May 2018

Numerical Solutions of Fractional Nonlinear Advection-Reaction-Diffusion Equations

Sophia Vorderwuelbecke
University of Wisconsin-Milwaukee

Numerical Solutions of Fractional Nonlinear Advection-Reaction-Diffusion Equations

by

Sophia Vorderwülbecke

A Thesis Submitted in
Partial Fulfillment of the
Requirements for the Degree of

Master of Science
in
Mathematics

at

The University of Wisconsin-Milwaukee
May 2018

ABSTRACT

Numerical Solutions of Fractional Nonlinear Advection-Reaction-Diffusion Equations

by

Sophia Vorderwülbecke

The University of Wisconsin-Milwaukee, 2018
Under the Supervision of Professor Bruce A. Wade

In this thesis nonlinear differential equations containing advection, reaction and diffusion terms are solved numerically, where the diffusion term is modelled by a fractional derivative. One of the methods employed is a finite difference method for temporal as well as spatial discretization. Furthermore, exponential time differencing schemes under consideration of different matrix exponential approximations are exploited for the temporal discretization, whereas finite differences are used for the spatial approximation. The schemes are applied to the homogeneous Burgers, Burgers-Fisher and Burgers-Huxley equation and compared with respect to convergence and efficiency in a numerical investigation.

Table of Contents

List of Figures

Motivation

1.1 Partial Differential Equations (PDEs)

Partial differential equations are used to model a variety of physical phenomena, such as heat transfer, fluid dynamics or quantuum mechanics. In applications there are usually relationships between rates of change and some variables or a conservation. Solving those equations, either analytically or numerically, results usually in good predictions for the given physical system.

If the relationship is nonlinear in some variables, the equation is called a nonlinear PDE. This nonlinearity implies a higher complexity of the problem and usually inadequate theoretical support. With that the equations are harder to solve analytically - in many cases which involve PDEs it is not possible at all. Following from that, it is not as easy to generalize from specific applications and to work out a more abstract theory as it is for linear PDEs, and therefore numerical methods become crucial tools to get solutions of nonlinear PDEs.

1.2 Fluid Dynamics

In this thesis, the main focus will be on nonlinear PDEs related to fluid dynamics. Fluids can be classified as gases or liquids, which is why common applications of fluid PDEs are in the context of aero- and hydrodynamics, for example, in models of ocean movement, wind turbines or jet engines.

An important assumption to model fluids is that they are continous, even if they consist of molecules. This can be justified if a 'macro-scale' approach is used, where so-called fluid parcels contain a huge amount of molecules, but mathematically represent an infinitesimal volume with constant properties. Further characterizations help to model the flow of the fluids properly. Some of them are listed in the following.

- (in)compressible: constant/not constant density of the fluid parcels
- (in)viscous: viscosity/diffusion is (not) considered because viscous forces[1] are (not) dominating inertial forces[2] indicated by a low (high) Reynolds number[3]
- (un)steady: fluid properties are not time-dependent (time-dependent)
- laminar/turbulent: fluid moves smoothly/irregular fluctuations in fluid flow
- (non-)Newtonian: stress[4] has (no) linear relationship to strain[5]

1.2.1 Navier-Stokes Equation

A wide range of fluid flows is governed by the Navier Stokes equations. The momentum in convective form for Newtonian flows is given by

$$\rho\left(\frac{\partial u}{\partial t} + u \cdot \nabla u\right) = -\nabla \bar{p} + \mu \nabla^2 u + \frac{\mu}{3}\nabla(\nabla \cdot u) + \rho g, \tag{1.2.1}$$

where u denotes the fluid velocity, ρ the density, $\bar{p}$ the mechanical pressure, μ the dynamic viscosity and g an external source, for example gravity. Moreover, the mechanical pressure $\bar{p}$ can be expressed in terms of the thermodynamical pressure p and volume viscosity ξ by $\bar{p} = p - \xi \nabla \cdot u$.

[1] $dynamic\ viscosity \cdot velocity/distance \cdot area$
[2] $mass \cdot acceleration$
[3] helpful to predict flow patterns and defined by the ratio of inertial to viscious forces
[4] tensile or compressive loading ($force/area$)
[5] amount of deformation ($length\ of\ deformation/original\ length$)

The operators acting on u are the gradient ∇, divergence $\nabla\cdot$ and Laplace operator ∇^2. The nonlinearity, introduced by the advection term $u \cdot \nabla u$ and the pressure term, is the main reason for the modelled fluid velocity solution being hard to find. It is actually one of the Millenium Prize Problems to prove that there exist global, smooth solutions to the Navier Stokes equation.

The Navier-Stokes equation in the form given in (1.2.1) can be rewritten as

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{1}{\rho}\nabla \bar{p} + \frac{\mu}{\rho}\nabla^2 u + \frac{\mu}{3\rho}\nabla(\nabla \cdot u) + g. \tag{1.2.2}$$

Incompressibility means, that the density flow is constant, so mathematically, incompressibility results in a non-existent divergence of the flow velocity $\nabla \cdot u = 0$ and therefore, the mechanical becomes equivalent to the thermodynamical pressure. Furthermore, in case of homogeneous fluids, which are characterized by uniform density, the pressure term $\frac{1}{\rho}\nabla p$ can be expressed as thermodynamic work ∇w, which is also called an internal source. Moreover, we define the kinematic viscosity $v = \frac{\mu}{\rho}$. With that, the equation (1.2.2) can be simplified to

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = v\nabla^2 u - \nabla w + g. \tag{1.2.3}$$

1.2.2 Burgers-Huxley's and Burgers-Fisher's Equation

Dropping the source terms of the incompressible Navier-Stokes equation (1.2.3) leads to the Burgers equation

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - v\nabla^2 u = 0. \tag{1.2.4}$$

The equation is named after Johannes Martinus Burgers, who developed the equation as a nonlinear extension of the heat equation by incorporation of the advection term $u \cdot \nabla u$ [Burgers, 1948].

The equation (1.2.4) is the viscid Burgers equation, whereas, in case of an absent diffusion term, it is called inviscous. In the latter case discontinuities in the solution can occur, which are related to shock waves, see [contributors, 2018]. In contrast to the Navier-Stokes equation, it is possible to linearize the equation (1.2.4) to the linear parabolic heat equation with help of the Cole-Hopf transformation [Hopf, 1950], and then to solve it exactly

The equation (1.2.4) is in its homogeneous form, whereas adding an external force field $F(u)$ called reaction term to the right hand side yields the inhomogeneous Burgers equation

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - v \nabla u = F(u). \tag{1.2.5}$$

Let the external source be $F(u) = uf_i(u)$ to consider mild nonlinear reactions. The function f_i could be, for example, Huxley's reaction $f_1(u) = (1-u)(u-\gamma)$ or Fisher's reaction law $f_2(u) = 1 - u$. The standard Fisher's equation (also called Kolmogorov-Petrovsky-Piskunov (KPP) equation) is a semi-linear reaction-diffusion equation with f_2 as reaction term, but without the advection term as in Burgers' equation. It was published by the biologist Ronald Fisher in the year 1937 in application to population dynamics [Fisher, 1937]. Huxley's reaction term was also first considered in the context of biology in 1952, to be specific, in the Hodgin-Huxley model to simulate neural action potentials[6] [Hodgkin and Huxley, 1952]. The combination of f_1 or f_2 and Burgers' equation produces traveling wave solutions, where successive curves of the solutions are displaced by a constant distance in the temporal variable t. In other words, the wave does not change its shape along the axis of the spatial variable x, see [Griffiths and Schiesser, 2010].

[6] Neural action potentials, also called nerve impulses, are defined by fast changing membrane potentials of an axon in a cell. Their importance lies in cell-to-cell communication. In that context, a neuron emitting a potential is commonly known as 'fire'.

1.2.3 Space-Fractional Burgers' Equation

Space-fractional calculus is used to extend existing models with the intention of more applicability. Equations can be improved by including non-local properties by substituting a regular derivative by a fractional derivative denoted by $\Delta^{\alpha/2}$. The non-locality implies that the solution of (1.2.6) in a certain configuration is not only influenced by the diffusive properties of the solutions at this point, but also from the surrounding environment. The fractional differentiation could be described as 'continuous differentiation', since additionally non-integer orders of derivatives are considered. In the space-fractional Burgers-Huxley/Burgers-Fisher equation the non-locality is modelled in the diffusion term.

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - v\Delta^{\alpha/2} u = F(u) \tag{1.2.6}$$

1.3 Numerical Methods for Nonlinear PDEs

As already mentioned, nonlinear PDEs seldomly have analytical solutions and the fractional extension makes them even more complicated to solve. Therefore, there is a great need of numerical methods. Unfortunately, in contrast to the broad theory regarding approximations of linear PDE solutions, scientists are still working to understand nonlinear ones. Therefore, most methods are developed for specific equations, but not for general types.

Possible general ideas to obtain analytic solutions for nonlinear PDEs could be to make use of symmetries, an ansatz or other tranformations resulting in reduced equations. Indeed, it is possible to convert the non-fractional Burgers equation (nonlinear) to the heat equation (linear) with help of the Cole-Hopf transformation by using a nonlinear substitution. If there is no reduction of the general equation possible, it is helpful to make assumptions, for example induced by real-life constraints, and consider a more simple equation, as for example Burgers' equation instead of the Navier-Stokes equation.

Traveling wave analytical solutions, which for example solve the Burgers-Huxley and Burgers-Fisher equation, can be calculated with an associated traveling wave method, as direct integration, factorization or expansion, or residual function method, where in the first step a solution is assumed, see [Griffiths and Schiesser, 2010].

Generally, numerical methods to solve nonlinear PDEs could be the method of lines, which transforms a space-time PDE into an ordinary differential equation (ODE), by employing discretization techniques for the spatial operators like finite differences, volume and element methods.

By focussing on a special type of equation which involves advection, reaction and diffusion terms (ARD equations), some theoretical results could be gathered. Especially, in case of pure advection or diffusion equations a more general theory has been established regarding positivity of the results and also different types of discretizations have been applied (for example, finite differences or flux-limiting), as can be seen in [Hundsdorfer and Verwer, 1996]. For equations where advection, reaction and diffusion are combined, splitting schemes have been developed, such a implicit-explicit (IMEX), on which exponential time differencing schemes are based, or alternating direction implicit (ADI), on which Rosenbrock methods rely (see [Hundsdorfer and Verwer, 1996]). On top of that, a backward differentiation formula (BDF) method applicable to general parabolic problems has been developed (see [Vigo-Aguiar et al., 2007]).

Over the years exponential time differencing schemes became popular and more and more publications arised. Some of them included Padé approximations of matrix exponentials (see [Khaliq et al., 2009], [Janssen, 2009], [Kleefeld et al., 2012] and [Yousuf et al., 2012]) and also a speed up of these methods was developed, see [Asante-Asamani and Wade, 2016].

In contrast to that, some years later the matrix exponentials were approximated by real distinct poles, as in [Asante-Asamani et al., 2016] in application to reaction-diffusion systems and [Iyiola and Wade, 2017], [Iyiola et al., 2017] in application to fractional reaction-diffusion models. Furthermore, new investigations will be published soon regarding methods for systems of nonlinear space-fractional models with super-diffusion processes in pattern formation in [Iyiola and Wade, 2018].

The numerical methods used in this thesis are generally applicable to nonlinear advection-reaction-diffusion PDEs of fractional order. The focus here is set on the homogeneous Burgers, Burgers-Huxley and Burgers-Fisher equation. A finite difference scheme is explained, which uses finite differences for the spatial as well as temporal approximation of the fractional Burgers equations with non-smooth initial conditions. Furthermore, two exponential time differencing schemes including a Padé and real distinct poles approximation for the matrix exponentials are exploited. A numerical investigation is performed to compare the methods with respect to convergence and efficiency.

Mathematical Introduction

In this chapter various mathematical foundations regarding fractional calculus and approximations, used at different stages of the thesis are mentioned. They play an important role in understanding the numerical methods and results explained in the following chapters.

2.1 Fractional Calculus

As already mentioned in the introductory chapter, fractional calculus, can be understood as expansion of integer-order to non-integer order integration and differentiation. Since this results in a more complex theory, there is the need of special functions, on which the theory can be built on.

2.1.1 Gamma Function

One of the special functions, which is essential to know in fractional calculus is the Gamma function. This function expands factorials $z!$ in the way, that z could be complex.

Definition 2.1.1. *Let $z \in \mathbb{C}$, then the Gamma function is defined as*

$$\Gamma(z) := \int_0^\infty e^{-t} t^{z-1} dt.$$

Basic properties of the Gamma function are the recursive and the limit definition, needed for a more convenient way to calculate frational centered differences later.

Lemma 2.1.2. *Properties of the Gamma function*

$$\Gamma(z+1) = z\Gamma(z)$$
$$\Gamma(z) = \lim_{n\to\infty} \frac{n!n^z}{z(z+1)\dots(z+n)} \qquad \text{, where } Re(z) > 0$$

Another important function for regular calculus is the exponential function, which is abstracted to fractional calculus by Mittag-Leffler's definition.

Definition 2.1.3. *Let $\beta, z \in \mathbb{C}$ and $Re(\alpha) > 0$, then the Mittag-Leffler function regarding one/two parameters is definded as*

$$E_\alpha(z) := \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + 1)}$$
$$E_{\alpha,\beta}(z) := \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + \beta)}.$$

2.1.2 Fractional Derivatives

Fractional derivatives are generally defined as an integer-order derivative of fractionally integrated term. In the following, let ${}_aD_x^\alpha f(x)$ denote the Riemann-Liouville and $D^\alpha f(x)$ the Riesz fractional derivative of order α. Let $a, b \in \mathbb{R}$ and $-\infty \le a < x < b \le \infty$.
For the sake of completeness, the general fractional Riemann-Liouville derivative is given by the following definition.

Definition 2.1.4. *Riemann-Liouville*

$${}_aD_x^\alpha f(x) := \left(\frac{d}{dx}\right)^{m+1} \int_a^x (x-\xi)^{m-\alpha} f(\xi) d\xi, \text{ where } m \in \mathbb{N}, m \le \alpha < m+1$$

For the purposes of this thesis the following definition is sufficient.

