrix}$$

are square matrices of block size 2^{p-j-1} .

2.4 A modification of LU factorization

In 1990 Rojo [14] proposed a new method for solving symmetric circulant tridiagonal linear systems and in recent years it has been modified to deal with matrices M having a special structure. For instance in [5] it is adapted to the case when the coefficient matrix M is pentadiagonal and strongly diagonally dominant. In [11] M is allowed to be not diagonally dominant. El-Sayed [4] extended Rojo's method to tridiagonal block matrices. His approach consists

in introducing a nonlinear matrix equation to solving the problem (1). The algorithms we propose in this paper are based on [4] and investigate different approaches for solving the nonlinear matrix equation of El-Sayed. We discuss Woodbury's formula and its numerical implementation.

Firstly, let us describe an algorithm for solving parametric linear systems of the form.

$$N y = f, \quad (15)$$

where

$$N = \begin{pmatrix} X & B & & & \\ B^* & A & & & 0 \\ & & \ddots & & \\ & & & \ddots & \\ 0 & & & & B \\ & & & B^* & A \end{pmatrix},$$

is a block tridiagonal matrix with block size n , X is a parameter block of size $m \times m$, and the vectors $y = \{y_i\}_{i=1,\dots,n}$, and $f = \{f_i\}_{i=1,\dots,n}$ are column vectors consisting of n blocks of size $m \times 1$.

The matrix N admits the following LU factorization

$$N = LU = \begin{pmatrix} I & & & & 0 \\ B^*X^{-1} & & & & \\ & \ddots & & & \\ 0 & & \ddots & & \\ & & & B^*X^{-1} & I \end{pmatrix} \begin{pmatrix} X & B & & & \\ & \ddots & & & 0 \\ & & \ddots & & \\ 0 & & & B & \\ & & & & X \end{pmatrix},$$

where I is the $m \times m$ identity matrix. The above factorization exists when the parameter X satisfies the nonlinear matrix equation

$$X + B^*X^{-1}B = A. \quad (16)$$

Thus, solving the linear system (15) is equivalent to solving two simpler systems

$$\begin{aligned} L z &= f, & z &= \{z_i\}_{i=1,\dots,n} \\ U y &= z, & y &= \{y_i\}_{i=1,\dots,n}, \end{aligned} \quad (17)$$

whose solutions are

$$\begin{aligned} z_1 &= f_1 \\ z_i &= f_i - B^*X^{-1}z_{i-1}, & i &= 2, 3, \dots, n, \\ y_n &= X^{-1}z_n \\ y_i &= X^{-1}(z_i - By_{i+1}), & i &= n-1, n-2, \dots, 1, \end{aligned} \quad (18)$$

respectively.

Now, we can find the solution of (1). The matrices M and N are related by relation

$$M = N + E_1 V_1^T,$$

where $E_1 = \begin{pmatrix} I \\ 0 \\ \vdots \\ 0 \end{pmatrix}$, $V_1^T = (A - X \ 0 \ \dots \ 0)$.

Using Woodbury's formula we have

$$M^{-1} = N^{-1} - N^{-1}E_1(I + V_1^T N^{-1}E_1)^{-1}V_1^T N^{-1}.$$

Therefore, the solution x of (1) is obtained from the vector y as follows

$$\begin{aligned} x &= M^{-1}f \\ &= N^{-1}f - N^{-1}E_1(I + V_1^T N^{-1}E_1)^{-1}V_1^T N^{-1}f \\ &= y - N^{-1}E_1(I + (A - X)E_1^T N^{-1}E_1)^{-1}V_1^T y \\ &= y - N^{-1}E_1(I + (A - X)E_1^T N^{-1}E_1)^{-1}(A - X)y_1. \end{aligned} \quad (19)$$

El-Sayed proposes following decomposition

$$N = LDV,$$

for computing $N^{-1}E_1$ and $E_1^T N^{-1}E_1$, where

$$L = \begin{pmatrix} I & & & \\ P & \cdot & & 0 \\ & \cdot & \cdot & \\ 0 & & \cdot & \cdot \\ & & P & I \end{pmatrix}, \quad V = \begin{pmatrix} I & Q & & \\ & \cdot & \cdot & 0 \\ & & \cdot & \cdot \\ 0 & & \cdot & Q \\ & & & I \end{pmatrix},$$

$$D = \text{diag}(X, X, \dots, X)$$

are square matrices of block size n , $P = B^* X^{-1}$ and $Q = X^{-1}B$. The matrix N^{-1} becomes

$$N^{-1} = V^{-1}D^{-1}L^{-1}.$$

If we denote $L^{-1} = (L_{ij})$ and $V^{-1} = (V_{ij})$, then

$$L_{ij} = \begin{cases} 0 & i < j \\ (-1)^{i-j} P^{i-j} & i \geq j, \end{cases}$$

$$V_{ij} = \begin{cases} 0 & i > j \\ (-1)^{j-i} Q^{j-i} & i \leq j, \end{cases}$$

$$D^{-1} = \text{diag}(X^{-1}, X^{-1}, \dots, X^{-1}).$$

Therefore, the blocks $(N^{-1}E_1)_i$ of the vector $N^{-1}E_1$ satisfy the formulas

$$\begin{aligned} (N^{-1}E_1)_i &= (V^{-1}D^{-1}L^{-1}E_1)_i \\ &= \sum_{s=i}^n (-1)^{i+1} Q^{s-i} X^{-1} P^{s-1} \quad \text{for } i = 1, \dots, n. \end{aligned} \quad (20)$$

Hence

$$E_1^T N^{-1} E_1 = (N^{-1} E_1)_1 = \sum_{s=1}^n Q^{s-1} X^{-1} P^{s-1}. \quad (21)$$

The coordinates x_i for $i=1, \dots, n$ of the vector x in (19) are given by

$$x_i = y_i - [\sum_{s=i}^n (-1)^{i+1} Q^{s-i} X^{-1} P^{s-1}] \times \\ \times [I + (A - X) \sum_{s=1}^n (-1)^{i+1} Q^{s-1} X^{-1} P^{s-1}]^{-1} (A - X) y_1.$$

Obviously formulas (20) and (21) are not convenient to implement directly, because they require a great number of redundant multiplications. For this reason we propose two algorithms for computing the vector $N^{-1} E_1$.

Algorithm F. Solve m linear systems of type (17) with right-hand sides the corresponding to different columns of E_1 . Note that this approach does not take into consideration the special structure of the right-hand sides vectors (having only a very sparse nonzero block). If the matrices A and B are real, the algorithm costs $O(8nm^3)$ flops and requires the storage of nm^2 real numbers.

Algorithm R. The blocks $(N^{-1} E_1)_i$ are recursively computed by the formula (20) using the following algorithm:

- Find the cells $Y_i = (-1)^{n-i} X^{-1} P^{n-i}$, for $i = 1, \dots, n$ by

$$Y_1 = X^{-1} \\ Y_i = Y_{i-1}(-P), \quad \text{for } i = 2, \dots, n.$$

- Compute the blocks $(N^{-1} E_1)_i$ by

$$(N^{-1} E_1)_n = Y_n \\ (N^{-1} E_1)_i = Y_i - Q(N^{-1} E_1)_{i+1}, \quad \text{for } i = n - 1 \div 1.$$

Our theoretical investigation shows that the algorithm R:

1. Requires half as many flops as the algorithm F, at the expense of minimal increase of storage memory. If the matrices A and B are real the algorithm costs $O(4nm^3)$ flops and needs to store $(n + 2)m^2$ real numbers;
2. Takes advantage of the special form of the matrix E_1 .

3 Algorithms for solving special block tridiagonal linear systems

In this section we compare the algorithms for solving special block tridiagonal linear systems described in Section 2 and there modifications.

3.1 Algorithm LU

1. Find the matrices A_i, B_i, F_i and G_i according to (3).
 2. Solve the system $L y = f$ by (4).
 3. Solve the system $U x = y$ by (5).
- end.

If the matrices A and B are real, this algorithm requires $O(\frac{15n}{3}m^3 + \frac{n}{3}m^3 + \frac{n}{3}m^3) = O(\frac{17n}{3}m^3)$ flops and the storage of $(3n + 1)m^2 + 2nm$ real numbers.

