NUMERICAL SOLUTIONS OF THE REACTION-DIFFUSION EQUATIONS BY EXPONENTIAL INTEGRATORS

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Mathematics

by
Melek SOFYALIOĞLU

July 2014
İZMİR

We approve the thesis of **Melek SOFYALIOĞLU**

Examining Committee Members:

Prof. Dr. Gamze TANOĞLU
Department of Mathematics, İzmir Institute of Technology

Assist. Prof. Dr. H.Seçil ARTEM
Department of Mechanical Engineering, İzmir Institute of Technology

Assist. Prof. Dr. Fatih ERMAN
Department of Mathematics, İzmir Institute of Technology

11 July 2014

Prof. Dr. Gamze TANOĞLU
Supervisor, Department of Mathematics
İzmir Institute of Technology

Prof. Dr. Oğuz YILMAZ
Head of the Department of
Mathematics

Prof. Dr. R. Tuğrul SENGER
Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGMENTS

This thesis is the consequence of a five-term study and now I would like to express my gratitude to all the people supporting me from all the aspects for the period of my thesis.
Firstly, I would like to thank and express my most sincere and deepest gratitude to my advisor Prof. Dr. Gamze TANOĞLU, for her understanding, help, encouragement and patience during my studies to prepare this thesis. And I would like to thank to The Scientific and Technological Research Council of Turkey (TÜBİTAK) for its financial support.
Finally, I am also grateful to my family and my friends Neslişah İMAMOĞLU, Barış ÇİÇEK, Yeşim ÇİÇEK and Gizem KAFKAS for their confidence to me and for their endless supports.

ABSTRACT

NUMERICAL SOLUTIONS OF THE REACTION-DIFFUSION EQUATIONS BY EXPONENTIAL INTEGRATORS

This thesis presents the methods for solving stiff differential equations and the convergency analysis of exponential integrators, namely the exponential Euler method, exponential second order method, exponential midpoint method for evolution equation. It is also concentrated on how to combine exponential integrators with the interpolation polynomials to solve the problems which has discrete force. The discrete force is approximated by using the Newton divided difference interpolation polynomials. The new error bounds are derived. The performance of these new combinations are illustrated by applying to some well-known stiff problems. In computational part, the methods are applied to linear ODE systems and parabolic PDEs. Finally, numerical results are obtained by using MATLAB programming language.

ÖZET

REAKSİYON DİFÜZYON DENKLEMLERİNİN ÜSTEL İNTEGRATÖRLERLE SAYISAL ÇÖZÜMLERİ

Bu tez stiff diferansiyel denklemleri çözmek için kullanılan yöntemleri ve üstel integratörlerin yakınsaklık analizini sunmaktadır. Ayrıca, ayrık güç içeren probremleri çözmek için üstel integratörlerin interpolasyon polinomlarıyla nasıl birleştirileceğine konsantre olunmuştur. Verilen ayrık güç Newton bölünmüş farklar interpolasyon polinomu kullanılarak yaklaşık olarak hesaplanmıştır. Yeni hata sınırları elde edilmiştir. Yapılan yeni kombinasyonların performansı bazı iyi bilinen stiff problemlere uygulanarak açıklanmıştır. Sayısal kısımda, yöntemler lineer adi diferansiyel denklem sistemlerine ve parabolik kısmi diferansiyel denklemlere uygulanmıştır. Son olarak, sayısal sonuçlar MATLAB programlama dili kullanılarak elde edilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES viii

LIST OF TABLES ix

CHAPTER 1. INTRODUCTION 1

CHAPTER 2. STIFF DIFFERENTIAL EQUATIONS 3
2.1. History of Stiff Differential Equations 3
2.2. Stiffness 4

CHAPTER 3. NUMERICAL METHODS FOR STIFF DIFFERENTIAL EQUATIONS 10
3.1. Traditional Implicit Methods 10
3.1.1. Backward Differentiation Formula 10
3.1.2. Implicit Runge Kutta Method 12
3.1.3. Trapezoid Method 13
3.2. Exponential Integrators 14
3.2.1. Derivation of the ϕ Functions 16
3.2.1.1. Derivation of the Methods with ϕ Functions 17
3.2.2. Exponential Euler Method 19
3.2.3. Second Order Method 20
3.2.4. Exponential Midpoint Rule 21
3.2.5. Exponential Rosenbrock Method 21
3.3. Error Analysis via Richardson Extrapolation 23

CHAPTER 4. ERROR ANALYSIS 25
4.1. Analytical Framework 25
4.2. Convergence of Exponential Euler Method 29
4.3. Convergence of the Second Order Method 36
4.4. Convergence of the Exponential Midpoint Method 39

CHAPTER 5. INTERPOLATION THEORY 44
5.1. Newton Divided Differences Polynomial Approximation 44

CHAPTER 6. ERROR ANALYSIS OF EXPONENTIAL INTEGRATORS WITH DISCRETE FORCE 49
6.1. Error Analysis of Exponential Euler Method 49
6.2. Error Analysis of Second Order Method 51
6.3. Error Analysis of Exponential Midpoint Method 52

CHAPTER 7. NUMERICAL EXPERIMENT 56
7.1. Application of Various Exponential Integrators on the Prothero-Robinson Equation 56
7.2. One Dimensional Example to Explain Stiffness 59
7.3. Two Dimensional ODE Problem 62
7.4. Example of Reaction-Diffusion Equations 66
7.4.1. Linear Problem 66
7.4.2. Semilinear Problem: The Fisher Equation 68
7.4.3. Semilinear Problem: The Allen Cahn Equation 71

CHAPTER 8. CONCLUSION 77

REFERENCES 78

APPENDIX A. MATLAB CODES FOR THE APPLICATIONS OF THE EXPONENTIAL INTEGRATORS 80

LIST OF FIGURES

Figure **Page**

Figure 7.1. Comparison of numerical methods with exact solution for time step Δt = 0.1. 58

Figure 7.2. Order plot for the exponential methods applied to the Prothero equation for α = 2. 58

Figure 7.3. Order plot for the exponential methods applied to the Prothero equation for α = 4. 59

Figure 7.4. Divided difference polynomial of given data points and cos(x). 60

Figure 7.5. Comparison of different exponential integrators with exact solution for Δx = 0.1. 61

Figure 7.6. Order of exponential methods. 63

Figure 7.7. Order plot of the exponential methods with divided difference. 64

Figure 7.8. Divided difference polynomial of given data points and e^t. 65

Figure 7.9. Comparison of numerical methods with exact solution for time step Δt = 0.015. 65

Figure 7.10. Order graphic. 68

Figure 7.11. Analytic and computed solutions of second order method for linear parabolic equation Δt = 0.05 and Δx = 0.01. 69

Figure 7.12. Second order method solution for different values of time with Δt = 0.05 and Δx = 0.01. 69

Figure 7.13. Order plot for the exponential Rosenbrock Euler method applied to Fisher equation. 72

Figure 7.14. Numerical solution of Fisher equation Δt = 0.01 and Δx = 0.5. 72

Figure 7.15. Computed solutions of Fisher equation for different values of time with Δt = 0.01 and Δx = 0.5. 73

Figure 7.16. Order plot for Allen Cahn equation. 75

Figure 7.17. Numerical Solution of Allen Cahn equation Δt = 0.1 and Δx = 0.05. 75

Figure 7.18. Computed solutions of Allen-Cahn equation for different values of time with Δt = 0.1 and Δx = 0.05. 76

LIST OF TABLES

Table **Page**

Table 5.1. Format for constructing divided differences of $g(t)$. 46
Table 6.1. g is given as a function of time 49
Table 6.2. g and g' are given as a function of time 53
Table 7.1. Comparison of errors for various exponential integrators different Δt values. 57
Table 7.2. Comparison of traditional methods and exponential euler method errors with discrete force for different Δx values. 61
Table 7.3. Comparison of different exponential integrators errors with discrete force for different Δx values. 61
Table 7.4. Given discrete data points. 62
Table 7.5. Comparison of different methods errors for different Δt values. 63
Table 7.6. Comparison of different exponential and traditional methods with divided difference errors for different Δt values. 64
Table 7.7. Divided difference table of $g(t)$. 67
Table 7.8. Comparison of different exponential integrators with divided difference errors for different Δt values when $\Delta x = 0.01$ in L_∞ norm. 68
Table 7.9. Comparison of exponential Rosenbrock Euler method errors in different norms for $\Delta x = 0.5$. 71
Table 7.10. Comparison of exponential Rosenbrock Euler method error for different norms. 74

CHAPTER 1

INTRODUCTION

Numerical methods for solving initial value problems have a long history in mathematics. These methods can be either explicit or implicit. This thesis is concentrated on numerical solution of the stiff problems. One of the technique to solve such problems is exponential integrators.

Stiff systems of ordinary differential equations arise frequently while solving partial differential equations by spectral method. These problems are mostly solved by implicit methods. Implicit methods are usually more costly than explicit methods but their stability properties enable us to study for large time steps and this compensate the extra cost.

Exponential integrators are explicit methods which is considered as generalizations of the implicit methods. This fact allows to use much more large time steps although explicit methods require much smaller time steps than implicit methods. For this reason, exponential integrators are suitable methods to solve stiff problems. We can find applications of exponential integrators in applied mathematics, physics, financial mathematics and statistics. More detail can be read from (Kandolf, 2011).

We will go on a historical background of exponential integrators. The idea behind exponential integrators dates back to late 1950's. The beginning of this methods takes place in the study of Hersch in 1958. In these years traditional integrators for ODEs can not compute the true solution when it is compared with the exact solution. When Hersch realized that traditional integrators have problems, he suggested a new exact integration scheme for constant coefficient linear ODEs (Michels& Sobottka & Weber, 2014). This scheme is the first exponential integrator. In 1960, Certaine developed the first multistep method by using variation of constants approach with an algebraic approximation of the nonlinear part. After this marks, the research on exponential integrators heightened. In 1963, Pope proposed linearizing a nonlinear differential equation, which involves exponential term, at each time step. This generates the basis for the Rosenbrock methods. (Pope, 1963). In 1967, Lawson solved an IVP by using a Runge-Kutta process numerically. In his article, he changed classical coefficients of Runge-Kutta for exponential functions. On the other words, exponential functions are used as coefficients. This article was the first Runge-Kutta based on exponential method. (Lawson, 1967)

In 1978, Friedli proposed higher order exponential Runge-Kutta methods to solve non-stiff differential equations. He used classical Taylor series expansion. In 1998, Hochbruck, Lubich

and Selhofer introduce a new integration method which uses multiplication of matrix-vectors with the exponential of the Jacobian (Hochbruck & Lubich & Selhofer , 1998).
In 2005, Hochbruck and Ostermann derived higher order exponential Runge-Kutta methods to solve stiff differential equations. (Hochbruck & Ostermann & Schweitzer, 2008)
In the literature, most papers use variation of constants formula to solve parabolic problems. The initial value problems under consideration can be written as follows:

$$\begin{aligned} u'(t) + Au(t) &= g(t, u(t)), \\ u(0) &= u_0, \end{aligned} \tag{1.1}$$

where $A \in \mathbb{R}^{n\times n}$ is a constant linear operator that represents higher order spatial derivatives and $g \in \mathbb{R}^n$ is a nonlinear operator. All of papers use e^{-hA}, which is the exponential function of a matrix $-hA$ in order to take a step from time t to time $t + h$ where h is the step size on time.
This thesis is organized as follows: Chapter 2 introduces information about stiff differential equations, the relevant definitions and examples about stiff ODE and PDE. We shall give a brief information about stiff differential equations and test the stiffness of ODEs and PDEs. Chapter 3 focuses on the numerical methods to solve stiff differential equations. This chapter is formed mainly three parts. The first part is traditional implicit methods, the second part is exponential methods and the last part is Richardson extrapolation which is well-known error estimation. In Chapter 4, we study in detail the stability issue for general case of the exponential methods by using the semigroup approaches. The convergence results for exponential methods are given. Our convergence proofs are based on a discrete type of variation of constants formula. We will define the numerical methods with the help of the variation of constants formula. In Chapter 5, we define Newton divided difference polynomial to approximate given uniform distinct datas. In Chapter 6, we deal with exponential integrators with discrete force. Therefore, we combine exponential integrators with Newton divided difference interpolation polynomial. Then, we find error bounds for this new combination.In Chapter 7, all methods, which are introduced in previous chapters, are applied to various stiff problems, in order to test the performance of these methods. Finally, the conclusion is given briefly in Chapter 8.

CHAPTER 2

STIFF DIFFERENTIAL EQUATIONS

In this chapter, we will review the various techniques to solve stiff problems. The classical methods do not work effectively for such problems. We will deal with firstly the history of stiff differential equations and then the detection of stiffness in ODEs and PDEs, respectively.

2.1. History of Stiff Differential Equations

The earliest detection of stiffness in differential equations presented by two chemists Curtiss and Hirschfelder in 1952. They coined the term and described the existence and the nature of stiffness. They profitted by the experience of John Tukey with the solution of ”stiff” equations (Curtis & Hirschfelder, 1952).

In 1963, Dahlquist defined the problem and pointed out the difficulties that the classical methods with solving stiff differential equations. He dealed with the problems in stability. He said in Aiken (1985) that ”... around 1960, thing became completely different and everyone became aware that the world was full of stiff problems.” (Hairer & Wanner, 2000)

In 1968, Gear became one of the most important names in this area. In 1979, Gear and Shampine wrote an article. The purpose of this article was to aid people who are interested in the solution of stiff ordinary differential equations. They identified the problem area and described the characteristics shared by methods for the numerical solution of stiff problems (Shampine & Gear, 1976). In 1970, Liniger designed efficient algorithms for solving stiff systems of ordinary differential equations (Liniger & Willoughby, 1970). In 1973, Lambert examined critically various qualitative statements including the notion of stiffness. One of them is ’A linear constant coefficient system is stiff if all of its eigenvalues have negative real part and the stiffness ratio is large.’ This statement is adopted as a definition of stiffness. He selected the most satisfactory of these statements as a ’definition’ of stiffness. This is: ’If a numerical method with a finite region of absolute stability, applied to a system with any initial condition, is forced to use in a certain interval of integration a step lentgh which is excessively small in relation to the smoothness of the exact solution in that interval, then the system is said to be stiff in that interval’ (Lambert, 2000). Lambert thought that the rate of stiffness in real life problems would become more important year by year. In 1996, Spijker

defined that 'Initial value problems are stiff if they are difficult to solve by ordinary, explicit step-by-step methods, whereas certain implicit methods perform quite well' (Spijker, 1995). In 2009, Brugnano , Mazzia And Trigiante rewieved the evolution of stiffness. According to historical background the definition of stiffness can be formalized as follows:

Definition 2.1 *A linear differential system*

$$u'(t) = Au(t) + f(t), \quad u(0) = u_0,$$

where $A \in \mathbb{R}^{n\times n}$ *and* $u, f, u_0 \in \mathbb{R}^n$.
This system is said to be stiff if and only if

i) *For all i,* $Re(\lambda_i) < 0$,

ii) $\frac{max|Re(\lambda_i)|}{min|Re(\lambda_i)|} \gg 1$, *where* λ_i *are eigenvalues of A for* $i = 1, 2, ..., n$.

We called $\frac{max|Re(\lambda_i)|}{min|Re(\lambda_i)|}$ *as stiffness ratio.*

We will check the stiffness of given any equation the aid of this definition. Next two examples are given in order to illuminate stiff differential equations.

2.2. Stiffness

In this section, we focus our attention on two examples to illustrate the stiff differential equation. We first consider the linear ODE system:

$$\begin{aligned} u_1' &= -u_1 + e^t, \quad u_1(0) = 1, \\ u_2' &= 2u_1 - 100u_2, \quad u_2(0) = 0, \end{aligned} \tag{2.1}$$

where $t \in [0, 0.3]$. We can rewrite equation (2.1) following matrix form,

$$u'(t) = \begin{bmatrix} -1 & 0 \\ 2 & -100 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} e^t \\ 0 \end{bmatrix}, \quad u_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{2.2}$$

This system is equivalent to following form

$$u'(t) = Au(t) + f(t), \;\; u(0) = u_0, \tag{2.3}$$

where $A \in \mathbb{R}^{2\times 2}$ and $u, f, u_0 \in \mathbb{R}^2$. Then, we can check the stiffness according to Definition (2.1). We have to find eigenvalues of the coefficient matrix. The eigenvalues of the given matrix A are $\lambda_1 = -100$ and $\lambda_2 = -1$. Both of the eigenvalues are negative and stiffness ratio=$\frac{|\lambda_2|}{|\lambda_1|} = 100 \gg 1$. So, linear equation system (2.1) is said to be stiff.
Our next example is the detection of stiffness in PDE. Let us consider the heat equation:

$$\frac{\partial u(x,t)}{\partial t} \;\; = \;\; \frac{\partial^2 u(x,t)}{\partial x^2},$$

with the initial condition and the boundary conditions

$$u(x,0) \;\; = \;\; f(x), \tag{2.4}$$
$$u(0,t) \;\; = \;\; u(1,t) = 0, \tag{2.5}$$

where $x \in (0,1]$, $t \in (0,T]$.
We will solve the diffusion problem using the finite difference method. The basic idea of the method is to replace the spatial derivation in partial differential equation with algebraic approximation. We approximate the spatial partial derivative of u_{xx} using the central difference formula. The approximate solution of $u(x,t)$ at $x = x_n$ is denoted by $u_n(t)$

$$\begin{aligned} \frac{\partial u_n(t)}{\partial t} \;\; &= \;\; \frac{u_{n+1}(t) - 2u_n(t) + u_{n-1}(t)}{(\Delta x)^2}, \\ u_0(t) \;\; &= \;\; u_N(t) = 0 \;\; , \;\; t \in (0,T], \\ u_n(0) \;\; &= \;\; f(x_n) \;\; , \;\; n = 1, ..., N-1, \end{aligned} \tag{2.6}$$

where $\Delta x = \frac{1}{N}$ and $x_n = n\Delta x$, $n = 1, ..., N-1$. Suppose that the step size $\Delta x = h$. It is convenient to write (2.4) in matrix form

$$\begin{bmatrix} u_1' \\ u_2' \\ \vdots \\ \\ u_{N-1}' \end{bmatrix} = \frac{1}{h^2} \begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & \ddots & \ddots & & \\ & & & \ddots & \ddots & 1 & \\ & & & & & -2 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \\ u_{N-1} \end{bmatrix}.$$

This system can be written as

$$u'(t) = Au(t), \;\; u(0) = u_0, \tag{2.7}$$

where $u_0 = [f(x_1), ..., f(x_{N-1})]^T$ is the initial condition. Here we consider the periodic boundary conditions. Boundary conditions are embedded into the matrix. To examine the stiffness of the given diffusion problem, we need to find the eigenvalues of A. These eigenvalues are real and can be given by the following equality

$$\lambda_j = -\frac{4}{h^2} sin^2 \frac{j\pi}{2N}, \quad 1 \leq j \leq N-1.$$

The proof can be obtained by showing a relationship between the characteristic polynomial for A and Chebyshev polynomials

$$\lambda_{N-1} \leq \lambda_j \leq \lambda_1, \tag{2.8}$$

with the following results for λ_{N-1} and λ_1

$$\begin{aligned} \lambda_{N-1} &= -\frac{4}{h^2} sin^2 \frac{(N-1)\pi}{2N} \approx -\frac{4}{h^2}, \\ \lambda_1 &= -\frac{4}{h^2} sin^2 \frac{\pi}{2N} \approx -\pi^2, \end{aligned}$$

with the approximations available for larger N. Let r be an eigenvector with corresponding eigenvalue λ for A.

For a vector $r = (r_1, r_2, ..., r_{N-1})$ to be an eigenvector for A with corresponding eigenvalue λ we must have

$$\frac{1}{h^2}(r_{n-1} - 2r_n + r_{n+1}) = \lambda r_n, \quad n = 2, ..., N-2, \tag{2.9}$$

It is usually not easy to find eigenvalues and eigenvectors for such a matrix, but if we have a prediction then it is very easy to check whether it fits or not. A good suggestion for r can be to take

$$r_n = sin(n\theta), \quad n = 1, ..., N-1, \tag{2.10}$$

we work out

$$\begin{aligned} r_{n-1} + r_{n+1} &= sin(n-1)\theta + sin(n+1)\theta, \\ &= 2sin(n\theta)cos(\theta), \\ &= 2r_n cos\theta. \end{aligned} \tag{2.11}$$

When we replace (2.11) in (2.9), this gives

$$\begin{aligned} \lambda &= \frac{1}{h^2}(-2 + 2cos(\theta)), \\ &= \frac{-4}{h^2}sin^2(\theta/2). \end{aligned}$$

In addition to the $N-3$ equations we must also have the similar relations for $n = 1$ and $n = N-1$:

$$\begin{aligned} 2r_1 - r_2 &= \lambda r_1, \\ -r_{N-2} + 2r_{N-1} &= \lambda r_{N-1}. \end{aligned}$$

These are fulfilled automatically if we can manage to have $r_0 = r_N = 0$. For r_N we must require

$$r_N = sin(N\theta) = 0. \tag{2.12}$$

Note that the roots are

$$N\theta_j = j\pi, \quad j = 1, 2, ... \tag{2.13}$$

We therefore define

$$\theta_j = \frac{j\pi}{N}, \quad j = 1, 2, ..., N-1, \tag{2.14}$$

and with these $N-1$ values of θ we have a set of $N-1$ orthogonal eigenvectors and corresponding eigenvalues for A:

$$\lambda_j = -4sin^2\Big(\frac{j\pi}{2N}\Big). \tag{2.15}$$

Directly examining this formula,

$$\lambda_{N-1} \leq \lambda_j \leq \lambda_1. \tag{2.16}$$

The least and the largest eigenvalues are can be obtained by using (2.15)

$$\lambda_{N-1} = \Big(\frac{-4}{h^2}\Big)sin^2\Big(\frac{(N-1)\pi}{2N}\Big) = \Big(\frac{-4}{h^2}\Big), \tag{2.17}$$

$$\lambda_1 = \Big(\frac{-4}{h^2}\Big)sin^2\Big(\frac{\pi}{2N}\Big) = -\pi^2. \tag{2.18}$$

Finally, the proportion of equations is obtained

$$\frac{\lambda_{N-1}}{\lambda_1} \approx \frac{4}{(\pi h)^2}, \tag{2.19}$$

it can be seen that it is a stiff system if h is small.

As a consequence, these two examples show that the stiffness of the differential equations can be identified by the Definition (2.1).

CHAPTER 3

NUMERICAL METHODS FOR STIFF DIFFERENTIAL EQUATIONS

In the previous chapter, we studied how to identify the stiffness of ordinary differential equations and partial differential equations. In the present chapter, we will answer the question 'What type of method can we use for solving stiff differential equations?'. In the first section, we define and construct traditional implicit methods. In what follows, we overview derivation of exponential integrators.

3.1. Traditional Implicit Methods

Traditional methods include explicit methods and implicit methods. To solve stiff problems numerically we prefer implicit methods. In this section, we will introduce several implicit methods. Our motivation the need for implicit methods is enlarging the stability region. This property overcome the stiffness.