Proposition 2.1.5. *Left/Right-sided Riemann-Liouville*

Let $\Omega = [a,b] \times [0,T]$, $T > 0$ *and* $1 < \alpha \leq 2$. *Moreover, assume* $u(x,t) = 0$ *forall* $x \notin [a,b]$ *and* $t \in [0,T]$. *Under these assumptions the left- and right-sided space-fractional Riemann-Liouville derivatives become*

$$
\begin{aligned}
{}_{-\infty}D_x^{\alpha} &= \frac{1}{\Gamma(2-\alpha)} \frac{\partial^2}{\partial x^2} \int_{-\infty}^{x} \frac{u(\xi,t)}{(x-\xi)^{\alpha-1}} d\xi \\
{}_{x}D_{+\infty}^{\alpha} &= \frac{1}{\Gamma(2-\alpha)} \frac{\partial^2}{\partial x^2} \int_{x}^{\infty} \frac{u(\xi,t)}{(\xi-x)^{\alpha-1}} d\xi.
\end{aligned}
$$

In the numerical methods explained in a later chapter there is made use of the Riesz fractional derivative defined by (2.1.7) under the same assumptions as for the left/right-sided Riemann-Liouville propostion. The main disadvantage of the Riemann-Liouville derivative is that the fractional derivative of a constant is not zero, see [Podlubny, 1998].

Definition 2.1.6. *Riesz*

$$
D^{\alpha} u(x,t) := \frac{-1}{2\cos(\frac{\pi\alpha}{2})\Gamma(2-\alpha)} \frac{\partial^2}{\partial x^2} \int_{-\infty}^{\infty} \frac{u(\xi,t)}{|x-\xi|^{\alpha-1}} d\xi \qquad \forall (x,t) \in \Omega. \tag{2.1.7}
$$

The relationship between Riesz and Riemann-Liouville fractional derivatives is given by

$$
D^{\alpha} u(x,t) = -\frac{1}{2\cos(\pi\alpha/2)} ({}_{-\infty}D_x^{\alpha} + {}_x D_{+\infty}^{\alpha}) u(x,t).
$$

2.1.3 Spatial Discretization of Fractional Derivatives

In case of the Riesz' fractional derivative one way of spatial discretization is using fractional centered differences. Another possible method would be the matrix transfer technique, see [Podlubny, 1998].

Definition 2.1.8. *Fractional centred differences*

Let $f : \mathbb{R} \to \mathbb{R}, f \in L^1(\mathbb{R}), h > 0$ and $\alpha > -1$, then α-th order fractional centred differences of f at point x is defined as

$$\Delta_h^\alpha f(x) := \sum_{k=-\infty}^{\infty} g_k^\alpha f(x - kh), \forall x \in \mathbb{R}.$$

The coefficients are given by

$$g_k^\alpha := \frac{(-1)^k \Gamma(\alpha + 1)}{\Gamma(\alpha/2 - k + 1)\Gamma(\alpha/2 + k + 1)}$$

The coefficicients g_k^α can be computed in a more convenient way by

$$\begin{aligned} g_0^\alpha &:= \frac{\Gamma(\alpha + 1)}{\Gamma(\alpha/2 + 1)^2} \\ g_{k+1}^\alpha &= \left(1 - \frac{\alpha + 1}{\alpha/2 + k + 1}\right) g_k, \forall k \in \mathbb{N} \cup \{0\} \end{aligned}$$

Let $\Omega = [a, b] \times [0, T]$, $T > 0$ and $u \in \mathcal{C}^5(\Omega)$. Moreover, we assume $u(x, t) = 0$ forall $x \notin [a, b]$ and $t \in [0, T]$. For our purpose of $1 < \alpha \leq 2$ the following is satisfied (see [Ortigueira, 2006])

$$\lim_{h \to 0} \frac{1}{h^\alpha} \Delta_h^\alpha u(x, t) = \lim_{h \to 0} \frac{1}{h^\alpha} \sum_{k=-\infty}^{\infty} g_k^\alpha u(x - kh, t) = D^\alpha u(x, t) + \mathcal{O}(h^2), \forall (x, t) \in \Omega.$$

In addition, it is good to know, that g_0^α is positive, while all other coefficients are negative. Furthermore, the coefficients are symmetric around g_0^α and the sum of all coefficients is zero.

2.2 Approximation of Matrix Exponentials

In a following chapter a semi-discretized time stepping scheme will be explained, which contains matrix exponentials. Since direct computation of these exponentials consumes a lot of time, there are at least two possible discretization techniques considered to increase efficiency.

2.2.1 Padé Approximation

The first possibility is to use an approximation published by Henri Padé in 1892, see [Padé, 1892].

Definition 2.2.1. *Let a function f be represented by a power series $f(x) := \sum_{l=0}^{\infty} c_l x^l$, where $c_0 \neq 0$. Then the (m,n)th-entry in the Padé-table is*

$$R_{m,n}(x) := \frac{P_{m,n}(x)}{Q_{m,n}(x)},$$

where $P_{m,n}(x)$ and $Q_{m,n}(x)$ are power series, cut off at m and n, respectively.

This is a very general definition working for all analytic functions. By considering the power series representation of the exponential function $e^{-x} = \sum_{k=0}^{\infty}(-x)^k/k!$ the following quotients can be obtained, see [Janssen, 2009].

Proposition 2.2.2. *Matrix exponential approximation with Padé*

$$P_{m,n}(-x) := \sum_{j=0}^{m} \frac{(m+n-j)!m!}{(m+n)!j!(m-j)!}(-x)^j$$
$$Q_{m,n}(-x) := \sum_{j=0}^{n} \frac{(m+n-j)!n!}{(m+n)!j!(n-j)!}(-x)^j$$

The error of this approximation is $R_{m,n}(-x) = e^{-x} + \mathcal{O}(|x|^{m+n+1})$ as $x \to 0, x \in \mathbb{C}$. The Padé-schemes which will be used later, are

$$R_{0,1}(-kA) := (I + kA)^{-1} \text{ and } R_{1,1}(-kA) := (2I - kA)(2I + kA)^{-1}.$$

The first one is locally second order convergent, while the second one is second order convergent.

2.2.2 Real Distinct Poles (RDP) Approximation

The second approach is to use a function $R(x)$ of the form (2.2.4), instead of $R_{m,n}(x)$ as in the Padé approximation. This time the denominator has a different form. The idea in the context of RDP approximations for ETD schemes is to determine the coefficients to approximate the Mittag-Leffler function in an appropriate way.

Definition 2.2.3. *RDP approximation: Let $a_1, a_2, a_3 \in \mathbb{R}$ and $a_2 \neq a_3$, then a real distinct poles approximating function has the following form:*

$$R(x) := \frac{1 + a_1 x}{(1 - a_2 x)(1 - a_3 x)} \tag{2.2.4}$$

By considering the relationships

$$a_1 + a_2 + a_3 = \frac{\Gamma(\beta)}{\Gamma(\alpha + \beta)} \qquad a_1 - a_2 a_3 = \frac{\Gamma(\beta)}{\Gamma(2\alpha + \beta)} - \frac{\Gamma(\beta)}{\Gamma(\alpha + \beta)} a_2$$

for the coefficients, a second order approximation for the Mittag-Leffler function can be obtained, i.e. $R(x) - \Gamma(\beta) E_{\alpha,\beta}(x) = Cx^3 + \mathcal{O}(x^4)$, as $x \to 0$, where C is an error constant. For Proof and formula of the error constant see [Iyiola and Wade, 2017], [Iyiola et al., 2018], [Voss and Khaliq, 1995]. To sharply design the coefficients one property of the approximation, called L-acceptance, is introduced at this point.

Definition 2.2.5. *L-acceptance: R(x) is said to be L-acceptable, if $|R(x)| < 1$ for $Re(x) < 0$ (A-acceptable) and additionally, $|R(x)| \to 0$ for $Re(x) \to -\infty$.*

Specifically for this thesis, the approximation for $E_{1,1}(-x) = e^{-x}$ is of utmost interest. In that case L-acceptance is immediately apparent, since we know, the exponential is smaller than one for negative x and also, the smaller x the closer is the exponential to zero. As can be seen in [Asante-Asamani et al., 2016], this leeds to the following coefficients and final, almost optimal approximation of the exponentials (see [Voss and Khaliq, 1995]).

Proposition 2.2.6. *RDP approximation for exponentials*

$$R(-x) = E_{1,1}(-x) = e^{-x} \approx \frac{1 - \frac{5}{12}x}{(1 + \frac{1}{4}x)(1 + \frac{1}{3}x)}, \text{ with the error constant } C = 0.041\bar{6}$$

Furthermore, an essential feature of the RDP approximaton is, that partial fraction expansion can be used to rewrite the fraction in terms of only simple real poles fractions, since R has only real poles. Later on, this opens more efficient possibilities to program the numerical schemes, see section 3.2.1.

Numerical Methods

Let $\Omega = (a, b) \times (0, T)$, where $a, b \in \mathbb{R}$, $a < b$ and $T \geq 0$. For initial and boundary conditions let ϕ be a function, which maps from the spatial region and ψ_1, ψ_2 be functions, which map from the temporal region to the real-valued numbers. Furthermore, $\psi_1, \psi_2 \in \mathcal{C}^1((a, b))$. The solution u, a function mapping from Ω to the real-valued numbers, is assumed to be sufficiently smooth and satisfying the following PDE for all $(x, t) \in \Omega$.

$$\mathcal{L}(u(x,t)) + \mathcal{R}(u(x,t)) + \mathcal{N}(u(x,t)) = F(u(x,t))$$

$$\text{s.t.} \begin{cases} u(x,0) = \phi(x) & \forall x \in (a,b) \\ u(a,t) = \psi_1(t) & \forall t \in (0,T) \\ u(b,t) = \psi_2(t) & \forall t \in (0,T) \end{cases}$$

In this general fractional advection-reaction-diffusion (ARD) equation the fractional differential operator is denoted by $\mathcal{L}$, the linear differential operator in t by $\mathcal{R}$, the nonlinear advection operator by $\mathcal{N}$ and F is a non-homogeneous source term.

Let $F(u(x,t)) = u(x,t)f_i(u(x,t))$ with $i = 1,2$ and $u \equiv u(x,t)$. Morover, suppose γ, λ are positive numbers, $\gamma < 1$ and $1 < \alpha \leq 2$. In case of Burgers-Fisher's with f_1 or Burgers-Huxley's equation with f_2 we get the following terms.

$$\begin{aligned}
\mathcal{L}(u) &:= -\frac{\partial^\alpha u}{\partial |x|^\alpha} \text{ with } 1 \leq \alpha < 2 \\
\mathcal{R}(u) &:= \frac{\partial u}{\partial t} \\
\mathcal{N}(u) &:= \lambda u \frac{\partial u}{\partial x} \\
f_1(u) &:= 1 - u \text{ or } f_2(u) := (1-u)(u-\gamma)
\end{aligned}$$

3.1 Finite Difference Method

The main idea of finite difference methods is to discretize the region and then use difference quotients for the approximation of functions on the region, differential operators and boundary operators specified in differential equations. This leads to equation systems which can be solved numerically. Since the considered PDEs are of fractional type, we make use of fractional centered differences as described in section 2.1.3.

3.1.1 Discretizations for Fractional Burgers' Equations

The discretizations of the regions can be seperated in the spatial mesh, defined by the stepsizes $h = (b-a)/M$, where $M+1$ is the number of spatial grid points and the temporal mesh, given by the temporal stepsize τ_n and the number of meshpoints $N+1$. With that, the discretization of the functions u and f are $u_j^n = u(x_j, t_n)$ and $f(u_j^n)$ with $x_j = a + jh$, $j \in 0, ..., M$ and $n \in 0, ..., N$.

For the variation in t the backward difference quotient of order 1 is used as approximation. Furthermore, centred differences of order 2 are used for the spatial derivative, i.e.

$$\frac{\partial u_j^n}{\partial x} = \frac{\left(u_{j+1}^n - u_{j-1}^n\right)}{(2h) + \mathcal{O}(h^2)}$$

and the fractional term is approximated by fractional centred differences, see section 2.1.3, which is also of second order. With that, the approximations of the linear differential, nonlinear advection and fractional differential operator are

$$\begin{aligned}
\tilde{\mathcal{R}}(u_j^n) &:= \frac{u_j^{n+1} - u_j^n}{\tau_n} \\
\tilde{\mathcal{N}}(u_j^n) &:= \lambda u_j^{n-1} \frac{u_{j+1}^n - u_{j-1}^n}{2h} \\
\tilde{\mathcal{L}}(u_j^n) &:= -\frac{1}{h^\alpha} \sum_{k=0}^{M} g_{j-k}^\alpha u_k^n.
\end{aligned}$$

Using the discretizations the following discrete non-homogeneous Burgers equation

$$\tilde{\mathcal{R}}(u_j^n) + \tilde{\mathcal{L}}(u_j^{n+1}) + \tilde{\mathcal{N}}(u_j^{n+1}) = u_j^{n+1} f(u_j^n) \text{ such that } \begin{cases} u_j^0 = \phi(x_j) & \forall j \in 0, 1, ..., M \\ u_0^n = \psi_1(t_n) = 0 & \forall n \in 0, 1, ..., N \\ u_M^n = \psi_2(t_n) = 0 & \forall n \in 0, 1, ..., N \end{cases}$$

can be obtained, where the time steps are choosen in the way that the nonlinearity occurs in the advection operator, as well as in the source term, see [Macías-Díaz, 2018].

3.1.2 Matrix-Vector Form

By rewriting the approximated non-homogeneous Burgers equation it is possible to obtain a matrix-vector representation, as can be seen in [Macías-Díaz, 2018], describing a system of equations, which can be solved numerically.