3.2 Algorithm CHOL

1. Find the matrices A_i and B_i according to (6).
 2. Solve the system $L y = f$ by (4).
 3. Solve the system $L^* x = y$ by (7).
- end.

If the matrices A and B are real, this algorithm requires $O(\frac{11n}{3}m^3 + \frac{n}{3}m^3 + \frac{n}{3}m^3) = O(\frac{13n}{3}m^3)$ flops and the storage of $\frac{3n+4}{2}m^2 + \frac{3nm}{2}$ real numbers.

3.3 Algorithm CR (Cyclic Reduction)

1. Find the matrices $A^{(j)}, B^{(j)}$ and $F^{(j)}$ for $j = 1, \dots, p$ by (12).
 2. Compute the vectors $f^{(j)}$ for $j = 1, \dots, p$ by (13).
 3. Solve the linear system $F^{(p)} x^{(p)} = f^{(p)}$.
 4. Restore the coordinates of the vector x according to (14).
- end.

If the matrices A and B are real, this algorithm requires $O([18p+2]m^3 + [4pm^3 + 4 * 2^p m^2] + 2m^3 + 6 * 2^p m^2) = O([22p+4]m^3 + 10 * 2^p m^2)$ flops and the storage of $(2p+9)m^2 + (8 * 2^p - 6)m$ real numbers, where $p = \log_2 n$.

Some identical computations are imposed by program realization on formulas (12) and (13). It is clear, that the vectors $f_-^{(j)}$ are not used in the next calculations. Based on this consideration and according to the special structure of the cells $D_1^{(j)}, D_2^{(j)}$ and $L^{(j)}$ the next new algorithm - modification of Cyclic Reduction is developed. It needs less flops and less storage.

3.4 Algorithm CRM (Cyclic Reduction-Modification)

1. Find the matrices $A^{(j)}, B^{(j)}, F^{(j)}$ and the vectors $f_+^{(j)}$ by the scheme
 - 1.1 Put

$$A^{(0)} = A, B^{(0)} = B, F^{(0)} = F,$$

$$f_{-k} = f_{2k-1}, \quad \text{for } k = 1, \dots, 2^{p-1}.$$

1.2 For $j = 1, 2, \dots, p-1$ compute

$$\begin{aligned} W_1 &= B^{(j-1)} A^{(j-1)^{-1}}, \\ W_2 &= B^{(j-1)*} A^{(j-1)^{-1}}, \\ W_3 &= W_1 B^{(j-1)*}, \\ A^{(j)} &= A^{(j-1)} - W_2 B^{(j-1)} - W_3, \\ B^{(j)} &= -W_1 B^{(j-1)}, \\ F^{(j)} &= F^{(j-1)} - W_3, \\ f_1 &= f_{-1} - W_1 f_{+1}^{(j-1)}, \\ f_k &= f_{-k} - W_2 f_{+k-1}^{(j-1)} - W_1 f_{+k}^{(j-1)}, \quad \text{for } k = 2, \dots, 2^{p-j}, \\ \left. \begin{aligned} f_{-k} &= f_{2k-1} \\ f_{+k}^{(j)} &= f_{2k} \end{aligned} \right\} \quad \text{for } k = 1, \dots, 2^{p-j-1}. \end{aligned}$$

1.3 Find

$$\begin{aligned} W_1 &= B^{(p-1)} A^{(p-1)^{-1}}, \\ F^{(p)} &= F^{(p-1)} - W_1 B^{(p-1)*}, \\ f_1 &= f_{-1} - W_1 f_{+1}^{(p-1)}. \end{aligned}$$

2. Solve the linear system $F^{(p)} x_1 = f_1$.

3. Retrieve the coordinates of the vector x by the scheme

$$\left. \begin{aligned} x_2 &= A^{(p-1)^{-1}} [f_{+1}^{(p-1)} - B^{(p-1)*} x_1], \\ x_{+k} &= A^{(j-1)^{-1}} [f_{+k}^{(j-1)} - B^{(j-1)*} x_k - B^{(j-1)} x_{k+1}] \\ &\quad \text{for } k = 1, \dots, 2^{p-j} - 1 \\ x_{+s} &= A^{(j-1)^{-1}} [f_{+s}^{(j-1)} - B^{(j-1)*} x_s], \quad s = 2^{p-j} \\ x_{2k-1} &= x_k, \quad x_{2k} = x_{+k} \quad \text{for } k = 2^{p-j}, \dots, 1 \end{aligned} \right\} j = p-1, \dots, 1$$

end.

If the matrices A and B are real, this algorithm requires $O([12pm^3 + 4 * 2^p m^2] + 2m^3 + 6 * 2^p m^2) = O([12p + 2]m^3 + 10 * 2^p m^2)$ flops and the storage of $(2p + 9)m^2 + (4 * 2^p - 1)m$ real numbers, where $p = \log_2 n$.

For the new algorithms based on the discussion in section 2.4, we must address the problem of solving the nonlinear matrix equation (16). In [6], Engwerda proves that, if A is a positive definite matrix, then solution of (16) is equivalent to the solution of following matrix equation

$$Z + \tilde{B}^* Z^{-1} \tilde{B} = I, \quad (22)$$

where $\tilde{B} = A^{-\frac{1}{2}} B A^{-\frac{1}{2}}$, $Z = A^{-\frac{1}{2}} X A^{-\frac{1}{2}}$. Thus, the results proposed in

[6,10] can be readily adapted to produce following algorithms for (16).
Let $\varepsilon > 0$ be a fixed tolerance.

Algorithm EI $_{-\gamma}$ (Engwerda; Ivanov)

1. Find the matrix $\tilde{B} = A^{-\frac{1}{2}}BA^{-\frac{1}{2}}$.
 2. Solve Equation (22) by the following algorithm:
 - 2.1 $Z_0 = \gamma I$, $(\frac{1}{2} \leq \gamma \leq 1)$.
 - 2.2 For $k = 1, 2, \dots$ compute

$$Z_{k+1} = I - \tilde{B}^* Z_k^{-1} \tilde{B},$$
 if $\|Z_k - Z_{k+1}\|_\infty = \|Z_k + \tilde{B}^* Z_k^{-1} \tilde{B} - I\|_\infty \leq \varepsilon$, then stop
 - 2.3 $Z_k \rightarrow Z_+$, where Z_+ is maximal solution of (22).
 3. Compute the solution $X = A^{\frac{1}{2}}Z_{k+1}A^{\frac{1}{2}}$.
- end.

If the matrices A and B are real, this algorithm requires $O(8m^3 + [2m^3 + 2m^3 + 2m^3]k + 4m^3) = O([6k + 12]m^3)$ flops, where k is number of iterations for solving (22). The algorithm needs to store $7m^2$ real numbers.

Since for each k , Z_k are positive definite matrices, can modify the above algorithms using the idea proposed by Zhan [15].

Algorithm EI $_{-\gamma}$ M (Engwerda; Ivanov - Modification)

1. Find the matrix $\tilde{B} = A^{-\frac{1}{2}}BA^{-\frac{1}{2}}$.
 2. Solve the equation (22) as follows:
 - 2.1 $Z_0 = \gamma I$, $(\frac{1}{2} \leq \gamma \leq 1)$.
 - 2.2 For $k = 1, 2, \dots$
 - Compute the Cholesky factorization of Z_k , $Z_k = \tilde{L}\tilde{L}^*$,
 - Solve the triangular matrix equation $\tilde{L}\tilde{Z} = \tilde{B}$,
 - Compute $Z_{k+1} = I - \tilde{Z}^*\tilde{Z}$,
 if $\|Z_k - Z_{k+1}\|_\infty = \|Z_k + \tilde{B}^* Z_k^{-1} \tilde{B} - I\|_\infty \leq \varepsilon$, then stop
 - 2.3 $Z_k \rightarrow Z_+$.
 3. Compute $X = A^{\frac{1}{2}}Z_{k+1}A^{\frac{1}{2}}$.
- end.

