Splitting Methods for a Blow-up Reaction-Diffusion Equation

APMA 922	Final Project
Submitted by:	Nathan King
Student #:	301222376
Submitted to:	Dr. Ralf Wittenberg
Date:	December 16, 2013

Contents

1 Introduction

Originating in the theory of solid fuel combustion, the Frank-Kamenetskii equation

$$u_t = \Delta u + e^u, \tag{1}$$

has been immensely studied. This PDE was derived by David Albertovic Frank-Kamenetskii, a Soviet theoretical physicist and chemist, in 1938 [1]. The Frank-Kamenetskii equation is of interest because it has a blow-up solution, that is

$$\lim_{t \to T_b} \|u(\mathbf{x}, t)\|_\infty = \infty,$$

where $T_b < \infty$ denotes the blow-up time.

In 1966, Fujita [2] gave the first indepth study of blow-up equations of this form. The series of four papers [2, 3, 4, 5] started with the problem $u_t = \Delta u + u^{1+\alpha}$, with $\alpha > 0$. The papers then extended to the Frank-Kamenetskii equation and more general reaction terms $F(u)$. The main result is that for certain conditions on $F(u)$ and large enough initial data (dependant on dimension) the PDE

$$u_t = \Delta u + F(u),$$

has a blow-up solution. From this work and extensions by others, results of where, when and how the blow-up occurs are known. The question of when is answered by upper and lower bounds on the blow-up time T_b [2]. The location of blow-up for $u_t = \Delta u + u^{1+\alpha}$ is known to be at only $x = 0$ [6]. The growth rate of the solution is also known for many problems. The historical review in [7] provides a summary of results known and numerous citations.

A numerical solution is difficult due to the large growth of the solution in small time intervals as $t \to T_b$. Methods for approximating the solution numerically began in the mid 1970s with the work of Nawagawa [8]. For a survey of numerical methods studied up to 1998 see Bandle and Brunner [9]. This paper is concerned with splitting methods [10], which are also called fractional step methods [11]. Splitting methods are applied to the reaction-diffusion equation

$$\begin{aligned} &u_t = u_{xx} + u^{1+\alpha}, \quad t > 0, \\ &u(-1, t) = u(1, t) = 0, \\ &u(x, 0) = \beta \cos(\pi x/2), \end{aligned} \tag{2}$$

where $\alpha > 0$ and $\beta \in \mathbb{R}$.

The methods explored here follow from Beck [7]. Splitting methods are also describe by Hairer, Wanner and Lubich [10] and historical developments are given in [12]. The main purpose of this paper is to show that the splitting approach for time stepping is superior to standard methods for PDE (2). Analysis must compare one-step errors between standard and splitting methods. Another basis of comparison is the growth rate of the numerical method compared to the theoretical growth rate. Note that we are mostly concerned with time stepping for PDE (2) and will not consider the spatial discretizations theoretically here.

Next splitting methods in general are described in section 2.1. Some specific splitting methods are derived in section 2.2 for the PDE (2). The one-step error and the growth rate

of first order methods are given in sections 2.3 and 2.4, respectively. Section 2.3 compares the one-step error of the standard backward Euler (BE) to that of a splitting method constructed with backward Euler (SpBE). Whereas, section 2.4 proves the growth rate for SpBE follows the theoretical growth rate given by Friedman and McLeod [13]. A numerical study of PDE (2) is carried out in section 3 for $\alpha = 1$. Sections 3.1 examines BE, SpBE, Crank-Nicolson (CN) and a splitting method constructed with Crank-Nicolson (SpCN). The latter all deals with fixed time step methods, however, adaptive time stepping is desirable. Since the solution increases rapidly as $t \to T_b$ the step size needed decreases as $t \to T_b$. It is therefore inefficient to use fixed time steps and adaptive time stepping is explored in section 3.2. Adaptive time stepping methods are implemented using first and second order pairs BE–CN and SpBE–SpCN.

This work began in the thesis [14] and during an Undergraduate Student Research Award (USRA) from the Natural Sciences and Engineering Research Council (NSERC). The thesis [14] focused on six adaptive time stepping initial value problem (IVP) solvers. The IVP solvers were applied to $y' = \lambda y + y^{1+\alpha}$, which is a scalar IVP that captures the form of one equation in the MOL IVP system for the PDE (2). The six IVP solvers were then applied to the full MOL system of IVPs from PDE (2). The initial condition and boundary conditions were different in [14]: $u(0,t) = u(\pi,t) = 0$ and $u(x,0) = \beta \sin(x)$. Splitting methods for blow-up problems were then explored during the USRA. Inspired by Beck [7], the goal was to apply the same techniques to blow-up integro-differential equations. Some of the methods in [7] were coded, namely SpBE and a second order splitting method based on explicit methods (denoted SoSpFE in [7]). The splitting method codes for SpCN and SpBE–SpCN were coded specifically for this paper, along with the standard CN and BE–CN.