Let $A^n \in \mathbb{R}^{(M+1)\times(M+1)}$, $u^n \in \mathbb{R}^{M+1}$ and $v^n \in \mathbb{R}^{M+1}$. With that the equation system is

$$\begin{cases} A^n u^{n+1} = v^n, \quad \forall n \in 0, 1, ..., N-1 \\ \text{s.t. } u^0 = u_0, \end{cases}$$

with

$$u_0 = \begin{pmatrix} \phi(x_0) \\ \phi(x_1) \\ \dots \\ \phi(x_{M-1}) \\ \phi(x_M) \end{pmatrix}, u^n = \begin{pmatrix} u_0^n \\ u_1^n \\ \dots \\ u_{M-1}^n \\ u_M^n \end{pmatrix}, v^n = \begin{pmatrix} \psi_1(t_{n+1}) \\ u_1^n \\ \dots \\ u_{M-1}^n \\ \psi_2(t_{n+1}) \end{pmatrix}.$$

The matrix A^n is relying on the notation

$$\begin{aligned} \xi_j^n &:= \frac{\tau_n}{h^\alpha} g_1^\alpha - \frac{\lambda \tau_n}{2h} u_j^n & \eta_j^n &:= 1 + \frac{\tau_n}{h^\alpha} g_0^\alpha - \tau_n f\binom{n}{j} \\ \nu_j^n &:= \frac{\tau_n}{h^\alpha} g_1^\alpha + \frac{\lambda \tau_n}{2h} u_j^n & \bar{g}_k^\alpha &:= \frac{\tau_n}{h^\alpha} g_k^\alpha, \end{aligned}$$

and becomes

$$A^n = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ \xi_1^n & \eta_1^n & \nu_1^n & \bar{g}_2^\alpha & \dots & \bar{g}_{M-3}^\alpha & \bar{g}_{M-2}^\alpha & \bar{g}_{M-1}^\alpha \\ \bar{g}_2^\alpha & \xi_2^n & \eta_2^n & \nu_2^n & \dots & \bar{g}_{M-4}^\alpha & \bar{g}_{M-3}^\alpha & \bar{g}_{M-2}^\alpha \\ & & & & \dots & & & \\ \bar{g}_{M-2}^\alpha & \bar{g}_{M-3}^\alpha & \bar{g}_{M-4}^\alpha & \bar{g}_{M-5}^\alpha & \dots & \eta_{M-2}^n & \nu_{M-2}^n & \bar{g}_2^\alpha \\ \bar{g}_{M-1}^\alpha & \bar{g}_{M-2}^\alpha & \bar{g}_{M-3}^\alpha & \bar{g}_{M-4}^\alpha & \dots & \xi_{M-1}^n & \eta_{M-1}^n & \nu_{M-1}^n \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

3.1.3 Features

For the fractional advection-reaction-diffusion equations, as for example the Burgers-Huxley equation, positive and bounded solutions are obtained by the numerical method analogously to the the exact solutions obtained in the non-fractional case. Furthermore, the method is consistent, stable and convergent with first-order in time and second-order in space. For a detailed proof refer to [Macías-Díaz, 2018]).

3.2 Exponential Time Differencing

Consider the advection-reaction-diffusion equation in the form

$$u_t(x,t) + Au(x,t) = \mathcal{F}\left(t, u(x,t)\right) \tag{3.2.1}$$

such that

$$\begin{cases} u(x,0) = \phi(x) & \forall x \in (a,b) \\ u(a,t) = \psi_1(t) & \forall t \in (0,T) \\ u(b,t) = \psi_2(t) & \forall t \in (0,T). \end{cases}$$

Exponential time differencing schemes (ETD) are time-stepping schemes to solve reaction-diffusion-advection PDEs numerically. The main idea is to rewrite the ARD-equation in the way that linear terms $Au(x,t)$ and nonlinear terms $\mathcal{F}(t,u)$ get seperated. The intention for the splitting is to use different time stepping schemes for the seperated parts and especially, to treat the linear term highly accurate. Schemes, which use an implicit method for one term and an explicit for the other one, are called implicit-explicit (IMEX), and are belonging besides strang, multi-component, ADI and AF splitting are belonging to the class of time-splitting schemes (see [Hundsdorfer and Verwer, 1996]).

At this point it has to be mentioned that the difference between regular ETD and fractional ETD schemes is, that different approximations of A are used. Therefore, the following section 3.2.1, which is only about how to handle the equation in time, is applicable to either regular or fractional equations.

3.2.1 Time-Stepping Scheme

For the development of the time-stepping scheme the Duhamel principle is used on (3.2.1) to obtain an exact solution on an interval $[t_n, t_{n+1}]$. The equation is similiar to Volterra's integral equation for ordinary differential equations. Let $u(x, t_n) \equiv u(t_n)$ and $k = \Delta t$, then an exact solution is

$$u(t_{n+1}) = e^{-kA}u(t_n) + \int_0^k e^{-A(k-\tau)}\mathcal{F}\left(t_n + \tau, u(t_n + \tau)\right) d\tau. \tag{3.2.2}$$

Let us first consider the nonlinear part $\mathcal{F}$ in the integrand in (3.2.1). Instead of approximating the integral by a quadrature formula, as for example in exponential Rosenbrock methods [Hochbruck et al., 2009], this ETD-scheme works with the following linearization of $\mathcal{F}$ around t_n

$$\mathcal{F}(t_n + \tau, u(t_n + \tau)) \approx \mathcal{F}(t_n, u(t_n)) + \tau\left(\frac{\mathcal{F}(t_{n+1}, u(t_{n+1})) - \mathcal{F}(t_n, u(t_n))}{k}\right). \tag{3.2.3}$$

Be aware, that k is the variable describing the temporal stepsize and not τ, as in chapter 3.1. In this case, τ is the variable with respect to which the integration is carried out. In the ODE case the idea of linearization of the integrand leads in the ODE case to the implicit trapezoidal or Crank-Nicolson (CN) method, which belongs to the class of Adams-Moulton methods.

By integration as described in (3.2.2) substituting $\mathcal{F}$ with its linearization (3.2.3), the semi-discretized scheme

$$u_{n+1} = e^{-Ak}u_n + A^{-1}(1 - e^{-Ak})\mathcal{F}(u_n) + \frac{A^{-2}}{k}(kA - I + e^{-Ak})\left(\mathcal{F}(t_{n+1}, u_{n+1}) - \mathcal{F}(t_n, u_n)\right) \tag{3.2.4}$$

of second order can be established, see also [Janssen, 2009]. It is only semi-discretized since A and $\mathcal{F}$ are not space-discretized, yet. With that it is not nessecary to use additional iterative methods to approximate the nonlinear part. However, (3.2.4) is fully implicit and therefore, an inital guess is needed first to start the iteration and then (3.2.4) can correct the guess. The predictor equation is obtained in the following way. Since we expect the nonlinear part of the solution not to contribute significantly to the variation of the solution for small perturbations we can approximate $\mathcal{F}$ with first-order accuracy by

$$\mathcal{F}(t_n + \tau, u(t_n + \tau)) \approx \mathcal{F}(t_n, u(t_n))$$

and with that the predictor equation consists of the first two terms of (3.2.4), that is

$$u^{\star}_{n+1} = e^{-Ak}u_n - A^{-1}(1 - e^{-Ak})\mathcal{F}(t_n, u_n) \tag{3.2.5}$$

as can be seen in [Janssen, 2009].

Altogether, the ETD scheme is a predictor-corrector method used for equations in form of (3.2.1) and is described by (3.2.4) and (3.2.5). It is a fixed point method, similiar to PECE-method used for ODEs. Usually one iteration is already good enough, since the predictors order is only one less than the correctors. Obviously, the approximation is not finished, since the exponentials are not computed. Common in application to ETD schemes are either Padé or real distinct poles (RDP) approximations, see section 2.2.

Crank-Nicolson

Using the the (0,1)-Padé scheme $R_{0,1}(-kA) := (I + kA)^{-1}$, where I is the identity, for the predictor (3.2.5) and the (1,1)-Padé scheme $R_{1,1}(-kA) := (2I - kA)(2I + kA)^{-1}$ for the corrector (3.2.4), we obtain the following ETD scheme

$$\begin{aligned} u^{\star}_{n+1} &= R_{1,1}(-kA)u_n + kR_{0,1}\left(-\frac{1}{2}kA\right)\mathcal{F}(t_n, u_n) \\ u_{n+1} &= u^{\star}_{n+1} + 2kR_{0,1}\left(-\frac{1}{2}kA\right)\left[\mathcal{F}(t_{n+1}, u^{\star}_{n+1}) - \mathcal{F}(t_n, u_n)\right]. \end{aligned} \tag{3.2.6}$$

Note that the scheme was also subject to the following algebraic transformations:
$-A^{-1}\left[R_{1,1}(-kA) - I\right] = kR_{0,1}\left(-\frac{1}{2}kA\right)$ and $\frac{1}{k}A^{-2}\left[R_{1,1}(-kA) - I + kA\right] = 2kR_{0,1}\left(-\frac{1}{2}kA\right)$

After considering the exponentials we still have to take a look at the matrices, which have to be inverted. In case of high condition numbers the ETD-method in form of (3.2.6) could become inaccurate due to high error amplification (small errors in input produces high error in output). That is why a splitting technique/partial fraction decomposition is used. In the beginning there is an initial damping step, which can be calculated directly by

$$(I + kA)u_{n+1} = u_n + k\mathcal{F}(t_n, u_n)$$

, and in the further steps the computationally efficient method is given by

$$\begin{cases} u^{\star}_{n+1} = (-1)u_n + a_n & \text{with } (2I + kA)a_n = 4u_n + 2k\mathcal{F}(t_n, u_n) \\ u_{n+1} = u^{\star}_{n+1} + b_n & \text{with } (2I + kA)b_n = k\left[\mathcal{F}(t_{n+1}, u^{\star}_{n+1}) - \mathcal{F}(t_n, u_n)\right]. \end{cases} \tag{3.2.7}$$

A version of this method is published in [Asante-Asamani and Wade, 2016] where a slight speedup is achieved with help of a splitting method.

.

Real Distinct Poles

The ETD-CN is very efficient and the matrix exponential approximations are A-acceptable, see [Janssen, 2009]. Unfortunately, in some equations high frequencies are not decaying fast enough/spurious oscillations are not damped out, see [Iyiola and Wade, 2017]. Another problem in ETD-CN is, that matrices with eigenvalues close to zero can result in numerical problems, since the matrices are inverted. For that reason, there was a partial fractional decomposition used in ETD-CN, see (3.2.7).

In real life application the equations are usually considered in large regions resulting in a big amount of data. Therefore, parallelization is very common to speed up the evolution and for that, a separation of poles in the rational approximation is helpful. Since this is not possible for Padé schemes, because the approximation is defined using truncated power series, there is another discretization considered, which is called real distinct poles (RDP) approximation, see 2.2.2.

The RDP approximation $R(z) := \left(1 + \frac{5}{12}z\right)\left[(1 - \frac{1}{3}z)(1 - \frac{1}{4}z)\right]^{-1}$ is nearly optimal for a second order rational approximation, see [Voss and Khaliq, 1995]. Again, as in ETD-CN, a (0,1)-Padé is used as a locally second order predictor (3.2.5) and the higher-order approximation $R(-Ak) := \left(I - \frac{5}{12}Ak\right)\left[\left(I + \frac{1}{3}Ak\right)\left(I + \frac{1}{4}Ak\right)\right]^{-1}$ is used for the matrix exponentials e^{-Ak} in the corrector (3.2.4). The resulting ETD scheme including RDP approximations is given by

$$\begin{aligned} u^{\star}_{n+1} &= R_{0,1}(-kA)u_n + A^{-1}(I - R_{0,1}(-Ak))\mathcal{F}(t_n, u_n) \\ u_{n+1} &= R(-kA)u_n + A^{-1}(I - R(-Ak))\mathcal{F}(t_n, u_n) \\ &\quad + \frac{A^{-2}}{k}(kA - I + R(-Ak)\left[\mathcal{F}(t_{n+1}, u^{\star}_{n+1}) - \mathcal{F}(t_n, u_n)\right]. \end{aligned} \tag{3.2.8}$$

Further partial fractional decompositions to improve efficiency, as can be seen in [Asante-Asamani et al., 2016], [Iyiola and Wade, 2017], leads to the following efficient and ready to program ETD-RDP scheme

$$\begin{cases} (I + kA)u^*_{n+1} = u_n + k\mathcal{F}(t_n, u_n) \\ u_{n+1} = a_n + b_n \\ \text{with } (I + \frac{1}{3}kA)a_n = 9u_n + 2k\mathcal{F}(t_n, u_n) + k\mathcal{F}(t_{n+1}, u^*_{n+1}) \\ \text{and } (I + \frac{1}{4}kA)b_n = -8u_n - \frac{3}{2}k\mathcal{F}(t_n, u_n) - \frac{k}{2}\mathcal{F}(t_{n+1}, u^\star_{n+1}). \end{cases} \tag{3.2.9}$$

Again before using (3.2.9) there is an inital damping step, which can be calculated directly by $(I + kA)u_{n+1} = u_n + k\mathcal{F}(t_n, u_n)$.

3.2.2 Spatial Discretization for Fractional Burgers' Equations

The general advection-reaction-diffusion equation can be specified to the fractional Burgers equation as mentioned in the beginning of this chapter. With the goal to apply the ETD scheme the equation has to be rewritten as

$$u_t(x,t) + Au(x,t) = \mathcal{F}(t, u(x,t)),$$

as discussed in section 3.2, where linear terms including fractional terms are described by $Au(x,t)$ and nonlinear terms by $\mathcal{F}(t, u(x,t))$. The temporal discretizations are (3.2.6) or (3.2.8). In a next step, the region of interest is discretized, similiar to the finite difference methods in section 3.1. Furthermore, the approximation of space-fractional derivatives is done by fractional centred differences,

$$-\delta^\alpha_h u^{(n)}_j = \frac{1}{h^\alpha} \sum_{k=0}^{M} g^\alpha_{j-k}, u^{(n)}_k$$

where δ^α_h denotes the fractional centred difference operator.

With the intent to bring the fractional centred differences into matrix-vector form let

$$\mathbf{u}^n = (u_0^n, ..., u_M^n)^T \text{ and } \mathbf{g}_j^\alpha = \left(g_{j-0}^\alpha, ..., g_{j-M}^\alpha\right) \text{ so that } \delta_h^\alpha u_j^n = -\frac{1}{h^\alpha}\mathbf{g}_j^\alpha \cdot \mathbf{u}^n.$$

With that, the matrix A^n is described in

$$\delta_h^\alpha \mathbf{u}^n = -\frac{1}{h^\alpha}\begin{pmatrix} \mathbf{g_0^\alpha} \cdot \mathbf{u}^n \\ ... \\ \mathbf{g_M^\alpha} \cdot \mathbf{u}^n \end{pmatrix} = -\frac{1}{h^\alpha}\begin{pmatrix} 1 & 0 & 0 & ... \\ g_1 & g_0 & g_{-1} & \\ ... & & & ... \\ ... & & 0 & 1 \end{pmatrix}\mathbf{u}^n = A^n\mathbf{u}^n. \tag{3.2.10}$$

By consideration of $g_i = g_{-i}$ for all $i > 0$ the matrix is known to be symmetric.