If the matrices A and B are real, this algorithm requires $O(8m^3 + [\frac{m^3}{3} + \frac{4m^3}{3} + 2m^3]k + 4m^3) = O([\frac{11}{3}k + 12]m^3)$ flops, where k is number of iterations. The algorithm stores $\frac{17m^2+m}{2}$ real numbers.

In case we wish to solve (16) by a direct solver, we can use the following adaption of the algorithm presented in [12].

Algorithm M (Meini)

1. Set $X_0 = A, A_0 = A, B_0 = B$.
 2. For $k = 1, 2, \dots$ compute

$$W = A_k^{-1} B_k,$$

$$B_{k+1} = B_k W,$$

$$W = B_k^* W,$$

$$A_{k+1} = A_k - B_k A_k^{-1} B_k^* - W,$$

$$X_{k+1} = X_k - W,$$
 if $\|X_{k+1} - X_k\|_\infty \leq \varepsilon$ for $\varepsilon > 0$, then stop.
 3. $X_k \rightarrow X_+$, where X_+ is maximal solution of (16).
- end.

If the matrices A and B are real, this algorithm requires $O(2m^3 + 5 * 2m^3) = O(12m^3)$ flops per iteration and the storage of $7m^2$ real numbers.

In analogy with the algorithm EI- γ M we can consider the following modification of Meini's algorithm.

Algorithm MM (Meini - Modification)

1. Set $X_0 = A, A_0 = A, B_0 = B$.
 2. For $k = 1, 2, \dots$
 - 2.1 Compute the Cholesky factorization of A_k , $A_k = \tilde{L}\tilde{L}^*$,
 - 2.2 Solve the triangular matrix equations

$$\tilde{L}\tilde{Y} = B_k,$$

$$\tilde{L}\tilde{Z} = B_k^*,$$
 - 2.3 Compute

$$W = \tilde{Y}^* \tilde{Y},$$

$$B_{k+1} = \tilde{Z}^* \tilde{Y},$$

$$A_{k+1} = A_k - \tilde{Z}^* \tilde{Z} - W,$$

$$X_{k+1} = X_k - W,$$
 if $\|X_k - X_{k+1}\|_\infty \leq \varepsilon$ for $\varepsilon > 0$, then stop.
 3. $X_k \rightarrow X_+$.
- end.

If the matrices A and B are real, this algorithm requires $O(\frac{m^3}{3} + \frac{8m^3}{3} + 3 * 2m^3) = O(\frac{27m^3}{3})$ flops, per iteration and the storage of $\frac{19m^2+m}{2}$ real numbers.

The algorithms $EI_\gamma M$ and MM require less operations per iteration at the expense of a minimal increase of the memory space. However, their MATLAB implementation shows that they are slower than EI_γ and M respectively. This is due because they call fewer built in MATLAB function.

The advantage of the algorithms M and MM is their quadratic convergence, which guarantees fewer iterations to reach the required accuracy.

Based on the discussion presented in this section, we propose the following new algorithm for (1).

3.5 Algorithm $\alpha(\beta)$

1. Solve the matrix equation (16) by algorithm α , where $\alpha \in \{EI_\gamma, EI_\gamma M, M, MM\}$.
2. Find the vector $y = N^{-1}f$ by formulas (18).
3. Compute the matrix $N^{-1}E_1$ by algorithm β , where $\beta \in \{F, R\}$.
4. Compute the solution x of (1) by formula (19) with successive calculation of the expressions:

$$C = (A - X)(N^{-1}E_1)_1 ; (I + C)^{-1} ; (A - X)y_1;$$

$$z = (I + C)^{-1}(A - X)y_1 ; x = y - N^{-1}E_1 z.$$

end.

For real matrices A and B , this algorithm requires $O(k*o_\alpha + 8nm^2 + o_\beta + 6m^3)$ flops and the storage of $m_\alpha + nm + m_\beta + (2m^2 + m)$ real numbers, where k is number of iterations for solving the matrix equation (16), o_α and m_α are respectively the number of operations and memory space required for implementation of the algorithm α . The numbers o_β and m_β are similarly defined.

4 Numerical experiments

In this section we wish to corroborate the discussion of Section 3 by solving (1), with M given as in (2), and exact solution $x = (1, 1, \dots, 1)^T$.

In our numerical experiments, M is real, symmetric, with several block size n and several size and structure of the cells A and B . The above algorithms are compared by means of execution times and accuracy of the solution.

The codes are written in MATLAB language and computations are done on a PENTIUM computer. The results of the experiments are given in separate

Table 1
Flops and memory space

Algorithm	flops	memory space
LU	$\frac{17n}{3}m^3 + O(nm^2)$	$3nm^2 + 2nm$
CHOL	$\frac{13n}{3}m^3 + O(nm^2)$	$\frac{3n}{2}m^2 + \frac{3n}{2}m$
CR	$22m^3 \log_2 n + 10nm^2 + O(nm)$	$2m^2 \log_2 n + 8nm$
CRM	$12m^3 \log_2 n + 10nm^2 + O(nm)$	$2m^2 \log_2 n + 4nm$
EI- γ (F)	$(8n + 6Iter)m^3 + O(nm^2)$	$nm^2 + (n + 1)m$
EI- γ (R)	$(4n + 6Iter)m^3 + O(nm^2)$	$nm^2 + (n + 1)m$
M(R)	$(4n + 12Iter)m^3 + O(nm^2)$	$nm^2 + (n + 1)m$

tables for each example. The following notation is used:

- LU stands for the LU algorithm.
- CHOL stands for the CHOL algorithm.
- CR stands for the CR algorithm.
- CRM stands for the CRM algorithm.
- EI-1(F) stands for the $\alpha(\beta)$ algorithm. The matrix equation (16) is solved by algorithm EI- γ with initial guess $Z_0 = I$, then $N^{-1}E_1$ is computed by algorithm F .
- EI-1(R) stands for the $\alpha(\beta)$ algorithm. The matrix equation (16) is solved by algorithm EI- γ with initial guess $Z_0 = I$, then $N^{-1}E_1$ is computed by algorithm R .
- EI- $\frac{1}{2}$ (R) stands for the $\alpha(\beta)$ algorithm. The matrix equation (16) is solved by algorithm EI- γ with initial guess $Z_0 = \frac{1}{2}I$, then $N^{-1}E_1$ is computed by algorithm R .
- M(R) stands for the $\alpha(\beta)$ algorithm. The matrix equation (16) is solved by algorithm M, then $N^{-1}E_1$ is computed by algorithm R .
- $n = 2^p$ is the block size of matrix M and m is the size of each block.

For all programs the value of ε is set to $\varepsilon = 10^{-14}$.

- $Iter$ is the smallest number k , for which:
$$\begin{aligned} \|Z_k - Z_{k+1}\|_\infty &\leq \varepsilon && \text{for algorithm EI-}\gamma, \\ \|X_k - X_{k+1}\|_\infty &\leq \varepsilon && \text{for algorithm M.} \end{aligned}$$
- $Err.$ = $\|x - \tilde{x}\|_\infty$, where \tilde{x} is the computed solution.

Table 1 reports the flops and memory space required for each program.

From Table 1. we see that algorithm CRM requires less memory space than the others. Its number of operation depends of the relationship between the sizes m and n . If $m < 5n/6p$ then algorithm CRM requires $O(10nm^2)$ flops and if $m > 5n/6p$ then - $O(12m^3 \log_2 n)$. The algorithms $EI_{\gamma}(R)$ and $M(R)$ are more effective than the classical LU and CHOL. Even under the assumption that the number of iterations $Iter$ is considerably less than n , for $m < 3$ and $p \geq 3$ they require less flops than algorithm CRM.

In the next examples the cells A and B of the matrix M are chosen in such way that they guarantee the existence of a positive definite solution of Equation (16) [9,12].

Example 1.

The cells A and B are chosen like in Example 7.3 from [9] i.e.

$$A = \begin{pmatrix} 1.20 & -0.30 & 0.10 \\ -0.30 & 2.10 & 0.20 \\ 0.10 & 0.20 & 0.65 \end{pmatrix}, \quad B = \begin{pmatrix} 0.37 & 0.13 & 0.12 \\ -0.30 & 0.34 & 0.12 \\ 0.11 & -0.17 & 0.29 \end{pmatrix}.$$

Here $\|\tilde{B}\|_2 = \|A^{-\frac{1}{2}}BA^{-\frac{1}{2}}\|_2 = 0.511$. To reach the required accuracy in solving Equation (16) Algorithm EI_1 needs 404 iterations while Algorithm M only 10 iterations. In Table 2 we present the execution time (in seconds) and the error, of each algorithm for different values of m and n .