3.1.1. Backward Differentiation Formula

Consider the ordinary differential equation as follows:

$$y'(t) = f(t, y). \tag{3.1}$$

In order to obtain a numerical solution of (3.1) by replacing the derivative on the left hand side of the equation. Let $r(t)$ be a polynomial of degree $\leq r$. This polynomial can be expressed in terms of interpolation of y(t) at the nodes $t_{n+1}, t_n, ..., t_{n-r+1}$ for $r \geq 1$

$$r(t) = \sum_{j=-1}^{r-1} y(t_{n-j}) l_{j,n}(t), \tag{3.2}$$

where $l_{j,n}(t)$ are the Langrange interpolation basis functions for the nodes $t_{n+1}, t_n, ..., t_{n-r+1}$. The interpolation polynomial can be written as

$$l_{j,n}(t) = \sum_{j=n+1}^{n-r+1} \frac{t - t_i}{t_j - t_i}, \qquad n+1 \leq i \leq n-r+1. \tag{3.3}$$

Then, differentiate r(t) at the point t_{n+1}

$$r'(t_{n+1}) \approx y'(t_{n+1}) = f(t_{n+1}, y(t_{n+1})).$$

Let $r(t)$ be a first degree polynomial and this polynomial interpolates $y(t)$ at the nodes t_{n+1}, t_n

$$\begin{aligned} r(t) &= y_{n+1}\frac{t - t_n}{t_{n+1} - t_n} + y_n\frac{t - t_{n+1}}{t_n - t_{n+1}}, && (3.4)\\ r'(t) &= y_{n+1}\frac{1}{h} - y_n\frac{1}{h}, \\ r'(t_{n+1}) &= f(t_{n+1}, y_{n+1}). && (3.5) \end{aligned}$$

As a result,

$$y_{n+1} = y_n + h f(t_{n+1}, y_{n+1}). \tag{3.6}$$

This method is called Backward Euler Method(BDF1).

In order to obtain an other formula, let choose $r(t)$ as a second degree polynomial. This polynomial interpolates y(t) at the nodes t_{n+1}, t_n, t_{n-1}. We construct the method using Lagrange interpolation basis functions:

$$\begin{aligned} r(t) &= y_{n+1}\frac{(t - t_n)(t - t_{n-1})}{(t_{n+1} - t_n)(t_{n+1} - t_{n-1})} + y_n\frac{(t - t_{n+1})(t - t_{n-1})}{(t_n - t_{n+1})(t_n - t_{n-1})} && (3.7)\\ &\quad + y_{n-1}\frac{(t - t_{n+1})(t - t_n)}{(t_{n-1} - t_{n+1})(t_{n-1} - t_n)}, \\ r'(t) &= y_{n+1}\frac{1}{2h^2}(t - t_n + t - t_{n-1}) - y_n\frac{1}{h^2}(t - t_{n+1} + t - t_{n-1}) && (3.8)\\ &\quad + y_{n-1}\frac{1}{2h^2}(t - t_{n+1} + t - t_n), \\ r'(t_{n+1}) &= \frac{3}{2h}y_{n+1} - \frac{2}{h}y_n + \frac{1}{2h}y_{n-1} = f(t_{n+1}, y_{n+1}), \end{aligned}$$

As a final result,

$$y_{n+1} = \frac{4}{3}y_n - \frac{1}{3}y_{n-1} + \frac{2}{3}hf(t_{n+1}, y_{n+1}). \tag{3.9}$$

This method is called BDF2 which is a multistep method.
As we see, backward differentiation formulas are implicit methods. For stiff problems, the importance of BDF methods lies in their super stability properties which allow them to take much larger step sizes than would be possible with explicit methods.

3.1.2. Implicit Runge Kutta Method

A simple example of an implicit 1-stage Runge-Kutta method can be obtain

$$\begin{aligned} y_{n+1} &= y_n + hb_1 f(t_n + c_1 h, k_1), \\ k_1 &= y_n + ha_{11} f(t_n + c_1 h, k_1). \end{aligned} \tag{3.10}$$

To determine the coefficients a_{11}, b_1 and c_1, we substitude the exact solution into (3.10) and use Taylor expansion

$$y(t_{n+1}) = y(t_n) + hb_1[f + c_1 h f_t + ha_{11} f f_y] + O(h^3), \tag{3.11}$$

when we compare the terms of (3.11) with the Taylor series of $y(t_{n+1})$

$$y(t_{n+1}) = y(t_n) + hf + \frac{h^2}{2}(f_t + f f_y) + O(h^3), \tag{3.12}$$

where $f = f(t_n, y(t_n))$. We will assume the following conditions

$$a_{11} = \frac{1}{2}, \;\; c_1 = \frac{1}{2}, \;\; b_1 = 1, \tag{3.13}$$

these coefficients yields

$$y_{n+1} = y_n + hf(t_n + h/2, k_1), \tag{3.14}$$

$$k_1 = y_n + \frac{h}{2} f(t_n + h/2, k_1). \tag{3.15}$$

This method is called implicit midpoint method. The detailed information can be found in (Hairer & Norsett & Wanner, 1993).

3.1.3. Trapezoid Method

To achieve the formula of the trapezoidal method, we will begin with the trapezoidal rule for numerical integration

$$\int_a^b g(\tau)d\tau \approx \frac{1}{2}(b-a)[g(a) + g(b)] - \frac{1}{12}h^3 g'''(\xi), \tag{3.16}$$

for $a < \xi < b$. Consider the equation

$$y'(t) = f(t, y(t)), \tag{3.17}$$

where $y \in \mathbb{R}$ and $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. We integrate the equation (3.17) from t_n to t_{n+1}

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(\tau, y(\tau))d\tau. \tag{3.18}$$

Using the trapezoidal rule (3.16) to approximate the integral, we get

$$y(t_{n+1}) = y(t_n) + \frac{1}{2}h[f(t_n, y(t_n)) + f(t_{n+1}, y(t_{n+1}))] - \frac{1}{12}h^3 y'''(\xi), \tag{3.19}$$

for $t_n < \xi < t_n + h$. By dropping the error term, an approximate solution y_n at time-step t_n for (3.17) the implicit trapezoidal method approximates the solution at time-step $t_{n+1} = t_n + h$ by

solving the following nonlinear equation

$$y_{n+1} = y_n + \frac{h}{2}[f(t_n, y_n) + f(t_{n+1}, y_{n+1})]. \tag{3.20}$$

This equation can be solved by both iterative solution method and Newton's method with an initial guess provided by the forward Euler's method.
The combination of forward Euler method and backward Euler method generates trapezoidal method. Although both forward Euler method and backward Euler method are first order methods, this combination gives us a second order method.

3.2. Exponential Integrators

Exponential Integrators are a class of numerical integrators for time integration of differential equations. They are most commonly used for stiff problems and oscillatory problems. In this thesis, we will just deal with stiff differential equations. Exponential integrators are purposed to be used on ODEs that can be split into a stiff linear part and a non-stiff nonlinear part. We already know that if an equation is stiff then the traditional explicit methods don't work well. Although exponential integrators are explicit methods, they are considered as generalizations of the implicit methods. For this reason, exponential integrators are suitable methods to solve stiff problems.
Our motivation the need for using exponential integrators is that they overcome the problem of stiffness taking large time steps.
To get an idea of what an Exponential Integrator looks like, we will begin with an evolution equation of the form

$$\begin{aligned} u'(t) &= F(t, u(t)), \\ u(0) &= u_0. \end{aligned} \tag{3.21}$$

The linearization of this equation at time t gives us following semilinear problem

$$\begin{aligned} u'(t) + Au(t) &= g(t, u(t)), \\ u(0) &= u_0, \end{aligned} \tag{3.22}$$

where A is a linear operator that represents the highest order of differential terms and g is a nonlinear operator. The linear part of (3.22)

$$u'(t) + Au(t) = 0, \quad u(0) = u_0, \tag{3.23}$$

can be solved exactly as

$$u(t) = e^{-tA}u_0. \tag{3.24}$$

To get the representation of the exact solution the problem (3.22) we will begin with multiplying Equation (3.22) through by integrating factor e^{tA}. Then, we obtain

$$\begin{aligned} e^{tA}u' + e^{tA}Au &= e^{tA}g(t, u(t)), \\ (e^{tA}u)' &= e^{tA}g(t, u(t)). \end{aligned}$$

Integrating both sides of this equation we get

$$\begin{aligned} \int_{t_n}^{t_n+h} \frac{d}{d\tau}(e^{\tau A}u(\tau))d\tau &= \int_{t_n}^{t_n+h} e^{\tau A}g(\tau, u(\tau))d\tau, \\ e^{(t_n+h)A}u(t_n+h) - e^{t_nA}u(t_n) &= \int_{t_n}^{t_n+h} e^{\tau A}g(\tau, u(\tau))d\tau, \\ u(t_n+h) &= e^{-(t_n+h)A}\left(e^{t_nA}u(t_n) + \int_{t_n}^{t_n+h} e^{\tau A}g(\tau, u(\tau))d\tau\right). \end{aligned}$$

As a result,

$$u(t_n+h) = e^{-hA}u(t_n) + \int_{t_n}^{t_n+h} e^{-(t_n+h-\tau)A}g(\tau, u(\tau))d\tau. \tag{3.25}$$

It is the required solution. Let $t_{n+1} = t_n + h$ and $\tau = t_n$ where h is the step size. Then change of the variables,

$$\begin{aligned} \tau - t_n = 0 &= v, \\ d\tau &= dv. \end{aligned}$$

So, the boundaries of integral change and we obtain

$$u(t_{n+1}) = e^{-hA}u(t_n) + \int_0^h e^{-(h-v)A}g(t_n + v, u(t_n + v))dv. \tag{3.26}$$

Finally, the exact solution of (3.22) can be written recursively as

$$u(t_{n+1}) = e^{-hA}u(t_n) + \int_0^h e^{-(h-\tau)A}g(t_n + \tau, u(t_n + \tau))d\tau, \tag{3.27}$$

this representation is called variation of constants formula. By approximating the integral with various quadrature formulas, different numerical schemes can be obtained. Our convergence proofs will be generally based on a discrete version of the variation of constants formula in following chapters.

As their name suggests, exponential integrators use the matrix exponential in the numerical integrator. We will call the matrix exponential as ϕ functions. In the following section we will define these functions.

3.2.1. Derivation of the ϕ Functions

The entire functions ϕ_k, for $k \geq 1$ can be defined by the integral representation as

$$\phi_0(x) = e^x, \tag{3.28}$$

$$\phi_k(x) = \int_0^1 e^{(1-\theta)x}\frac{\theta^{k-1}}{(k-1)!}d\theta. \tag{3.29}$$

The argument x can be a scalar or a matrix. The ϕ functions are defined recursively by

$$\phi_0(x) = e^x, \tag{3.30}$$

$$\phi_{k+1}(x) = \frac{\phi_k(x) - \frac{1}{k!}}{x}, \quad k \geq 0. \tag{3.31}$$

In this case, ϕ_0 is the matrix exponential. The first few ϕ functions are

$$\begin{aligned}
\phi_0(x) = e^x &= 1 + x + \frac{1}{2}x^2 + \frac{1}{3!}x^3 + \ldots, && (3.32)\\
\phi_1(x) = \frac{e^x - 1}{x} &= 1 + \frac{1}{2}x + \frac{1}{3!}x^2 + \frac{1}{4!}x^3 + \ldots, && (3.33)\\
\phi_2(x) = \frac{e^x - 1 - x}{x^2} &= \frac{1}{2} + \frac{1}{3!}x + \frac{1}{4!}x^2 + \frac{1}{5!}x^3 + \ldots, && (3.34)\\
\phi_3(x) = \frac{e^x - 1 - x - \frac{1}{2}x^2}{x^3} &= \frac{1}{3!} + \frac{1}{4!}x + \frac{1}{5!}x^2 + \frac{1}{6!}x^3 + \ldots. && (3.35)
\end{aligned}$$

The Taylor series representation of these functions is given by (Schmelzer& Trefethen, 2007) as

$$\phi_k(x) = \sum_{l=k}^{\infty} \frac{1}{l!} z^{l-k}. \tag{3.36}$$

3.2.1.1. Derivation of the Methods with ϕ Functions

Consider the linear problem

$$\begin{aligned}
u'(t) &= Au(t) + g(t), && (3.37)\\
u(0) &= u_0.
\end{aligned}$$

Every stage in an exponential integrator can be expressed as a linear combination of ϕ function. Firstly, we assume $g(t)$ as a constant

$$\dot{u} = Au + a_1, \quad u(0) = u_0, \tag{3.38}$$

the solution of this equation is given by

$$\begin{aligned}
u(t) &= e^{tA}u_0 + \int_0^t e^{(t-\tau)A} a d\tau = e^{tA}u_0 + e^{tA}\frac{1}{-A}e^{-\tau A}a_1\Big|_0^t, && (3.39)\\
&= e^{tA}u_0 + e^{tA}\frac{e^{-tA} - 1}{-A}a_1 = e^{tA}u_0 + t\frac{e^{tA} - 1}{tA}a_1. && (3.40)
\end{aligned}$$

The solution in terms of the ϕ_k for $k = 0, 1, ...$ is given by

$$u(t) = e^{tA}u_0 + t\phi_1(tA)a_1. \tag{3.41}$$

Secondly, take $g(t)$ as a polynomial of first degree then the equation

$$\dot{u} = Au + a_1 + a_2 t, \quad u(0) = u_0, \tag{3.42}$$

using the variation of constants formula

$$\begin{aligned} u(t) &= e^{tA}u_0 + \int_0^t e^{(t-\tau)A}(a_1 + a_2\tau)d\tau, & (3.43)\\ &= e^{tA}u_0 + t\frac{e^{tA}-1}{tA}a_1 + t^2\frac{e^{tA}-1-tA}{t^2A^2}a_2. & (3.44) \end{aligned}$$

So, given equation has a solution of the form of ϕ functions

$$u(t) = e^{tA}u_0 + t\phi_1(tA)a_1 + t^2\phi_2(tA)a_2. \tag{3.45}$$

In general, when we take $g(t)$ as a polynomial equation

$$\dot{u} = Au + a_1 + a_2 t + a_3\frac{t^2}{2!} + ... + a_n\frac{t^{n-1}}{(n-1)!}, \quad u(0) = u_0, \tag{3.46}$$

where $u(t), a_1, a_2, a_3, ..., a_n$ are vectors and A is a matrix. The solution of the equation admits the following form

$$\begin{aligned} u(t) &= e^{tA}u_0 + \int_0^t e^{(t-\tau)A}\left(a_1 + a_2 t + a_3\frac{t^2}{2!} + ... + a_n\frac{t^{n-1}}{(n-1)!}\right)d\tau, & (3.47)\\ &= e^{tA}u_0 + t\frac{e^{tA}-1}{tA}a_1 + t^2\frac{e^{tA}-1-tA}{t^2A^2}a_2 + t^3\frac{e^{tA}-1-tA-\frac{1}{2}tA^2}{t^3A^3}a_3 \\ &\quad +... + \int_0^t e^{(t-\tau)A}a_n\frac{t^{n-1}}{(n-1)!}d\tau, & (3.48)\\ &= e^{tA}u_0 + t\phi_1(tA)a_1 + t^2\phi_2(tA)a_2 + t^3\phi_3(tA)a_3 + ... + t^n\phi_n(tA)a_n. & (3.49) \end{aligned}$$

Finally, this identity leads to the following general solution

$$u(t) = e^{tA}u_0 + \sum_{i=1}^{n} t^i \phi_i(tA)a_i. \tag{3.50}$$

3.2.2. Exponential Euler Method

In this section, we will derive the exponential Euler method. Consider the evolution equation

$$\begin{aligned} u'(t) + Au(t) &= g(t, u(t)), \\ u(t_0) &= u_0. \end{aligned} \tag{3.51}$$

From the variation of constants formula the exact solution of (3.51) can be written as

$$u(t_{n+1}) = e^{-hA}u(t_n) + \int_0^h e^{-(h-\tau)A} g(t_n + \tau, u(t_n + \tau))d\tau. \tag{3.52}$$

The main idea behind constructing a method is to approximate the function g in the equation (3.52). We will shortly denote $g(t_n, u(t_n)) = g(t_n)$. Taylor expansion of $g(t_n + \tau)$ at t_n leads to

$$g(t_n + \tau) = g(t_n) + O(\tau). \tag{3.53}$$

This expansion motivates the following equation

$$\begin{aligned} u(t_{n+1}) &= e^{-hA}u(t_n) + \int_0^h e^{-(h-\tau)A} g(t_n + \tau)d\tau, && (3.54) \\ &= e^{-hA}u(t_n) + \int_0^h e^{-(h-\tau)A} g(t_n)d\tau + O(h^2), && (3.55) \\ &= e^{-hA}u(t_n) + h\frac{e^{-hA} - I}{-hA} g(t_n) + O(h^2), && (3.56) \\ &= e^{-hA}u(t_n) + h\phi_1(-hA)g(t_n) + O(h^2). && (3.57) \end{aligned}$$

The numerical solution of (3.51) takes the form

$$u_{n+1} = e^{-hA}u_n + h\phi_1(-hA)g(t_n), \tag{3.58}$$

where ϕ_1 is given in Equation (3.32). At time $t_{n+1} = (n+1)h$, $n \in 0, 1, ...$ the exact solution of (3.51) is approximated by exponential Euler method where h is the time step.

3.2.3. Second Order Method

In order to show that the previous ideas are not limited to the exponential Euler method, a second- order exponential scheme can be constructed in a similar way. To obtain the second order method we will use Taylor expansion of $g(t_n + \tau)$ at t_n

$$g(t_n + \tau) = g(t_n) + \tau g'(t_n) + O(\tau^2). \tag{3.59}$$

Substituting $g'(t_n) = \frac{g(t_{n+1}) - g(t_n)}{h}$ in this expansion leads to

$$g(t_n + \tau) = g(t_n) + \frac{\tau}{h}\Big(g(t_{n+1}) - g(t_n)\Big) + O(\tau^2). \tag{3.60}$$

Inserting (3.60) into (3.52), we have

$$\begin{aligned} u_{n+1} &= e^{-hA}u_n + \int_0^h e^{-(h-\tau)A}\Big(g(t_n) + \frac{\tau}{h}(g(t_{n+1}) - g(t_n))\Big)d\tau, \\ &= e^{hA}u_n + \int_0^h e^{-(h-\tau)A}g(t_n)d\tau + \frac{1}{h}\int_0^h e^{-(h-\tau)A}\tau g(t_{n+1})d\tau - \frac{1}{h}\int_0^h e^{-(h-\tau)A}\tau g(t_n)d\tau. \end{aligned}$$

Applying integration by parts to the integral, we get the following expression

$$\begin{aligned} u_{n+1} &= e^{-hA}u_n + h\phi_1(-hA)g(t_n) + h\phi_2(-hA)g(t_{n+1}) - h\phi_2(-hA)g(t_n), \\ &= e^{-hA}u_n + h(\phi_1(-hA) - \phi_2(-hA))g(t_n) + h\phi_2(-hA)g(t_{n+1}). \end{aligned}$$

Replacing the function g yields the following approximation to the exact solution at time t_{n+1},

$$u_{n+1} = e^{-hA}u_n + h(\phi_1(-hA) - \phi_2(-hA))g(t_n) + h\phi_2(-hA)g(t_{n+1}). \tag{3.61}$$

This exponential method is called second order method.

3.2.4. Exponential Midpoint Rule

We present here a construction for the numerical solution of (3.51) based on the exact solution given by variation of constants formula

$$u(t_{n+1}) = e^{-hA}u(t_n) + \int_0^h e^{-(h-\tau)A}g(t_n + \tau)d\tau.$$

Midpoint rule is obtained by rewriting $g(t_n + \tau)$ in exact solution as

$$g(t_n + \tau) = g((t_n + h/2) + (\tau - h/2)) = g(t_n + h/2) + (\tau - h/2)g'(t_n + h/2). \tag{3.62}$$

Taking into account (3.62), the scheme is calculated by

$$\begin{aligned} u_{n+1} &= e^{-hA}u_n + \int_0^h e^{-(h-\tau)A}g(t_n + h/2)d\tau, \\ &= e^{-hA}u_n + h\phi_1(-hA)g(t_n + h/2), \\ &= e^{-hA}u_n + h\phi_1(-hA)g(t_n) + \frac{h^2}{2}\phi_1(-hA)g'(t_n) + O(h^3). \end{aligned}$$

After all, the numerical method is called exponential midpoint rule

$$u_{n+1} = e^{-hA}u_n + h\phi_1(-hA)g(t_n) + \frac{h^2}{2}\phi_1(-hA)g'(t_n), \tag{3.63}$$

where u_{n+1} denote the numerical approximation to the solution at time t_{n+1}.

3.2.5. Exponential Rosenbrock Method

In this section, we are concerned with the exponential Rosenbrock method for time discretization of autonomous evolution equations of the following form

$$\begin{aligned} u'(t) &= F(u(t)) = Au(t) + g(u(t)), \\ u(t_0) &= u_0. \end{aligned} \tag{3.64}$$

From the variation of constants formula, the exact solution of (3.64) can be written as

$$u(t_{n+1}) = e^{hA}u(t_n) + \int_0^h e^{(h-\tau)A} g(u(t_n + \tau))d\tau. \tag{3.65}$$

The method depends on a linearization of (3.64) at each step

$$u'(t) = J_n u(t) + g_n(u(t)), \quad t_n \le t \le t_{n+1}, \tag{3.66}$$

where

$$J_n = \frac{\partial F}{\partial u}(u_n), \quad g_n(u(t)) = F(u(t)) - J_n u(t). \tag{3.67}$$

Applying the exponential Euler method to linearized problem (3.66)

$$\begin{aligned} u_{n+1} &= e^{hJ_n}u_n + h\phi(hJ_n)g_n(u_n), \\ &= e^{hJ_n}u_n + h\phi(hJ_n)[F(u_n) - J_n u_n]. \end{aligned}$$

After regulating the equation, we obtain

$$u_{n+1} = u_n + h\phi(hJ_n)F(u_n). \tag{3.68}$$

This numerical scheme is called exponential Rosenbrock-Euler method. Exponential Rosenbrock-Euler method is explicit time stepping scheme. Its implementation is standard, apart from

the calculation of the exponential and related functions. This method is computationally attractive since it involves just one matrix function in each step. To implement exponential Rosenbrock-Euler method it is important to approximate the application of matrix functions to vectors efficiently. More information can be found in (Caliari & Ostermann, 2009).

3.3. Error Analysis via Richardson Extrapolation

Richardson extrapolation holds on an interpretation about the form of the error terms in a numerical approximation. Our estimation depends on the step size h. Suppose that $R(h)$ is a numerical approximation to an exact result $R(0)$. If the exact result is achieved as $h \to 0$, then our estimate is consistent (Burg& Erwin , 2008).
$R(h)$ can be expanded as

$$R(h) = R(0) + Kh^p + K'h^{p+1} + K''h^{p+2} + ... \tag{3.69}$$

We already know that the notation $O(h^{p+1})$ is used for a sum of terms of order h^{p+1} and higher. Therefore, the numerical approximation can be written as

$$R(h) = R(0) + Kh^p + O(h^{p+1}). \tag{3.70}$$

We will use the expansion for $R(h)$ and $R(rh)$

$$\begin{aligned} R(h) &= R(0) + Kh^p + O(h^{p+1}), \\ R(rh) &= R(0) + Kr^p h^p + O(h^{p+1}). \end{aligned}$$

By multiplying $R(h)$ by r^p and subtracting $R(rh)$, we obtain

$$r^p R(h) - R(rh) = (r^p - 1)R(0) + O(h^{p+1}). \tag{3.71}$$

Therefore, dividing this equation by $r^p - 1$ we get desired form:

$$\frac{r^p R(h) - R(rh)}{r^p - 1} = R(0) + O(h^{p+1}). \tag{3.72}$$

Suppose that $r = 2$ and $p = 1$

$$2R(h) - R(2h) = R(0) + O(h^2). \tag{3.73}$$

We generated an approximation whose error is order 2. As a result, we will use Richardson extrapolation when the exact solution is not known.