The further procedure is to develop an approximation for $\mathcal{F}$, which contains the nonlinear terms. In application to the non-homogeneous Burgers equation, that are the advection and reaction term, so that $\mathcal{F}(u,t) = -\lambda u(x,t)u_x(x,t) + u(x,t)f(u(x,t))$. As second-order approximation of the derivative in space a backward difference quotient

$$\partial_x u_j^{(n)} = \frac{u_{j+1}^{(n)} - u_{j-1}^{(n)}}{2h} + \mathcal{O}(h^2),$$

is used, which can be rewritten to a matrix vector form for each time step t_n as

$$\partial_x \mathbf{u^{(n)}} = \frac{1}{2h}\begin{pmatrix} 1 & 0 & 0 & ... \\ -1 & 0 & 1 & \\ 0 & -1 & 0 & 1 \\ ... & & & ... \\ & & & 0 & 1 \end{pmatrix}\mathbf{u^{(n)}}, \text{ where } \mathbf{u^{(n)}} = \begin{pmatrix} \psi_1(t_n) \\ u_1^{(n)} \\ ... \\ u_{M-1}^{(n)} \\ \psi_2(t_n) \end{pmatrix}.$$

With that we get the final approximation for the nonlinear part of the equation, that is

$$\mathcal{F}(\mathbf{u^{(n)}}) = -\lambda \mathbf{u^{(n)}} \partial_x \mathbf{u^{(n)}} + \mathbf{u^{(n)}} f(\mathbf{u^{(n)}}). \tag{3.2.11}$$

Altogether, the obtained ETD scheme for the fractional Burgers equation is either 3.2.6 or 3.2.9, where the matrix A is approximated by $A^{(n)}$, as in 3.2.10, and $\mathcal{F}(t_n, u_n)$ by $\mathcal{F}(\mathbf{u^{(n)}})$, as in 3.2.11.

Numerical Results

4.1 Example and its Solutions

Let the space-fractional Burgers equation with reaction term be considered for each $(x,t) \in \Omega$ with $\Omega = (a,b) \times (0,T)$ in the form

$$\frac{\partial}{\partial t}u(x,t) = \frac{\partial^\alpha}{\partial |x|^\alpha}u(x,t) - \lambda u(x,t)\frac{\partial}{\partial x}u(x,t) + u(x,t)f(u(x,t)) \tag{4.1.1}$$

from now on, with the same initial and boundary conditions as before.

The example, used for the comparison of the numerical schemes regarding the space-fractional Burgers equation in homogeneous form or with reaction term, is the following.

Example 4.1.2. *Let $\Omega = (-200, 200) \times (0, 12)$, spatial stepsize $h = 1$ and $\lambda = 1$ or $\lambda = 0.1$. The boundary conditions are given by*

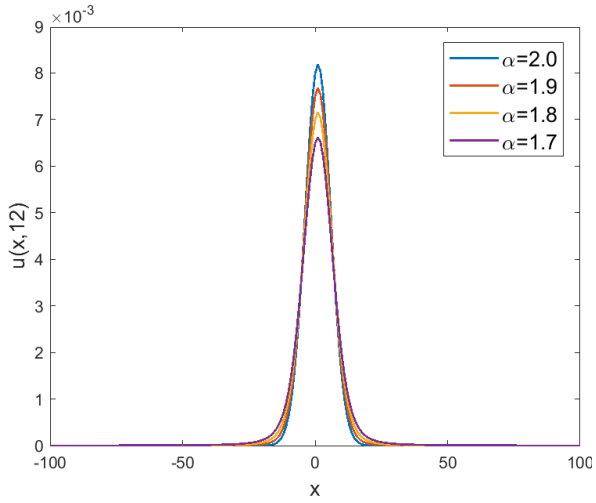
$$\psi_1(t) = \psi_2(t) = 0 \qquad \forall t \in (0, 12)$$

and the initial condition by

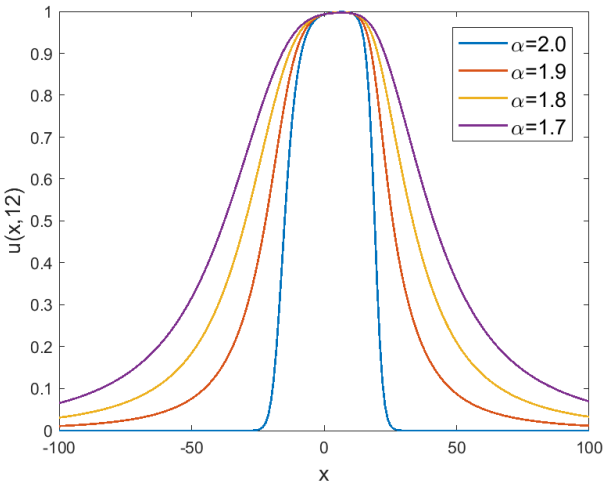
$$\phi(x) = \begin{cases} 0.1 & : x = 0 \\ 0 & : otherwise. \end{cases}$$

For the moment let the solutions be obtained regarding various fractional parameters, advection coefficients or time steps. Furthermore, let $\lambda = 1$, $\tau = 0.0125$ and the method to construct the solution be FETD-RDP.

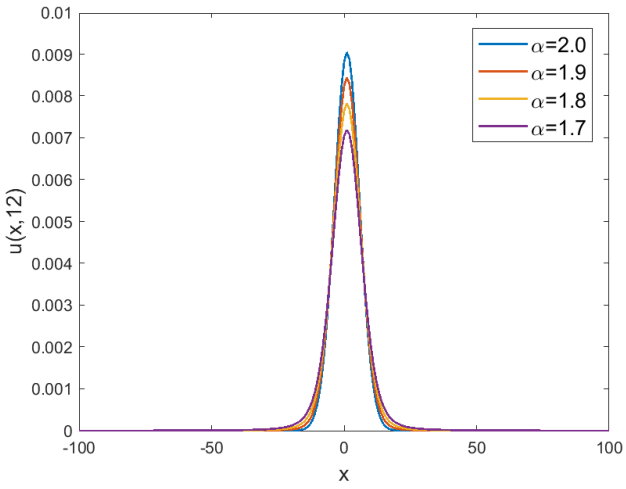
How the choice of the fractional parameter α influences the solution of (4.1.1) can be seen in 4.1, where the solution is plotted over the interval $(-100, 100)$. The smaller α, the more the curve differs from its initial condition. A reason for that is that the smaller the fraction is, the more diffusive properties are taken into account. In other words, the closer α is to two, the more the fractional derivative behaves like regular diffusion.



(a) Homogeneous Burgers



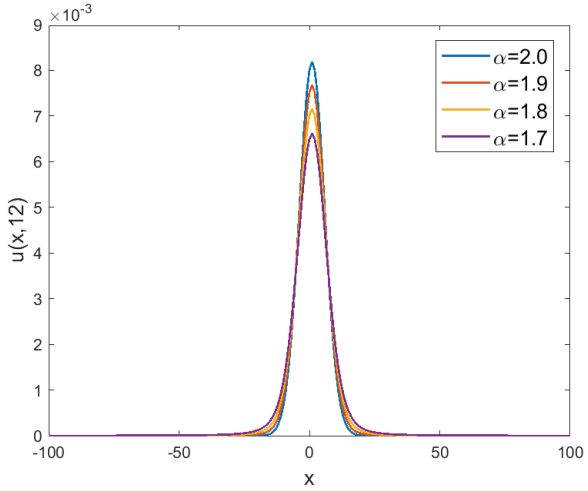
(b) Burgers-Fisher



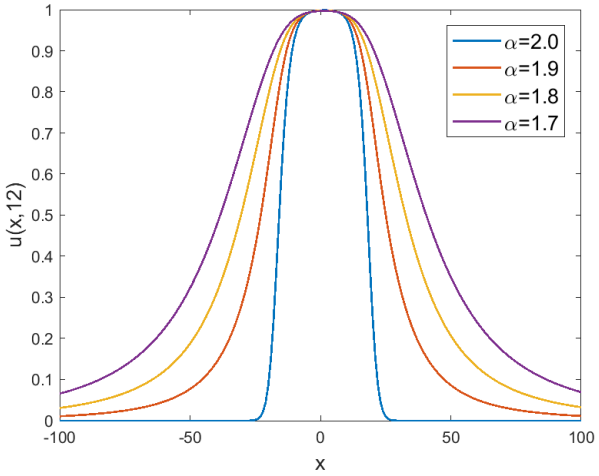
(c) Burgers-Huxley

Figure 4.1: Comparison of solutions for various α with $\lambda = 1$

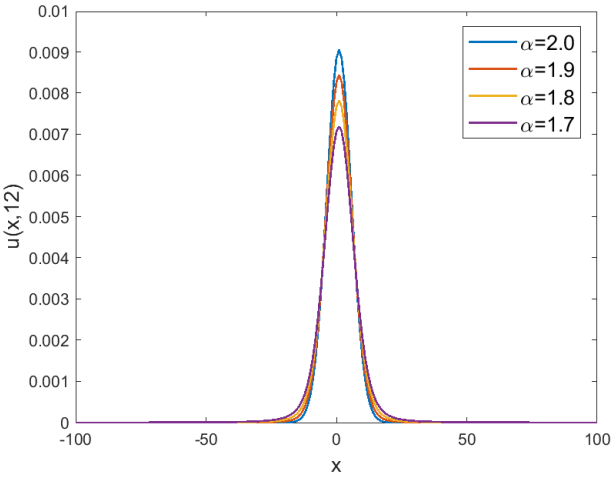
By varying the parameter λ, different solutions as in 4.2, where $\lambda = 0.1$, can be obtained. Since λ is the constant multiplied with the convection term, it describes how much convection occurs in the solution. As already mentioned earlier, the convection term consists of the solution multiplied with its derivative.



(a) Homogeneous Burgers



(b) Burgers-Fisher



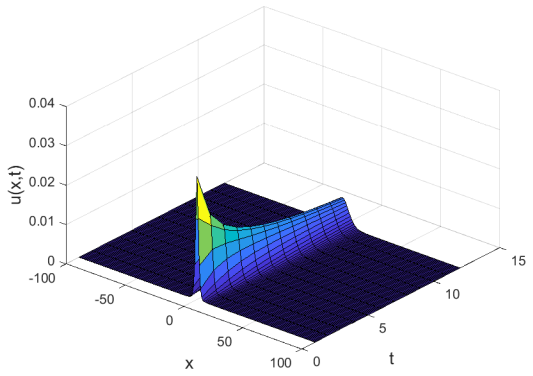
(c) Burgers-Huxley

Figure 4.2: Comparison of solutions for various α with $\lambda = 0.1$

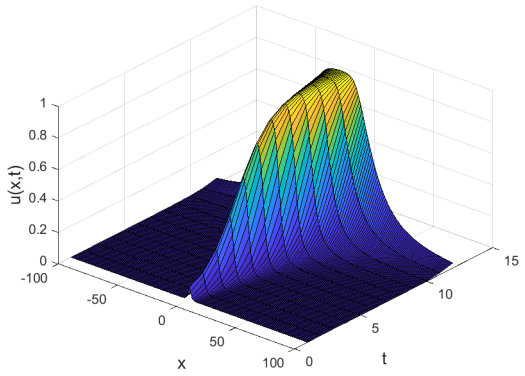
First, note that the solutions are positive. Futhermore, the rate of change is positive for negative values, while it is negative for positive values. For all those reasons, the curve is shifted in a certain way to the right for positive x and to the left for negative x. In the plot, this becomes most obvious with Fisher's reaction term and for the positive x-values close to zero, since the maximum of the function is not at $x = 0$ anymore, but shifted to the right. Note, the higher λ, the more the previously described shift becomes obvious in the solution.

The plots for Burgers-Huxley's and homogeneous Burgers' equation look very similiar, though the maximum in the former one is slightly higher. Considering Huxley's reaction $f_2(u) = (1-u)(u-\gamma)$ in the whole source term $F(u) = uf_2(u)$, the solutions are approximately squared and then multplied by $1-u$. Furthermore, taking into account that the initial condition at $x = 0$ is 0.1, it is explained, why the maximum of Burgers-Huxley's is situated between the orders of magnitude 10^{-3} to 10^{-2}. In case of the homogeneous Burgers equation there is no reaction term at all, which is why there the maximum is even smaller than in Burgers-Huxley's solution. In comparison to that, the solutions for Burgers-Fisher's, where the reaction term is just $1-u$, are generally higher due to the fact the reaction term is high enough to dominate the terms, which influence the solutions to become smaller.

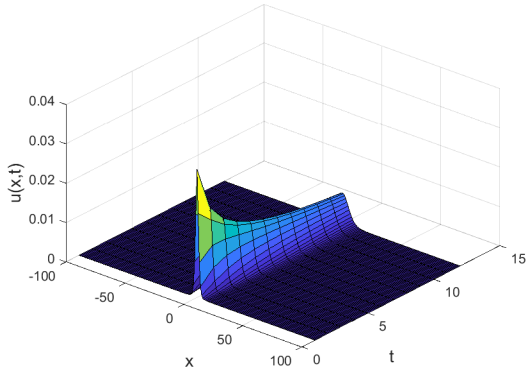
In the figures 4.3 we can see how the solutions develop over time. Therefore, let again $\lambda = 1$ and also the fractional parameter is set, in this case $\alpha = 1.8$. The solutions are picked from the time steps $T = 1, 2, \ldots, 12$ and a surface is fitted over the solutions. The comparison of the reaction terms, explained above, becomes more obvious in these plots. Especially, the solutions clearly decrease over time for homogeneous Burgers' and Burger-Huxley's equation, while they increase for Burgers-Fisher's.



(a) Homogeneous Burgers



(b) Burgers-Fisher



(c) Burgers-Huxley

Figure 4.3: Comparison of solutions for $\alpha = 1.8$ and $\lambda = 1$

4.2 Comparison regarding to Convergence and Efficiency

In this chapter we adress, which method works best for an example with non-smooth initial conditions, but homogeneous Dirichlet boundary conditions, given by 4.1.2. The quality of the numerical algorithm can be understood, on the one hand, as how much the approximations produced by the algorithm improve in terms of the error while letting the temporal stepsize decrease (convergence), and on the other hand, how much time elapses while the error gets improved (efficiency).

For the purpose of comparing the numerical methods described in chapter 2.2.2 the following definitions of error and convergence rate are used. To the best of my knowledge, no exact solutions exist for example 4.1.2. Therefore, let u be an almost exact solution, produced by FETD-CN with a very small temporal step size $\tau = 0.0001$, whereas $\tilde{u}_\tau$ is the approximated solution, produced with stepsize τ. Then, the relative error in max-norm is defined by $E_\tau = ||u - \tilde{u}_\tau||_\infty / ||u||_\infty$ and the approximated rate of convergence defined in terms of the relative error by

$$p \approx \frac{\log\left(\frac{E_\tau}{E_{\tau/2}}\right)}{log(2)}. \tag{4.2.1}$$

In the following, the comparison is outlined in detail for the homogeneous Burgers equation and from there, it is contrasted how including Fisher's and Huxley's reaction term, respectively, affects convergence and efficiency.

4.2.1 Homogeneous Burgers' Equation

For Burgers' equation with no source term and the convection parameter $\lambda = 1$ the plots showing convergence and efficiency results can be seen in the figure 4.4. FETD-RDP and FETD-CN show similiar convergence and effiency results. However, RDP seems to be slightly better convergence, whereas in case of CN the elapsed time is less (on cost of a higher error). In contrast to that, the FD method costs slightly less time than ETD-CN for same temporal step amount, but therefore, has a much higher error.