Example 2.

We let the cells of the matrix M to be the matrices of example 5.2 from [12], i.e. $A = I$, and $B = (b_{i,j})$ a symmetric matrix, whose entries are determined as below: by the scheme:

1. Fix a real $0 \leq \alpha \leq 1/2$.
2. For $i = 1, \dots, m$
 - For $j = i, \dots, m$

$$b_{i,j} = 2 * i + j$$
 - end j .
 - Compute $s_1 = \sum_{j=1}^{i-1} b_{i,j}, \quad s_2 = \sum_{j=i}^m b_{i,j}$.
 - For $j = i, \dots, m$

$$b_{i,j} = \frac{b_{i,j}(1/2-\alpha-s_1)}{s_2}, \quad b_{j,i} = b_{i,j}$$
 - end j .
- end i .

The matrix B is symmetric, nonnegative and such that $Be = (1/2-\alpha)e$, where e is the vector having all it's entries equal to 1. Thus $\|\tilde{B}\|_2 = \|A^{-\frac{1}{2}}BA^{-\frac{1}{2}}\|_2 = 1/2 - \alpha$.

Table 2
Execution time (in seconds) and errors for Example 1

Algorithm	$m = 3$			
	$n = 2^6 = 64$		$n = 2^{10} = 1024$	
	Err.	time	Err.	time
LU	2.0650e-014	0.06	3.3640e-014	1.04
CHOL	1.3545e-014	0.05	5.1292e-014	0.88
CR	1.2079e-013	0.05	3.2019e-013	1.42
CRM	6.1284e-014	0.06	1.6398e-013	0.77
EI_1(F)	1.4211e-014	0.16	8.4155e-014	1.81
EI_1(R)	1.9540e-014	0.11	8.4155e-014	0.83
M(R)	1.2879e-014	0.06	5.8620e-014	0.77
	$n = 2^8 = 256$		$n = 2^{12} = 4096$	
	Err.	time	Err.	time
LU	2.8089e-014	0.27	3.3640e-014	5.99
CHOL	4.6407e-014	0.27	5.1292e-014	4.45
CR	3.0931e-013	0.33	3.2019e-013	11.32
CRM	1.5998e-013	0.22	1.6398e-013	3.40
EI_1(F)	2.9532e-014	0.55	3.0065e-013	9.72
EI_1(R)	2.9976e-014	0.28	3.0065e-013	4.34
M(R)	3.2863e-014	0.22	2.5757e-013	4.23

We consider the following two cases:

1. $\alpha = 0.4$; then $\|\tilde{B}\|_2 = 0.1$. To obtain the required accuracy for (16), Algorithm EI_1 needs 8 iterations, Algorithm M 4 iterations.
2. $\alpha = 0$; then $\|\tilde{B}\|_2 = \frac{1}{2}$. In this case, we solve the equation (22) by algorithm EI_ γ , with initial guess $Z_0 = \frac{1}{2}I$ ($\gamma = \frac{1}{2}$). To reach the required accuracy for (16), Algorithm EI_ $\frac{1}{2}$ needs 9 iterations. The use of Algorithm EI_1 is not recommended, because it needs more than $3 \cdot 10^6$ iterations. Algorithm M needs of 32 iterations.

In Tables 3 and 4 we give execution time (in seconds) and errors, for $\alpha = 0$ and $\alpha = 0.4$ respectively.

Example 3.

Let

$$A = \text{circ}(20, -8, 1, \dots, 1, -8),$$

be a circulant matrix and $B = I$. In this case $\|\tilde{B}\|_2 = \|A^{-\frac{1}{2}}BA^{-\frac{1}{2}}\|_2 = 0.1667$. To reach the required accuracy for (16) Algorithm EI_1 needs 10 iterations, Algorithm M - 5 iterations.

In Table 5 we give the execution time (in seconds) and the error of each algorithms for different values of m and n .

Table 3
Execution time (in seconds) and errors for Example 2: ($\alpha = 0$)

Algorithm	$m = 3$		$m = 5$		$m = 10$	
	$n = 2^6 = 64$					
	Err.	time	Err.	time	Err.	time
LU	8.8818e-015	0.11	4.3299e-015	0.11	1.3989e-014	0.11
CHOL	1.1990e-014	0.06	7.7716e-015	0.06	3.9968e-015	0.11
CR	6.3505e-014	0.05	4.9960e-015	0.05	7.6605e-015	0.11
CRM	4.3521e-014	0.06	1.6431e-014	0.06	2.2204e-015	0.06
EI_ $\frac{1}{2}$ (R)	6.3949e-014	0.05	7.1054e-014	0.06	8.5265e-014	0.06
M(R)	1.7764e-013	0.05	1.5632e-013	0.05	2.8422e-014	0.06
$n = 2^8 = 256$						
LU	5.4845e-014	0.22	2.7645e-014	0.28	8.7153e-014	0.55
CHOL	1.3767e-013	0.22	5.1958e-014	0.28	6.4615e-014	0.44
CR	1.0281e-012	0.27	4.8628e-014	0.27	9.7033e-014	0.33
CRM	6.8057e-013	0.16	2.4425e-013	0.22	4.0079e-014	0.22
EI_ $\frac{1}{2}$ (R)	1.0658e-012	0.17	4.2633e-012	0.22	2.5295e-012	0.22
M(R)	5.6843e-013	0.17	8.5265e-013	0.22	7.1054e-013	0.22
$n = 2^{10} = 1024$						
LU	3.9635e-013	1.10	6.2017e-013	1.26	2.7101e-013	2.58
CHOL	3.6748e-013	0.88	1.4135e-012	1.04	3.4261e-013	2.03
CR	1.6434e-011	1.42	6.9700e-013	1.44	1.3676e-012	1.48
CRM	1.0729e-011	0.82	3.9986e-012	0.83	6.7812e-013	0.88
EI_ $\frac{1}{2}$ (R)	3.8426e-011	0.71	6.7075e-012	0.75	3.6380e-011	1.04
M(R)	3.0809e-011	0.73	2.0236e-011	0.77	8.1968e-011	1.05
$n = 2^{12} = 4096$						
LU	5.5140e-012	5.99	4.9095e-012	6.70	4.1102e-012	19.17
CHOL	8.9115e-012	4.45	4.0730e-012	5.05	2.1547e-012	11.32
CR	2.6242e-010	11.73	1.1125e-011	11.87	2.1823e-011	11.98
CRM	1.7121e-010	3.35	6.3752e-011	3.40	1.0165e-011	3.51
EI_ $\frac{1}{2}$ (R)	1.0141e-009	4.17	9.7543e-010	4.34	5.6480e-010	5.77
M(R)	6.2664e-010	4.23	4.4201e-010	4.34	7.0941e-010	5.76

5 Conclusions

From the discussion and the results obtained by numerical experiments, we can conclude that:

1. The proposed modifications of formulas (20) and (21) (Algorithm R) lead to a considerable decrease in the number of operations, for computing the block vector $N^{-1}E_1$. That explains why the execution time for for Algorithms EI_ γ (R) and M(R) is less than for Algorithm EI_ γ (F).
2. The adapted algorithms CRM, EI_ γ (R) and M(R) essentially take advantage of the special structure of the matrix M , and this makes them more effective than the classical algorithms LU, CHOL, CR as far as

Table 4
Execution time (in seconds) and errors for Example 2: ($\alpha = 0.4$)