CHAPTER 4

ERROR ANALYSIS

In this chapter, we will derive error bounds of exponential integrators for parabolic problem. Throughout present chapter, we will consider problems with smooth solutions. Therefore, we can always expand the solution in a Taylor series.

4.1. Analytical Framework

We will use the same analytical framework given in (Hochbruck & Ostermann, 2010). The detail discussion can be also found in the given work. In order to clarify our analysis, we will give basic definitions of the semigroup theory. Consider the following semilinear equation

$$\begin{aligned} u'(t) + Au(t) &= g(t, u(t)), \\ u(t_0) &= u_0, \end{aligned} \tag{4.1}$$

where A is time invariant operator and u is a vector function of time. The solution of (4.1) can be written as

$$u(t_{n+1}) = e^{-tA}u(t_n) + \int_0^h e^{-(h-\tau)A} g(t_n + \tau, u(t_n + \tau))d\tau, \quad 0 \le t \le T. \tag{4.2}$$

If A is a linear bounded operator in Banach space, $u(t)$ still has this form. However, in many interesting cases, it is unbounded which don't admit this form. This is some extend shows the richness of semigroup theory. For its application, semigroup theory uses abstract methods of operator to treat initial boundary value problem for linear and nonlinear equations that describe the evolution of a system. We will ground our analysis on an abstract formulation of strongly continuous semigroups on a Banach space X with norm $\| \cdot \|$ where $(\mathcal{D}(A), A)$ is linear unbounded operators in X and $g : [0, T] \times X \to X$. In our proofs we will shortly denote $g(t_n, u(t_n)) \approx g(t_n)$.

Definition 4.1 *An strongly continuous semigroup (C_0-semigroup) on a real or complex Banach space X, $T(t)_{t\geq 0}$, satisfying*

$$T : X \mapsto X$$

i) *T(0)=I,*

ii) $T(t+s) = T(t)T(s), \quad for\ all \quad t, s \geq 0,$

iii) $\forall x \in X$ *such that* $||T(t)x - x|| \to 0$ as $t \to 0$.

We usually write $T(t) = e^{-At}$.

Definition 4.2 *An analytic semigroup on a Banach space X, $T(t)_{t\geq 0}$, satisfying*

i) *T(0)=I,*

ii) $T(t+s) = T(t)T(s), \quad for\ all \quad t, s \geq 0,$

iii) $\forall x \in X$ *such that* $||T(t)x - x|| \to 0$ as $t \to 0^+$,

iv) $t \to T(t)$ *is real analytic on* $0 < t < \infty$ *(Pazy, 1983).*

Definition 4.3 *The infinitesimal generator A of given semigroup $T(t)$ is defined by*

$$Ax := \lim_{t\to 0^+} \frac{1}{t}(T(t)x - x),$$

the limit exists in the domain $\mathcal{D}(A)$, which consists of all $x \in X$.

Proposition 4.1 *Let $A : \mathcal{D}(A) \to X$ be sectorial, A is a densely defined and closed linear operator on X satisfying the resolvent condition*

$$|| (\lambda I - A)^{-1} || \leq \frac{C}{|\lambda - m|}, \tag{4.3}$$

on the sector $\lambda \in \mathbb{C}$, $v \leq |arg(\lambda - m)| \leq \pi$, $\lambda \neq m$ for $C \geq 1$,$m \in \mathbb{R}$ and $0 < v < \pi/2$. (Hochbruck & Ostermann, 2005)

Theorem 4.1 *If A is sectorial operator , then $-A$ is the infinitesimal generator of analytic semigroup $e^{-tA}{}_{t\geq 0}$.*
Converse is also true, if $-A$ generates an analytic semigroup, then A is sectorial.

Proof The proof can be read from (Henry, 1981). □

Theorem 4.2 *Let $T(t)$ be a semigroup. There exist constants $\omega \in \mathbb{R}$ and $C \geq 1$ such that*

$$\| T(t) \| \leq Ce^{\omega t} \quad for \quad t \geq 0. \tag{4.4}$$

Proof Let choose a constant $C \geq 1$ such that $\| T(t) \| \leq C$ for all $0 \leq t \leq 1$. Suppose that $\omega = logC$. Then, for each $t > 0$ and $n \leq t$, where n is integer

$$\| T(t) \| = \| T\Big(\sum_{k=1}^{n} \frac{t}{n} \Big) \| .$$

Using the semigroup property $T(t + s) = T(t) + T(s)$ we get

$$\| \prod_{k=1}^{n} \Big(T(\frac{t}{n}) \Big) \| = \| \Big(T(\frac{t}{n}) \Big)^n \| \leq C^n \leq C^{t+1} = Ce^{\omega t}.$$

The proof is completed. (Sheree, 2003) □

Theorem 4.3 *(Hille-Yosida Theorem) A linear unbounded operator A is the generator of a C_0 semigroup if and only if*

i) *A is a closed operator,*

ii) *A has a dense domain $\mathcal{D}(A)$,*

iii) *$\lambda \in \rho(A)$ for each $\lambda > 0$,where $\rho(A)$ is the resolvent set of A,*
$\rho(A) = \{\lambda \in \mathbb{C} | \lambda I - A$ *is invertable*$\}$,

iv) $\| (\lambda I - A)^{-1} \| \leq \frac{1}{\lambda}$.

Proof The proof can be read from (Sheree, 2003). □

Assumption 4.1 *Let X be a Banach space with norm $\| \cdot \|$. We assume that A is a linear operator on X and that (-A) is the infinitesimal generator of a strongly continuous semigroup (C_0 semigroup) e^{-tA} on X.*

This assumption refers that there exist constants $\omega \in \mathbb{R}$ and $C \geq 1$ such that

$$\| e^{-tA} \| \leq Ce^{\omega t} \quad for \quad t \geq 0. \tag{4.5}$$

Assumption 4.2 *Let X be a Banach space with norm* $\| \ . \ \|$. *We assume that A is a linear operator on X and that (-A) is the infinitesimal generator of an analytic semigroup* e^{-tA} *on X.*

Remark 4.1 *The generators of analytic semigroups let us to define fractional powers of the operator.*

The assumption (4.2) means that there exist constants $C = C(\gamma)$

$$\| e^{-tA} \| + \| t^{\gamma} A^{\gamma} e^{-tA} \| \leq C \quad for \quad \gamma, t \geq 0. \tag{4.6}$$

Our proofs will be based on the substraction of the numerical solution from the exact solution. So, our error estimation will include bounds of terms of the following form

$$e^{-t_n A}(g(t_n, u(t_n)) - g(t_n, u_n)). \tag{4.7}$$

We can bound the difference $(g(t_n, u(t_n)) - g(t_n, u_n))$. Firstly, multiply (4.7) with $A^{\alpha} A^{-\alpha}$

$$\| e^{-t_n A} A^{\alpha} A^{-\alpha} (g(t_n, u(t_n)) - g(t_n, u_n)) \| = \| e^{-t_n A} A^{\alpha} \| \| A^{-\alpha} (g(t_n, u(t_n)) - g(t_n, u_n)) \| . \tag{4.8}$$

Then, multiply (4.8) with ${t_n}^{\alpha} {t_n}^{-\alpha}$

$${t_n}^{-\alpha} \| e^{-t_n A} {t_n}^{\alpha} A^{\alpha} \| \| A^{-\alpha} (g(t_n, u(t_n)) - g(t_n, u_n)) \| \leq C {t_n}^{-\alpha} \| A^{-\alpha} (g(t_n, u(t_n)) - g(t_n, u_n)) \| . \tag{4.9}$$

Assumption 4.3 *Let* $V = \{v \in X | A^{\alpha} v \in X\}$ *be a Banach space with norm* $\| v \|_V = \| A^{\alpha} v \|$ *for* $0 \leq \alpha \leq 1$. *We assume that* $g : [0, T] \times V \to X$ *is locally Lipshitz continuous in a strip along the exact solution u. There exist a real number* $L = L(R, T)$ *such that, for all* $t \in [0, T]$,

$$\| g(t, v) - g(t, w) \| \leq L \| v - w \|_V, \tag{4.10}$$

where $max(\| v - u(t) \|_V, \| w - u(t) \|_V) \leq R$.

Lemma 4.1 *Let the initial value problem*

$$u'(t) + Au(t) = g(t, u(t)), \ u(t_0) = u_0, \tag{4.11}$$

A is an $n \times n$ matrix. The exact solution of (4.11) can be represented by

$$u(t_n + \tau) = e^{-\tau A}u(t_n) + \sum_{i=0}^{m-1} \tau^{i+1}\phi_{i+1}(-\tau A)g^{(i)}(t_n) + \delta_n(m, \tau), \tag{4.12}$$

$$\delta_n(m, \tau) = \int_0^\tau e^{-(\tau-\sigma)A} \int_0^\sigma \frac{(\sigma - \eta)^{m-1}}{(m-1)!} g^{(m)}(t_n + \eta)d\eta d\sigma, \tag{4.13}$$

is provided when g is a sufficiently smooth function.

Proof The Taylor series expansion of $g(t_n + \sigma)$ at t_n leads to

$$g(t_n + \sigma) = \sum_{i=0}^{m-1} \frac{\tau^i}{i!} g^{(i)}(t_n) + \gamma_n(m, \sigma), \tag{4.14}$$

$$\gamma_n(m, \sigma) = \int_0^\sigma \frac{(\sigma - \eta)^{m-1}}{(m-1)!} g^{(m)}(t_n + \eta)d\eta, \tag{4.15}$$

and the entire functions

$$\begin{aligned} \phi_0(z) &= e^z, \\ \phi_i(z) &= \int_0^1 e^{(1-\tau)z} \frac{\tau^{i-1}}{(i-1)!} d\tau, \ \ i \geq 1. \end{aligned} \tag{4.16}$$

Substituting the expansion (4.14) and ϕ function definition (4.16) into the following variation of constants formula

$$u(t_n + \tau) = e^{-\tau A}u(t_n) + \int_0^\tau e^{-(\tau-\sigma)A} g(t_n + \sigma)d\sigma, \tag{4.17}$$

we obtain the desired result. □

In the following section, we will give convergence of exponential Euler method for semilinear parabolic problems.

The convergence proofs in this section will be based on (Hochbruck & Ostermann, 2010).

4.2. Convergence of Exponential Euler Method

Let the initial value problem

$$\begin{aligned} u'(t) + Au(t) &= g(t, u(t)), \\ u(0) &= u_0. \end{aligned} \tag{4.18}$$

Our proof will be based on the representation of exact solution by the variation of constants formula

$$u(t_{n+1}) = e^{hA}u(t_n) + \int_0^h e^{(h-\tau)A} g(t_n + \tau) d\tau. \tag{4.19}$$

We expand $g(t_n + \tau)$ in a Taylor series with integral form of the reminder

$$g(t_n + \tau) = g(t_n) + \int_0^\tau g'(t_n + \sigma) d\sigma. \tag{4.20}$$

Then, we substitude (4.20) in the (4.19)

$$u(t_{n+1}) = e^{-hA}u(t_n) + h\phi_1(-hA)g(t_n) + \delta_{n+1}, \tag{4.21}$$

$$\delta_{n+1} = \int_0^h e^{-(h-\tau)A} \int_0^\tau g'(t_n + \sigma) d\sigma d\tau. \tag{4.22}$$

For the reminder δ_{n+1} we have following estimation.

Lemma 4.2 *Let the initial value problem*

$$u'(t) + Au(t) = g(t, u(t)), \;\; u(0) = u_0,$$

satisfy Assumption (4.2). Moreover, let $0 < \beta \leq 1$ *and* $A^{\beta-1}g' \in L^\infty(0, T)$. *Then,*

$$\| \sum_{j=0}^{n-1} e^{-jhA} \delta_{n-j} \|_V \leq Chsup_{0 \leq t \leq t_n} \| A^{\beta-1} g'(t, u(t)) \|_V, \tag{4.23}$$

holds with a constant C, uniformly in $0 \leq t_n \leq T$.

Proof Firstly, we represent the supremum by M, where M is

$$M := \sup_{0 \leq t \leq t_n} \| A^{\beta-1} g'(t, u(t)) \|_V \ . \tag{4.24}$$

Then, we will construct the expression on the left hand side of the expression (4.23). We can write the defect δ_{n-j} as

$$\delta_{n-j} = \int_0^h e^{-(h-\tau)A} \int_0^\tau g'(t_{n-j-1} + \sigma) d\sigma d\tau. \tag{4.25}$$

multiply the equation by $e^{-jhA} A^{1-\beta} A^{\beta-1}$

$$e^{-jhA} \delta_{n-j} = e^{-jhA} \int_0^h e^{-(h-\tau)A} A^{1-\beta} \int_0^\tau A^{\beta-1} g'(t_{n-j-1} + \sigma) d\sigma d\tau. \tag{4.26}$$

For $j = 0$, the defect is

$$\delta_n = \int_0^h e^{-(h-\tau)A} A^{1-\beta} \int_0^\tau A^{\beta-1} g'(t_{n-1} + \sigma) d\sigma d\tau.$$

To use the stability bound (4.6) multiply the equation by $(h-\tau)^{1-\beta} (h-\tau)^{\beta-1}$

$$\delta_n = \int_0^h (h-\tau)^{1-\beta} (h-\tau)^{\beta-1} e^{-(h-\tau)A} A^{1-\beta} \int_0^\tau A^{\beta-1} g'(t_{n-1} + \sigma) d\sigma d\tau.$$

We can estimate this in V by

$$\| \delta_n \|_V \leq C \int_0^h (h-\tau)^{\beta-1} \int_0^\tau A^{\beta-1} g'(t_{n-1} + \sigma) d\sigma d\tau.$$

From our representation (4.24), we have

$$\begin{aligned} \| \delta_n \|_V &\leq CM \int_0^h (h-\tau)^{\beta-1} \int_0^\tau d\sigma d\tau, \\ &= CM \int_0^h \tau (h-\tau)^{\beta-1} d\tau. \end{aligned}$$

We evaluate the integral using integration by parts and obtain

$$\| \delta_n \|_V \leq CMh^{\beta+1}.$$

The remaining terms for $j = 1, ..., n-1$ are defined as

$$e^{-jhA}\delta_{n-j} = \int_0^h e^{-(h-\tau)A} e^{-jhA} A^{1-\beta} (jh)^{1-\beta} (jh)^{\beta-1} \int_0^\tau A^{\beta-1} g'(t_{n-j-1}+\sigma) d\sigma d\tau.$$

This identity can be bounded in V by

$$\| e^{-jhA}\delta_{n-j} \|_V \leq CM \int_0^h e^{-(h-\tau)A} (jh)^{\beta-1} \tau d\tau,$$

by means of integration by parts we get

$$\begin{aligned} \| e^{-jhA}\delta_{n-j} \|_V &\leq CM(jh)^{\beta-1} \phi_2(-hA) h^2, \\ &\leq CM(jh)^{\beta-1} h^2. \end{aligned}$$

Hence, the remaining sum verifies that

$$\begin{aligned} \| \sum_{j=1}^{n-1} e^{-jhA}\delta_{n-j} \|_V &\leq CM \sum_{j=1}^{n-1} h^2 (jh)^{\beta-1}, \\ &\leq CMh \int_0^{t_{n-1}} t^{\beta-1} dt, \\ &\leq CMh. \end{aligned}$$

Finally, we get the desired estimation

$$\| \sum_{j=0}^{n-1} e^{-jhA}\delta_{n-j} \|_V \leq Ch \sup_{0\leq t\leq t_n} \| A^{\beta-1}g'(t,u(t)) \|_V .$$

□

Theorem 4.4 *Consider the initial value problem*

$$u'(t) + Au(t) = g(t,u(t)), \quad u(t_0) = u_0,$$

for its numerical solution the exponential Euler method

$$u_{n+1} = e^{-hA}u_n + h\phi_1(-hA)g(t_n,u_n).$$

Let given IVP satisfy Assumption (4.2) and Assumption (4.3). Furthermore, $g : [0,T]\times X \to X$ *is differentiable, and* $A^{\beta-1}g' \in L^\infty(0,T)$*, where* $\beta \in (0,1]$*. Then the error bound can be written as*

$$\| u_n - u(t_n) \|_V \leq Ch \sup_{0\leq t\leq t_n} \| A^{\beta-1}g'(t,u(t)) \|_V, \tag{4.27}$$

holds uniformly in $0 \leq nh \leq T$*. The constant C is independent of n and h.*

Proof The proof will be based on the difference of numerical and exact solution. Our numerical method is exponential Euler method

$$u_{n+1} = e^{-hA}u_n + h\phi_1(-hA)g(t_n,u_n). \tag{4.28}$$

Representation of exact solution will be given by variation of constants formula. The solution can be written as

$$u(t_{n+1}) = e^{-hA}u(t_n) + h\phi_1(-hA)g(t_n,u(t_n)) + \delta_{n+1}, \tag{4.29}$$

we define the reminder

$$\delta_{n+1} = \int_0^h e^{-(h-\tau)A} \int_0^\tau g'(t_n + \sigma, u(t_n + \sigma))d\sigma d\tau.$$

Substract (4.29) from (4.28) and denote $e_n = u_n - u(t_n)$, we have

$$e_{n+1} = e^{-hA}e_n + h\phi_1(-hA)(g(t_n, u_n) - g(t_n, u(t_n))) - \delta_{n+1}. \tag{4.30}$$

Solving this recursion leads to

$$e_n = \sum_{j=0}^{n-1} e^{-(n-j-1)hA}h\phi_1(-hA)(g(t_j, u_j) - g(t_j, u(t_j))) - \sum_{j=0}^{n-1} e^{-jhA}\delta_{n-j}. \tag{4.31}$$

We now proceed to verify (4.31) by showing inductively that the formula holds true for each n.

(Base Step) : Let $G_j = g(t_j, u_j) - g(t_j, u(t_j))$. When $n = 1$

$$e_1 = h\phi_1(-hA)G_0 - \delta_0,$$

so it is true for $n = 1$.

(Assumption step) : Let $k \in \mathbb{Z}^+$ is given and suppose given summation is true for $n = k - 1$. This yields

$$e_{k-1} = \sum_{j=0}^{k-2} e^{-(k-j-2)hA}h\phi_1(-hA)G_j - \sum_{j=0}^{k-2} e^{-jhA}\delta_{k-j-1}.$$

(Induction Step) : Now, we will show it is also true for $n = k$

$$\begin{aligned} e_k &= e^{-hA}e_{k-1} + h\phi_1(-hA)G_{k-1} - \delta_k, \\ &= \sum_{j=0}^{k-2} e^{-(k-j-1)hA}h\phi_1(-hA)G_j - \sum_{j=0}^{k-2} e^{-(j+1)hA}\delta_{k-j-1} + h\phi_1(-hA)G_{k-1} - \delta_k, \\ &= \sum_{j=0}^{k-1} e^{-(k-j-1)hA}h\phi_1(-hA)G_j - \sum_{j=0}^{k-1} e^{-jhA}\delta_{k-j}. \end{aligned}$$

Thus, Equation (4.31) holds for $n = k$, and the proof of the induction step is complete.
We can estimate e_n in V by using the inequality (4.6), the Lipschitz condition in Assumption (4.3) and Lemma (4.2) :

$$\begin{aligned}
\| e_n \|_V &= h \sum_{j=0}^{n-1} \| e^{-(n-j-1)hA} \|_V \| \phi_1(-hA) \|_V \| (g(t_j, u_j) - g(t_j, u(t_j))) \|_V \\
&\quad + \| \sum_{j=0}^{n-1} e^{-jhA} \delta_{n-j} \|_V, \\
&\leq hL \sum_{j=0}^{n-1} \| e^{-(n-j-1)hA} \|_V \| \phi_1(-hA) \|_V \| e_j \|_V + CMh, \\
&= hL \sum_{j=0}^{n-2} \| e^{-(n-j-1)hA} \|_V \| \phi_1(-hA) \|_V \| e_j \|_V + hL \| \phi_1(-hA) \|_V \| e_{n-1} \|_V \\
&\quad + CMh.
\end{aligned}$$

We will use following remark to bound ϕ_1.

Remark 4.2 *We can find a bound for ϕ_1 using integral representation of ϕ function (3.28) and semigroup property*

$$\| \phi_1(-hA) \| \leq C.$$

With the aid of this inequality, the estimation of e_n can be written as

$$\begin{aligned}
\| e_n \|_V &= hL \sum_{j=0}^{n-2} \| e^{-(n-j-1)hA} A^{\alpha} [(n-j-1)h]^{\alpha} \| [(n-j-1)h]^{-\alpha} \| \phi_1(-hA) \|_V \| e_j \|_V \\
&\quad + hL \| \phi_1(-hA) \|_V \| e_{n-1} \|_V + CMh, \\
&\leq hLC \sum_{j=0}^{n-2} [(n-j-1)h]^{-\alpha} \| e^{-hA} \|_V \| e_j \|_V + hL \| e(-hA) A^{\alpha} h^{\alpha} \| h^{-\alpha} \| e_{n-1} \|_V \\
&\quad + CMh, \\
&\leq hLC \sum_{j=0}^{n-2} [(n-j-1)h]^{-\alpha} \| e_j \|_V + hLCh^{-\alpha} \| e_{n-1} \|_V \\
&\quad + Ch \sup_{0 \leq t \leq t_n} \| A^{\beta-1} g'(t, u(t)) \|_V .
\end{aligned}$$

Finally, we obtain the desired error bound

$$\| e_n \|_V \leq Ch \sum_{j=0}^{n-2} t_{(n-j-1)h}^{-\alpha} \| e_j \|_V + Ch^{1-\alpha} \| e_{n-1} \|_V + Ch \sup_{0 \leq t \leq t_n} \| A^{\beta-1} g'(t, u(t)) \|_V, \quad (4.32)$$

with a constant C. In order to regularize inequality (4.32) we need following lemma, which is called Discrete Gronwall Lemma.

Lemma 4.3 *For $h > 0$ and $T > 0$, let $0 \leq t_n = nh \leq T$. Further assume that the sequence of non-negative numbers μ_n satisfy the following inequality*

$$\mu_n \leq ph \sum_{v=1}^{n-1} t_{n-v}^{-\sigma} + q t_n^{-\rho},$$

for $\sigma \geq 0, \rho < 1$ and $p, q \geq 0$. Then the discrete Gronwall inequality is

$$\mu_n \leq C q t_n^{-\rho},$$

where C depends on σ, ρ, p and T.

The proof will be ended by application of Discrete Gronwall Lemma to the inequality (4.32).

$$\| u_n - u(t_n) \|_V \leq Ch \sup_{0 \leq t \leq t_n} \| A^{\beta-1} g'(t, u(t)) \|_V . \quad (4.33)$$

□

We used the convergence analysis of exponential Euler method presented in (Hochbruck & Ostermann, 2010). In the next section, we will adopt the error bounds for second order method to linear parabolic problems by following similar way.