Especially in case of $\alpha = 2$, FD shows the best results regarding elapsed time. The reason for that is, that the matrix A becomes of a banded structure, since the fractional diffusion is actually regular diffusion and with that the fractional operator gets reduced to the Laplace operator. With that property it is possible to improve the efficieny of the algorithm. However, to obtain similiar good approximations as in the FETD schemes, the time step in FD would have to be chosen smaller and with that the elapsed time would increase.

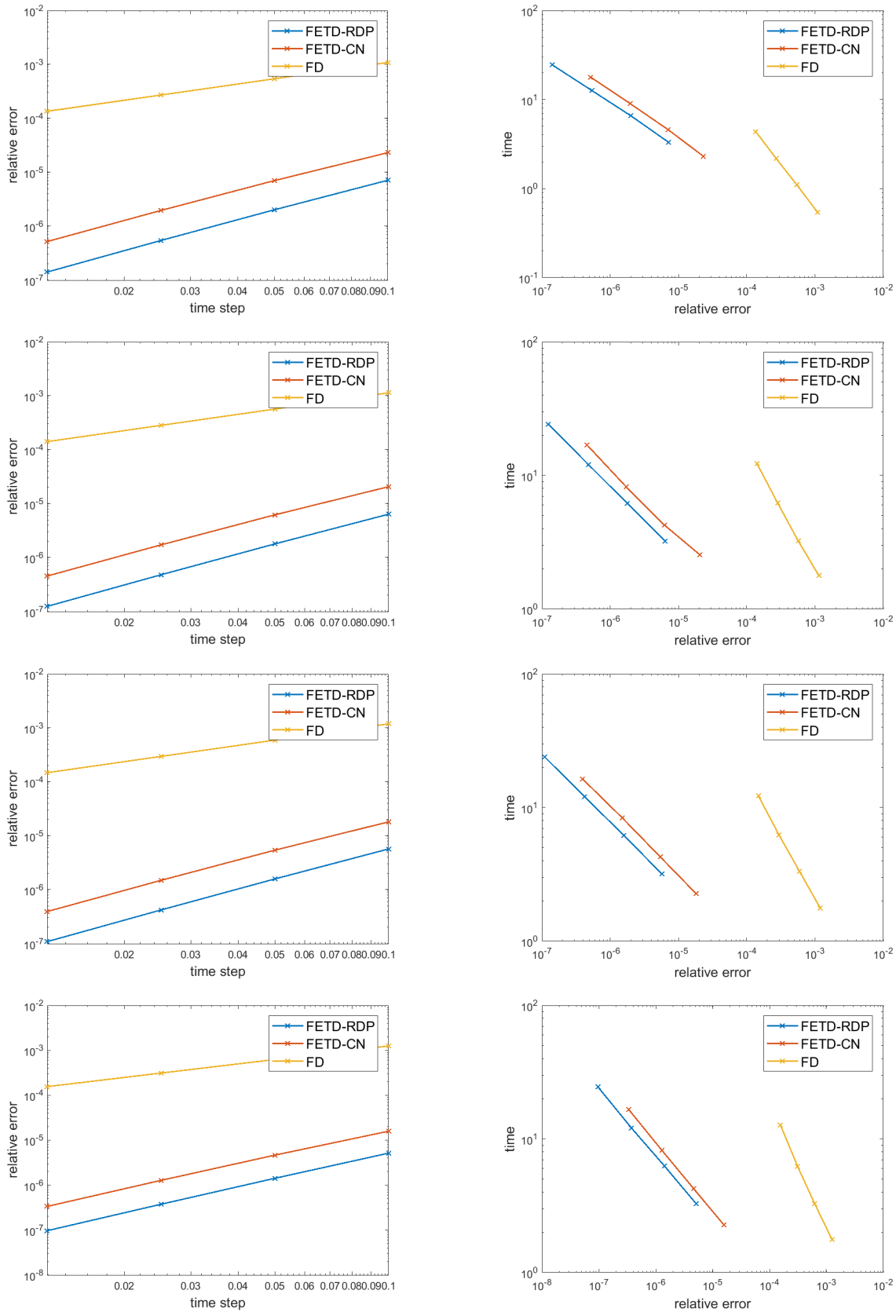


Figure 4.4: Homogeneous Burgers' equation with $\lambda = 1$: convergence plots (left) and efficiency plots (right) for fractional parameters $\alpha = 2.0, 1.9, 1.8, 1.7$ (from top to bottom)

A deeper insight into the results can be provided by the tables in the figure 4.5. Since FETD-RDP and FETD-CN show similiar convergence and efficiency, the tables regarding FETD-CN are excluded there. The results for FETD-CN can be found in the appendix A. As expected, the convergences rate depends on the fractional parameter. In case of FETD-RDP $p = 2$ and p increases with decreasing τ. In contrast to that, FD converges approximately with order 1 and the order increases slightly for decreasing τ.

alpha	**tau**	**error**	**time**	**rate**
2	0.1	7.1228e-06	3.3275	0
2	0.05	2.0163e-06	6.5632	1.8207
2	0.025	5.4238e-07	12.5931	1.8943
2	0.0125	1.4112e-07	24.5897	1.9423
1.9	0.1	6.3783e-06	3.218	0
1.9	0.05	1.7921e-06	6.1635	1.8315
1.9	0.025	4.7979e-07	12.0568	1.9012
1.9	0.0125	1.2454e-07	24.2569	1.9458
1.8	0.1	5.7009e-06	3.1878	0
1.8	0.05	1.5842e-06	6.1967	1.8474
1.8	0.025	4.2135e-07	12.0573	1.9107
1.8	0.0125	1.0906e-07	24.0343	1.9498
1.7	0.1	5.1472e-06	3.2715	0
1.7	0.05	1.4188e-06	6.2862	1.8591
1.7	0.025	3.7541e-07	12.0698	1.9182
1.7	0.0125	9.6777e-08	24.4962	1.9557

(a) FETD-RPD method

alpha	**tau**	**error**	**time**	**rate**
2	0.1	0.0010889	0.54228	0
2	0.05	0.00054287	1.1006	1.0042
2	0.025	0.00027103	2.1674	1.0022
2	0.0125	0.00013541	4.3797	1.0011
1.9	0.1	0.0011385	1.7797	0
1.9	0.05	0.00056756	3.2386	1.0043
1.9	0.025	0.00028335	6.2247	1.0022
1.9	0.0125	0.00014157	12.2412	1.0011
1.8	0.1	0.001193	1.7698	0
1.8	0.05	0.00059469	3.3212	1.0044
1.8	0.025	0.00029688	6.2428	1.0022
1.8	0.0125	0.00014833	12.2241	1.0011
1.7	0.1	0.0012532	1.7714	0
1.7	0.05	0.00062464	3.2618	1.0045
1.7	0.025	0.00031182	6.2355	1.0023
1.7	0.0125	0.00015579	12.7269	1.0012

(b) FD method

Figure 4.5: Convergence table for homogeneous Burgers' equation

4.2.2 Burgers-Fisher's Equation

Tables 4.5 and 4.8 illustrate, that the errors for the Burgers-Fisher equation are generally slightly higher than in the homogeneous Burgers equation and the numerical methods have higher convergence rates.

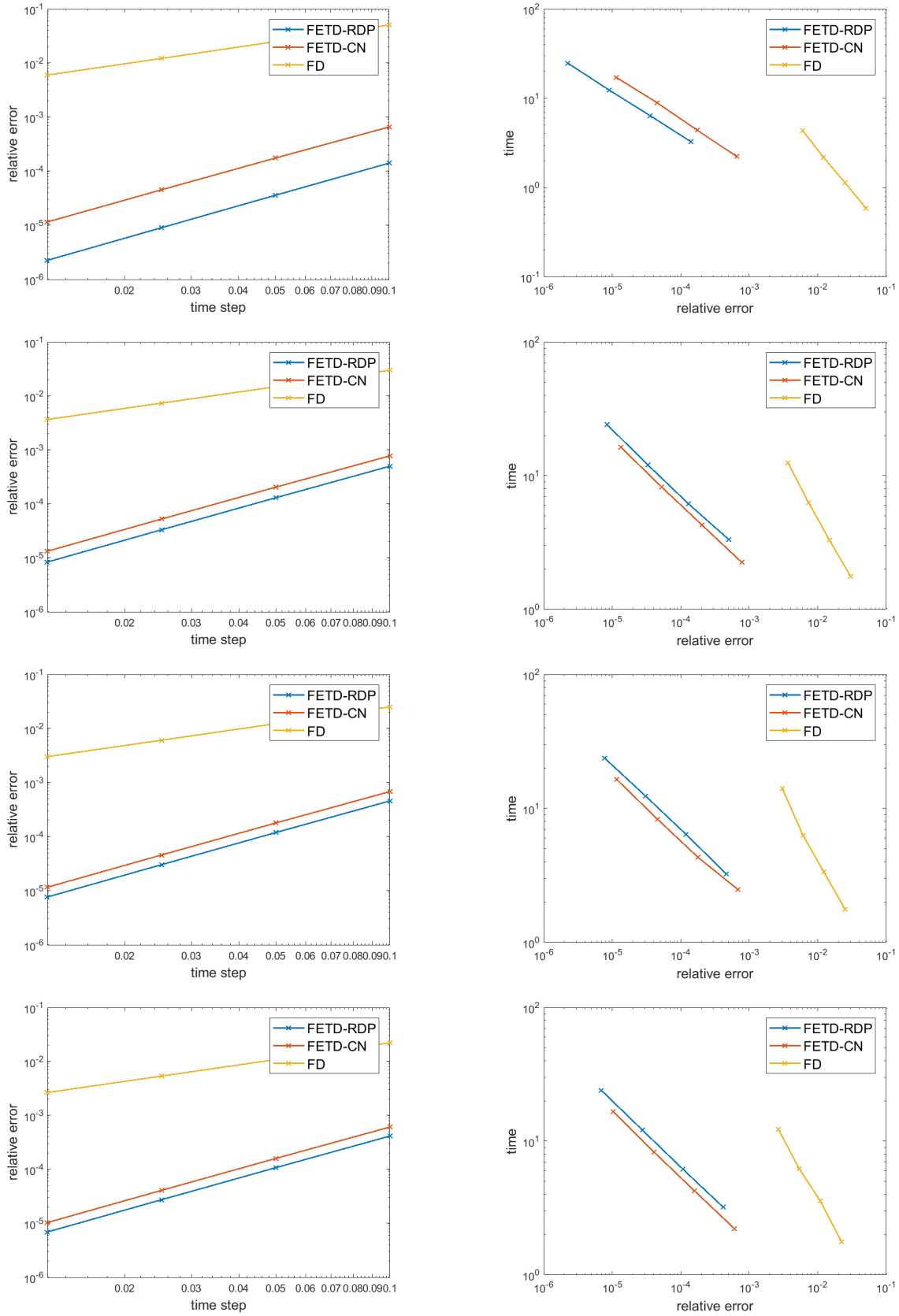


Figure 4.6: Burgers-Fisher's equation with $\lambda = 1$: convergence plots (left) and efficiency plots (right) for fractional parameters $\alpha = 2.0, 1.9, 1.8, 1.7$ (from top to bottom)

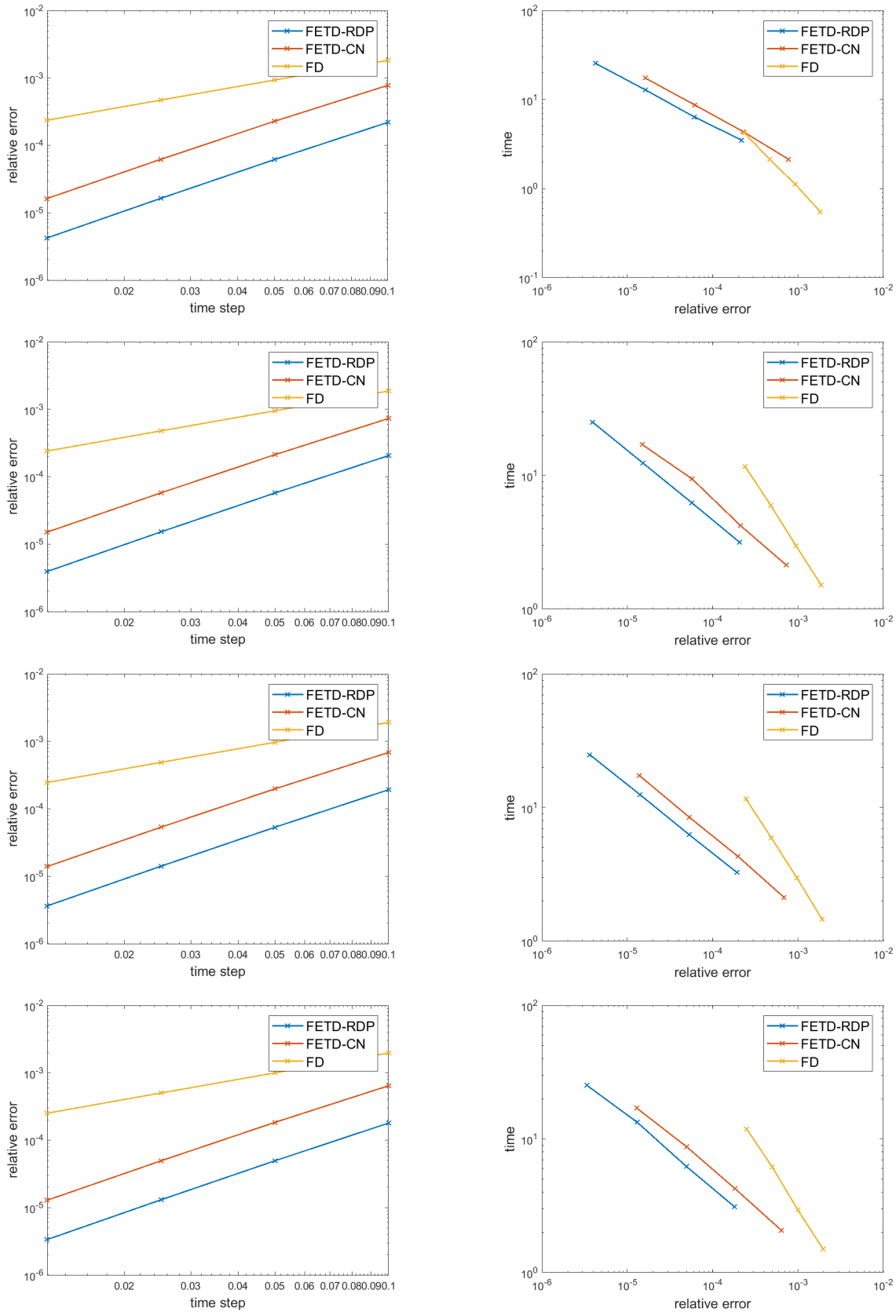


Figure 4.7: Burgers-Huxley's equation with $\lambda = 1$: convergence plots (left) and efficiency plots (right) for fractional parameters $\alpha = 2.0, 1.9, 1.8, 1.7$ (from top to bottom)