Algorithm	$m = 3$		$m = 5$		$m = 10$	
	$n = 2^6 = 64$					
	Err.	time	Err.	time	Err.	time
LU	4.4409e-016	0.05	4.4409e-016	0.11	8.8818e-016	0.11
CHOL	6.6613e-016	0.05	5.5511e-016	0.06	8.8818e-016	0.11
CR	4.4409e-016	0.05	5.5511e-016	0.05	6.6613e-016	0.06
CRM	4.4409e-016	0.05	5.5511e-016	0.05	6.6613e-016	0.06
EI_1(F)	6.6613e-016	0.11	3.3307e-016	0.17	08.8818e-016	0.33
EI_1(R)	6.6613e-016	0.05	3.3307e-016	0.06	8.8818e-016	0.06
M(R)	4.4409e-016	0.05	5.5511e-016	0.05	7.7716e-016	0.06
$n = 2^8 = 256$						
LU	4.4409e-016	0.21	4.4409e-016	0.27	8.8818e-016	0.55
CHOL	6.6613e-016	0.22	5.5511e-016	0.27	8.8818e-016	0.44
CR	4.4409e-016	0.27	5.5511e-016	0.28	6.6613e-016	0.28
CRM	4.4409e-016	0.18	5.5511e-016	0.20	6.6613e-016	0.22
EI_1(F)	6.6613e-016	0.44	3.3307e-016	0.60	8.8818e-016	1.16
EI_1(R)	6.6613e-016	0.17	3.3307e-016	0.22	8.8818e-016	0.22
M(R)	4.4409e-016	0.16	5.5511e-016	0.16	7.7716e-016	0.22
$n = 2^{10} = 1024$						
LU	4.4409e-016	1.04	4.4409e-016	1.27	8.8818e-016	2.59
CHOL	6.6613e-016	0.94	5.5511e-016	1.05	8.8818e-016	1.97
CR	4.4409e-016	1.43	5.5511e-016	1.42	6.6613e-016	1.48
CRM	4.4409e-016	0.77	5.5511e-016	0.88	6.6613e-016	0.83
EI_1(F)	6.6613e-016	1.76	3.3307e-016	2.52	8.8818e-016	4.78
EI_1(R)	6.6613e-016	0.76	3.3307e-016	0.83	8.8818e-016	1.10
M(R)	4.4409e-016	0.72	5.5511e-016	0.77	7.7716e-016	0.99
$n = 2^{12} = 4096$						
LU	4.4409e-016	5.99	4.4409e-016	6.70	8.8818e-016	18.51
CHOL	6.6613e-016	4.45	5.5511e-016	5.11	8.8818e-016	10.93
CR	4.4409e-016	11.81	5.5511e-016	12.14	6.6613e-016	12.31
CRM	4.4409e-016	3.35	5.5511e-016	3.40	6.6613e-016	3.57
EI_1(F)	6.6613e-016	9.62	3.3307e-016	13.74	8.8818e-016	24.33
EI_1(R)	6.6613e-016	4.22	3.3307e-016	4.28	8.8818e-016	5.99
M(R)	4.4409e-016	4.23	5.5511e-016	4.34	7.7716e-016	5.76

the number of operation, memory requirements and execution time are concerned.

3. The Algorithms CRM, EI $_{\gamma}$ (R) and M(R) are comparable for accuracy of the computed solution, execution time for required flops (for $m \leq 3$ and $p \geq 3$). For large values of the m , Algorithm CRM requires the least flops, which, together with the fact that it uses the least memory space, suggest es that it is most suitable when the block size of M is a power of two.

Table 5
Execution time (in seconds) and errors for Example 3

Algorithm	$m = 5$		$m = 7$		$m = 10$	
	$n = 2^6 = 64$					
	Err.	time	Err.	time	Err.	time
LU	6.6613e-016	0.05	6.6613e-016	0.05	1.3323e-015	0.11
CHOL	1.1102e-015	0.05	6.6613e-016	0.05	8.8818e-016	0.11
CR	6.6613e-016	0.06	1.1102e-015	0.06	1.7764e-015	0.06
CRM	7.7716e-016	0.05	1.7764e-015	0.06	1.5543e-015	0.06
EI_1(F)	3.7748e-015	0.11	2.4425e-015	0.22	2.8866e-015	0.33
EI_1(R)	3.7748e-015	0.05	2.4425e-015	0.06	2.8866e-015	0.06
M(R)	5.5511e-016	0.05	4.4409e-016	0.05	6.6613e-016	0.06
$n = 2^8 = 256$						
LU	6.6613e-016	0.33	6.6613e-016	0.38	1.3323e-015	0.50
CHOL	1.1102e-015	0.22	6.6613e-016	0.33	8.8818e-016	0.44
CR	6.6613e-016	0.27	1.1102e-015	0.28	1.7764e-015	0.28
CRM	7.7716e-016	0.17	1.7764e-015	0.22	1.5543e-015	0.22
EI_1(F)	3.7748e-015	0.60	2.4425e-015	0.83	2.8866e-015	1.15
EI_1(R)	3.7748e-015	0.22	2.4425e-015	0.22	2.8866e-015	0.23
M(R)	5.5511e-016	0.16	4.4409e-016	0.17	6.6613e-016	0.27
$n = 2^{10} = 1024$						
LU	6.6613e-016	1.26	6.6613e-016	1.82	1.3323e-015	2.59
CHOL	1.1102e-015	1.05	6.6613e-016	1.21	8.8818e-016	1.98
CR	6.6613e-016	1.48	1.1102e-015	1.42	1.7764e-015	1.48
CRM	7.7716e-016	0.82	1.7764e-015	0.88	1.5543e-015	0.88
EI_1(F)	3.7748e-015	2.58	2.4425e-015	3.46	2.8866e-015	4.83
EI_1(R)	3.7748e-015	0.77	2.4425e-015	0.83	2.8866e-015	1.10
M(R)	5.5511e-016	0.77	4.4409e-016	0.82	6.6613e-016	0.99
$n = 2^{12} = 4096$						
LU	6.6613e-016	6.70	6.6613e-016	7.58	1.3323e-015	18.290
CHOL	1.1102e-015	5.05	6.6613e-016	5.71	8.8818e-016	11.15
CR	6.6613e-016	11.84	1.1102e-015	11.91	1.7764e-015	12.58
CRM	7.7716e-016	3.69	1.7764e-015	3.74	1.5543e-015	3.87
EI_1(F)	3.7748e-015	13.89	2.4425e-015	18.29	2.8866e-015	24.71
EI_1(R)	3.7748e-015	4.45	2.4425e-015	4.67	2.8866e-015	5.93
M(R)	5.5511e-016	4.40	4.4409e-016	4.62	6.6613e-016	5.77

4. If n is not a power of two and the cells of the matrix M satisfy the conditions for existence of the solution of equation (16), then the use of Algorithms M(R) is recommended instead.

Acknowledgments

This work was partially supported by Shoumen University under contract N 17/2001 and by Norwegian Research Council through the GI4PDE project, contract no. 142955/431. I wish to thank Ivan Ivanov and Antonella Zanna for their helpful remarks and comments.

References

- [1] D. Bini, B. Meini, Solving Block Banded Block Teoplitz Systems with Structured Blocks: New Algorithms and Open Problems, in: Large-Scale Scientific Computations of Engineering and Environmental Problems II, *Notes on Numerical Fluid Mechanics*, vol.73 Vieweg, 2000, pp. 15-24.
- [2] B.L. Buzbee, G.H. Golub, C.W. Nielson, On direct methods for solving Poisson's equations, *SIAM J. Numer. Analysis* **7** (1970) 627-656.
- [3] F. Diele, L. Lopez, The use of the factorization of five-diagonal matrices by tridiagonal Teoplitz matrices, *Appl. Math. Lett.* **11** (1998) 61-69.
- [4] S.M. El-Sayed, Study of special matrices and numerical methods for special matrix equations, Ph.D. Thesis, Sofia, 1996 (in Bulgarian).
- [5] S.M. El-Sayed, I.G. Ivanov, M.G. Petkov, A new modification of the Rojo Method for solving symmetric circulant five-diagonal systems of linear equations, *Computers Math. Applic.* **35** (1998) 35-44 .
- [6] J.C. Engwerda, C.M. Ran Andre, A.L. Rijkeboer, Necessary and Sufficient Conditions for the Existence of a Positive Definite Solution of the Matrix Equation $X + A^*X^{-1}A = Q$, *Linear Algebra Appl.* **186** (1993) 255-275.
- [7] G. Fiorentino, S. Serra, Multigrid methods for symmetric positive definite block Teoplitz matrices with nonnegative generating functions, *SIAM J. Sci. Computing* **17** (1996) 1068-1081.
- [8] G. Golub, C. Van Loan, *Matrix Computation*, The John Hopkins University Press, Baltimore, 1989.
- [9] C.-H. Guo and P. Lancaster, Iterative solution of two matrix equations, *The Mathematics of Computation* **68** (1999) 1589-1603.
- [10] I. Ivanov, V. Hasanov, F.Uhlig, Iterative methods for computing a positive definite solution of matrix equations $X \pm A^*X^{-1}A = I$, submitted to Mathematics of Computation.
- [11] I. Ivanov, B. Mintchev, A Method for Solving Special Circulant Pentadiagonal Linear System, in: Large-Scale Scientific Computations of Engineering and Environmental Problems II, *Notes on Numerical Fluid Mechanics*, vol.73, Vieweg, 2000, pp. 144-151.