4.3. Convergence of the Second Order Method

Consider the linear initial value problem

$$u'(t) + Au(t) = g(t), \; u(0) = u_0, \quad (4.34)$$

The solution of (4.34) at time $t_{n+1} = t_n + h$, $n = 0, 1, ...$ is given by the variation of constants formula

$$u(t_{n+1}) = e^{-hA}u(t_n) + \int_0^h e^{-(h-\tau)A}g(t_n + \tau)d\tau. \tag{4.35}$$

We expand $g(t_n + \tau)$ in a Taylor series with integral form of the reminder

$$g(t_n + \tau) = g(t_n) + \frac{\tau}{h}g(t_{n+1} - g(t_n)) + \int_0^\tau (\tau - \sigma)g''(t_n + \sigma)d\sigma. \tag{4.36}$$

Then, we insert (4.36) in the exact solution with integral form of the reminder yields

$$u(t_{n+1}) = e^{-hA}u(t_n) + h(\phi_1(-hA) - \phi_2(-hA))g(t_n) + h\phi_2(-hA)g(t_{n+1}) + \delta_{n+1} \tag{4.37}$$

$$\delta_{n+1} = \int_0^h e^{-(h-\tau)A} \int_0^\tau (\tau - \sigma)g''(t_n + \sigma)d\sigma d\tau. \tag{4.38}$$

For the defect δ_{n+1} we have following lemma.

Lemma 4.4 *Let the initial value problem*

$$u'(t) + Au(t) = g(t),\ u(0) = u_0 \tag{4.39}$$

satisfy Assumption (4.2). Moreover, let $0 < \beta \leq 1$ *and* $A^{\beta-1}g'' \in L^\infty(0, T)$*. Then,*

$$\| \sum_{j=0}^{n-1} e^{-jhA}\delta_{n-j} \|_V \leq Ch^2 sup_{0\leq t\leq t_n} \| A^{\beta-1}g''(t) \|_V, \tag{4.40}$$

holds with a constant C, uniformly in $0 \leq t_n \leq T$.

Proof Firstly, we represent the supremum by M, where M is denoted by

$$M := \sup_{0\leq t\leq t_n} \| A^{\beta-1}g''(t) \|_V . \tag{4.41}$$

We will begin with constructing the left hand side of the expression (4.40). Firstly, we will write δ_{n-j} as follows:

$$\delta_{n-j} = \int_0^h e^{-(h-\tau)A} \int_0^\tau (\tau - \sigma) g''(t_{n-j-1} + \sigma) d\sigma d\tau, \tag{4.42}$$

multiply the equation (4.42)by $e^{-jhA} A^{1-\beta} A^{\beta-1}$

$$e^{-jhA}\delta_{n-j} = e^{-jhA} \int_0^h e^{-(h-\tau)A} A^{1-\beta} \int_0^\tau A^{\beta-1}(\tau - \sigma) g''(t_{n-j-1} + \sigma) d\sigma d\tau. \tag{4.43}$$

For $j = 0$, this equality takes the form

$$\delta_n = \int_0^h e^{-(h-\tau)A} A^{1-\beta} \int_0^\tau A^{\beta-1}(\tau - \sigma) g''(t_{n-j-1} + \sigma) d\sigma d\tau.$$

To use the stability bound (4.6) multiply this equation by $(h-\tau)^{1-\beta}(h-\tau)^{\beta-1}$

$$\delta_n = \int_0^h (h-\tau)^{1-\beta}(h-\tau)^{\beta-1} e^{-(h-\tau)A} A^{1-\beta} \int_0^\tau A^{\beta-1}(\tau - \sigma) g''(t_{n-j-1} + \sigma) d\sigma d\tau.$$

This expression is bounded in V by

$$\begin{aligned} \| \delta_n \|_V &\leq C \int_0^h (h-\tau)^{\beta-1} \int_0^\tau A^{\beta-1}(\tau - \sigma) g''(t_{n-j-1} + \sigma) d\sigma d\tau, \\ &\leq CM \int_0^h (h-\tau)^{\beta-1} \int_0^\tau (\tau - \sigma) d\sigma d\tau, \\ &= CM \int_0^h \frac{\tau^2}{2} (h-\tau)^{\beta-1} d\tau. \end{aligned}$$

Applying integration by parts, we obtain

$$\| \delta_n \|_V \leq CMh^{\beta+2}.$$

The remaining terms for $j = 1, ..., n-1$ are defined as

$$e^{-jhA}\delta_{n-j} = \int_0^h e^{-(h-\tau)A} e^{-jhA} A^{1-\beta}(jh)^{1-\beta}(jh)^{\beta-1} \int_0^\tau A^{\beta-1}(\tau-\sigma)g''(t_{n-j-1}+\sigma)d\sigma d\tau.$$

Hence, this term can be bounded in V by

$$\| e^{-jhA}\delta_{n-j} \|_V \leq CM \int_0^h e^{-(h-\tau)A}(jh)^{\beta-1} \int_0^\tau (\tau-\sigma)d\tau,$$

using integration by parts we obtain

$$\begin{aligned} \| e^{-jhA}\delta_{n-j} \|_V &\leq \frac{CM}{2}(jh)^{\beta-1}\phi_3(-hA)h^3, \\ &\leq \frac{CM}{2}(jh)^{\beta-1}h^3. \end{aligned}$$

Thus, the remaining sum is obtained by

$$\begin{aligned} \| \sum_{j=1}^{n-1} e^{-jhA}\delta_{n-j} \|_V &\leq CM \sum_{j=1}^{n-1} h^3(jh)^{\beta-1}, \\ &\leq CMh^2 \int_0^{t_{n-1}} t^{\beta-1}dt, \\ &\leq CMh^2. \end{aligned}$$

Finally, we are in the position to state the convergence result for second order method. The error can be written as

$$\| \sum_{j=0}^{n-1} e^{-jhA}\delta_{n-j} \|_V \leq Ch^2 \sup_{0\leq t\leq t_n} \| A^{\beta-1}g''(t) \|_V .$$

□

In the following section, we will derive the error bounds for exponential midpoint method of linear parabolic problems.

4.4. Convergence of the Exponential Midpoint Method

The analysis of exponential midpoint method follows with the same ideas that were used in previous section. Assume that the initial value problem

$$u'(t) + Au(t) = g(t),\ u(0) = u_0. \tag{4.44}$$

The solution of (4.44) at time $t_{n+1} = t_n + h,\ n = 0, 1, ...$ can be given by the variation of constants formula

$$u(t_{n+1}) = e^{-hA}u(t_n) + \int_0^h e^{-(h-\tau)A}g(t_n + \tau)d\tau. \tag{4.45}$$

This scheme is obtained by approximating the function g within the integral by midpoint rule. We expand the term $g(t_n + \tau)$ in the variation of constants formula in a Taylor series with integral form of the reminder

$$\begin{aligned} g(t_n + \tau) &= g((t_n + h/2) + (\tau - h/2)), \\ &= g(t_n + h/2) + \int_0^{\tau - h/2} g'(t_n + h/2 + \sigma)d\sigma, \end{aligned} \tag{4.46}$$

and then we also expand $g(t_n + h/2)$

$$g(t_n + h/2) = g(t_n) + \frac{h}{2}g'(t_n) + \int_0^{h/2} \frac{h}{2}g''(t_n + \tau)d\tau. \tag{4.47}$$

Finally, we substitude (4.47) and (4.46) in the (4.45)

$$u(t_{n+1}) = e^{-hA}u(t_n) + h\phi_1(-hA)g(t_n) + \frac{h^2}{2}\phi_1(-hA)g'(t_n) + \delta_{n+1}, \tag{4.48}$$

where the reminder δ_{n+1} leads to

$$\delta_{n+1} = \int_0^h e^{-(h-\tau)A} \int_0^{\frac{h}{2}} (\frac{h}{2} - \sigma)g''(t_n + \sigma)d\sigma d\tau + \int_0^h e^{-(h-\tau)A} \int_0^{\tau - \frac{h}{2}} g'(t_n + \frac{h}{2} + \sigma)d\sigma d\tau.$$

For this reminder we have following estimation.

Lemma 4.5 *Let the initial value problem (4.44) satisfy Assumption (4.2). Moreover, let* $0 < \beta \leq 1$ *and* $A^{\beta-1}g'' \in L^{\infty}(0,T)$. *Then,*

$$\| \sum_{j=0}^{n-1} e^{-jhA}\delta_{n-j} \|_V \leq Ch^2 \, sup_{0\leq t\leq t_n} \| A^{\beta-1}g''(t) \|_V + Ch \, sup_{0\leq t\leq t_n} \| A^{\beta-1}g'(t) \|_V, \qquad (4.49)$$

holds with a constant C, uniformly in $0 \leq t_n \leq T$.

Proof Firstly, we represent the supremum by M_1 and M_2 as

$$M_1 := \sup_{0\leq t\leq t_n} \| A^{\beta-1}g''(t) \|_V,$$
$$M_2 := \sup_{0\leq t\leq t_n} \| A^{\beta-1}g'(t) \|_V .$$

To construct the left hand side of the expression (4.49), we may write δ_{n-j} with the help of equation (4.49)

$$\begin{aligned} \delta_{n-j} &= \int_0^h e^{-(h-\tau)A}\Big[\int_0^{\frac{h}{2}} (\frac{h}{2} - \sigma)g''(t_{n-j-1} + \sigma)d\sigma \\ &\quad + \int_0^{\tau-\frac{h}{2}} g'(t_{n-j-1} + \frac{h}{2} + \sigma)d\sigma \Big] d\tau. \end{aligned} \qquad (4.50)$$

Multiply the equation (4.50) by $e^{-jhA}A^{1-\beta}A^{\beta-1}$

$$\begin{aligned} e^{-jhA}\delta_{n-j} &= e^{-jhA}\int_0^h e^{-(h-\tau)A}A^{1-\beta}A^{\beta-1}\Big[\int_0^{\frac{h}{2}} (\frac{h}{2} - \sigma)g''(t_{n-j-1} + \sigma)d\sigma \\ &\quad + \int_0^{\tau-\frac{h}{2}} g'(t_{n-j-1} + \frac{h}{2} + \sigma)d\sigma \Big] d\tau. \end{aligned} \qquad (4.51)$$

For $j = 0$ and multiplying the equation by $(h-\tau)^{1-\beta}(h-\tau)^{\beta-1}$

$$\begin{aligned} \delta_n &= \int_0^h e^{-(h-\tau)A}(h-\tau)^{1-\beta}A^{1-\beta}(h-\tau)^{\beta-1}\Big[\int_0^{\frac{h}{2}} A^{\beta-1}(\frac{h}{2} - \sigma)g''(t_{n-1} + \sigma)d\sigma \\ &\quad + \int_0^{\tau-\frac{h}{2}} A^{\beta-1}g'(t_{n-1} + \frac{h}{2} + \sigma)d\sigma \Big] d\tau. \end{aligned}$$

By using the stability bound (4.6), we take the norm of this identity in V

$$\| \delta_n \|_V \leq C \int_0^h (h-\tau)^{\beta-1} [\int_0^{\frac{h}{2}} A^{\beta-1} (\frac{h}{2} - \sigma) g''(t_{n-1} + \sigma) d\sigma + \int_0^{\tau - \frac{h}{2}} A^{\beta-1} g'(t_{n-1} + \frac{h}{2} + \sigma) d\sigma] d\tau.$$

We thus have

$$\begin{aligned} \| \delta_n \|_V &\leq C \int_0^h (h-\tau)^{\beta-1} \Big[M_1 \int_0^{\frac{h}{2}} (\frac{h}{2} - \sigma) d\sigma + M_2 \int_0^{\tau - \frac{h}{2}} d\sigma \Big] d\tau, \\ &= C \int_0^h (h-\tau)^{\beta-1} \Big[M_1 \frac{h^2}{8} + M_2 (\tau - \frac{h}{2}) \Big] d\tau, \\ &= C M_1 \frac{h^2}{8} \int_0^h (h-\tau)^{\beta-1} d\tau + C M_2 \int_0^h (h-\tau)^{\beta-1} (\tau - \frac{h}{2}) d\tau, \end{aligned}$$

and this gives us the bound

$$\begin{aligned} \| \delta_n \|_V &\leq C M_1 \frac{1}{8\beta} h^{\beta+2} + C M_2 \frac{1}{(\beta+1)(\beta+2)} h^{\beta+2} - C M_2 \frac{1}{2\beta} h^{\beta+1}, \\ &\leq C M h^{\beta+2}. \end{aligned}$$

The remaining terms for $j = 1, ..., n-1$,

$$\begin{aligned} e^{-jhA} \delta_{n-j} &= \int_0^h e^{-(h-\tau)A} e^{-jhA} A^{1-\beta} (jh)^{1-\beta} (jh)^{\beta-1} \Big[\int_0^{\frac{h}{2}} A^{\beta-1} (\frac{h}{2} - \sigma) g''(t_{n-j-1} + \sigma) d\sigma \\ &+ \int_0^{\tau - \frac{h}{2}} A^{\beta-1} g'(t_{n-j-1} + \frac{h}{2} + \sigma) d\sigma \Big] d\tau. \end{aligned}$$

The stability estimate (4.6) enables us to define the bounded operators. The bound can be written in V as follows:

$$\| e^{-jhA} \delta_{n-j} \|_V \leq C \int_0^h e^{-(h-\tau)A} (jh)^{\beta-1} \Big[M_1 \int_0^{\frac{h}{2}} (\frac{h}{2} - \sigma) d\sigma + M_2 \int_0^{\tau - \frac{h}{2}} d\sigma \Big] d\tau.$$

After some calculations we obtain

$$\| e^{-jhA} \delta_{n-j} \|_V \leq \frac{C M_1}{8} (jh)^{\beta-1} \phi_1(-hA) h^3 + C M_2 (jh)^{\beta-1} \phi_2(-hA) h^2 + \frac{C M_2}{2} (jh)^{\beta-1} \phi_1(-hA) h^2.$$

We then write the remaining sum as

$$\begin{aligned} \| \sum_{j=1}^{n-1} e^{-jhA}\delta_{n-j} \|_V &\leq \frac{CM_1}{8}h^3 \sum_{j=1}^{n-1}(jh)^{\beta-1} + CM_2h^2 \sum_{j=1}^{n-1}(jh)^{\beta-1} + \frac{CM_2}{2}h^2 \sum_{j=1}^{n-1}(jh)^{\beta-1}, \\ &\leq CM_1h^2 \int_0^{t_{n-1}} t^{\beta-1}dt + CM_2h \int_0^{t_{n-1}} t^{\beta-1}dt, \\ &\leq CM_1h^2 + CM_2h^2. \end{aligned}$$

Finally, we get the desired estimation

$$\| \sum_{j=0}^{n-1} e^{-jhA}\delta_{n-j} \|_V \leq Ch^2 \sup_{0\leq t\leq t_n} \| A^{\beta-1}g''(t) \|_V + Ch \sup_{0\leq t\leq t_n} \| A^{\beta-1}g'(t) \|_V .$$

□

CHAPTER 5

INTERPOLATION THEORY

In this chapter, we will suppose that there is an unknown function for which its exact values at some distinct points are given. To solve such problems we need to transform given distinct data points to a continuous function. This is the subject of interpolation theory. Various ways can be used to write an interpolation polynomial such as Lagrange, Newton divided differences, etc. but the polynomial will be the same according to following theorem

Theorem 5.1 *If $t_0, t_1, ..., t_n$ are n+1 distinct points, $(t_0, g(t_0)), (t_1, g(t_1)), ..., (t_n, g(t_n))$ are given. There is a polynomial $p(t)$ that interpolates $g(t_i)$ at t_i , $i = 0, 1, ..., n$. The polynomial $p(t)$ of degree $\leq n$ is unique.*

Proof The proof can be read from (Atkinson, 1988). □

In present chapter, we will use Newton divided difference interpolation polynomial.

5.1. Newton Divided Differences Polynomial Approximation

In this manner, we will firstly give a brief information about Newton Divided Differences. Our aim is to approximate a set of distinct points by a simple polynomial function. Consider a polynomial of degree at most n that passes through the $n + 1$ points.

t	t_0	t_1	t_2	...	t_n
$g(t)$	$g(t_0)$	$g(t_1)$	$g(t_2)$	...	$g(t_n)$

We would like to begin with developing a procedure to compute $d_1, d_2, ..., d_n$ such that an interpolation polynomial has the following form

$$p_n(x) = d_0 + d_1(t - t_0) + d_2(t - t_0)(t - t_1) + ... + d_n(t - t_0)...(t - t_{n-1}), \quad (5.1)$$

$$= d_0 + \sum_{j=1}^{n} d_j \prod_{k=0}^{j-1} (t - t_k). \quad (5.2)$$

The interpolation polynomial of divided differences can be built recursively. Having a recursive formula will be advantageous for us. Let degree$(p_n) = n$ and degree$(p_{n-1}) = n - 1$. We

can write

$$p_n(t) = p_{n-1}(t) + K(t). \tag{5.3}$$

where correction term $K(t) = d_n(t-t_0)...(t-t_{n-1})$. At the distinct nodes, $K(t)$ satisfies following equalities

$$K(t_i) = p_n(t_i) - p_{n-1}(t_i) = g(t_i) - g(t_i) = 0, \;\; i = 0, ..., n-1.$$

Since $p_n(t_n) = g(t_n)$, the coefficients d_n in (5.1) are given by

$$\begin{aligned} d_n &= \frac{g(t_n) - p_{n-1}(t_n)}{(t - n - t_0)...(t_n - t_{n-1})}, && (5.4) \\ &= g[t_0, t_1, ..., t_n]. && (5.5) \end{aligned}$$

The coefficient d_n is called the n-th order Newton divided difference of g. Moreover, the interpolation formula becomes

$$p_n(t) = p_{n-1}(t) + (t - t_0)...(t - t_{n-1})g[t_0, t_1, ..., t_n]. \tag{5.6}$$

The divided differences are generated by

$$g[t_i] = g(t_i), \;\; i = 0, 1, ..., n. \tag{5.7}$$

The first divided differences are defined as

$$g[t_i, t_{i+1}] = \frac{g(t_{i+1}) - g(t_i)}{t_{i+1} - t_i}. \tag{5.8}$$

The second divided differences can be determined by

$$g[t_i, t_{i+1}, t_{i+2}] = \frac{g[t_{i+1}, t_{i+2}] - g[t_i, t_{i+1}]}{t_{i+2} - t_i}. \tag{5.9}$$

A useful formula for computing $g[t_0, t_1, ..., t_n]$ order of n is

$$g[t_0, t_1, ..., t_n] = \frac{g[t_1, t_2, ..., t_n] - g[t_0, t_1, ..., t_{n-1}]}{t_n - t_0}. \tag{5.10}$$

We can show the construction of divided differences in the following table From (5.6) we

t_i	$g(t_i)$	$g[t_i, t_{i+1}]$	$g[t_i, t_{i+1}, t_{i+2}]$	$g[t_i, t_{i+1}, t_{i+2}, t_{i+3}]$	$g[t_i, t_{i+1}, t_{i+2}, t_{i+3}, t_{i+4}]$
t_0	$g(t_0)$				
		$g[t_0, t_1]$			
t_1	$g(t_1)$		$g[t_0, t_1, t_2]$		
		$g[t_1, t_2]$		$g[t_0, t_1, t_2, t_3]$	
t_2	$g(t_2)$		$g[t_1, t_2, t_3]$		$g[t_0, t_1, t_2, t_3, t_4]$
		$g[t_2, t_3]$		$g[t_1, t_2, t_3, t_4]$	
t_3	$g(t_3)$		$g[t_2, t_3, t_4]$		
		$g[t_3, t_4]$			
t_4	$g(t_4)$				

Table 5.1. Format for constructing divided differences of $g(t)$.

obtain following formulas

$$\begin{aligned}
p_0(t) &= g(t_0), && (5.11)\\
p_1(t) &= g(t_0) + (t - t_0)g[t_0, t_1], && (5.12)\\
p_2(t) &= g(t_0) + (t - t_0)g[t_0, t_1] + (t - t_0)(t - t_1)g[t_0, t_1, t_2], && (5.13)\\
&\vdots \\
p_n(t) &= g(t_0) + (t - t_0)g[t_0, t_1] + (t - t_0)(t - t_1)g[t_0, t_1, t_2] \\
&\quad +... + (t - t_0)...(t - t_{n-1})g[t_0, t_1, ..., t_n]. && (5.14)
\end{aligned}$$

This is called Newton's divided difference formula for the interpolation polynomial.
The Newton interpolating polynomial passing through the uniformly spaced $(n+1)$ data points has degree $\leq n$. This polynomial is the same as the Lagrange interpolating polynomial. Therefore, the error of Newton interpolation will be also the same as the error of the Lagrange interpolation. The difference between divided differences and Lagrange interpolation depends only on the computational appearance. The advantage of using Newton interpolation is the efficiency of the use of nested multiplication and the easiness to add more data points for higher-order interpolating polynomials. For instance, we can have interpolation polynomial

of f at the nodes $t_0, t_1, ..., t_n$,

$$p_n(t) = g(t_0) + (t - t_0)g[t_0, t_1] + ... + (t - t_0)...(t - t_{n-1})g[t_0, t_1, ..., t_n],$$

if we want to add a discrete point $(t_{n+1}, g(t_{n+1}))$ to our interpolation, we can do it by using

$$p_{n+1}(t) = p_n(t) + (t - t_0)...(t - t_n)g[t_0, t_1, ..., t_{n+1}],$$

in just one step.

When Newton divided difference polynomials $p_n(t)$ are used for interpolation to compute values of a function $g(t)$, we expect a difference between the approximation value $p_n(t)$ and exact function value $g(t)$. In other words, we wish to know how accurately is our interpolating polynomial approximates to $g(t)$.

$$E_n(t) = g(t) - p_n(t). \tag{5.15}$$

We shouldn't expect to apply (5.15) when $g(t)$ is not known. In such a case we can use the Next Term Rule. According to this rule, $p_{n+1}(t)$ is thought as exact function

$$E_n(t) = p_{n+1}(t) - p_n(t). \tag{5.16}$$

The interpolation error formula of divided differences can be defined as follows:

Suppose $t_0, t_1, ..., t_n$ are distinct nodes in the interval $[a, b]$ and t is a real number in this interval which is different from node points. Construction of the interpolation polynomial $g(t)$ at these points can be written as

$$\begin{aligned} p_{n+1}(t) &= g(t_0) + (t - t_0)g[t_0, t_1] + ... \qquad (5.17)\\ &\quad +(t - t_0)...(t - t_n)g[t_0, t_1, ..., t_n] \\ &\quad +(t - t_0)...(t - t_n)g[t_0, t_1, ..., t_n, t], \\ &= p_n(t) + (t - t_0)(t - t_1)...(t - t_n)g[t_0, t_1, ..., t_n, t]. \qquad (5.18) \end{aligned}$$

Since $p_{n+1}(t) = g(t)$ from the Next Term Rule

$$g(t) - p_n(t) = (t - t_0)(t - t_1)...(t - t_n)g[t_0, t_1, ..., t_n, t]. \tag{5.19}$$

This gives us error formula $E_n(t) = g(t) - p_n(t)$. According to Theorem (5.1), the error of divided difference interpolation is equal to the error of Lagrange polynomial which is given by

$$g(t) - p_n(t) = (t - t_0)(t - t_1)...(t - t_n)\frac{g^{n+1}(\xi)}{(n+1)!}, \tag{5.20}$$

for some $\xi \in \{t_0, t_1, ..., t_n, t\}$.(More information about Lagrange interpolation polynomial can be read from (Atkinson, 1988)). Comparing these two error formulas we get

$$g[t_0, t_1, ..., t_n, t] = \frac{g^{n+1}(\xi)}{(n+1)!}. \tag{5.21}$$

The error bound for $E_n(t)$ can be written as

$$\|E_n(t)\| = \|(t - t_0)(t - t_1)...(t - t_n)\| \ \|g[t_0, t_1, ..., t_n, t]\|. \tag{5.22}$$

From (5.4) we have $d_{n+1} = g[t_0, t_1, ..., t_n, t]$ is a constant. In the following chapter we will combine Newton divided difference polynomial with exponential integrators.