The reason for the higher errors is probably, that by taking a source term into account, more basic calculations are needed in the algorithm and with that there more computational errors, produced for example by a limited representation of decimals, are introduced. Comparing the different fractional parameters for the same temporal stepsizes, the rate increases for the homogeneous Burgers equation, but decreases for Burgers-Fisher's. With that we definitely know, that the convergence rate is not only depending on the choice of the fractional parameter, but also on the chosen source term.

alpha	**tau**	**error**	**time**	**rate**
2	0.1	0.0001414	3.2594	0
2	0.05	3.5895e-05	6.3792	1.9779
2	0.025	9.0017e-06	12.3701	1.9955
2	0.0125	2.2513e-06	24.7617	1.9995
1.9	0.1	0.00050052	3.3373	0
1.9	0.05	0.00013013	6.1525	1.9434
1.9	0.025	3.3192e-05	12.0383	1.9711
1.9	0.0125	8.3824e-06	24.1432	1.9854
1.8	0.1	0.00045979	3.2548	0
1.8	0.05	0.00011936	6.4088	1.9456
1.8	0.025	3.0423e-05	12.4255	1.9721
1.8	0.0125	7.6804e-06	23.8402	1.9859
1.7	0.1	0.00041499	3.2139	0
1.7	0.05	0.00010767	6.173	1.9465
1.7	0.025	2.7434e-05	12.1361	1.9726
1.7	0.0125	6.9247e-06	24.0447	1.9862

(a) FETD-RPD method

alpha	**tau**	**error**	**time**	**rate**
2	0.1	0.050686	0.58708	0
2	0.05	0.024963	1.1377	1.0218
2	0.025	0.012173	2.1815	1.0362
2	0.0125	0.0060058	4.3717	1.0192
1.9	0.1	0.030089	1.7709	0
1.9	0.05	0.014884	3.289	1.0155
1.9	0.025	0.0073936	6.2635	1.0094
1.9	0.0125	0.0036829	12.484	1.0054
1.8	0.1	0.025172	1.761	0
1.8	0.05	0.012329	3.3471	1.0298
1.8	0.025	0.0060904	6.3044	1.0174
1.8	0.0125	0.0030264	14.1437	1.0089
1.7	0.1	0.022264	1.77	0
1.7	0.05	0.01088	3.5804	1.0331
1.7	0.025	0.0053676	6.2267	1.0193
1.7	0.0125	0.0026659	12.2818	1.0097

(b) FD method

Figure 4.8: Convergence table for Burgers-Fisher's equation with $\lambda = 1$

For $\alpha = 2$ FETD-RDP has lower errors than for low fractional parameters. Since this phenomen does not occur in the results for the homogeneous Burgers equation, it has to be related to the reaction term. FETD-RDP always has a smaller error than FETD-CN, and a much smaller one than FD, as in the previous test, see figure 4.5.

4.2.3 Burgers-Huxley's Equation with $\gamma = 0.005$

Shown in figure 4.7, FETD-CN has almost the same error for Burgers-Huxley's as for Burgers-Fisher's, whereas the FD and FETD-RDP method have slightly lower errors, but all of them are still higher than in the homogeneous Burgers equation. This is caused by the reaction term $u(1-u)(u-\gamma)$, which has less influence than $u(1-u)$, but more than in the homogeneous case. The reason for that could be similiar to Burgers-Fisher's higher errors. Since the source term is included, there are more calculations needed. However, even if in the Burgers-Huxley more calculations are needed in the reaction term than in the Burgers-Fisher equation, the solutions themselves are smaller and with that the errors in each step have less influence on the error of the final solution. The FD method yields the best efficiency results for the Burgers-Huxley equation, which can be recognized by the efficiency curve being closer to the FETD methods.

alpha	tau	error	time	rate
2	0.1	0.00021921	3.5017	0
2	0.05	6.127e-05	6.3723	1.839
2	0.025	1.6335e-05	12.8452	1.9072
2	0.0125	4.2279e-06	25.5735	1.95
1.9	0.1	0.00020554	3.1743	0
1.9	0.05	5.7158e-05	6.2398	1.8464
1.9	0.025	1.5187e-05	12.4201	1.9121
1.9	0.0125	3.9229e-06	25.0754	1.9529
1.8	0.1	0.00019229	3.272	0
1.8	0.05	5.3214e-05	6.2637	1.8534
1.8	0.025	1.4094e-05	12.507	1.9168
1.8	0.0125	3.6336e-06	24.9001	1.9556
1.7	0.1	0.00017945	3.1193	0
1.7	0.05	4.9429e-05	6.2113	1.8602
1.7	0.025	1.3051e-05	13.3513	1.9212
1.7	0.0125	3.3589e-06	25.2225	1.9581

(a) FETD-RPD method

alpha	tau	error	time	rate
2	0.1	0.0018275	0.55014	0
2	0.05	0.00092987	1.1213	0.97475
2	0.025	0.00046945	2.1161	0.98606
2	0.0125	0.00023592	4.2137	0.99266
1.9	0.1	0.0018702	1.513	0
1.9	0.05	0.00094989	2.975	0.97732
1.9	0.025	0.00047907	5.9869	0.98753
1.9	0.0125	0.00024063	11.6591	0.99345
1.8	0.1	0.001917	1.4587	0
1.8	0.05	0.00097201	2.9817	0.9798
1.8	0.025	0.00048975	5.9199	0.98892
1.8	0.0125	0.00024586	11.607	0.99419
1.7	0.1	0.0019687	1.4995	0
1.7	0.05	0.00099655	2.9426	0.9822
1.7	0.025	0.00050165	6.1479	0.99025
1.7	0.0125	0.00025172	11.8231	0.99489

(b) FD method

Figure 4.9: Convergence table for Burgers-Huxley's equation with $\lambda = 1$

The convergence rates, as can be seen in 4.9, are slightly higher for the FETD applied on Burgers-Huxley's than on homogeneous Burgers' equation, whereas, the convergence rate of the FD method is slightly less in comparison, but overall very similiar. The dynamics of the convergence rate by variation of the fractional parameter are comparable to Burgers-Fisher's equation.

Conclusion

In this thesis a finite difference scheme and two exponential time differencing schemes including a Padé and real disinct poles approximation for the matrix exponentials are explained. The numerical investigation underlines the first order convergence of the former scheme and second order of the latter. In all considered applications, the ETD schemes with real distinct poles approximation works best in terms of accuracy of the results. This demonstrates the dominance of ETD-RDP over FD for the considered nonlinear fractional PDEs.

One possible way to improve the exponential time differencing scheme with real distinct poles approximation is to parallelize the method which is actually a major feature of that scheme. Another idea would be to choose the temporal stepsize adaptively indicated by an error boundary. This could be applied to either the FD or one of the ETD schemes.

In the future the scheme can also be further extended to describe advection reaction diffusion models with two dimensions in space. For the non-fractional homogeneous Burgers equation some methods with one dimension in time and two dimensions in space have already been investigated for a modified cubic B-spline differential quadrature method combined with a fourth order Runge-Kutta scheme, see [Shukla et al., 2014], a Cole-Hopf transformation plus finite element method, see [Zhao et al., 2011], a fully implicit finite difference scheme solved with Adomian decomposition method, see [Zhu et al., 2010] and reduced order models based on Galerkin projection and discrete empirical interpolation in case of high Reynolds number, see [Wang et al., 2016].

Bibliography

[Asante-Asamani et al., 2016] Asante-Asamani, E., Khaliq, A., and Wade, B. A. (2016). A real distinct poles exponential time differencing scheme for reaction–diffusion systems. *Journal of Computational and Applied Mathematics*, 299:24–34.

[Asante-Asamani and Wade, 2016] Asante-Asamani, E. and Wade, B. A. (2016). A dimensional splitting of etd schemes for reaction-diffusion systems. *Communications in Computational Physics*, 19(5):1343–1356.

[Burgers, 1948] Burgers, J. M. (1948). A mathematical model illustrating the theory of turbulence. In *Advances in applied mechanics*, volume 1, pages 171–199. Elsevier.

[contributors, 2018] contributors, W. (2018). Burgers' equation — wikipedia, the free encyclopedia. [Online; accessed 26-March-2018].

[Fisher, 1937] Fisher, R. A. (1937). The wave of advance of advantageous genes. *Annals of Human Genetics*, 7(4):355–369.

[Griffiths and Schiesser, 2010] Griffiths, G. and Schiesser, W. E. (2010). *Traveling wave analysis of partial differential equations: numerical and analytical methods with MATLAB and Maple*. Academic Press.

[Hochbruck et al., 2009] Hochbruck, M., Ostermann, A., and Schweitzer, J. (2009). Exponential rosenbrock-type methods. *SIAM Journal on Numerical Analysis*, 47(1):786–803.

[Hodgkin and Huxley, 1952] Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544.

[Hopf, 1950] Hopf, E. (1950). The partial differential equation ut + uux = xx. *Communications on Pure and Applied Mathematics*, 3(3):201–230.

[Hundsdorfer and Verwer, 1996] Hundsdorfer, W. and Verwer, J. (1996). Numerical solution of advection-diffusion-reaction equations. *CWI Report NMN9603, Centrum voor Wiskunde en Informatica, Amsterdam*, 24:30.

[Iyiola et al., 2017] Iyiola, O., Asante-Asamani, E., Furati, K., Khaliq, A., and Wade, B. (2017). Efficient time discretization scheme for nonlinear space fractional reaction–diffusion equations. *International Journal of Computer Mathematics*, pages 1–18.

[Iyiola et al., 2018] Iyiola, O., Asante-Asamani, E., and Wade, B. (2018). A real distinct poles rational approximation of generalized mittag-leffler functions and their inverses: Applications to fractional calculus. *Journal of Computational and Applied Mathematics*, 330:307–317.

[Iyiola and Wade, 2018] Iyiola, O. and Wade, B. (2018). Exponential integrator methods for systems of non-linear space-fractional models with super-diffusion processes in pattern formation. *Computers & Mathematics with Applications*.

[Iyiola and Wade, 2017] Iyiola, O. S. and Wade, B. (2017). Exponential integrator methods for nonlinear fractional reaction-diffusion models.

[Janssen, 2009] Janssen, B. (2009). *An efficient exponential time differencing method for nonlinear reaction diffusion problems*. The University of Wisconsin-Milwaukee.

[Khaliq et al., 2009] Khaliq, A., Martin-Vaquero, J., Wade, B., and Yousuf, M. (2009). Smoothing schemes for reaction-diffusion systems with nonsmooth data. *Journal of Computational and Applied Mathematics*, 223(1):374–386.

[Kleefeld et al., 2012] Kleefeld, B., Khaliq, A., and Wade, B. (2012). An etd crank-nicolson method for reaction-diffusion systems. *Numerical Methods for Partial Differential Equations*, 28(4):1309–1335.

[Macías-Díaz, 2018] Macías-Díaz, J. (2018). A dynamically consistent method to solve nonlinear multidimensional advection-reaction equations with fractional diffusion. *Journal of Computational Physics (to appear)*.

[Ortigueira, 2006] Ortigueira, M. D. (2006). Riesz potential operators and inverses via fractional centred derivatives. *International Journal of Mathematics and Mathematical Sciences*, 2006.

[Padé, 1892] Padé, H. (1892). *Sur la représentation approchée d'une fonction par des fractions rationnelles*. Number 740. Gauthier-Villars et fils.

[Podlubny, 1998] Podlubny, I. (1998). *Fractional differential equations: an introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications*, volume 198. Elsevier.

[Shukla et al., 2014] Shukla, H., Tamsir, M., Srivastava, V. K., and Kumar, J. (2014). Numerical solution of two dimensional coupled viscous burger equation using modified cubic b-spline differential quadrature method. *AIP Advances*, 4(11):117134.

[Vigo-Aguiar et al., 2007] Vigo-Aguiar, J., Martín-Vaquero, J., and Wade, B. (2007). Adapted bdf algorithms applied to parabolic problems. *Numerical Methods for Partial Differential Equations*, 23(2):350–365.

[Voss and Khaliq, 1995] Voss, D. and Khaliq, A. (1995). Parallel lod methods for second order time dependent pdes. *Computers & Mathematics with Applications*, 30(10):25–35.

[Wang et al., 2016] Wang, Y., Navon, I. M., Wang, X., and Cheng, Y. (2016). 2d burgers equation with large reynolds number using pod/deim and calibration. *International Journal for Numerical Methods in Fluids*, 82(12):909–931.

[Yousuf et al., 2012] Yousuf, M., Khaliq, A., and Kleefeld, B. (2012). The numerical approximation of nonlinear black–scholes model for exotic path-dependent american options with transaction cost. *International Journal of Computer Mathematics*, 89(9):1239–1254.

[Zhao et al., 2011] Zhao, G., Yu, X., and Zhang, R. (2011). The new numerical method for solving the system of two-dimensional burgers equations. *Computers & Mathematics with Applications*, 62(8):3279–3291.

[Zhu et al., 2010] Zhu, H., Shu, H., and Ding, M. (2010). Numerical solutions of two-dimensional burgers equations by discrete adomian decomposition method. *Computers & Mathematics with Applications*, 60(3):840–848.