- [12] B. Meini, Matrix Equations and Structures: Efficient Solution of Special discrete Algebraic Riccati Equations, in: *Second Conference on Numerical Analysis and Applications*, LNIS 1988, Springer, Berlin, 2000, pp. 578-585.
- [13] J. Rissanen, Solution of linear equations with Hancel and Teoplitz matrices, *Numer. Math.* **22** (1974) 361-366.
- [14] O. Rojo, A new method for solving symmetric circulant tridiagonal systems of linear equations, *Computers Math. Applic.* **20** (1990) 61-67.
- [15] X. Zhan, Computing the extremal positive definite solutions of a matrix equation, *Siam J. Sci. Comput.* **17** (1996) 1167-1174.

Article 3

A Method for Solving Hermitian Pentadiagonal Block Circulant
Systems of Linear Equations

Published in
LNCS 2907, Springer
481-488, 2004

A Method for Solving Hermitian Pentadiagonal Block Circulant Systems of Linear Equations

Borislav V. Minchev^a, Ivan G. Ivanov^b

^a*Department of Computer Science, University of Bergen,
Thormhøllensgate 55, N-5020 Bergen, Norway*

^b*Faculty of Economics and Business Administration,
Sofia University, Sofia 1113, Bulgaria*

Abstract

A new effective method and its two modifications for solving Hermitian pentadiagonal block circulant systems of linear equations are proposed. New algorithms based on the proposed method are constructed. Our algorithms are then compared with some classical techniques as far as implementation time is concerned, number of operations and storage. Numerical experiments corroborating the effectiveness of the proposed algorithms are also reported.

Key words: linear system, block circulant matrix, matrix equation, Woodbury's formula

1991 MSC: 65F10

1 Introduction

Linear systems of equations having circulant coefficient matrices appear in many applications. For example, in finite difference approximations to elliptic equations subject to periodic boundary conditions [2,8] and in approximations of periodic functions using splines [1,9]. In case when multidimensional problems are concerned the coefficient matrices of the resulting linear systems are with block circulant structure [7].

Email addresses: Borko.Minchev@ii.uib.no (Borislav V. Minchev),
i_ivanov@feb.uni-sofia.bg (Ivan G. Ivanov).

URL: <http://www.ii.uib.no/~borko> (Borislav V. Minchev).

In this paper we propose a new method and its two modifications for solving Hermitian pentadiagonal block circulant systems of linear equations. It is known that these systems have the form

$$W x = f, \quad (1)$$

where

$$W = \begin{pmatrix} M & N & S & & & S^* & N^* \\ N^* & M & N & S & & & S^* \\ S^* & N^* & . & . & . & 0 & \\ & S^* & . & . & . & . & \\ & & . & . & . & . & \\ & & 0 & . & . & . & N & S \\ S & & & S^* & N^* & M & N \\ N & S & & & S^* & N^* & M \end{pmatrix} \quad (2)$$

is Hermitian pentadiagonal block circulant matrix with block size n . M , N and S are $m \times m$ matrices, $x = \{x_i\}_{i=1,\dots,n}$, $f = \{f_i\}_{i=1,\dots,n}$, are column vectors with block size n , x_i and f_i , are blocks with size $m \times 1$.

Our goal is to construct a new effective method for solving (1) and then to compare it with some classical techniques.

The paper is organized as follows: in Section 2 we present the new method and discuss its two modifications based on different applications of the Woodbury's formula [4]; in Section 3 we report some numerical experiments corroborating the effectiveness of the proposed algorithms.

2 A Modification of LU factorizations

Adapting the ideas suggested in [6], we construct a new method for solving linear systems with coefficient matrices of the form (2). Our approach is based on the solution of a special nonlinear matrix equation. One can find the solution of (1) using the following steps:

Step 1. Solve the parametric linear system

$$T y = f, \quad (3)$$

where

$$T = \begin{pmatrix} X & Y & S & & & \\ Y^* & Z & N & . & & 0 \\ S^* & N^* & M & . & . & \\ & & . & . & . & S \\ & 0 & . & . & . & N \\ & & S^* & N^* & M & \end{pmatrix}$$

is pentadiagonal matrix with block size n . It has a block Teoplitz structure except for the north-western corner, $y = \{y_i\}_{i=1,\dots,n}$, and $f = \{f_i\}_{i=1,\dots,n}$ are column vectors with blocks size n , y_i and f_i are blocks with size $m \times 1$.

The matrix T admits the following LU factorization

$$T = LU = \begin{pmatrix} I_m & & & & \\ Y^* X^{-1} & \cdot & & & 0 \\ S^* X^{-1} & \cdot & \cdot & & \\ & \cdot & \cdot & \cdot & \\ 0 & & S^* X^{-1} & Y^* X^{-1} & I_m \end{pmatrix} \begin{pmatrix} X & Y & S & 0 \\ \cdot & \cdot & \cdot & \\ & \cdot & \cdot & S \\ 0 & \cdot & Y & \\ & & X & \end{pmatrix},$$

where I_m is the identity matrix with size $m \times m$.

The above decomposition exists when the parameters $X = X^*$, Y and $Z = Z^*$ satisfy the relations

$$\begin{cases} Z = Y^* X^{-1} Y + X \\ N = Y^* X^{-1} S + Y \\ M = S^* X^{-1} S + Z \end{cases}. \quad (4)$$

Let us introduce the following notations

$$F = \begin{pmatrix} X & Y \\ Y^* & Z \end{pmatrix}, \quad Q = \begin{pmatrix} S & 0 \\ N & S \end{pmatrix}, \quad R = \begin{pmatrix} M & N \\ N^* & M \end{pmatrix}. \quad (5)$$

If F is a positive definite solution of the matrix equation

$$F + Q^* F^{-1} Q = R \quad (6)$$

and $X = X^* > 0$, $Z = Z^* > 0$ then the blocks X , Y and Z satisfy the system (4).

Thus, solving the linear system (3) is equivalent to solve two simpler systems

$$\begin{aligned} L z &= f, & z &= \{z_i\}_{i=1,\dots,n} \\ U y &= z, & y &= \{y_i\}_{i=1,\dots,n}. \end{aligned}$$

Step 2. Solve the pentadiagonal block Teoplitz linear system

$$P u = f, \quad (7)$$

where

$$P = \begin{pmatrix} M & N & S & & & \\ N^* & M & N & . & & 0 \\ S^* & N^* & M & . & . & \\ & . & . & . & . & \\ & & . & . & . & S \\ 0 & & . & . & . & N \\ & & & S^* & N^* & M \end{pmatrix} \quad (8)$$

is Hermitian pentadiagonal block Teoplitz matrix with block size n , $u = \{u_i\}_{i=1,\dots,n}$ and $f = \{f_i\}_{i=1,\dots,n}$ are column vectors with block size n , u_i and f_i are blocks with size $m \times 1$.

The matrices T and P satisfy the relation $P = T + J_2 \hat{V}$, where

$$J_2 = \begin{pmatrix} I_m & 0 & 0 & \dots & 0 \\ 0 & I_m & 0 & \dots & 0 \end{pmatrix}^T, \quad \hat{V} = \begin{pmatrix} M - X & N - Y & 0 & \dots & 0 \\ N^* - Y^* & M - Z & 0 & \dots & 0 \end{pmatrix}$$

are matrices with block size $n \times 2$ and $2 \times n$ respectively.