CHAPTER 6

ERROR ANALYSIS OF EXPONENTIAL INTEGRATORS WITH DISCRETE FORCE

In this chapter, we will assume that there is an unknown function for which its exact values at some distinct points are given. To solve such problems we need to transform given distinct data points to a continuous function. This polynomial function is constructed by using Newton divided difference interpolation polynomial. Consider the initial value problem

$$u'(t) + Au(t) = g(t), \quad u(t_0) = u_0, \tag{6.1}$$

where we do not know continuous function $g(t)$, we just know $n + 1$ distinct points given in the following table

t	t_0	t_1	...	t_n
$g(t)$	$g(t_0)$	$g(t_1)$	...	$g(t_n)$

Table 6.1. g is given as a function of time

$p_n(t)$ denotes the Newton divided difference interpolation polynomial of degree $\leq n$ that passes through distinct points. To simplify the notation, we will use notational convention as $p_n(t) = p(t)$ and interpolation error E_n of degree $\leq n$, $E_n(t) = E(t)$.

6.1. Error Analysis of Exponential Euler Method

Theorem 6.1 *Numerical solution of the equation (6.1) with exponential Euler method is*

$$u_{n+1} = e^{-hA}u_n + h\phi_1(-hA)p(t_n). \tag{6.2}$$

Let given initial value problem satisfy Assumption (4.2). Furthermore, $g : [0, T] \to X$ *is differentiable, and* $A^{\beta-1}g' \in L^{\infty}(0, T)$ *where* $\beta \in (0, 1]$. *Then we can bound the error* $\| u_n - u(t_n) \|_V$ *uniformly in* $0 \leq nh \leq T$.

Proof The proof will be based on the differences of the numerical and the exact solutions. Our numerical method is exponential Euler method

$$u_{n+1} = e^{-hA}u_n + h\phi_1(-hA)p(t_n). \tag{6.3}$$

The exact solution will be given by variation of constants formula

$$u(t_{n+1}) = e^{-hA}u(t_n) + h\phi_1(-hA)g(t_n) + \delta_{n+1}, \tag{6.4}$$

we define the reminder term in integral form

$$\delta_{n+1} = \int_0^h e^{-(h-\tau)A} \int_0^\tau g'(t_n + \sigma)d\sigma d\tau. \tag{6.5}$$

We shortly denote $e_n = u_n - u(t_n)$. Substracting (6.4) from (6.3) leads to

$$e_{n+1} = e^{-hA}e_n + h\phi_1(-hA)(p(t_n) - g(t_n)) - \delta_{n+1}. \tag{6.6}$$

Substitute $E_n(t) = g(t) - p_n(t)$ in the equation (6.6) then we have

$$\begin{aligned} e_{n+1} &= e^{-hA}e_n + h\phi_1(-hA)(-E(t_n)) - \delta_{n+1}, & (6.7)\\ &= e^{-hA}e_n - h\phi_1(-hA)E(t_n) - \delta_{n+1}, & (6.8) \end{aligned}$$

where $E_n(t) = (t - t_0)(t - t_1)...(t - t_n)g[t_0, t_1, ..., t_n, t]$. For short we can write

$$e_{n+1} = e^{-hA}e_n - h\phi_1(-hA)E_n - \delta_{n+1}. \tag{6.9}$$

Solving this recursion gives

$$e_n = -\sum_{j=0}^{n-1} e^{-(n-j-1)hA} h\phi_1(-hA)E_j - \sum_{j=0}^{n-1} e^{-jhA}\delta_{n-j}.$$

We can estimate and find a bound for e_n in V as we did in Chapter 4. We have an extra term $\sum_{j=0}^{n-1} e^{-(n-j-1)hA} h\phi_1(-hA)E_j$ which comes from the interpolation error. It is also possible to bound E_n as follows:

$$Max|E_n| \leq \frac{c_{n+1}}{(n+1)!} Max|\gamma_n(t)|,$$

where $c_{n+1} = Max|g^{n+1}(t)|$ and $\gamma_n(t) = (t-t_0)(t-t_1)...(t-t_n)$. □

6.2. Error Analysis of Second Order Method

Theorem 6.2 *Consider the initial value problem (6.1). Numerical solution of this equation with the second order method is given by*

$$u_{n+1} = e^{-hA}u_n + h\Big(\phi_1(-hA) - \phi_2(-hA)\Big)p(t_n) + h\phi_2(-hA)p(t_{n+1}). \tag{6.10}$$

Let Assumption (4.2) be fulfilled. Furthermore, $g : [0,T] \to X$ is differentiable, and $A^{\beta-1}g'' \in L^\infty(0,T)$ where $\beta \in (0,1]$. Then we can bound the error $\| u_n - u(t_n) \|_V$ uniformly in $0 \leq nh \leq T$.

Proof The proof will follow the same argumentation as in the proof of Theorem (6.1). We will deal with the differences of the numerical and the exact solutions. Our numerical method is exponential second order method

$$u_{n+1} = e^{-hA}u_n + h\Big(\phi_1(-hA) - \phi_2(-hA)\Big)p(t_n) + h\phi_2(-hA)p(t_{n+1}). \tag{6.11}$$

Exact solution of given IVP (6.1) is represented by

$$u(t_{n+1}) = e^{-hA}u(t_n) + h\Big(\phi_1(-hA) - \phi_2(-hA)\Big)g(t_n) + h\phi_2(-hA)g(t_{n+1}) + \delta_{n+1}, \tag{6.12}$$

the reminder term δ_{n+1} in integral form leads to

$$\delta_{n+1} = \int_0^h e^{-(h-\tau)A} \int_0^\tau (\tau - \sigma) g''(t_n + \sigma) d\sigma d\tau. \tag{6.13}$$

The representation of the errors e_{n+1} can be written by substracting (6.12) from (6.11). We then obtain

$$\begin{aligned} e_{n+1} &= e^{-hA} e_n + h\Big(\phi_1(-hA) - \phi_2(-hA)\Big)\Big(p(t_n) - g(t_n)\Big) + h\phi_2(-hA)\Big(p(t_{n+1}) - g(t_{n+1})\Big) \\ &\quad -\delta_{n+1}. \end{aligned} \tag{6.14}$$

Substituting $E_n(t) = g(t) - p_n(t)$ in the equation (6.14)

$$e_{n+1} = e^{-hA} e_n - h\Big(\phi_1(-hA) - \phi_2(-hA)\Big)E(t_n) + h\phi_2(-hA)(-E(t_{n+1})) - \delta_{n+1}, \tag{6.15}$$

$$= -h\Big(\phi_1(-hA) - \phi_2(-hA)\Big)E_n - h\phi_2(-hA)E_{n+1} - \delta_{n+1}, \tag{6.16}$$

where $E_n(t) = (t - t_0)(t - t_1)...(t - t_n)g[t_0, t_1, ..., t_n, t]$. As a solution of this recursion we get

$$e_n = -\sum_{j=0}^{n-1} e^{-(n-j-1)hA} h\Big(\phi_1(-hA) - \phi_2(-hA)\Big)E_j + h\phi_2(-hA)E_{j+1} - \sum_{j=0}^{n-1} e^{-jhA}\delta_{n-j}.$$

We can estimate and find a bound for e_n as we did before. An important consequence of the discrete force, we have an extra term

$$\sum_{j=0}^{n-1} e^{-(n-j-1)hA} h\Big(\phi_1(-hA) - \phi_2(-hA)\Big)E_j + h\phi_2(-hA)E_{j+1}),$$

which comes from the interpolation error. We can bound E_n as follows:

$$Max|E_n| \leq \frac{c_{n+1}}{(n+1)!} Max|\gamma_n(t)|,$$

where $c_{n+1} = Max|g^{n+1}(t)|$ and $\gamma_n(t) = (t - t_0)(t - t_1)...(t - t_n)$. □

6.3. Error Analysis of Exponential Midpoint Method

t	t_0	t_1	...	t_n
$g(t)$	$g(t_0)$	$g(t_1)$	...	$g(t_n)$
$g'(t)$	$g'(t_0)$	$g'(t_1)$	...	$g'(t_n)$

Table 6.2. g and g' are given as a function of time

$p(t)$ is a polynomial that interpolates $g(t_i)$ and $q(t)$ is a polynomial that interpolates $g'(t_i)$ at t_i, $i = 0, 1, ..., n$.

Theorem 6.3 *Let Assumption (4.2) be fulfilled for the initial value problem (6.1). Numerical solution of this equation with exponential midpoint method is given by*

$$u_{n+1} = e^{-hA}u_n + h\phi_1(-hA)p(t_n) + \frac{h^2}{2}\phi_1(-hA)q(t_n). \tag{6.17}$$

Moreover, $g : [0, T] \to X$ is differentiable, and $A^{\beta-1}g', A^{\beta-1}g'' \in L^\infty(0, T)$ where $\beta \in (0, 1]$. Then we can bound the error $\| u_n - u(t_n) \|_V$ uniformly in $0 \leq nh \leq T$.

Proof The proof will follow the same pattern as previous theorems. Our numerical method is exponential midpoint method for this theorem. We will find bounds for the differences of the numerical and the exact solutions. Exponential midpoint method can be written as

$$u_{n+1} = e^{-hA}u_n + h\phi_1(-hA)p(t_n) + \frac{h^2}{2}\phi_1(-hA)q(t_n), \tag{6.18}$$

where $q(t)$ denotes divided difference polynomial that passes through distinct points of $g'(t)$. Exact solution of given IVP (6.1) is represented by variation of constants formula and given by

$$u(t_{n+1}) = e^{-hA}u(t_n) + h\phi_1(-hA)g(t_n) + \frac{h^2}{2}\phi_1(-hA)g'(t_n) + \delta_{n+1}, \tag{6.19}$$

we define the reminder term in integral form

$$\delta_{n+1} = \int_0^h e^{-(h-\tau)A} \int_0^{\frac{h}{2}} (\frac{h}{2} - \sigma) g''(t_n + \sigma) d\sigma d\tau + \int_0^h e^{-(h-\tau)A} \int_0^{\tau - \frac{h}{2}} g'(t_n + \frac{h}{2} + \sigma) d\sigma d\tau.$$

Then, we briefly denote the error as $e_n = u_n - u(t_n)$. In order to obtain error, we substract (6.19) from (6.18)

$$e_{n+1} = e^{-hA} e_n + h\phi_1(-hA)(p(t_n) - g(t_n)) + \frac{h^2}{2}\phi_1(-hA)(q(t_n) - g'(t_n)) - \delta_{n+1}. \tag{6.20}$$

Substituting $E_n(t) = g(t) - p_n(t)$ and $R_n(t) = g'(t) - q_n(t)$ in the equation (6.20) we obtain

$$e_{n+1} = e^{-hA} e_n + h\phi_1(-hA)(-E(t_n)) + \frac{h^2}{2}\phi_1(-hA)(-R(t_n)) - \delta_{n+1}, \tag{6.21}$$

where $E_n(t) = (t - t_0)(t - t_1)...(t - t_n) g[t_0, t_1, ..., t_n, t]$ and
$R_n(t) = (t - t_0)(t - t_1)...(t - t_n) g'[t_0, t_1, ..., t_n, t]$.

Shortly, we can denote the recursion as

$$e_{n+1} = -h\phi_1(-hA)E_n - \frac{h^2}{2}\phi_1(-hA)R_n - \delta_{n+1}. \tag{6.22}$$

The solution of this recursion leads to

$$e_n = -\sum_{j=0}^{n-1} e^{-(n-j-1)hA} h\phi_1(-hA)E_j - \frac{h^2}{2}\phi_1(-hA)R_j - \sum_{j=0}^{n-1} e^{-jhA}\delta_{n-j}.$$

We can estimate and find a bound for e_n. We have an extra term

$$\sum_{j=0}^{n-1} e^{-(n-j-1)hA} h\phi_1(-hA)E_j - \frac{h^2}{2}\phi_1(-hA)R_j,$$

which comes from the interpolation error. It is also possible to bound E_n and R_n as follows:

$$Max|E_n| \leq \frac{c_{n+1}}{(n+1)!} Max|\gamma_n(t)|, \quad Max|R_n| \leq \frac{d_{n+1}}{(n+1)!} Max|\gamma_n(t)|,$$

where $c_{n+1} = Max|g^{n+1}(t)|$, $d_{n+1} = Max|g^{n+2}(t)|$ and $\gamma_n(t) = (t - t_0)(t - t_1)...(t - t_n)$. □

CHAPTER 7

NUMERICAL EXPERIMENT

In this chapter, we illustrate the performance of different exponential integrators. For this purpose we solved numerically some class of ODE and PDE problems. We will work with a uniform mesh in time. This chapter includes mainly three parts. Firstly, we will solve Prothero-Robinson equation by using exponential integrators. Secondly, we will deal with the solution of one dimensional and two dimensional problems with discrete force. Finally, our last examples will be Reaction-Diffusion equations. Krylov subspace methods are used for approximating the product of the matrix functions with the corresponding vectors. Last, all simulations will be obtained by running the problems which are written in Matlab programming language.

7.1. Application of Various Exponential Integrators on the Prothero-Robinson Equation

Our first problem is the Prothero-Robinson equation. This equation is a model problem for stiff equations.

$$u' = P(u - h(t)) + h'(t). \tag{7.1}$$

Let we suppose $h(t) = \begin{pmatrix} cos(t) \\ cos(2t) \end{pmatrix}$ and $P = \begin{pmatrix} 1 & 0 \\ -10^{\alpha} & -10^{\alpha} \end{pmatrix}$ with the initial condition $u(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $t \in [0, 1]$.

The exact solution for this problem is given by $u_{ex}(t) = \begin{pmatrix} cos(t) \\ cos(2t) \end{pmatrix}$. The initial condition and exact solution is barrowed from (El-Azab, 2012).

$||P|| \backsimeq 10^{\alpha}$, and the eigenvalues are -10^{α} and 1, then the stiffness of the problem depends on the α.

To solve this problem numerically, we may organize the given informations of (7.1) for $\alpha = 2$.

$$\begin{aligned} u' &= \begin{pmatrix} 1 & 0 \\ -10^2 & -10^2 \end{pmatrix} u - \begin{pmatrix} 1 & 0 \\ -10^2 & -10^2 \end{pmatrix} \begin{pmatrix} cos(t) \\ cos(2t) \end{pmatrix} + \begin{pmatrix} -sin(t) \\ -2sin(2t) \end{pmatrix}, \\ u(0) &= \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \end{aligned}$$

Finally, we obtain

$$u' = \begin{pmatrix} 1 & 0 \\ -10^2 & -10^2 \end{pmatrix} u - \begin{pmatrix} -cost - sint \\ 10^2(cos(t) + cos(2t)) - 2sin(2t) \end{pmatrix}. \tag{7.2}$$

We will solve this equation using various exponential methods. These methods are exponential Euler method, second order method and exponential midpoint method, respectively. The errors of these methods, which are listed in the following Table 7.1., are measured in the maximum norm. Comparison of numerical methods and exact solution is plotted in Figure 7.1. Moreover, all orders of these methods are confirmed by the results illustrated for $\alpha = 2$ in Figure 7.2.

Δt	u	$error_{ExponentialEuler}$	$error_{SecondOrder}$	$error_{MidpointRule}$
0.1	u_1	0.04309863013	0.00181544973	0.00429585398
	u_2	0.20569922274	0.00209595990	0.11316444546
0.01	u_1	0.00421840195	0.00001814987	0.00004327682
	u_2	0.01220267611	0.00004116969	0.00227224660
0.001	u_1	0.00042084677	0.00000018149	0.00000043308
	u_2	0.00102237666	0.00000041825	0.00002308565
0.0001	u_1	0.00004207466	0.00000000181	0.00000000433
	u_2	0.00010022378	0.00000000418	0.00000023089

Table 7.1. Comparison of errors for various exponential integrators different Δt values.

As a result, we can see clearly that the second order method works more effectively than the other methods.

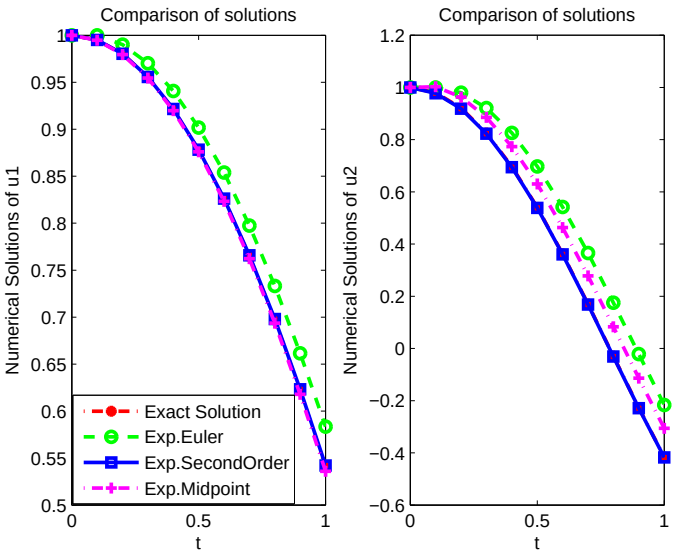

Figure 7.1. Comparison of numerical methods with exact solution for time step $\Delta t = 0.1$.

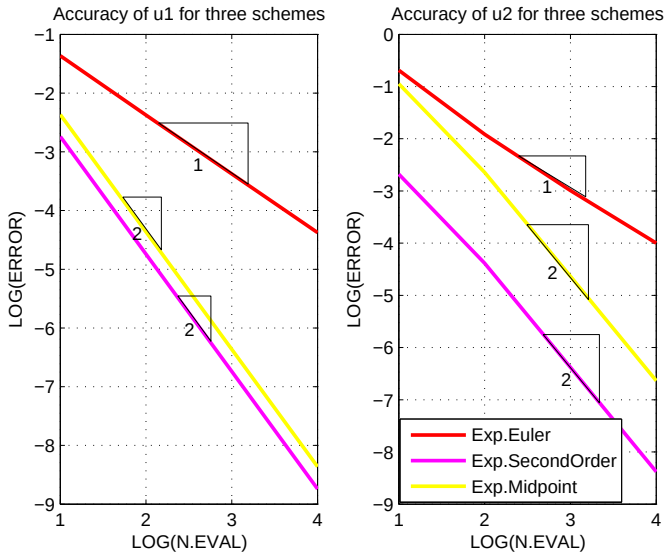

Figure 7.2. Order plot for the exponential methods applied to the Prothero equation for $\alpha = 2$.

When we take $\alpha = 4$, the stiffness ratio of the problem increases. This increase causes order reduction for the equation which is presented in Figure 7.3.

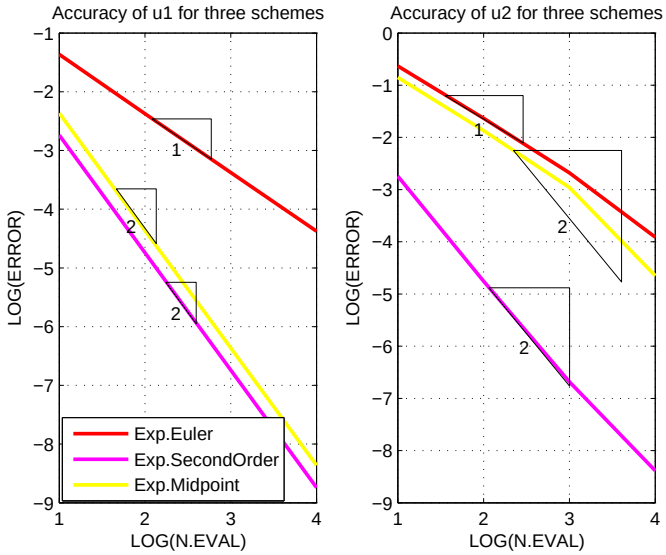

Figure 7.3. Order plot for the exponential methods applied to the Prothero equation for $\alpha = 4$.

7.2. One Dimensional Example to Explain Stiffness

Consider the one dimensional problem

$$y' = -50y + 50f(x), \quad x \in (0, \pi), \tag{7.3}$$

with initial condition $y(0) = 0$. We will use $f(x)$ as a interpolation polynomial of following data points:

x	0	$\pi/3$	$2\pi/3$	π
$f(x)$	1	0.5	-0.5	-1

Discretize in space

$$x_j = j\Delta x, \quad j = 1, 2, ..., N, \tag{7.4}$$

where $\Delta x = \frac{x_N - x_0}{N} = \frac{\pi}{N}$ is the space between the node points. We divided $(0, \pi)$ into N parts of equal length. In order to test the efficiency of the divided difference interpolation polynomial, we construct this problem choosing the distinct values from continuous trigonometric function $cos(x)$. In other words, $f(x) = cos(x)$. Therefore, the exact solution may be written as

$$y(x) = -\frac{2500}{2501}e^{-50x} + \frac{2500}{2501}cos(x) + \frac{50}{2501}sin(x). \tag{7.5}$$

Figure 7.4. shows Newton the divided difference approximation polynomial of given discrete points and $cos(x)$.

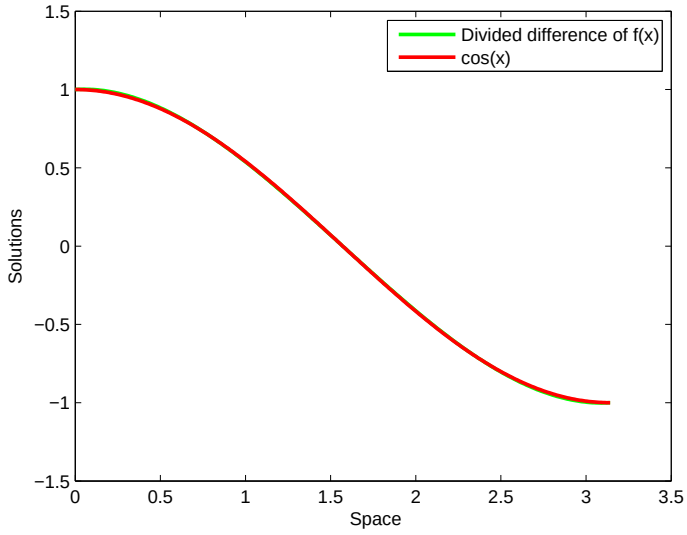

Figure 7.4. Divided difference polynomial of given data points and cos(x).