Appendix

A Convergence Tables for FETD-CN

alpha	tau	error	time	rate
2	0.1	2.3155e-05	2.3103	0
2	0.05	7.0117e-06	4.61	1.7235
2	0.025	1.9563e-06	9.0237	1.8416
2	0.0125	5.1897e-07	17.8206	1.9144
1.9	0.1	2.0642e-05	2.5439	0
1.9	0.05	6.1994e-06	4.2606	1.7354
1.9	0.025	1.7201e-06	8.2057	1.8496
1.9	0.0125	4.547e-07	16.9734	1.9195
1.8	0.1	1.8145e-05	2.267	0
1.8	0.05	5.4101e-06	4.2774	1.7459
1.8	0.025	1.4938e-06	8.4038	1.8567
1.8	0.0125	3.9375e-07	16.3922	1.9236
1.7	0.1	1.5867e-05	2.2648	0
1.7	0.05	4.6452e-06	4.2435	1.7722
1.7	0.025	1.2772e-06	8.2428	1.8628
1.7	0.0125	3.3583e-07	16.5976	1.9272

(a) Burgers-Fisher

alpha	tau	error	time	rate
2	0.1	0.00065897	2.2262	0
2	0.05	0.00017502	4.3828	1.9127
2	0.025	4.5271e-05	8.9188	1.9509
2	0.0125	1.1526e-05	17.0885	1.9737
1.9	0.1	0.00077894	2.2458	0
1.9	0.05	0.00020452	4.2744	1.9293
1.9	0.025	5.2495e-05	8.2422	1.962
1.9	0.0125	1.3305e-05	16.3815	1.9802
1.8	0.1	0.00068406	2.4822	0
1.8	0.05	0.00017929	4.3074	1.9318
1.8	0.025	4.5964e-05	8.2964	1.9637
1.8	0.0125	1.1641e-05	16.4522	1.9812
1.7	0.1	0.00060994	2.2126	0
1.7	0.05	0.00015967	4.2384	1.9336
1.7	0.025	4.09e-05	8.2829	1.9649
1.7	0.0125	1.0354e-05	16.6719	1.9819

(b) Burgers-Fisher

alpha	tau	error	time	rate
2	0.1	0.00077714	2.1356	0
2	0.05	0.00022698	4.3758	1.7756
2	0.025	6.2007e-05	8.6922	1.8721
2	0.0125	1.6258e-05	17.4413	1.9313
1.9	0.1	0.00073135	2.1313	0
1.9	0.05	0.00021198	4.2146	1.7866
1.9	0.025	5.7622e-05	9.4158	1.8793
1.9	0.0125	1.5065e-05	16.996	1.9354
1.8	0.1	0.00068655	2.1282	0
1.8	0.05	0.00019753	4.319	1.7973
1.8	0.025	5.3442e-05	8.464	1.8861
1.8	0.0125	1.3934e-05	17.3509	1.9393
1.7	0.1	0.00064275	2.0635	0
1.7	0.05	0.00018362	4.2391	1.8075
1.7	0.025	4.9455e-05	8.7534	1.8925
1.7	0.0125	1.2862e-05	17.0628	1.943

(c) Burgers-Huxley

Figure 10: Convergence tables for FETD-CN ($\lambda = 1$)

alpha	tau	error	time	rate
2	0.1	5.4824e-06	2.3271	0
2	0.05	1.4227e-06	4.4552	1.9462
2	0.025	3.7164e-07	8.8051	1.9366
2	0.0125	9.5607e-08	17.2944	1.9587
1.9	0.1	5.5929e-06	2.3306	0
1.9	0.05	1.4434e-06	4.2514	1.9541
1.9	0.025	3.6796e-07	8.2908	1.9719
1.9	0.0125	9.3066e-08	16.4315	1.9832
1.8	0.1	5.7277e-06	2.2337	0
1.8	0.05	1.4713e-06	4.2354	1.9609
1.8	0.025	3.7385e-07	8.3553	1.9765
1.8	0.0125	9.4355e-08	16.6716	1.9863
1.7	0.1	5.8881e-06	2.2841	0
1.7	0.05	1.5066e-06	4.2863	1.9665
1.7	0.025	3.8179e-07	8.3656	1.9804
1.7	0.0125	9.6171e-08	16.365	1.9891

(a) Burgers-Fisher

alpha	tau	error	time	rate
2	0.1	0.00043479	2.1926	0
2	0.05	0.00011511	4.4857	1.9174
2	0.025	2.9717e-05	8.5518	1.9536
2	0.0125	7.5578e-06	16.8442	1.9752
1.9	0.1	0.0007191	2.2258	0
1.9	0.05	0.00018868	4.2006	1.9302
1.9	0.025	4.8405e-05	8.3405	1.9627
1.9	0.0125	1.2265e-05	16.6515	1.9807
1.8	0.1	0.00065044	2.1516	0
1.8	0.05	0.00017042	4.2879	1.9323
1.8	0.025	4.3679e-05	8.3202	1.9641
1.8	0.0125	1.1061e-05	16.3032	1.9815
1.7	0.1	0.00058791	2.2534	0
1.7	0.05	0.00015387	4.2198	1.9339
1.7	0.025	3.9408e-05	8.2508	1.9651
1.7	0.0125	9.9753e-06	16.4049	1.9821

(b) Burgers-Fisher

alpha	tau	error	time	rate
2	0.1	0.00077684	2.6487	0
2	0.05	0.00022688	4.8092	1.7757
2	0.025	6.1977e-05	8.6998	1.8722
2	0.0125	1.625e-05	17.4487	1.9313
1.9	0.1	0.00073109	2.1535	0
1.9	0.05	0.0002119	4.2594	1.7867
1.9	0.025	5.7596e-05	8.4969	1.8793
1.9	0.0125	1.5058e-05	17.1801	1.9354
1.8	0.1	0.00068633	2.1643	0
1.8	0.05	0.00019746	4.2916	1.7973
1.8	0.025	5.342e-05	8.4563	1.8861
1.8	0.0125	1.3929e-05	17.006	1.9393
1.7	0.1	0.00064256	2.1265	0
1.7	0.05	0.00018356	4.2651	1.8076
1.7	0.025	4.9437e-05	8.5151	1.8926
1.7	0.0125	1.2857e-05	16.9298	1.943

(c) Burgers-Huxley

Figure 11: Convergence tables for FETD-CN ($\lambda = 0.1$)

B Programs

B.1 Produce Solutions

```
function produceSolutions()
    %exact solutions has to be produced before to save the right errors
    %afters using produceSolutions the first times all result gets saved
    %next time it can be run in mode 'loade'
    mode='first';
    methodList=["fetdrdp","fetdcn","diaz"];
    reactionList=["huxley","fisher","homogen"];
    lambdaList=[1,0.1];
    for method = methodList
        for reaction = reactionList
            for lambda = lambdaList
                skript_example58(mode,method,reaction,lambda);
            end
        end
    end
end

%run the examle used in paper by macias-diaz
function skript_example58(mode,method,reaction,lambda)
    addpath('./fetdrdp');
    addpath('./fetdcn');
    addpath('./diaz');

    %values from example58
    a=-200;
    b=200.;
    T=12.;
    h=1.;
    M=(b-a)/h;
    j=0:M;
    x=a+h*j;

    %fractional parameter
    alpha=[2 1.9 1.8 1.7];

    %time step
    if mode=='exact'
        tau=[0.0001];
    else
        tau=[0.1 0.05 0.025 0.0125];
    end

    %vary time step->solutions in different directories test1,2,...
    for j=1:length(tau)
        t=0:tau(j):T;
        N=length(t)-1;

        %vary fractional parameter->results as sol1,2,...
        for i=1:length(alpha)
```

```
50
51          %save solutions in first calculation
52          if mode=='first'
53              tic
54              sol=eval(strcat(method,'(lambda,t,x,M,N,a,b,alpha(i),tau(j),
                    reaction)'));
55              elapsedTime=toc;
56              %save solutions
57              %saving takes some time if you do not need it uncomment it
58              path=char(strcat('./',method,'/',reaction,'/example58_',
                    num2str(lambda*10000),'/test',num2str(j)));
59              mkdir(strcat(path));
60              save(strcat(path,'/sols',num2str(i)), 'sol');
61              %save elapsed time for computation
62              fileID = fopen(strcat(path,'/timeanderrors.txt'),'a');
63              fprintf(fileID,'%5s %d\n',strcat('time',num2str(i)),
                    elapsedTime);
64              %save error in file (relative, infinity norm)
65              load(strcat('./refSolution/fetdcn/',reaction,'/example58_',
                    num2str(lambda*10000),'/exactSol',num2str(i)),'exactSol');
66              err=norm(abs(sol(end,:)-exactSol(end,:)),inf)/norm(exactSol(
                    end,:));
67              fprintf(fileID,'%6s %d\n',strcat('error',num2str(i)),err);
68              fclose(fileID);
69              makePics(alpha,i,j,x,t,tau,sol,path)
70
71          %load solutions in next uses
72          elseif mode=='loade'
73              path=char(strcat('./',method,'/',reaction,'/example58_',
                    num2str(lambda*10000),'/test',num2str(j)));
74              load(strcat(path,'/sols',num2str(i)));
75              makePics(alpha,i,j,x,t,tau,sol,path)
76
77          %in case of calculating reference solution
78          elseif mode=='exact'
79              tic
80              exactSol=eval(strcat(method,'(lambda,t,x,M,N,a,b,alpha(i),tau(
                    j),reaction)'));
81              elapsedTime=toc;
82              %save solutions
83              path=char(strcat('./refSolution/',method,'/',reaction,'/
                    example58_',num2str(lambda*10000)));
84              mkdir(path);
85              save(strcat(path,'/exactSol',num2str(i)), 'exactSol');
86              %save elapsed time for computation
87              fileID = fopen(strcat(path,'/timeanderrors.txt'),'a');
88              fprintf(fileID,'%5s %d\n',strcat('time',num2str(i)),
                    elapsedTime);
89              fclose(fileID);
90              makePics(alpha,i,j,x,t,tau,exactSol,path)
91          end
92      end
93
94      figure(length(alpha)+1)
```

```
95          xlabel('x','FontSize',13)
96          ylabel(strcat('u(x,',num2str(T),')'),'FontSize',13);
97          legend({'\alpha=2.0','\alpha=1.9','\alpha=1.8','\alpha=1.7'},'FontSize
                ',13);
98          if mode~='exact'
99              print(strcat('-f',num2str(length(alpha)+1)),strcat('./',method,'/'
                    ,reaction,'/example58_',num2str(lambda*10000),'/test',num2str(
                    j),'/comparefrac',num2str(length(alpha)+1)),'-dpng');
100         end
101         figure(length(alpha)+1)
102         hold off
103     end
104     hold off
105 end
106
107 %method makes 2D plots considering different fractional parameters
108 %and 3d plots to plot surface at over different time steps
109 function makePics(alpha,i,j,x,t,tau,sol,path)
110             %2D-plot
111             figure(length(alpha)+1)
112             plot(-100:100,sol(end,100:300),'LineWidth',1.3);
113             hold on
114
115             %3D-plot
116             figure(i)
117             surf(x(101:301),t(1/tau(j):1/tau(j):end),sol(1/tau(j):1/tau(j):end
                    ,101:301));
118             az=40;el=40;view(az,el);
119             xlabel('x','FontSize',13);
120             ylabel('t','FontSize',13);
121             zlabel('u(x,t)','FontSize',13);
122             print(strcat('-f',num2str(i)),strcat(path,'/surf',num2str(i)),'-
                    dpng');
123 end
```

B.2 Produce Exact Solutions

```
function produceExactSolutions()
    %this method produces highly exact results which are used as reference
    %solution when comparing numerical results
    mode='exact';
    methodList=["fetdrdp", "fetdcn", "diaz"];
    reactionList=["huxley","fisher","homogen"];
    lambdaList=[1, 0.1];
    for method = methodList
        for reaction = reactionList
            for lambda = lambdaList
                skript_example58(mode,method,reaction,lambda);
            end
        end
    end
end

%run the examle used in paper by macias-diaz
function skript_example58(mode,method,reaction,lambda)
    addpath('./fetdrdp');
    addpath('./fetdcn');
    addpath('./diaz');

    %values from example58
    a=-200;
    b=200.;
    T=12.;
    h=1.;
    M=(b-a)/h;
    j=0:M;
    x=a+h*j;

    %fractional parameter
    alpha=[2 1.9 1.8 1.7];

    %time step
    if mode=='exact'
        tau=[0.0001];
    else
        tau=[0.1 0.05 0.025 0.0125];
    end

    %vary time step->solutions in different directories test1,2,...
    for j=1:length(tau)
        t=0:tau(j):T;
        N=length(t)-1;

        %vary fractional parameter->results as sol1,2,...
        for i=1:length(alpha)

            %save solutions in first calculation
            if mode=='first'
```

```
52                  tic
53                  sol=eval(strcat(method,'(lambda,t,x,M,N,a,b,alpha(i),tau(j),
                        reaction)'));
54                  elapsedTime=toc;
55                  %save solutions
56                  path=char(strcat('./',method,'/',reaction,'/example58_',
                        num2str(lambda*10000),'/test',num2str(j)));
57                  mkdir(strcat(path));
58                  save(strcat(path,'/sols',num2str(i)), 'sol');
59                  %save elapsed time for computation
60                  fileID = fopen(strcat(path,'/timeanderrors.txt'),'a');
61                  fprintf(fileID,'%5s %d\n',strcat('time',num2str(i)),
                        elapsedTime);
62                  %save error in file (relative, infinity norm)
63                  load(strcat('./refSolution/fetdcn/',reaction,'/example58_',
                        num2str(lambda*10000),'/exactSol',num2str(i)),'exactSol');
64                  err=norm(abs(sol(end,:)-exactSol(end,:)),inf)/norm(exactSol(
                        end,:));
65                  fprintf(fileID,'%6s %d\n',strcat('error',num2str(i)),err);
66                  fclose(fileID);
67                  makePics(alpha,i,j,x,t,tau,sol,path)
68
69              %load solutions in next uses
70              elseif mode=='loade'
71                  path=char(strcat('./',method,'/',reaction,'/example58_',
                        num2str(lambda*10000),'/test',num2str(j)));
72                  load(strcat(path,'/sols',num2str(i)));
73                  makePics(alpha,i,j,x,t,tau,sol,path)
74
75              %in case of calculating reference solution
76              elseif mode=='exact'
77                  tic
78                  exactSol=eval(strcat(method,'(lambda,t,x,M,N,a,b,alpha(i),tau(
                        j),reaction)'));
79                  elapsedTime=toc;
80                  %save solutions
81                  path=char(strcat('./refSolution/',method,'/',reaction,'/
                        example58_',num2str(lambda*10000)));
82                  mkdir(path);
83                  save(strcat(path,'/exactSol',num2str(i)), 'exactSol');
84                  %save elapsed time for computation
85                  fileID = fopen(strcat(path,'/timeanderrors.txt'),'a');
86                  fprintf(fileID,'%5s %d\n',strcat('time',num2str(i)),
                        elapsedTime);
87                  fclose(fileID);
88                  makePics(alpha,i,j,x,t,tau,exactSol,path)
89              end
90          end
91          if mode~='exact'
92              print(strcat('-f',num2str(length(alpha)+1)),strcat('./',method,'/'
                    ,reaction,'/example58_',num2str(lambda*10000),'/test',num2str(
                    j),'/comparefrac',num2str(length(alpha)+1)),'-dpng');
93          end
94          hold off
```

```
    end
    hold off
end

function makePics(alpha,i,j,x,t,tau,sol,path)
            %2D-plot
            figure(length(alpha)+1)
            plot(-100:100,sol(end,100:300));
            hold on

            %3D-plot
            figure(i)
            surf(x(101:301),t(1/tau(j):1/tau(j):end),sol(1/tau(j):1/tau(j):end
                ,101:301));
            az=40;el=40;view(az,el);
            xlabel('x');
            ylabel('t');
            zlabel('u(x,t)');
            print(strcat('-f',num2str(i)),strcat(path,'/surf',num2str(i)),'-
                dpng');
end
```