The work of the USRA then turned to replicating the results of Glowinski et al. [15] for an integro-differential equation. We also tried to find a blow-up integro-differential equations where the dominant part close to blow-up possesses an exact solution. If the dominant part possessed an exact solution it would be an ideal candidate for applying the splitting methods discussed below. The main work of this project was to theoretically understand the methods in [7], which was not achieved during the USRA. None of the theoretical work presented here was explored in detail during the USRA. The majority of the methods were coded for just for this project also, as explained above. The APMA 922 course gave a foundation to be able to understand these methods fully.

2 Splitting and Analysis of First-Order Methods

2.1 The Notion of Splitting

Splitting methods are specific to each differential equation to be solved. Consider a PDE $v_t = f(v)$ that can be split into a parts

$$v_t = f^{[1]}(v) + f^{[2]}(v) + \cdots + f^{[s]}(v),$$

where $f^{[i]}(v)$ could be a function or an operator. The basic idea is to solve s subproblems $v_t = f^{[i]}(v)$, for $i = 1, 2, \ldots, s$, to obtain an approximate solution to the PDE. It is possible that this solution is exact if the operators $f^{[i]}(v)$ commute for all $i = 1, 2, \ldots, s$ (see [11]

or [12]). In general, the operators do not commute and exact solutions to subproblems are not known. In most cases numerical solutions to each subproblem are needed. This introduces numerical error along with the existing splitting error. For the PDE (2) we write $u_t = \Lambda(u) + F(u)$, where $\Lambda(u) \equiv \Delta u$ and $F(u) \equiv u^{1+\alpha}$. The reaction term, $F(u)$, causes blow-up and dominants the solution of PDE (2) as $t \to T_b$. The motivation to use a splitting approach is that $u_t = F(u)$ has an exact solution. Therefore this information can be exploited when constructing a numerical integrator. We hope that using the solution to $u_t = F(u)$ will allow the numerical method to better capture the solution behaviour. Remember methods here are only semi-discretizations in time and therefore another discretization scheme must be used for the spatial component.

Notation for splitting methods can be cumbersome. One of the simplest ways to think about splitting is as the composition of multiple flows. Each flow to a subproblem is used as the initial condition to the next subproblem. The *flow* of a differential equation over time t is a mapping, which associates the value of $v(t)$ of the solution with initial condition $v(0) = v^0$. This map, denoted φ_t, is thus defined by

$$\varphi_t(v^0) = v(t) \text{ if } v(0) = v^0.$$

If subproblems are solved exactly we have the composition of flows to each differential equation. If a numerical method is used the concept of a numerical flow must be introduced. A *numerical flow* is a map that can represent any numerical method by

$$\mathcal{N}_k : v^n \to v^{n+1},$$

which computes the solution at time t^{n+1} with time step k. For example, the backward Euler method applied to $v_t = f(v)$ is given by

$$\mathcal{N}_k(v^n) = v^n + kf(v^{n+1}).$$

To create a numerical method for the PDE (2) any numerical flow $\mathcal{N}_k^{[\Lambda]}$ for $u_t = \Lambda(u)$ can be used. Composing the exact flow $\varphi_k^{[F]}$ to $u_t = F(u)$ with the numerical flow $\mathcal{N}_k^{[\Lambda]}$ gives a numerical method for PDE (2),

$$\mathcal{N}_k = \varphi_k^{[F]} \circ \mathcal{N}_k^{[\Lambda]}. \tag{3}$$

The numerical scheme (3) is a first-order splitting method. This method is the same first order method given by LeVeque [11]. Another first order method can be constructed by computing the adjoint method of (3). The *adjoint method*, $\mathcal{B}_k^*$, of the method $\mathcal{B}_k$ is an inverse map of the original method with reversed time step $-k$, that is $\mathcal{B}_k^* = \mathcal{B}_{-k}^{-1}$ [10]. For example, backward Euler is the adjoint method of forward Euler. Note that the adjoint method satisfies the natural properties $(\mathcal{B}_k^*)^* = \mathcal{B}_k$ and $(\mathcal{B}_k \circ \mathcal{C}_k)^* = \mathcal{C}_k^* \circ \mathcal{B}_k^*$. Therefore another first-order splitting method for PDE (2) is

$$\mathcal{N}_k^* = \mathcal{N}_k^{[\Lambda]*} \circ \varphi_k^{[F]}, \tag{4}$$

where $\varphi_k^{[F]*} = \varphi_k^{[F]}$ since it is an exact flow.