The errors of explicit Euler, implicit Euler, Runge-Kutta and exponential Euler methods are listed in the following Table 7.2. In Table 7.3., we compare the error of exponential methods. These errors are measured in the maximum norm.

Δx	$ExplicitEuler$	RK_4	$ImplicitEuler$	$ExponentialEuler$
0.1	$4.6081.10^{18}$	$1.7150.10^{35}$	0.1565	0.0806
0.01	0.1183	0.0182	0.0765	0.0128
0.001	0.0109	0.0116	0.0112	0.0111
0.0001	0.0109	0.0110	0.0110	0.0109

Table 7.2. Comparison of traditional methods and exponential euler method errors with discrete force for different Δx values.

Δx	$ExponentialEuler$	$SecondOrderMethod$	$MidpointRule$
0.1	0.0806	0.0102	0.0319
0.01	0.0128	0.0109	0.0110
0.001	0.0111	0.0109	0.0109
0.0001	0.0109	0.0109	0.0109

Table 7.3. Comparison of different exponential integrators errors with discrete force for different Δx values.

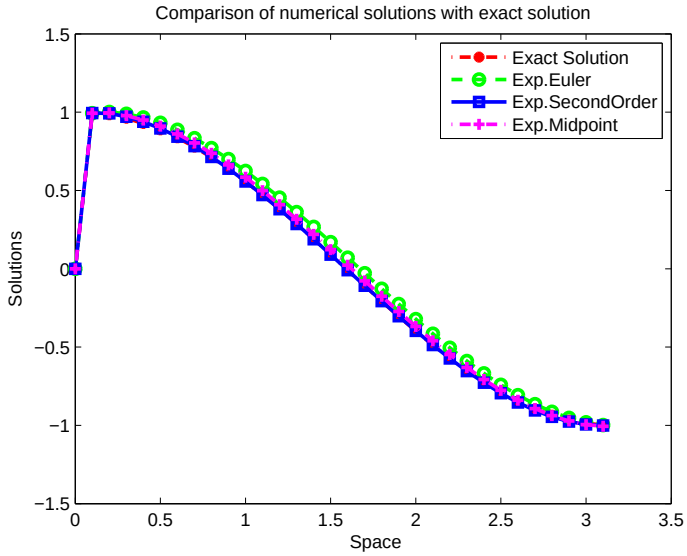

Figure 7.5. Comparison of different exponential integrators with exact solution for $\Delta x = 0.1$.

Figure 7.5. indicates the exact solution and exponential numerical solutions for considered problem for $\Delta x = 0.1$. As a consequence, we can see clearly that the second order method works more effectively than the other methods.

7.3. Two Dimensional ODE Problem

Consider a linear initial value problem

$$y_1' = -y_1 + f(t), \ y_1(0) = 1,$$
$$y_2' = 2y_1 - 100y_2, \ y_2(0) = 0,$$

$f(t)$ is the discrete force which is applied $t \in [0, 0.3]$. Suppose that we have four distinct values of force at time t.

t	0	0.1	0.2	0.3
$f(t)$	1	1.1052	1.2214	1.3499

Table 7.4. Given discrete data points.

In order to solve the problem, firstly we illustrate interpolation polynomial using the Newton divided differences. This system is equivalent to following form

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 2 & -100 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} f(t) \\ 0 \end{bmatrix}, \ u_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{7.6}$$

Discretize in time

$$t_j = j\Delta t, \quad j = 1, 2, ..., N, \tag{7.7}$$

where $\Delta t = \frac{t_N - t_0}{N} = \frac{0.3}{N}$ is the space between the node points. We divided $[0, 0.3]$ into N parts of equal length. To test the efficiency of the interpolation polynomial, we construct this problem choosing the distinct values from continuous exponential function e^t. In other words,

$f(t) = e^t$. So, the exact solution can be given by

$$\begin{aligned} y_1(t) &= \frac{1}{2}(e^{-t} + e^t), \\ y_2(t) &= -\frac{200}{9999}e^{-100t} + \frac{1}{990e^{-t}} + \frac{1}{101}e^t. \end{aligned}$$

Δt	y	$ExpEuler$	$ExpSecond$	$ExpMidpoint$	$ExplicitEuler$	$ImpEuler$
0.06	y_1	0.0091	$9.1323.10^{-5}$	$2.7125.10^{-4}$	0.0081	0.0101
	y_2	$1.7128.10^{-4}$	$1.7913.10^{-6}$	$3.2474.10^{-6}$	62.5063	0.0028
0.03	y_1	0.0046	$2.2837.10^{-5}$	$6.8169.10^{-5}$	0.0040	0.0051
	y_2	$8.6503.10^{-5}$	$4.4370.10^{-7}$	$1.1758.10^{-6}$	20.4822	0.0040
0.015	y_1	0.0023	$5.7096.10^{-6}$	$1.7086.10^{-5}$	0.0020	0.0026
	y_2	$4.3635.10^{-5}$	$1.1047.10^{-7}$	$3.3047.10^{-7}$	0.0145	0.0035

Table 7.5. Comparison of different methods errors for different Δt values.

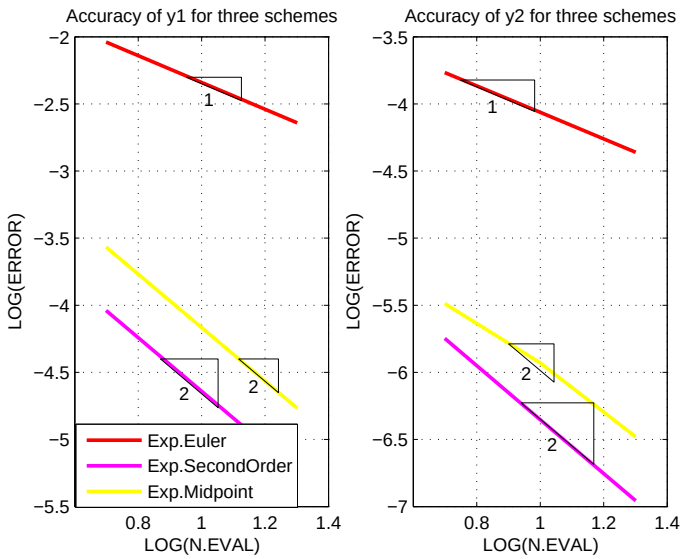

Figure 7.6. Order of exponential methods.

Table 7.5. shows the errors of the exponential Euler method, second order method, exponential midpoint method, traditional explicit and implicit Euler method in L_∞ norm for

different time steps. Figure 7.6. indicates the numerical convergence rate of the exponential methods.

In the following Table 7.6., we will deal with the errors of methods where $f(t)$ is constructed applying Newton divided difference approximation to the distinct values of force. The errors at the endpoint $t = 0.3$ are computed in a discrete L_∞ norm.

Δt	y	$ExpEuler$	$ExpSecond$	$ExpMidpoint$	$ExplicitEuler$	$ImpEuler$
0.06	y_1	0.0091	$9.5455.10^{-5}$	$2.6815.10^{-4}$	0.0081	0.0101
	y_2	$1.7122.10^{-4}$	$1.8678.10^{-6}$	$3.2474.10^{-6}$	62.5063	0.0028
0.03	y_1	0.0046	$2.6977.10^{-5}$	$6.4534.10^{-5}$	0.0040	0.0051
	y_2	$8.6434.10^{-5}$	$5.2063.10^{-7}$	$1.1758.10^{-6}$	20.4822	0.0040
0.015	y_1	0.0023	$9.8524.10^{-6}$	$1.3193.10^{-5}$	0.0020	0.0026
	y_2	$4.3562.10^{-5}$	$1.8756.10^{-7}$	$3.3951.10^{-7}$	0.0145	0.0035

Table 7.6. Comparison of different exponential and traditional methods with divided difference errors for different Δt values.

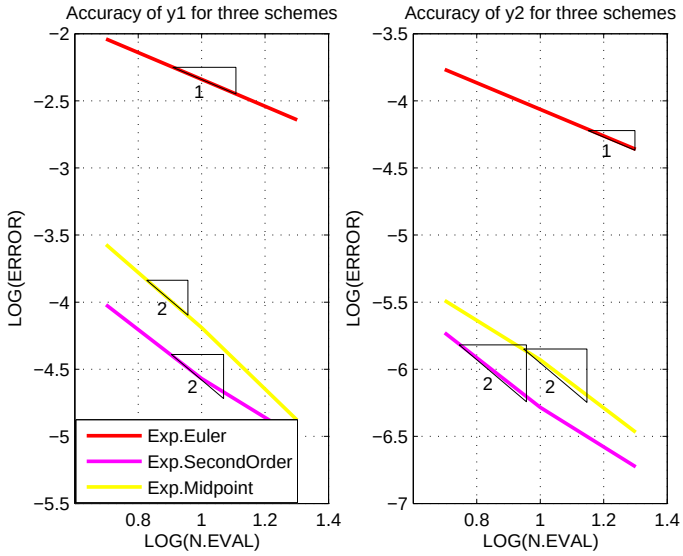

Figure 7.7. Order plot of the exponential methods with divided difference.

Figure 7.7. displays the numerical convergence rate of the exponential methods with discrete force. Divided difference polynomial of given data points and e^t are plotted in Figure 7.8.

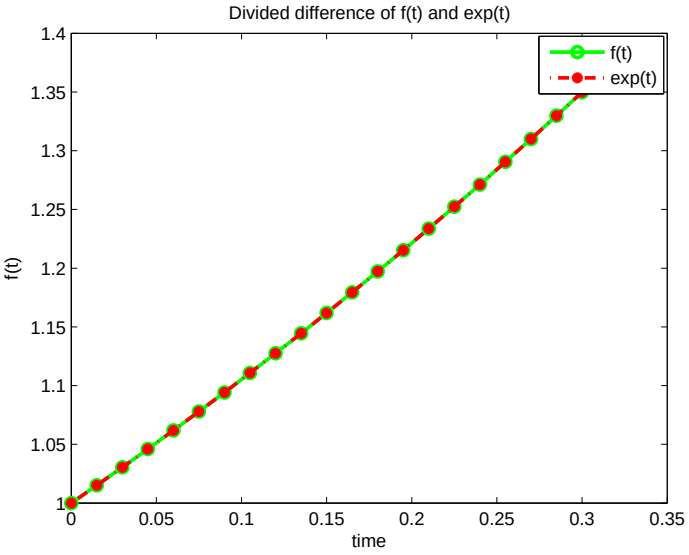

Figure 7.8. Divided difference polynomial of given data points and e^t.

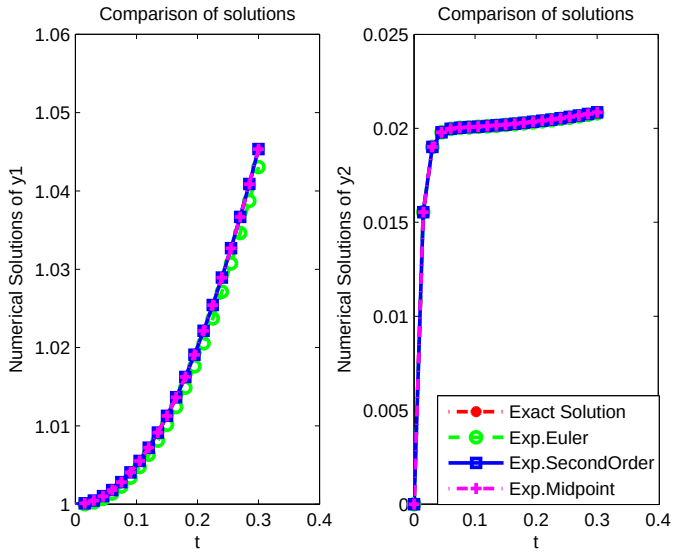

Figure 7.9. Comparison of numerical methods with exact solution for time step $\Delta t = 0.015$.

The comparison of first component y_1 and second component y_2 with exact solution and exponential methods are illustrated in Figure 7.9. As we see, second order method works more efficiently both continuous form e^t and divided difference polynomial $f(t)$ of given discrete data points. There is an order reduction which proceeds from discrete force.

7.4. Example of Reaction-Diffusion Equations

Let us consider how to simulate the reaction diffusion equation by exponential integrators. We will use finite difference approximations for reaction term. In finite difference method, the space is divided into appropriately sized elements. The differential equation is expressed in a matrix equation by representing a discrete set of space points. In this section, we will present the results from numerical experiments on the linear problem and semi-linear problems Fisher and Allen Cahn equations. To solve linear problem we will use exponential Euler, second order and exponential midpoint methods and for solving semilinear problems, we will use the exponential Rosenbrock-Euler method.

7.4.1. Linear Problem

Our first example in this section is,

$$u_t - u_{xx} = (2 + x(1 - x))g(t), \;\; t, x \in [0, 1], \tag{7.8}$$

with initial condition and boundary conditions

$$u(0, x) = x(1 - x), \tag{7.9}$$

$$u(t, 0) = u(t, 1) = 0. \tag{7.10}$$

Table 7.7. shows the Newton divided difference polynomial coefficients.

t	$e^t = g(t_i)$	$g[t_i, t_{i+1}]$	$g[t_i, t_{i+1}, t_{i+2}]$	$g[t_i, t_{i+1}, t_{i+2}, t_{i+3}]$	$g[t_i, t_{i+1}, t_{i+2}, t_{i+3}, t_{i+4}]$
0	1	1.1360	0.6456	0.1374	0.0390
0.25	1.2840	1.4588	0.8288	0.1764	
0.5	1.6487	1.8732	1.0640		
0.75	2.1170	2.4052			
1	2.7183				

Table 7.7. Divided difference table of $g(t)$.

We will solve this equation numerically by using exponential Euler, second order and exponential midpoint methods. We begin using central finite difference quotient approximation for u_{xx}

$$u_{xx}\Big|_{(t,x_i)} \approx \frac{u(t, x_{i+1}) - 2u(t, x_i) + u(t, x_{i-1})}{(\Delta x)^2}, \tag{7.11}$$

we obtain the following semi-discrete differential equation

$$u'(t) = \frac{1}{\Delta x^2} A u(t) + (2 + x(1 - x))g(t), \tag{7.12}$$

where $u(t)$ in equation (7.12) is in the form of $u(t) = (u(t, x_1), u(t, x_2), ..., u(t, x_{N-1}))^T$ and A is $(N - 1) \times (N - 1)$ tridiagonal matrix. $u(0) = (u(0, x_1), u(0, x_1), ..., u(0, x_{N-1}))^T$ is the initial condition and the boundary conditions $u(0, x_0)$ and $u(0, x_N)$ are embedded into the matrix. Then, the space interval $[0, 1]$ is discretized

$$x_i = x_0 + i\Delta x, \;\; i = 0, 1, ..., N, \tag{7.13}$$

where $\Delta x = \frac{x_N - x_0}{N} = \frac{1}{N}$ is the space between the node points. To test the efficiency of the interpolation polynomial, we construct this problem choosing the distinct values from continuous exponential function e^t. Therefore, the exact solution of (7.8) is given by

$$u(t, x) = x(1 - x)e^t. \tag{7.14}$$

Table 7.8 shows the errors of the exponential methods in L_∞ norm for the different time steps

when Δx is fixed at 0.01. The numerical convergence rate of the exponential methods is plotted in Figure 7.10.

Δt	$ExponentialEuler$	$SecondOrder$	$MidpointRule$
0.2	0.08318	0.00212	0.02355
0.1	0.03800	0.00057	0.00592
0.05	0.01795	0.00015	0.00141

Table 7.8. Comparison of different exponential integrators with divided difference errors for different Δt values when $\Delta x = 0.01$ in L_∞ norm.

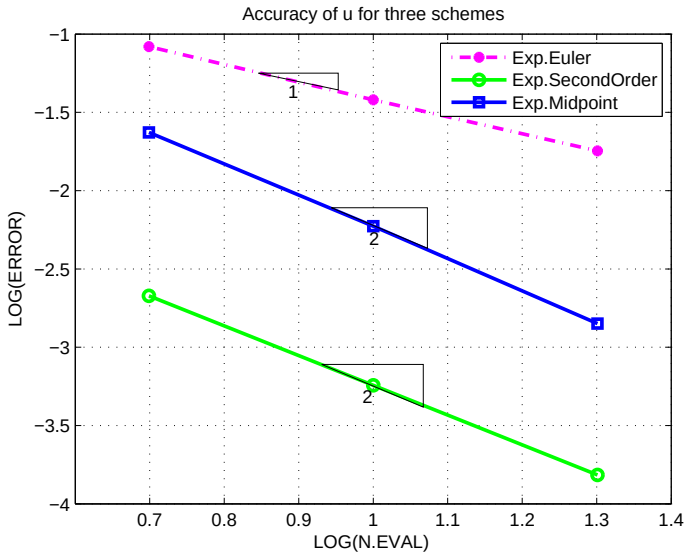

Figure 7.10. Order graphic.

The numerical solutions and the analytic solution of given linear parabolic equation is displayed in Figure 7.11. Figure 7.12 shows the second order method solution for different values of time.

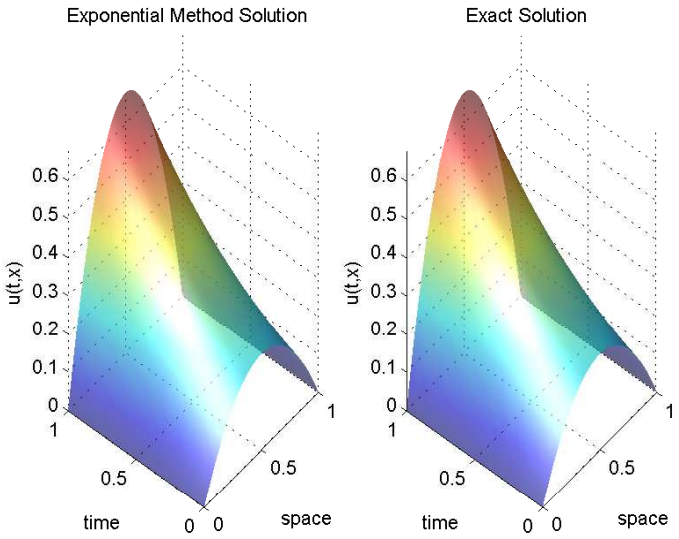

Figure 7.11. Analytic and computed solutions of second order method for linear parabolic equation $\Delta t = 0.05$ and $\Delta x = 0.01$.

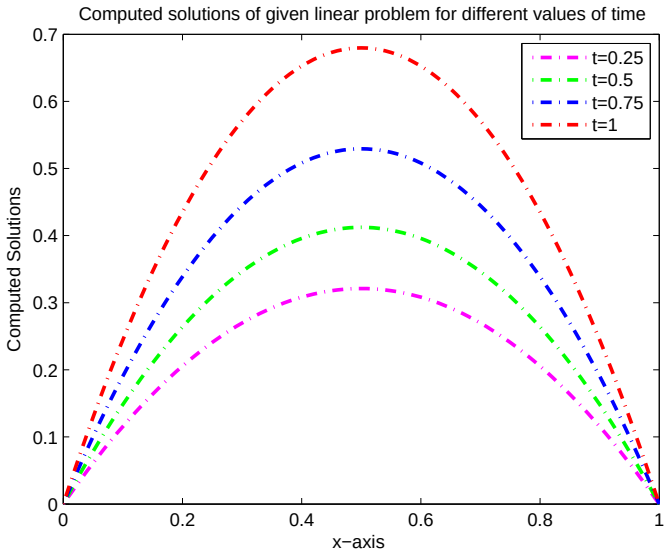

Figure 7.12. Second order method solution for different values of time with $\Delta t = 0.05$ and $\Delta x = 0.01$.

7.4.2. Semilinear Problem: The Fisher Equation

The first example of semilinear parabolic problems is the Fisher's reaction diffusion equation. This equation is in the form

$$u_t = Du_{xx} + ru(1-u), \quad x \in [-10, 10], \quad 0 \leq t \leq 1, \tag{7.15}$$

where $ru(1-u)$ is called Fisher's potential. This equation describes the effects of linear diffusion term u_{xx} and nonlinear reaction term $u(1-u)$. We will take $D = 0.1$ and $r = 1$ with initial and boundary conditions

$$u(0, x) = sech^2(x), \tag{7.16}$$

$$u(t, -10) = 0, \tag{7.17}$$

$$u(t, 10) = 0. \tag{7.18}$$

We will solve this equation numerically by using exponential Rosenbrock-Euler method. We begin with using central finite difference quotient approximation for Equation (7.15) instead of the differential term of space u_{xx}

$$u_{xx}\Big|_{(t,x_i)} \approx \frac{u(t, x_{i+1}) - 2u(t, x_i) + u(t, x_{i-1})}{(\Delta x)^2}, \tag{7.19}$$

we obtain the following semi-discrete differential equation

$$u'(t) = D\frac{1}{\Delta x^2}Au(t) + ru(t)(1-u(t)), \tag{7.20}$$

where $u(t)$ in equation (7.20) is in the form of $u(t) = (u(t, x_1), u(t, x_2), ..., u(t, x_{N-1}))^T$ and A is $(N-1) \times (N-1)$ tridiagonal matrix. $u(0) = (u(0, x_1), u(0, x_1), ..., u(0, x_{N-1}))^T$ is the initial condition and the boundary conditions $u(0, x_0)$ and $u(0, x_N)$ are embedded into the matrix. Then, the space interval $[-10, 10]$ is discretized

$$x_i = x_0 + i\Delta x, \;\; i = 0, 1, ..., N, \tag{7.21}$$

where $\Delta x = \frac{x_N - x_0}{N} = \frac{20}{N}$ is the space between the node points. We now turn to the time discretization of Equation (7.15). Exponential Rosenbrock Euler method is based on continuous linearization of Fisher equation along the numerical solution. For a given point u_n this linearization is

$$\begin{aligned} u'(t) &= J_n u(t) + g_n(u(t)), && (7.22)\\ J_n &= \frac{\partial}{\partial u}\Big(\frac{DA}{\Delta x^2}u(t) + ru(t)(1-u(t))\Big)\Big|_{u_n}, && (7.23)\\ g_n(u(t)) &= \frac{DA}{\Delta x^2}u(t) + ru(t)(1-u(t)) - J_n u(t), && (7.24) \end{aligned}$$

where J_n denotes the Jacobian and g_n denotes the nonlinear reminder term evaluated at u_n. Therefore, the exponential Rosenbrock-Euler method for considered problem is given by

$$u_{n+1} = e^{hJ_n}u_n + h\phi(hJ_n)g_n(u_n), \quad u_0 = u(0,x). \tag{7.25}$$

We will use (7.25) in order to solve Fisher equation numerically. Also, we will use Richardson extrapolation to calculate the errors. At the beginning, around the $x = 0$ diffusion term has a large absolute value, but the reaction term is quite small. In other words, the effect of diffusion dominates over the effect of reaction, so the peak goes down quickly and gets flatter. After the peak arrives at the lowest level, the reaction term dominates the diffusion. Table 7.9. shows the errors of the exponential Rosenbrock-Euler method in L_1, L_2 and L_∞ norm for the different time steps and fixed $\Delta x = 0.5$. Figure 7.13 shows the accuracy of the method for these norms.