Using the Woodbury's formula we have

$$P^{-1} = T^{-1} - T^{-1} J_2 [I_{2m} + \hat{V} T^{-1} J_2]^{-1} \hat{V} T^{-1}, \quad (9)$$

where I_{2m} is the identity matrix with size $2m \times 2m$. Therefore, the solution u of (7) is given by

$$u = P^{-1} f = y - T^{-1} J_2 [I_{2m} + \hat{V} T^{-1} J_2]^{-1} \hat{V} y.$$

One can find the matrix $T^{-1} J_2$ by solving $2m$ linear systems of type (3) with right-hand sides the corresponding two different columns of J_2 . This approach does not take into account the very sparse nonzero structure of J_2 . For real M , N and S it costs $O(20nm^3)$ flops and needs to store $2nm^2$ real numbers. In order to decrease the number of operations needed to compute $T^{-1} J_2$, we consider a new approach which is motivated by the ideas suggested in [5]. Let us denote the block columns vectors of J_2 with E_1 and E_2 respectively i.e.

$$E_1 = \begin{pmatrix} I_m & 0 & 0 & \dots & 0 \end{pmatrix}^T, \quad E_2 = \begin{pmatrix} 0 & I_m & 0 & \dots & 0 \end{pmatrix}^T.$$

Put $A = Y^* X^{-1}$ and $B = S^* X^{-1}$.

The matrix T admits the following decomposition

$$T = LDL^*, \quad \text{where } L = \begin{pmatrix} I_m & & & & \\ A & . & & & 0 \\ B & . & . & & \\ & . & . & . & \\ 0 & B & A & I_m \end{pmatrix}$$

is a square matrix of block size n and $D = \text{diag}(X, \dots, X)$.

Let $(L^{-1})_{ij}$ be the blocks of the matrix L^{-1} . We have

$$(L^{-1})_{ij} = \begin{cases} 0 & i < j \\ Z_{i-j+1} & i \geq j, \end{cases} \text{ where } \begin{matrix} Z_1 = I_m, & Z_2 = -A, & Z_3 = A^2 - B, \\ Z_j = -AZ_{j-1} - BZ_{j-2} & \text{for } j = 4 \dots n. \end{matrix}$$

Obviously $D^{-1} = \text{diag}(X^{-1}, \dots, X^{-1})$.

We propose to compute $T^{-1}J_2$ by consecutive calculations of $T^{-1}E_1$ and $T^{-1}E_2$ using the following algorithm:

Algorithm RP Recursive computations for Pentadiagonal system

- Find the cells $K_i = X^{-1}Z_i$ for $i = 1, \dots, n$ by the formulas

$$\begin{aligned} K_1 &= X^{-1} \\ K_2 &= -X^{-1}A \\ K_i &= -K_{i-1}A - K_{i-2}B \quad \text{for } i = 3, \dots, n. \end{aligned}$$

- Compute the blocks $(T^{-1}E_1)_i$ and $(T^{-1}E_2)_i$ by the formulas

$$\begin{aligned} (T^{-1}E_1)_n &= K_n \\ (T^{-1}E_1)_{n-1} &= K_{n-1} - A^*(T^{-1}E_1)_n \\ (T^{-1}E_1)_i &= K_i - A^*(T^{-1}E_1)_{i+1} - B^*(T^{-1}E_1)_{i+2} \quad \text{for } i = n-2, \dots, 2 \\ (T^{-1}E_1)_1 &= X^{-1} - A^*(T^{-1}E_1)_2 - B^*(T^{-1}E_1)_3 \\ (T^{-1}E_2)_n &= K_{n-1} \\ (T^{-1}E_2)_{n-1} &= K_{n-2} - A^*(T^{-1}E_2)_n \\ (T^{-1}E_2)_i &= K_{i-1} - A^*(T^{-1}E_2)_{i+1} - B^*(T^{-1}E_2)_{i+2} \quad \text{for } i = n-2, \dots, 2 \\ (T^{-1}E_2)_1 &= -A^*(T^{-1}E_2)_2 - B^*(T^{-1}E_2)_3. \end{aligned}$$

If the blocks M , N and S are real the algorithm RP costs $O(12nm^3)$ flops and needs to store $(3n+2)m^2$ real numbers. According to the above algorithm, in the next step, we consider two different approaches for solving (1).

Step 3. Solve the system (1)

3.1 The matrix W satisfies the relation

$$W = P + \tilde{U}\tilde{V},$$

where

$$\tilde{U} = \begin{pmatrix} I_m & 0 & 0 & 0 \\ 0 & I_m & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & I_m & 0 \\ 0 & 0 & 0 & I_m \end{pmatrix}, \quad \tilde{V} = \begin{pmatrix} 0 & 0 & \dots & S^* & N^* \\ 0 & 0 & \dots & 0 & S^* \\ S & 0 & \dots & 0 & 0 \\ N & S & \dots & 0 & 0 \end{pmatrix}$$

are matrices with block size $n \times 4$ and $4 \times n$ respectively.
Using the Woodbury's formula we have

$$W^{-1} = P^{-1} - P^{-1}\tilde{U} \left[I_{4m} + \tilde{V}P^{-1}\tilde{U} \right]^{-1} \tilde{V}P^{-1},$$

where I_{4m} is the identity matrix of size $4m \times 4m$.

The solution x of (1) is obtained from the vector u by the formula

$$x = W^{-1}f = u - P^{-1}\tilde{U} \left[I_{4m} + \tilde{V}P^{-1}\tilde{U} \right]^{-1} \tilde{V}u.$$

Denote the block columns vectors of \tilde{U} by E_1, E_2, E_{n-1} and E_n respectively. Thus, the computation of $P^{-1}\tilde{U}$ can be done by consecutive calculations of $P^{-1}E_1, P^{-1}E_2, P^{-1}E_{n-1}$ and $P^{-1}E_n$ using the formula (9). For $i = 1, 2, n-1, n$ we have

$$P^{-1}E_i = T^{-1}E_i - T^{-1}J_2 \left[I_{2m} + \hat{V}T^{-1}J_2 \right]^{-1} \hat{V}T^{-1}E_i. \quad (10)$$

Note that the numerical implementation of formulas (10) is very "cheap", since we already know from Step 2 the elements $T^{-1}J_2$ and $\left[I_{2m} + \hat{V}T^{-1}J_2 \right]^{-1}$. We recommend formulas (10) instead of solving $4m$ linear system of the form (7) with right hand side the corresponding column vectors of \tilde{U} . It is easy to observe that the blocks of $T^{-1}E_{n-1}$ and $T^{-1}E_n$ satisfy the relations

$$\begin{aligned} (T^{-1}E_{n-1})_n &= K_2^*, \\ (T^{-1}E_{n-1})_i &= K_{n-i}^* - K_{n-i+1}^*A \quad \text{for } i = n-1, \dots, 1, \\ (T^{-1}E_n)_i &= K_{n+1-i}^* \quad \text{for } i = n, \dots, 1, \end{aligned}$$

where K_i for $i = 1, \dots, n$ are the blocks from algorithm RP.

3.2 In order to decrease the size of the inverse matrix in the Woodbury's formula, we propose the following decomposition of the matrix W

$$W = \begin{pmatrix} P & V \\ V^* & R \end{pmatrix},$$

where P is from (8), with block size $n-2 \times n-2$, R is from (5) and

$$V^* = \begin{pmatrix} S & 0 & \dots & S^* & N^* \\ N & S & \dots & 0 & S^* \end{pmatrix}$$

is a matrix with block size $2 \times n-2$.