B.3 Convergence and Efficiency

```
function convergenceandefficiency()
    %all numerical results can be reproduced by this method
    %the programs for each case has to be run before
    %the method creates convergence and efficiency plots for all given
    %parameters
    %furthermore it makes one convergence table regarding all given fractional
    %parameters for a comparison
    methodList=["fetdrdp","fetdcn","diaz"];
    reactionList=["huxley", "fisher", "homogen"];
    lambdaList=[1, 0.1];
    alphaList=[1 2 3 4];%means alpha=2 1.9 1.8 1.7
    stepList=["test1","test2","test3","test4"];
    produceData(methodList,reactionList,lambdaList,alphaList,stepList);
end

function produceData(methodList,reactionList,lambdaList,alphaList,stepList)
    row={'2' '1.9' '1.8' '1.7'};
    for lambda = lambdaList %set lambda
        path=strcat('./Thesis/convergence/example58_',num2str(lambda*10000));
        mkdir(path);
        for reaction = reactionList %set reaction term
            conv=zeros(length(stepList)*length(alphaList),5,length(methodList)
                );
            for alpha=alphaList %set alpha
                table1=[];
                table2=[];
                for method = methodList
                    steparray1=[];
                    steparray2=[];
                    for step=stepList
                        %collect data for convergence and efficiency plot
                        path=char(strcat('./',method,'/',reaction,'/example58_
                            ',num2str(lambda*10000),'/',step,'/timeanderrors.
                            txt'));
                        fileID = fopen(path,'r');
                        formatSpec = '%s %12.6f';
                        A = textscan(fileID,formatSpec);
                        %convergence
                        steparray1=[steparray1 (A{2}(2*alpha))];
                        %efficiency
                        steparray2=[steparray2 (A{2}(2*alpha-1))];
                        fclose(fileID);
                    end
                    table1=[table1;steparray1];
                    table2=[table2;steparray2];
                end

                %generate convergence plot for each method
                xvalues=[0.1 0.05 0.025 0.0125];
                for i=1:length(methodList)
                    loglog(xvalues,table1(i,:),'-x','LineWidth',1.3);
```

```
                hold on
            end
            xlabel('time step','FontSize',13);
            ylabel('relative error','FontSize',13);
            legend({"FETD-RDP","FETD-CN","FD"},'FontSize',13);
            path=strcat('./Thesis/convergence/example58_',num2str(lambda
                *10000),'/',char(reaction));
            mkdir(path);
            print(strcat('-f',num2str(1)),strcat(path,'/
                fractionalparametercase',num2str(alpha)),'-dpng');
            hold off

            %Generate efficiency plot for each method
            for i=1:length(methodList)
                loglog(table1(i,:),table2(i,:),'-x','LineWidth',1.3);
                hold on
            end
            xlabel('relative error','FontSize',13);
            ylabel('time','FontSize',13);
            legend({"FETD-RDP","FETD-CN","FD"},'FontSize',13);
            path=strcat('./Thesis/efficiency/example58_',num2str(lambda
                *10000),'/',char(reaction));
            mkdir(path);
            print(strcat('-f',num2str(1)),strcat(path,'/
                fractionalparametercase',num2str(alpha)),'-dpng');
            hold off

            %collect data for convergence table for every method
            for j=1:length(methodList)
                rate=[0];
                alphavec=[str2double(row{alpha})];
                for i=1:length(stepList)-1
                    tmp=log(table1(j,i)/table1(j,i+1))/log(2);
                    rate=[rate tmp];
                    alphavec=[alphavec str2double(row{alpha})];
                end
                subtable=[alphavec' xvalues' table1(j,:)' table2(j,:)'
                    rate'];
                (alpha)*length(stepList)
                (alpha-1)*length(stepList)+1
                for m=1:5
                    conv((alpha-1)*length(stepList)+1:alpha*length(
                        stepList),m,j)=subtable(:,m);
                end
            end
        end

        %Generate latex convergence matrix for each method after
        %considering of all alphas
        for j=1:length(methodList)
            path=strcat('./Thesis/table/example58_',num2str(lambda*10000),
                '/',char(reaction));
            mkdir(path);
            col={'alpha' 'tau' 'error' 'time' 'rate'};
```

```
96                  matrix2latex(conv(:,:,j),strcat(path,'/',methodList(j),'
                        _fractionalparametercaseall','.tex'),'columnLabels',col,'
                        alignment', 'c');
97              end
98          end
99      end
100 end
```

B.4 Numerical Methods

```
function y=fetdrdp(lambda,t,x,M,N,a,b,alpha,tau,reaction)
    %t is the time interval partioned in N+1 parts by tau
    %[a,b] is the space interval and should get partitioned in M+1 parts
    %u0=[ksi(x0),fi(x1),...,fi(xM-1),ksi(xM)] is vector for t=0 defined on [a,
        b]
    %ksi is the function for the boundaries a and b
    %initializations
    h=(b-a)/M;
    A=zeros(M+1,M+1);
    coeffs=g(M,alpha);
    u=fi(a,b,M,h);
    y=zeros(N,M+1);

    %Construct A
    for i=1:M
        for j=1:i-1
            A(i,j)=coeffs(i-j+1,1); %make use of on diagonals always the same
                ??
        end
    end
    A(1,:)=zeros(M+1,1);
    A(M+1,:)=zeros(M+1,1);
    A=(A+transpose(A)+eye(M+1)*coeffs(1,1))*(1/(h^alpha)); %use symmetry

    %Construct B
    e = ones(M+1,1);
    B = spdiags([-1*e 0*e 1*e], -1:1, M+1, M+1);
    B(1,1)=1;
    B(1,2)=0;
    B(M+1,M+1)=1;
    B(M+1,M-1)=0;
    B=1/(2*h)*B;

    I=eye(M+1,M+1);
    %Construct F
    %attention fetp does only get applied on inner knots
    corr=u(:,1);
    for k=1:N %time steps
        F_corr=F(corr,lambda,B,M,reaction);

        pred=(I+A*tau)\(corr+tau*F_corr);
        F_pred=F(pred,lambda,B,M,reaction);

        an=(I+1./3*A*tau)\(9*corr+2*tau*F_corr+tau*F_pred);
        bn=(I+1./4*A*tau)\(-8*corr-3/2*tau*F_corr-1/2*tau*F_pred);
        corr=an+bn;

        y(k,:)=corr;
    end
end

```

```
%Gruenwald-Letnikov-Coefficients iterative
function coeffs=g(M,alpha)
    coeffs=zeros(M+1,1);
    coeffs(1,1)=gamma(alpha+1)/(gamma(alpha/2.+1)^2);
    for i=2:M+1
        coeffs(i,1)=coeffs(i-1,1)*(1-((alpha+1)/(alpha/2.+(i-2)+1)));
    end
end

function y=fisher(u)
    y=1-u;
end

function y=huxley(u)
    gamma=0.005;
    y=(1-u).*(u-gamma);
end

function y=homogen(u)
    y=0*u;
end


%function for non-linear part
function y=F(u,lambda,B,M,reaction)
    y=zeros(M+1,1);
    y(:,1)=u(:,1);
    y=-lambda.*y.*(B*y)+y.*eval(strcat(reaction,'(y)'));
    y=y(:,1);
end


%functions for boundary and initial conditions
function y=ksi1(t)
    y=0;
end
function y=ksi2(t)
    y=0;
end
function y=fi(a,b,M,h)
    counter=1;
    y=zeros(M+1,1);
    for x=a:h:b
        if x==0
            y(counter,1)=0.1;
        else
            y(counter,1)=0;
        end
        counter=counter+1;
    end
    y(1)=ksi1(0);
    y(M+1)=ksi2(0);
end
```

```
function y=fetdcn(lambda,t,x,M,N,a,b,alpha,tau,reaction)
    %t is the time interval partioned in N+1 parts by tau
    %[a,b] is the space interval and should get partitioned in M+1 parts
    %u0=[ksi(x0),fi(x1),...,fi(xM-1),ksi(xM)] is vector for t=0 defined on [a,
        b]
    %ksi is the function for the boundaries a and b

    %initializations
    h=(b-a)/M;
    A=zeros(M+1,M+1);
    coeffs=g(M,alpha);
    u=fi(a,b,M,h);
    y=zeros(N,M+1);

    %Construct A
    for i=1:M
        for j=1:i-1
            A(i,j)=coeffs(i-j+1,1); %make use of on diagonals always the same
                ??
        end
    end
    A(1,:)=zeros(M+1,1);
    A(M+1,:)=zeros(M+1,1);
    A=(A+transpose(A)+eye(M+1)*coeffs(1,1))*(1/(h^alpha)); %use symmetry

    %Construct B
    e = ones(M+1,1);
    B = spdiags([-1*e 0*e 1*e], -1:1, M+1, M+1);
    B(1,1)=1;
    B(1,2)=0;
    B(M+1,M+1)=1;
    B(M+1,M-1)=0;
    B=1/(2*h)*B;
    I=eye(M+1,M+1);
    %Construct F
    %attention fetd does only get applied on inner knots
    %inital damping step we can calculate directly
    pred=u(:,1);
    F_pred=F(pred,lambda,B,M,reaction);
    corr=(I+tau*A)\(pred+tau*F_pred);
    y(1,:)=corr;

    for k=2:N %time steps
        F_corr=F(corr,lambda,B,M,reaction);

        an=(2*I+tau*A)\(4*corr+2*tau*F_corr);
        pred=(-1)*corr+an;
        F_pred=F(pred,lambda,B,M,reaction);

        bn=(2*I+tau*A)\(tau*(F_pred-F_corr));
        corr=pred+bn;
        y(k,:)=corr;
    end
end
```

```
53
54 %Gruenwald-Letnikov-Coefficients iterative
55 function coeffs=g(M,alpha)
56     coeffs=zeros(M+1,1);
57     coeffs(1,1)=gamma(alpha+1)/(gamma(alpha/2.+1)^2);
58     for i=2:M+1
59         coeffs(i,1)=coeffs(i-1,1)*(1-((alpha+1)/(alpha/2.+(i-2)+1)));
60     end
61 end
62
63 function y=fisher(u)
64     y=1-u;
65 end
66
67 function y=huxley(u)
68    gamma=0.005;
69    y=(1-u).*(u-gamma);
70 end
71
72 function y=homogen(u)
73    y=0*u;
74 end
75
76 %function for non-linear part
77 function y=F(u,lambda,B,M,reaction)
78     y=zeros(M+1,1);
79     y(:,1)=u(:,1);
80     y=-lambda.*y.*(B*y)+y.*eval(strcat(reaction,'(y)'));
81     y=y(:,1);
82 end
83
84
85 %functions for boundary and initial conditions
86 function y=ksi1(t)
87     y=0;
88 end
89 function y=ksi2(t)
90     y=0;
91 end
92
93 function y=fi(a,b,M,h)
94     counter=1;
95     y=zeros(M+1,1);
96     for x=a:h:b
97         if x==0
98             y(counter,1)=0.1;
99         else
100            y(counter,1)=0;
101        end
102        counter=counter+1;
103    end
104    y(1)=ksi1(0);
105    y(M+1)=ksi2(0);
106 end
```

```
function sol=diaz(lambda,t,x,M,N,a,b,alpha,tau,reaction)
    %M+1 is the space dimension
    %N+1 is the time dimension
    %time interval is [0,T] and t is the mesh on it
    %spatial interval is [a,b]
    %alpha is order of fractional derivation
    %ksi1 and ksi2 are the boundary cond. while fi is the initial condition
    %(in seperate functions above)

    %INITILIAZATION
    %start coefficients
    gk=coefficients(alpha,M-2);
    %spatial mesh step
    h=(b-a)/M;
    %initial condition
    u=zeros(N,M+1); %actually we only need u as a vector
    for i=1:M+1
        u(1,i)=fi(x(i)); %from  0 to M
    end
    %matrix
    A=zeros(M+1,M+1);
    A(1,1)=1;
    A(M+1,M+1)=1;
    %NUMERICAL METHOD
    for n=0:N-1
        %time step
        tau=t(n+2)-t(n+1);

        %A
        eta=1+tau/(h^alpha)*gk(1)-tau*eval(strcat(reaction,'(u(n+1,2:M))'));
        d1=tau/(h^alpha)*gk(2);
        d2=((tau*lambda)/(2*h))*u(n+1,2:M);
        delta_p=d1+d2;
        delta_m=d1-d2;
        gkhat=tau/(h^alpha)*gk(3:length(gk));%gkhat only includes g2 to gM-1
        for i=2:M
            A(i,i)=eta(i-1);
            A(i,i-1)=delta_m(i-1);
            A(i,i+1)=delta_p(i-1);
        end
        for j=1:M-2%sub diagonals
            for i=(j+2):M
                A(i,j)=gkhat(i-j-1);
            end
        end
        for j=4:(M+1)%super diagonals
            for i=2:j-2
                A(i,j)=gkhat(j-i-1);
            end
        end

        %v_u^n
        v=transpose([ksi1(t(n+2)) u(n+1,2:M) ksi2(t(n+2))]);
        %final step, solve equation system
```

```
55          u(n+2,:)=A\v;
56      end
57      sol=u(:,:);
58  end
59
60  %iterative method for coefficient of the fractional centred differences
61  function gk=coefficients(alpha,K) %K=0,... caluclates g1,...
62      gk=zeros(K+2);
63      gk(1)=gamma(alpha+1)/(gamma(alpha/2+1)^2);
64      for k=0:K
65          gk(k+2)=(1-((alpha+1)/(alpha/2+k+1)))*gk(k+1);
66      end
67  end
68
69  function y=fisher(u)
70      y=1-u;
71  end
72
73  function y=huxley(u)
74      gamma=0.005;
75      y=(1-u).*(u-gamma);
76  end
77
78  function y=homogen(u)
79      y=0*u;
80  end
81
82  %functions for boundary and initial conditions
83  function y=ksi1(t)
84      y=0;
85  end
86  function y=ksi2(t)
87      y=0;
88  end
89  function y=fi(x)
90      if x==0
91          y=0.1;
92      else
93          y=0;
94      end
95  end
```