Δt	$L_1 norm$	$L_2 norm$	$L_\infty norm$
0.05	8.4532e-004	3.0527e-004	1.5319e-004
0.025	2.1241e-004	7.6486e-005	3.8432e-005
0.0125	5.3253e-005	1.9148e-005	9.6270-006
0.01	3.4102e-005	1.2258e-005	6.1638e-006

Table 7.9. Comparison of exponential Rosenbrock Euler method errors in different norms for $\Delta x = 0.5$.

Figure 7.14. shows the exponential Rosenbrock Euler solution of the Fisher equation. Finally, Figure 7.15. displays the layer behaviour of the Fisher equation for different values of time t.

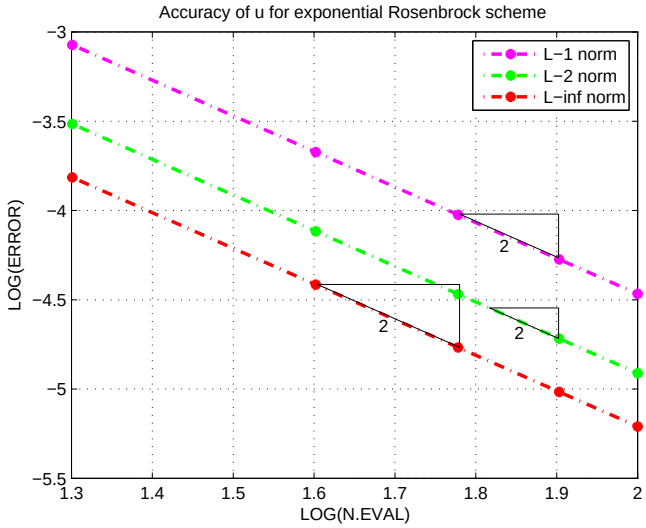

Figure 7.13. Order plot for the exponential Rosenbrock Euler method applied to Fisher equation.

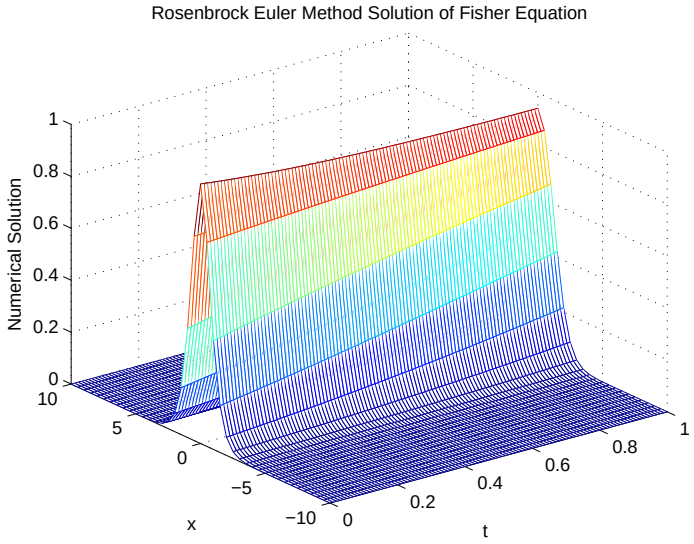

Figure 7.14. Numerical solution of Fisher equation $\Delta t = 0.01$ and $\Delta x = 0.5$.

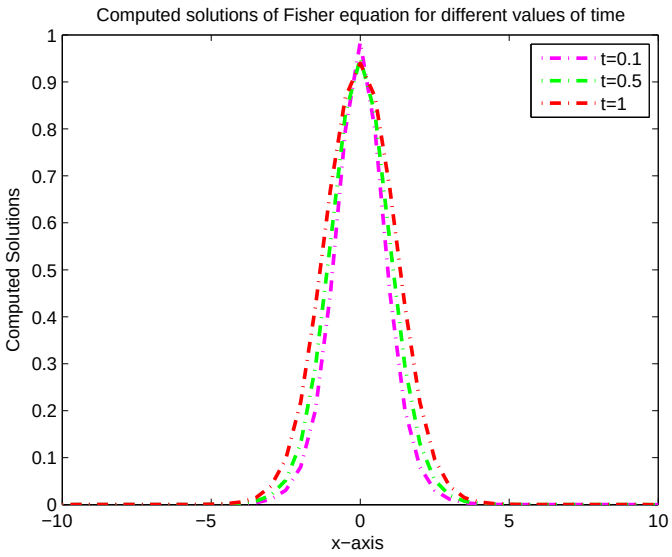

Figure 7.15. Computed solutions of Fisher equation for different values of time with $\Delta t = 0.01$ and $\Delta x = 0.5$.

7.4.3. Semilinear Problem: The Allen Cahn Equation

The second semilinear example is Allen-Cahn equation, which is a well-known equation from the area of reaction diffusion systems :

$$u_t = Du_{xx} + u(1 - u^2), \quad x \in [-1, 1], \tag{7.26}$$

where $D = 0.01$ with initial and boundary conditions barrowed from (Trefethen & Kassam)

$$\begin{aligned} u(0, x) &= 0.53x + 0.47sin(-1.5\pi x), \\ u(t, -1) &= -1, \\ u(t, 1) &= 1. \end{aligned} \tag{7.27}$$

This equation has a stable equilibria at $u = 1$ and $u = -1$ also has an unstable equilibrium at $u = 0$. One of the interesting features of this equation is the phonomenon of metastability. Regions of the solution that are near $1, -1$ will be flat, and the interface between such areas can remain unchanged over a very long timescale before changing suddenly. (Trefethen &

Kassam)

In order to solve the equation numerically, we performed a special discretization with grid length parameter Δx that is get by dividing the interval into N parts of equal length. As we defined before, the spatial derivative of u_{xx} is approximated with the central finite difference scheme is

$$u_{xx}\Big|_{(t,x_i)} \approx \frac{u(t,x_{i+1}) - 2u(t,x_i) + u(t,x_{i-1})}{(\Delta x)^2}, \tag{7.28}$$

where Δx is the spatial stepping in space and $i = 1, ..., N+1$. We will solve this equation by exponential Rosenbrock-Euler method numerically. The exponential Rosenbrock-Euler method for considered autonomous problem is given by

$$u_{n+1} = e^{hJ_n}u_n + h\phi(hJ_n)g_n(u_n), \quad u_0 = u(0,x), \tag{7.29}$$

where J_n denotes the Jacobian and g_n denotes the nonlinear reminder term evaluated at u_n. We calculate the errors by using Richardson extrapolation method at time $t = 1$, which are listed in Table 7.10. Then, order graphic is presented in Figure 7.16.

Δx	Δt	$L_1 norm$	$L_2 norm$	$L_\infty norm$
0.05	0.1	1.8283e-005	7.4222e-006	4.0037e-006
	0.05	4.7878e-006	1.9433e-006	1.0446e-006
	0.025	1.2247e-006	4.9704e-007	2.6671e-007

Table 7.10. Comparison of exponential Rosenbrock Euler method error for different norms.

Moreover, Figure 7.17 displays the exponential Rosenbrock Euler solution of the Allen Cahn equation. Finally, Figure 7.18. shows the layer behaviour of the Allen Cahn equation for different values of time t.

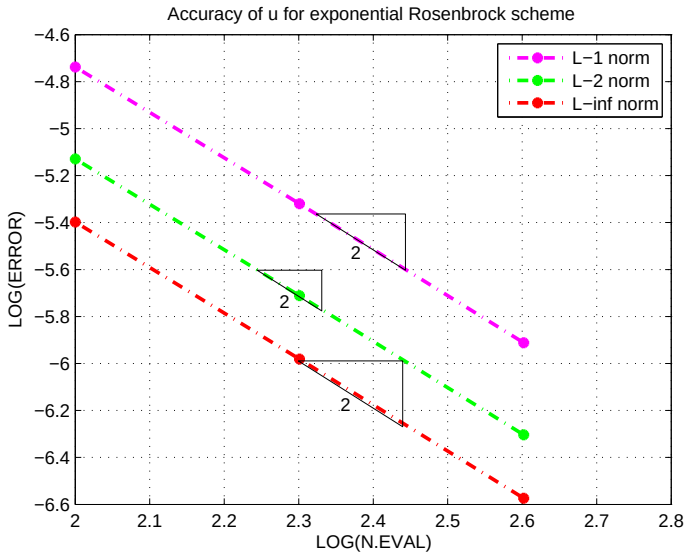

Figure 7.16. Order plot for Allen Cahn equation.

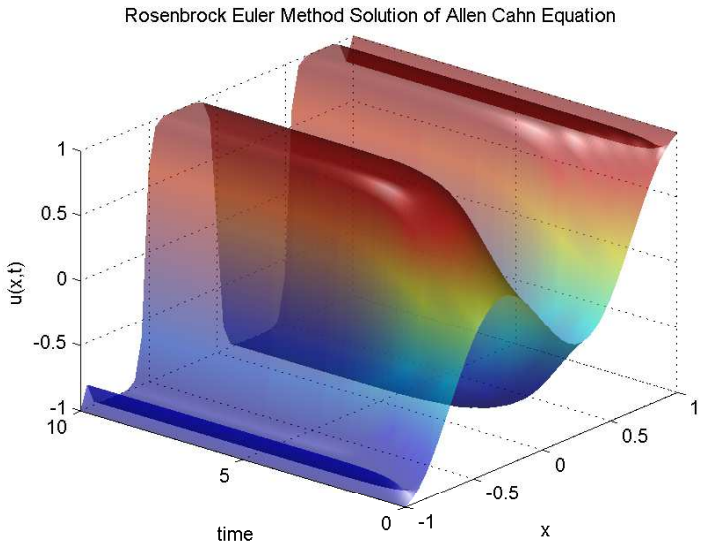

Figure 7.17. Numerical Solution of Allen Cahn equation $\Delta t = 0.1$ and $\Delta x = 0.05$.

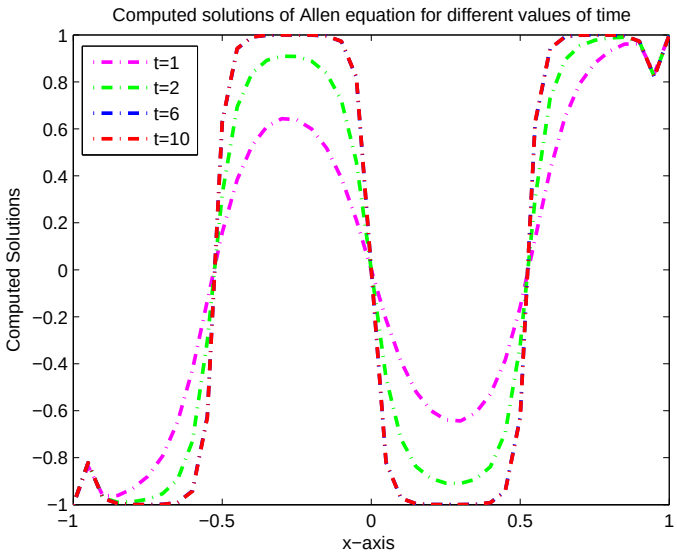

Figure 7.18. Computed solutions of Allen-Cahn equation for different values of time with $\Delta t = 0.1$ and $\Delta x = 0.05$.

CHAPTER 8

CONCLUSION

In this thesis, we studied various types of exponential integrators, namely, exponential Euler method, second order method, exponential midpoint method and exponential Rosenbrock-Euler method. We overviewed the derivation of these integrators and the error analysis for exponential Euler method. We derived error bound for second order method and exponential midpoint method. In these proofs, we follow similar way with (Hochbruck & Ostermann, 2010). We also consider some class of stiff ODE and PDE problems with a discrete force. In such problems, we approximate given data points by using Newton divided difference polynomials. We embedded the approximated polynomial into the differential equation. We obtained the error bounds for these problems. Several examples are illustrated in order to confirm our theoretical results.

REFERENCES

Atkinson, K. 1988: An introduction to Numerical Analysis. *John Wiley & Sons,* **Second Edition**.

Burg, C. and Erwin, T. 2008: Application of Richardson Extrapolation to the Numerical Solution of Partial Differential Equations. *Wiley,* **25(4)** 810-832.

Caliari, M. and Ostermann, A. 2009: Implementation of Exponential Rosenbrock-Type Integrators. *Elsevier Applied Numerical Mathematics,* **59(3),** 568-581.

Curtis, C.F. and Hirschfelder, J.O. 1952: Integration of Stiff Equations. *Proc. Natl. Acad. Sci. USA,* **38(3),** 235-243.

El-Azab, T.M.A. 2012: Exponential Peer Methods. *Martin-Luther-Universität Halle-Wittenberg*.

Hairer, E., Norsett,S.P. and Wanner,G. 1993: Solving Ordinary Differential Equations I: Nonstiff Problems. *Springer,* **Second Edition**.

Hairer, E. and Wanner,G. 2000: Solving Ordinary Differential Equations II. *Springer,* **Second Edition**.

Henry, D. 1981: Geometric Theory of Semilinear Parabolic Equations. *Springer-Verlag* .

Hochbruck, M. and Ostermann, A. 2010: Exponential Integrators. *Applied Numerical Mathematics,* **Cambridge University Press,** 209-286.

Hochbruck, M., Ostermann, A. and Schweitzer,J. 2008: Explicit Exponential Runge-Kutta Methods for Semilinear Parabolic Problems. *SIAM Journal on Numerical Analysis,* **47(1),** 786-803.

Hochbruck, M. and Ostermann, A. 2005: Exponential Runge-Kutta Methods for Parabolic Problems. *Applied Numerical Mathematics,* **53(2-4),** 323-339.

Hochbruck, M., Lubich, C. and Selhofer, H. 1998: Exponential Integrators for Large Systems of Differential Equations. *SIAM J. Sci. Comput.,* **19(5),** 1552-1574.

Kandolf, P. 2011: Exponential Integrators. *McMaster University*.

Lambert, J.D 1991: Numerical Methods for Ordinary Differential Systems. *John Wiley & Sons*.

Lawson, J.D. 1967: Generalized Runge-Kutta Processes for Stable Systems with Large Lipschitz Constants. *SIAM Journal on Numerical Analysis,* **4(3),** 372-380.

Liniger, W. and Willoughby, R.A. 1970: Efficient Integration Methods for Stiff Systems of Ordinary Differential Equations. *SIAM Journal on Numerical Analysis,* **7(1)**.

Michels,D.L., Sobottka,G.A., Weber,A.G. 2014: Exponential Integrators for Stiff Elastodynamic Problems. *ACM Transactions on Graphics* **33(1)**.

Minchev,B.V., Wright,W.M. ,2005: A review of exponential integrators for first order semilinear problems. *NTNU* **2005(2)**.

Pazy, A. 1983: Semigroups of Linear Operators and Applications to Partial Differential Equations. *Springer-Verlag* .

Pope, D.A. 1963: An Exponential Method of Numerical Integration of Ordinary Differential Equations. *Communications of the ACM,* **6(8),** 491-493.

Schmelzer,T., Trefethen,L.N. 2007: Evaluating Matrix Functions for Exponential Integrators via Caratheodory-Fejer Approximation and Contour Integrals. *Electronic Transactions on Numerical Analysis* **29,** 1-18.

Shampine, L.F. and Gear, C.W. 1976: A User's View of Solving Ordinary Differential Equations. *Department of Computer Science University of Illinois at Urbana.*

Sheree, L.L. 2003: Semigroup of Linear Operators.

Spijker, M.N 1995: Stiffness in numerical initial-value problems. *Journal of Computational and Applied Mathematics,***72(1996),** 393-406.

Trefethen, L.N. and Kassam, A.K. 2005: Fourth-order time stepping for stiff PDEs . *SIAM J. Sci. Comput.,* **26(4),** 1214-1233.