Put

$$\begin{aligned} \hat{x} &= (x_1, \dots, x_{n-2})^T, \quad \tilde{x} = (x_{n-1} \ x_n)^T, \quad x = \begin{pmatrix} \hat{x} \\ \tilde{x} \end{pmatrix}, \\ \hat{f} &= (f_1, \dots, f_{n-2})^T, \quad \tilde{f} = (f_{n-1} \ f_n)^T, \quad f = \begin{pmatrix} \hat{f} \\ \tilde{f} \end{pmatrix}. \end{aligned}$$

In this notations system (1) can be written in the form

$$\begin{pmatrix} P & V \\ V^* & R \end{pmatrix} \begin{pmatrix} \hat{x} \\ \tilde{x} \end{pmatrix} = \begin{pmatrix} \hat{f} \\ \tilde{f} \end{pmatrix},$$

which is equivalent to

$$\begin{cases} G\hat{x} = r \\ \tilde{x} = R^{-1}(\tilde{f} - V^*\hat{x}), \end{cases}$$

where $G = P - VR^{-1}V^*$, $r = \hat{f} - VR^{-1}\tilde{f}$.

By Woodbury's formula we have

$$G^{-1} = P^{-1} + P^{-1}V[R - V^*P^{-1}V]^{-1}V^*P^{-1}.$$

Therefore,

$$\hat{x} = G^{-1}r = z + P^{-1}V[R - V^*P^{-1}V]^{-1}V^*z,$$

where $z = P^{-1}r$ can be computed by means of Step 2.

Denote the block columns vectors of V with H_1 and H_2 respectively. The computation of $P^{-1}V$ can also be done by consecutive calculations of $P^{-1}H_1$ and $P^{-1}H_2$ using the formula (9). For $i = 1, 2$

$$P^{-1}H_i = T^{-1}H_i - T^{-1}J_2[I_{2m} + \hat{V}T^{-1}J_2]^{-1}\hat{V}T^{-1}H_i.$$

The numerical implementation of the last formulas is again “cheap”, since we already know from Step 2 the elements $T^{-1}J_2$ and $[I_{2m} + \hat{V}T^{-1}J_2]^{-1}$. The blocks of $T^{-1}H_1$ and $T^{-1}H_2$ satisfy the relations

$$\begin{aligned} (T^{-1}H_1)_i &= (T^{-1}E_1)_i S^* + K_{n-2-i}^* S + K_{n-i-1}^* \tilde{Q} \quad \text{for } i = 1, \dots, n-3 \\ (T^{-1}H_1)_{n-2} &= (T^{-1}E_1)_{n-2} S^* + K_1^* \tilde{Q} \\ (T^{-1}H_2)_i &= (T^{-1}E_1)_i N^* + (T^{-1}E_2)_i S^* + K_{n-1-i}^* S \quad \text{for } i = 1, \dots, n-2, \end{aligned}$$

where $\tilde{Q} = N - AS$ and K_i for $i = 1, \dots, n$ are the blocks from Algorithm RP.

3 Numerical experiments

In this section we compare our algorithms with some classical techniques for solving (1), with W given as in (2), and the exact solution $x = (1, 1, \dots, 1)^T$.

In our numerical experiments, W is Hermitian pentadiagonal block circulant with several block sizes n . The algorithms are compared by means of execution time and accuracy of the solution.

Table 1
Execution time (in seconds) and errors for Example 1

Algorithm	$m = 3$					
	$n = 4000$		$n = 6000$		$n = 8000$	
	Err.	time	Err.	time	Err.	time
LU	1.4482e-015	2.95	1.4482e-015	6.04	1.4482e-015	10.15
CHOL	2.2888e-015	2.00	2.2888e-015	3.23	2.2888e-015	5.44
M_RP(4m)	4.2635e-014	1.95	4.1064e-014	3.14	4.7126e-014	5.02
M_RP(2m)	3.3956e-014	1.63	4.1081e-014	2.62	6.0280e-014	4.34

The codes are written in MATLAB language and the computations are done on an AMD computer. The results of the experiments are given in different tables for each example.

The following notations are used: LU stands for classical LU factorization; $CHOL$ stands for the classical Cholesky factorization; M_RP(4m) stands for algorithm based on the proposed new method using Step 3.1; M_RP(2m) stands for algorithm based on the proposed new method using Step 3.2; $Err. = \|x - \tilde{x}\|_\infty$, where \tilde{x} is the computed solution.

To solve the system (1) we need to compute a positive definite solution of the matrix equation (6). The sufficient condition for the existence of a positive definite solution is $\|R^{-\frac{1}{2}} Q R^{-\frac{1}{2}}\| \leq \frac{1}{2}$, (see [3]). In the next two examples the cells of the matrix W , which form the matrices R and Q , are chosen to satisfy this condition.

Example 1 Let

$$M = \begin{pmatrix} 8 & 1-i & 1.5 \\ 1+i & 9 & 1 \\ 1.5 & 1 & 8 \end{pmatrix}, \quad N = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 1-i & 0 & 0 \end{pmatrix},$$

$$S = \begin{pmatrix} 1.2-3i & -0.3-i & 0.1 \\ -0.30 & 2.1 & 0.2 \\ 0.1 & 0.2 & 0.65+2i \end{pmatrix}.$$

In Table 1, we present the execution time (in seconds) and the error for each algorithm for different values of n .

Example 2 Let

$$M = \text{circ}(22, -8, 1, \dots, 1, -8), \quad N = \text{circ}(-7.2, 1.8, \dots, 1.8)$$

are circulant matrices and $S = I$.

The results from the numerical experiments for this example are given in Table 2.

Table 2
Execution time (in seconds) and errors for Example 2

Algorithm	$m = 7$					
	$n = 4000$		$n = 6000$		$n = 8000$	
	Err.	time	Err.	time	Err.	time
LU	1.7764e-015	3.49	1.7764e-015	6.66	1.7764e-015	11.01
CHOL	1.9984e-015	2.39	1.9984e-015	3.90	1.9984e-015	5.98
M_RP(4m)	2.6182e-014	2.14	3.2072e-014	3.40	3.7038e-014	4.88
M_RP(2m)	2.5537e-014	1.87	3.1282e-014	2.83	3.6124e-014	4.47

4 Conclusions

The proposed new algorithms M_RP(2m) and M_RP(4m) are faster than the classical *LU* and *CHOL*. From the theoretical discussions and the numerical experiments, we can conclude that Algorithm M_RP(2m) is most suitable for implementation. This is due to the size of the inverse matrix in the Woodbury's formula. The inverse matrix in M_RP(2m) is two times smaller than the inverse matrix in M_RP(4m). This leads to a considerable decrease in the execution time.

The complexity of the proposed new algorithms is $O(nm^3)$. For comparison, algorithm based on the Fast Fourier Transform (FFT) has complexity $O(nm \log(n))$, but it can be implemented only when the block size of the matrix W is power of two. Our method does not have these restrictions. The only restriction on the applicability of our method is related with the existence of a solution of the matrix equation (6).

References

- [1] J. H. Ahlberg, E. N. Nilson, J. L. Walsh, *The Theory of Splines and Their Applications*, Academic Press, New York, 1967.
- [2] R. Chan, T. Chan, Circulant preconditioners for elliptic problems, *sJ. Numer. Linear Algebra Appl.* **1**, (1992), 77-101.
- [3] J. C. Engwerda, On the Existence of a Positive Definite Solution of the Matrix Equation $X + A^T X^{-1} A = I$, *Linear Algebra Appl.*, **194**, (1993), 91-108.
- [4] G. Golub, C. Van Loan, *Matrix Computation*, The John Hopkins University Press, Baltimore, 1989.
- [5] B. Minchev, Some Algorithms for Solving Special Tridiagonal Block Teoplitz Linear Systems, *J. Comp. and Appl. Math.* vol. **156/1**, (2003), 179-200.

- [6] B. Mintchev, I. Ivanov, On the Computer Realization of Algorithms for Solving Special Block Linear Systems, *Applications of Mathematics in Engineering and Economics'27*, Heron Press, Sofia (2002), 303-313.
- [7] C. Mingkui, On the Solution of circulant linear systems, *SIAM J. Numer. Analysis* **24**, (1987), 668-683.
- [8] W. L. Wood, Periodicity effects on the iterative solution of elliptic difference equations, *SIAM J. Numer. Analysis* **8**, (1971), 439-464.
- [9] U. Zavyalov, B. Kvasov, V. Miroshnichenko, *Spline Functions Methods* (in Russian), Moscow, 1980.

ISBN 82-92610-01-4
Bergen, Norway 2004