APPENDIX A

MATLAB CODES FOR THE APPLICATIONS OF THE EXPONENTIAL INTEGRATORS

```
      %%EXPONENTIAL METHODS FOR PROTHERO ROBINSON EQUATION
clc clear all close all
format long tic
for e=1:4
A=[1 0; -10^2 -10^2]; N=10^e; h=1/N; step(e)=N;
t=0:h:1;
f1 = [-cos(t)-sin(t);(10^2)*(cos(t)+cos(2*t))-2*sin(2*t) ]
[V,D]=eig(h*A); d=diag(D);
IT(1:2,1)=[1;1]; IT1(1:2,1)=[1;1]; IT2(1:2,1)=[1;1];
for i=1:N
 IT(:,i+1)=expm(A*h)*IT(:,i)+h*V*diag(phi1(d,h,1))*inv(V)
   *[-cos(t(i))-sin(t(i));(10^2)*(cos(t(i))+cos(2*t(i)))
   -2*sin(2*t(i)) ]%exp.euler
 IT1(:,i+1)=expm(A*h)*IT1(:,i)
   +h*V*diag(phi1(d,h,1)-phi2(d,h,1))*inv(V)
   *[-cos(t(i))-sin(t(i));(10^2)*(cos(t(i))+cos(2*t(i)))
   -2*sin(2*t(i))]+h*V*diag(phi2(d,h,1))*inv(V)*[-cos(t(i+1))
   -sin(t(i+1));(10^2)*(cos(t(i+1))+cos(2*t(i+1)))
   -2*sin(2*t(i+1))];%2nd order
 IT2(:,i+1)=expm(A*h)*IT2(:,i)+h*V*diag(phi1(d,h,1))*inv(V)
   *[-cos(t(i))-sin(t(i));(10^2)*(cos(t(i))+cos(2*t(i)))
   -2*sin(2*t(i))]+0.5*(h^2)*V*diag(phi1(d,h,1))*inv(V)*
   [sin(t(i))-cos(t(i));(10^2)*(-sin(t(i))-2*sin(2*t(i)))
   -4*cos(2*t(i))];%midpoint
end
for i=1:N+1
ITex(:,i)=[cos(t(i)) cos(2*t(i))];
end
ITex
```

```
for i=1:N+1
errorex(i)=max(abs(IT(1,1:i)-ITex(1,1:i)));
errorex1(i)=max(abs(IT1(1,1:i)-ITex(1,1:i)));
errorex2(i)=max(abs(IT2(1,1:i)-ITex(1,1:i)));
end
errore=errorex(N+1);
errore1=errorex1(N+1);
errore2=errorex2(N+1);
u11(e)=norm(errore,inf) %%u1 için max. error
u21(e)=norm(errore1,inf)
u31(e)=norm(errore2,inf)
for i=1:N+1
errorex(i)=max(abs(IT(2,1:i)-ITex(2,1:i)));
errorex1(i)=max(abs(IT1(2,1:i)-ITex(2,1:i)));
errorex2(i)=max(abs(IT2(2,1:i)-ITex(2,1:i)));
end
errore=errorex(N+1);
errore1=errorex1(N+1);
errore2=errorex2(N+1);
u12(e)=norm(errore,inf) %%u2 için max. error
u22(e)=norm(errore1,inf)
u32(e)=norm(errore2,inf)
IT(1,:); IT(2,:); dt(e)=h;
if e>1
order11(e)=abs((log(u11(e)/u11(e-1)))/(log(dt(e)/dt(e-1))));
order21(e)=abs((log(u21(e)/u21(e-1)))/(log(dt(e)/dt(e-1))));
order31(e)=abs((log(u31(e)/u31(e-1)))/(log(dt(e)/dt(e-1))));
order12(e)=abs((log(u12(e)/u12(e-1)))/(log(dt(e)/dt(e-1))));
order22(e)=abs((log(u22(e)/u22(e-1)))/(log(dt(e)/dt(e-1))));
order32(e)=abs((log(u32(e)/u32(e-1)))/(log(dt(e)/dt(e-1))));
else
order11(e)=0; order21(e)=0; order31(e)=0;
order12(e)=0; order22(e)=0; order32(e)=0;
end
end
norm(A)
ordereuler=[order11;order12]
```

```
ordersecond=[order21;order22]
ordermidpoint=[order31;order32]
erroreuler=[(u11);(u12)]
errorsecond=[(u21);(u22)]
errormidpoint=[(u31);(u32)]
figure
subplot(1,2,1) ;plot(log10(step),log10(u11),'-r',
...'LineWidth',2)
hold all
plot(log10(step),log10(u21),'-m',...'LineWidth',2)
hold all
plot(log10(step),log10(u31),'-y',...'LineWidth',2)
hold all
xlabel('LOG(N.EVAL)')
ylabel('LOG(ERROR)')
title('Accuracy of u1 for three schemes')
hold off
subplot(1,2,2) ;
plot(log10(step),log10(u12),'-r',...'LineWidth',2)
hold all
plot(log10(step),log10(u22),'-m',...'LineWidth',2)
hold all
plot(log10(step),log10(u32),'-y',...'LineWidth',2)
hold all
xlabel('LOG(N.EVAL)')
ylabel('LOG(ERROR)')
title('Accuracy of u2 for three schemes')
hold off
legend('Exp.Euler','Exp.SecondOrder','Exp.Midpoint')
figure
subplot(1,2,1) ;plot(t,ITex(1,:),'-.r*',
...'LineWidth',2)
hold all
plot(t,IT(1,:),'--go',...'LineWidth',2)
hold all
plot(t,IT1(1,:),'-bs',...'LineWidth',2)
hold all
```

```
plot(t,IT2(1,:),'-.m+',...'LineWidth',2)
xlabel('t')
ylabel('Numerical Solutions of u1')
title('Comparison of solutions')
hold off
subplot(1,2,2) ;plot(t,ITex(2,:),'-.r*',
...'LineWidth',2)
hold all
plot(t,IT(2,:),'--go',...'LineWidth',2)
hold all
plot(t,IT1(2,:),'-bs',...'LineWidth',2)
hold all
plot(t,IT2(2,:),'-.m+',...'LineWidth',2)
xlabel('t')
ylabel('Numerical Solutions of u2')
title('Comparison of solutions')
hold off
legend('Exact','Exp.Euler','Exp.SecondOrder','Exp.Midpoint')
toc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%ONE DIMENSIONAL EXAMPLE
%y'=-50y+50cosx
clc clear all close all
x0=0; X=pi; N=10*pi;
h=(X-x0)/N;
x=x0:h:X;
X1=[0 pi/3 2*(pi/3) pi];
Y1=[1 0.5 -0.5 -1];
x1=0:h:pi;
X2=[0 pi/3 2*(pi/3) pi];
Y2=[0 -0.8660 -0.8660 0];
f1 = new_div_diff(X1, Y1, x1);
f2 = new_div_diff(X2, Y2, x1);
A=-50;
[V,D]=eig(h*A);
d=diag(D);
y(1)=0; y1(1)=0; y2(1)=0; y3(1)=0;
```

```
for i=1:N
y(i+1)=(1-h*(-50))\(y(i)+h*50*f1(i));%%imlicit euler
%y1(i+1)=y1(i)+h*((expm(-50*h)-1)/(-50*h))*
 (50*cos(x(i))-50*y1(i));%%exp eu
y1(i+1)=y1(i)+h*phi1(A*h)*(50*f1(i)-50*y1(i));
 %%expeuler with div. diff
%y2(i+1)=expm(A*h)*y2(i)+h*(phi1(d,h,1)-phi2(d,h,1))
 *50*cos(x(i))
 +h*(phi2(d,h,1)*50*cos(x(i+1)));%%2nd order
y2(i+1)=expm(A*h)*y2(i)+h*(phi1(d,h,1)-phi2(d,h,1))*50*f1(i)
 +h*(phi2(d,h,1)*50*f1(i+1));%%2nd order with div.dif
%y3(i+1)=expm(A*h)*y3(i)+h*V*diag(phi1(A*h))*inv(V)*50*cos(x(i))
 -0.5*(h^2)*(phi1(A*h)*50*sin(x(i+1)));%%midpoint
y3(i+1)=expm(A*h)*y3(i)+h*V*diag(phi1(A*h))*inv(V)*50*f1(i)
 +0.5*(h^2)*(phi1(A*h)*50*f2(i));%%midpoint with div. difference
end
y; y1; y2; y3;
for i=1:N+1
yex(i)=-(2500/2501)*exp(-50*x(i))+(2500/2501)*cos(x(i))
 +(50/2501)*sin(x(i));
end
yex;
for i=1:N+1
error1(i)=norm(abs(y1(i)-yex(i)),inf); %%%Linf - norm
error2(i)=norm(abs(y2(i)-yex(i)),inf); %%%Linf - norm
error3(i)=norm(abs(y3(i)-yex(i)),inf); %%%Linf - norm
error4(i)=norm(abs(y(i)-yex(i)),inf);  %%%Linf - norm
end
u1=max(error1); u2=max(error2);
u3=max(error3); u4=max(error4);
plot(x,yex,'-.r*',...
'LineWidth',2)%%% exact and numerical solution
hold all     %%%% y1:exponential euler
plot(x,y1,'--go',... %y2:exponential second order m.
'LineWidth',2)
hold all
plot(x,y2,'-bs',...
```

```
'LineWidth',2)
hold all
plot(x,y3,'-.m+',...
'LineWidth',2) %%y3:exponential midpoint
hold all
plot(x,y,'-y*') %%y:implicit euler
xlabel(' x ')
ylabel(' Solutions ')
title('Comparison of numerical solutions with exact solution')
hold off
legend('Exact Solution','Exp.Euler',
'Exp.Ostermann','Exp.Midpoint',
'Implicit.Euler')
%%Explicit Euler
eu(1)=0;
for i=1:N
eu(i+1)=eu(i)+h*(-50*eu(i)+50*f1(i));
end
eu;
%%Runge Kutta
rk(1)=0;
for j=1:N
a=h*(-50*rk(j)+50*f1(j));
b=h*(-50*(rk(j)+a/2)+50*(f1(j)+h/2));
c=h*(-50*(rk(j)+b/2)+50*(f1(j)+h/2));
d=h*(-50*(rk(j)+c)+50*(f1(j)+h)) ;
s=(a+2*(b+c)+d)/6;
rk(j+1)=rk(j)+s;
end
rk;
for i=1:N+1
erroreu(i)=norm(abs(eu(i)-yex(i)),inf);
errorrk(i)=norm(abs(rk(i)-yex(i)),inf);%Linf- norm
end
ereu=max(erroreu)
errk=max(errorrk)
figure
```

```
y=cos(x1);Divided Difference and cos
plot(x1,f1,'y')
hold all
plot(x1,y,'r')
legend('Divided difference of cos(x)','cos(x)')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Divided Difference Code
function y = new_div_diff(X, Y, x)
n = length(X);
if n ~= length(Y)
error('X and Y must be the same length.');
end
y = Y(1);
p = 1;
for i = 1:(n-1)
for j = 1:(n-i)
Y(j) = (Y(j+1) - Y(j))/(X(j+i) - X(j));
end
for k = i
p = p.*(x-X(i));
end
y = y + p.*Y(1)
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%TWO DIMENSIONAL EXAMPLE
for e=1:3
A=[-1 0; 2 -100];
N=5*2^(e-1) h=0.3/N; step(e)=N;
t=0:h:0.3;
X1=[0 0.1 0.2 0.3];
Y1=[1 1.1052 1.2214 1.3499];
x1=0:h:0.3;
f1 = new_div_diff(X1, Y1, x1);
[V,D]=eig(h*A);
d=diag(D);
IT(1:2,1)=[1;0];
IT1(1:2,1)=[1;0];
```

```
IT2(1:2,1)=[1;0];
for i=1:N
% IT(:,i+1)=expm(A*h)*IT(:,i)+h*V*diag(phi1(d,h,1))
%  *inv(V)*([f1(i);0])%%exp.euler with div. dif.
% IT1(:,i+1)=expm(A*h)*IT1(:,i)+h*V*diag(phi1(d,h,1)-phi2(d,h,1))
%  *inv(V)*[f1(i);0]+h*V*diag(phi2(d,h,1))*inv(V)*[f1(i+1);0];
%  %%second order with div. dif.
% IT2(:,i+1)=expm(A*h)*IT2(:,i)+h*V*diag(phi1(d,h,1))*inv(V)*
%  [f1(i);0]+0.5*(h^2)*V*diag(phi1(d,h,1))*inv(V)*[f1(i);0];
%  %%midpoint with div. dif.
 IT(:,i+1)=expm(A*h)*IT(:,i)
  +h*V*diag(phi1(d,h,1))*inv(V)*([exp(t(i));0]);%%euler
 IT1(:,i+1)=expm(A*h)*IT1(:,i)+h*V*diag(phi1(d,h,1)-phi2(d,h,1))
  *inv(V)*[exp(t(i));0]
  +h*V*diag(phi2(d,h,1))*inv(V)*[exp(t(i+1));0];%%2nd order
 IT2(:,i+1)=expm(A*h)*IT2(:,i)
  +h*V*diag(phi1(d,h,1))*inv(V)*[exp(t(i));0]
  +0.5*(h^2)*V*diag(phi1(d,h,1))*inv(V)*[exp(t(i));0];
  %%midpoint
end
for i=1:N+1
 ITex(:,i)=[0.5*exp(-t(i))+0.5*exp(t(i))(-200/(99*101))
  *exp(-100*t(i))+(1/99)*exp(-t(i))+(1/101)*exp(t(i))];
end
ITex
for i=1:N+1
errorex(i)=max(abs(IT(1,1:i)-ITex(1,1:i)));
errorex1(i)=max(abs(IT1(1,1:i)-ITex(1,1:i)));
errorex2(i)=max(abs(IT2(1,1:i)-ITex(1,1:i)));
end
errore=errorex(N+1);
errore1=errorex1(N+1);
errore2=errorex2(N+1);
u11(e)=norm(errore,inf)%%for u1 max. error
u21(e)=norm(errore1,inf)
u31(e)=norm(errore2,inf)
for i=1:N+1
```

```
errorex(i)=max(abs(IT(2,1:i)-ITex(2,1:i)));
errorex1(i)=max(abs(IT1(2,1:i)-ITex(2,1:i)));
errorex2(i)=max(abs(IT2(2,1:i)-ITex(2,1:i)));
end
errore=errorex(N+1); errore1=errorex1(N+1);
errore2=errorex2(N+1);
u12(e)=norm(errore,inf) %%for u2  max. error
u22(e)=norm(errore1,inf) u32(e)=norm(errore2,inf)
IT(1,:); IT(2,:); dt(e)=h;
if e>1
order11(e)=abs((log(u11(e)/u11(e-1)))/(log(dt(e)/dt(e-1))));
order21(e)=abs((log(u21(e)/u21(e-1)))/(log(dt(e)/dt(e-1))));
order31(e)=abs((log(u31(e)/u31(e-1)))/(log(dt(e)/dt(e-1))));
order12(e)=abs((log(u12(e)/u12(e-1)))/(log(dt(e)/dt(e-1))));
order22(e)=abs((log(u22(e)/u22(e-1)))/(log(dt(e)/dt(e-1))));
order32(e)=abs((log(u32(e)/u32(e-1)))/(log(dt(e)/dt(e-1))));
else
order11(e)=0; order21(e)=0; order31(e)=0;
order12(e)=0; order22(e)=0; order32(e)=0;
end
end
norm(A)
ordereuler=[order11;order12]
ordersecond=[order21;order22]
ordermidpoint=[order31;order32]
erroreuler=[(u11);(u12)]
errorsecond=[(u21);(u22)]
errormidpoint=[(u31);(u32)]
figure
subplot(1,2,1) ;plot(log10(step),log10(u11),'-r',...
'LineWidth',2)
hold all
plot(log10(step),log10(u21),'-m',...
'LineWidth',2)
hold all
plot(log10(step),log10(u31),'-y',...
'LineWidth',2)
```

```
hold all
xlabel('LOG(N.EVAL)') ylabel('LOG(ERROR)')
title('Accuracy of y1 for three schemes')
grid on hold off
subplot(1,2,2) ;
plot(log10(step),log10(u12),'-r',...'LineWidth',2)
hold all
plot(log10(step),log10(u22),'-m',...'LineWidth',2)
hold all
plot(log10(step),log10(u32),'-y',...'LineWidth',2)
hold all
xlabel('LOG(N.EVAL)') ylabel('LOG(ERROR)')
title('Accuracy of y2 for three schemes')
grid on hold off
legend('Exp.Euler','Exp.SecondOrder','Exp.Midpoint')
figure
subplot(1,2,1) ;plot(t,ITex(1,:),'-.r*',...
'LineWidth',2) %%exact and numerical solutions
hold all
plot(t,IT(1,:),'--go',...'LineWidth',2)
hold all
plot(t,IT1(1,:),'-bs',...'LineWidth',2)
hold all
plot(t,IT2(1,:),'-.m+',...'LineWidth',2)
xlabel('t')
ylabel('Numerical Solutions of y1')
title('Comparison of solutions')
hold off
subplot(1,2,2) ;plot(t,ITex(2,:),'-.r*',...
'LineWidth',2) %%exact and numerical solutions
hold all
plot(t,IT(2,:),'--go',...'LineWidth',2)
hold all
plot(t,IT1(2,:),'-bs',...'LineWidth',2)
hold all
plot(t,IT2(2,:),'-.m+',...'LineWidth',2)
xlabel('t')
```

```
ylabel('Numerical Solutions of y2')
title('Comparison of solutions')
hold off
legend('Exact','Exp.Euler','Exp.SecondOrder','Exp.Midpoint')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
REACTION-DIFFUSION EQUATIONS
%%LINEAR PROBLEM
for e=1:3
hx=0.01; x1=0; x2=1;
N=(x2-x1)/hx;
x=x1:hx:x2;
t1=0; t2=1; Nt=5*(2^(e-1)); ht=(t2-t1)/Nt;
step(e)=Nt;
t=t1:ht:t2;
X1=[0 0.25 0.5 0.75 1];
Y1=[1 1.2840 1.6487 2.117 2.7183];
divd = new_div_diff(X1, Y1,t)
g(1,1:N+1)=x.*(1-x); %%initial value
u(1:N-1,1)=(g(1,2:N))';
u2f(1:N-1,1)=(g(1,2:N))';
A=fin(N) %%u_xx in finite difference açılımı
AA=((1/hx)^2)*A;
m(1,1:N+1)=(2+x.*(1-x))
f(1:N-1,1)=(m(1,2:N))'
[V,D]=eig(ht*AA);
d=diag(D);
for i=1:Nt
u(:,i+1)=expm(AA*ht)*u(:,i)+ht*V*diag(phi1(d,ht,1))*inv(V)
 *f(1:N-1,1)*divd(i);%%exp.euler  divided diff
u11(:,i+1)=expm(AA*ht)*u11(:,i)+ht*V*diag(phi1(d,ht,1)
 -phi2(d,ht,1))*inv(V)*f(1:N-1,1)*divd(i)
 +ht*V*diag(phi2(d,ht,1))*inv(V)*f(1:N-1,1)*divd(i+1);%%2nd order
u22(:,i+1)=expm(AA*ht)*u22(:,i)+ht*V*diag(phi1(d,ht,1))*inv(V)
 *f(1:N-1,1)*divd(i)+0.5*(ht^2)*V*diag(phi1(d,ht,1))
 *inv(V)*f(1:N-1,1)*divd(i); %%midpoint
end
u; u11; u22;
```

```
for i=1:N+1
for j=1:Nt+1
 y(i,j)=x(i)*(1-x(i))*exp(t(j));%%exact
end
end
y;
s=y(:,Nt+1)'; v1(:,1:Nt+1)=0; v2(:,1:Nt+1)=0;
k1=vertcat(v1,u,v2) as1=k1(:,Nt+1)'
k2=vertcat(v1,u11,v2) as2=k2(:,Nt+1)'
k3=vertcat(v1,u22,v2) as3=k3(:,Nt+1)'
errorinf1=norm(s-as1,inf); errorinf2=norm(s-as2,inf);'
errorinf3=norm(s-as3,inf); '
u1(e)=max(errorinf1); u2(e)=max(errorinf2);'
u3(e)=max(errorinf3); dt(e)=ht;'
if e>1'
order1(e)=abs((log(u1(e)/u1(e-1)))/(log(dt(e)/dt(e-1))));'
order2(e)=abs((log(u2(e)/u2(e-1)))/(log(dt(e)/dt(e-1))));'
order3(e)=abs((log(u3(e)/u3(e-1)))/(log(dt(e)/dt(e-1))));'
else'
order1(e)=0; order2(e)=0; order3(e)=0;'
end'
ht'
end'
erroreuler=u1 errorsecond=u2 errormidpoint=u3'
ordereuler=order1 ordersecond=order2 ordermidpoint=order3'
figure'
plot(log10(step),log10(u1),'-.m*',...'
'LineWidth',2)'
hold all'
plot(log10(step),log10(u2),'-go',...'
'LineWidth',2)'
hold all'
plot(log10(step),log10(u3),'-bs',...'
'LineWidth',2)'
hold all'
grid on'
xlabel('LOG(N.EVAL)')'
```

```
ylabel('LOG(ERROR)')'
title('Accuracy of u for three schemes')'
hold off'
legend('Exp.Euler','Exp.SecondOrder','Exp.Midpoint')'
figure'
surf(x, t, k1', ... 'FaceColor','interp',...
'EdgeColor','none',... 'FaceLighting','phong')
axis tight
view(-50,30)    camlight left    alpha(0.6);
xlabel('space'); ylabel('time'); zlabel('u(t,x)');
title('Exponential Method Solution')
figure ; mesh(x,t,y')' xlabel('x');
ylabel('time'); zlabel('u(t,x)');'
title(' Solution of linear parabolic equation') '
s=y(:,Nt+1)';'
error1=norm(s-as1,1);'
error2=norm(s-as1,2);'
errorinf=norm(s-as1,inf);'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%u_xx in finite difference açılımı
function A=fin(N)
%A=zeros(N-1,N-1);
for i=1:N-1
    for j=1:N-1
        if i==j
            A(i,j)=-2;
        end
        if (i-j)==1
            A(i,j)=1;
        end
        if (i-j)==-1
            A(i,j)=1;
        end
    end
end
A;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%SEMILINEAR PROBLEMS-FISHER EQUATION
clear all  close all clc tic
for e=1:5
N=40; hx=20/N; x=-10:hx:10; Nt=20*e;
step(e)=Nt; ht=1/Nt; t=0:ht:1;
A=((1/hx)^2)*fin(N); a=0.1; b=1;
f(1,:)=(sech(x)).^2;%%initial condition
u(1:N-1,1)=(f(1,2:N))'; u2(1:N-1,1)=u(1:N-1,1);
for i=1:Nt
J=a*A+b*eye(N-1)-2*b*diag(u(1:N-1,i));
[V,D]=eig(ht*J); d=diag(D);
gb(:,i)=a*A*u(1:N-1,i)+b*(u(:,i)-(u(:,i)).^2)-(a*A*u(1:N-1,i)
 +b*eye(N-1)*u(1:N-1,i)-2*b*u(1:N-1,i).^2)
u(1:N-1,i+1)=expm(J*ht)*u(:,i)
 +ht*V*diag(phi1(d,ht,1))*inv(V)*gb(:,i);%%exp.euler
end
v1(:,1:Nt+1)=0; v2(:,1:Nt+1)=0; k3=vertcat(v1,u,v2);
figure ;
mesh(t,x,k3);
xlabel('t') ylabel('x')
zlabel('Numerical Solution')
title('Rosenbrock Euler Method Solution of Fisher Equation')
for i=1:Nt/2
 u2(1:N-1,i+1)=u(1:N-1,2*i+1);
end
N1=Nt/2; v3(:,1:Nt/2+1)=0; v4(:,1:Nt/2+1)=0;
k1=vertcat(v3,u2,v4);
t=0:2*ht:1; f1(1,1:N+1)=(sech(x)).^2;
u1(:,1)=(f1(1,2:N))';
A=((1/(hx))^2)*fin(N);
for i=1:N1
 J=a*A+b*eye(N-1)-2*b*diag(u1(1:N-1,i));
 [V,D]=eig(2*ht*J);
 d=diag(D);
 g1(:,i)=a*A*u1(1:N-1,i)+b*(u1(:,i)-(u1(:,i)).^2)-
  (a*A*u1(1:N-1,i)+b*eye(N-1)*u1(1:N-1,i)-2*b*u1(1:N-1,i).^2)
 u1(1:N-1,i+1)=expm(J*2*ht)*u1(:,i)+
```

```
  2*ht*V*diag(phi1(d,2*ht,1))*inv(V)*g1(:,i);%%exp.euler
end
k2=vertcat(v3,u1,v4);
ref=(2*k1-k2);
er1=norm(abs(ref(:,N1+1)-k1(:,N1+1)),1)
er2=norm(abs(ref(:,N1+1)-k1(:,N1+1)),2)
erinf=norm(abs(ref(:,N1+1)-k1(:,N1+1)),inf)
u11(e)=max(er1); u22(e)=max(er2); u33(e)=max(erinf);
dt(e)=ht; if e>1
order1(e)=abs((log(u11(e)/u11(e-1)))/(log(dt(e)/dt(e-1))));
order2(e)=abs((log(u22(e)/u22(e-1)))/(log(dt(e)/dt(e-1))));
order3(e)=abs((log(u33(e)/u33(e-1)))/(log(dt(e)/dt(e-1))));
else
order1(e)=0; order2(e)=0; order3(e)=0;
end ht
end
figure
plot(log10(step),log10(u11),'-.m*',...'LineWidth',2)
hold all
plot(log10(step),log10(u22),'-.g*',...'LineWidth',2)
hold all
plot(log10(step),log10(u33),'-.r*',...'LineWidth',2)
grid on
xlabel('LOG(N.EVAL)') ylabel('LOG(ERROR)')
legend('L-1 norm','L-2 norm','L-inf norm')
title('Accuracy of u for exponential Rosenbrock scheme ')
figure
plot(x,k3(:,11),'-.m',...'LineWidth',2)
hold all
plot(x,k3(:,51),'-.g',...'LineWidth',2)
hold all
plot(x,k3(:,end),'-.r',...'LineWidth',2)
xlabel('x-axis ')
ylabel('Computed Solutions')
title('Computed solutions of Fisher equation
for different values of time')
hold off
```

```
legend('t=0.1','t=0.5','t=1')
erroreuler1=u11 erroreuler2=u22 erroreuler3=u33
ordereuler1=order1 ordereuler2=order2 ordereuler3=order3
toc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%ALLEN CAHN EQUATION
clear all close all clc
for e=1:3
N=40; hx=(2)/N; x=-1:hx:1; Nt=100*2^(e-1);
step(e)=Nt; ht=10/Nt; t=0:ht:10;
A=((1/hx)^2)*fin(N); a=0.001; b=1;
f(1,1:N+1)=0.53*x+0.47*sin(-1.5*pi*x);%%I.C.
u(1:N-1,1)=(f(1,2:N))';
for i=1:Nt
 J=a*A+eye(N-1)-3*diag(u(1:N-1,i).^2)
 [V,D]=eig(ht*J); d=diag(D);
 g(:,i)=a*A*u(:,i)+(u(:,i)-(u(:,i)).^3)-(J*u(:,i));
 u(1:N-1,i+1)=expm(J*ht)*u(:,i)
  +ht*V*diag(phi1(d,ht,1))*inv(V)*g(:,i);%%exp.euler
end
v1(:,1:Nt+1)=-1; v2(:,1:Nt+1)=1; k3=vertcat(v1,u,v2)
figure
surf(x, t, k3', ...'FaceColor','interp',...
'EdgeColor','none',...'FaceLighting','phong')
axis tight view(-50,30) camlight left alpha(0.6);
xlabel('x'); ylabel('time'); zlabel('u(x,t)');
title('Rosenbrock Euler Method Solution of Allen Cahn Equation')
u2(1:N-1,1)=u(1:N-1,1);
for i=1:Nt/2
    u2(1:N-1,i+1)=u(1:N-1,2*i+1);
end
v3(:,1:Nt/2+1)=-1; v4(:,1:Nt/2+1)=1;
k1=vertcat(v3,u2,v4) N1=Nt/2; t=0:2*ht:1;
f1(1,1:N+1)=0.53*x+0.47*sin(-1.5*pi*x);
u1(:,1)=(f1(1,2:N))'; A=((1/(hx))^2)*fin(N)
for i=1:N1
 J=a*A+eye(N-1)-3*diag(u1(1:N-1,i).^2)
```

```
 [V,D]=eig(2*ht*J); d=diag(D);
 g1(:,i)=a*A*u1(:,i)+u1(:,i)-(u1(:,i)).^3-J*u1(:,i);
 u1(1:N-1,i+1)=expm(J*2*ht)*u1(:,i)
  +2*ht*V*diag(phi1(d,2*ht,1))*inv(V)*g1(:,i);%%exp.euler
end
v3(:,1:Nt/2+1)=-1; v4(:,1:Nt/2+1)=1;
k2=vertcat(v3,u1,v4) ref=(2*k1-k2)
er1=norm(abs(ref(:,N1+1)-k1(:,N1+1)),1)
er2=norm(abs(ref(:,N1+1)-k1(:,N1+1)),2)
erinf=norm(abs(ref(:,N1+1)-k1(:,N1+1)),inf)
u11(e)=max(er1); u22(e)=max(er2); u33(e)=max(erinf);
dt(e)=ht;
if e>1
order1(e)=abs((log(u11(e)/u11(e-1)))/(log(dt(e)/dt(e-1))));
order2(e)=abs((log(u22(e)/u22(e-1)))/(log(dt(e)/dt(e-1))));
order3(e)=abs((log(u33(e)/u33(e-1)))/(log(dt(e)/dt(e-1))));
else
order1(e)=0; order2(e)=0; order3(e)=0;
end
ht
end
figure
plot(log10(step),log10(u11),'-.m*',...'LineWidth',2)
hold all
plot(log10(step),log10(u22),'-.g*',...'LineWidth',2)
hold all
plot(log10(step),log10(u33),'-.r*',...'LineWidth',2)
grid on
xlabel('LOG(N.EVAL)') ylabel('LOG(ERROR)')
legend('L-1 norm','L-2 norm','L-inf norm')
title('Accuracy of u for exponential Rosenbrock scheme ')
figure
plot(x,k3(:,11),'-.m',...'LineWidth',2)
hold all
plot(x,k3(:,21),'-.g',...'LineWidth',2)
hold all
plot(x,k3(:,61),'-.b',...'LineWidth',2)
```

```
hold all
plot(x,k3(:,end),'-.r',...'LineWidth',2)
xlabel('x-axis ')
ylabel('Computed Solutions')
title('Computed solutions of Allen equation
for different values of time')
hold off
legend('t=1','t=2','t=6','t=10')
erroreuler1=u11 erroreuler2=u22 erroreuler3=u33
ordereuler1=order1 ordereuler2=order2 ordereuler3=order3
```