Higher order methods can be constructed as presented in Hairer et al. [10]. Let $\mathcal{C}_k$ be a one-step method of order p and $\gamma_1, \ldots, \gamma_s$ be real numbers. If

$$\begin{aligned} \gamma_1 + \cdots + \gamma_s &= 1, \\ \gamma_1^{p+1} + \cdots + \gamma_s^{p+1} &= 0, \end{aligned} \tag{5}$$

then the composition method $\mathcal{B}_k = \mathcal{C}_{\gamma_1 k} \circ \cdots \circ \mathcal{C}_{\gamma_s k}$ is a method of at least order $p+1$. This says that if we take a composition of a p-th order method with the right combination of step sizes, $\gamma_1 k, \ldots, \gamma_s k$, then we can construct a splitting method of at least order $p+1$. The conditions (5) have no real solution for even p. However, a result using the adjoint method generalizes for even p. Let $\mathcal{C}_k^*$ be the adjoint method of $\mathcal{C}_k$ and both $\beta_1, \ldots, \beta_s$ and $\nu_1, \ldots, \nu_s$ be real numbers. If

$$\begin{aligned} \beta_1 + \nu_1 + \cdots + \beta_s + \nu_s &= 1, \\ (-1)^p \beta_1^{p+1} + \nu_1^{p+1} + \cdots + (-1)^p \beta_s^{p+1} + \nu_s^{p+1} &= 0, \end{aligned} \tag{6}$$

then the composition method $\mathcal{B}_k = \mathcal{C}_{\nu_1 k} \circ \mathcal{C}^*_{\beta_1 k} \circ \cdots \circ \mathcal{B}_{\nu_s k} \circ \mathcal{B}^*_{\beta_s k}$ is a method of at least order $p+1$. In this paper we also explore a second-order splitting method, which is given by $\nu_1 = \beta_1 = 1/2$ and $p = s = 1$. With these choices the method

$$\mathcal{B}_k = \mathcal{C}_{k/2} \circ \mathcal{C}^*_{k/2}, \tag{7}$$

takes any consistent first order one-step method and creates the second order symmetric method $\mathcal{B}_k$. The method (7) is *symmetric* since $\mathcal{B}_k^* = \mathcal{B}_k$.

2.2 Specific Splitting Methods

Many possible combinations of methods can be used to construct a splitting method. As mentioned in the last section, an order $p+1$ method can be constructed from an order p method. The order p method could simply be any standard one-step method. It could however be constructed from an order $p-1$ method, which could have been constructed from an order $p-2$ method, etc. Splitting methods are therefore specialized methods that are constructed to best suit the problem at hand.

For PDE (2) we would like to construct a splitting method that utilizes the exact solution of $u_t = F(u)$. The exact solution to the reaction part of $u_t = \Delta u + F(u)$ can be found in general, with some assumptions on F. One can find a derivation for the more general F in [7]. Here we explore $F(u) = u^{1+\alpha}$, which makes it easy to determine

$$\varphi_t^{[F]}(U^n) = \left[(U^n)^{-\alpha} - \alpha t \right]^{-\frac{1}{\alpha}}, \quad \text{for } \; t < \frac{1}{\alpha (U^n)^\alpha}, \tag{8}$$

where $U^n = U^n(x) \approx u(x, t^n)$ is the initial condition at time t^n. The exact flow (8) is used to construct some first and second order splitting methods. More examples of constructing splitting methods for blow-up problems are given by Beck in [7]. These include the Frank-Kamenetskii equation, a quasilinear parabolic PDE with power-type nonlinearities, a system of semilinear parabolic PDEs and a wave equation.

When constructing a first order splitting method backward Euler is used for $\mathcal{N}_k^{[\Lambda]}$. This is because $u_t = \Lambda(u)$ is the heat equation, which is a stiff problem. From (4) we have

$$\mathcal{N}_k^*(U^n) = \mathcal{N}_k^{[\Lambda]*} \circ \varphi_k^{[F]}(U^n),$$

where $\mathcal{N}_k^{[\Lambda]*}(v^n) = v^n + k\Lambda(v^{n+1})$ and $\varphi_t^{[F]}(U^n)$ is given in (8). Composition notation is simplistic, however it is not practical for implementation. Instead one writes this method as

$$\begin{aligned} V &= \left[(U^n)^{-\alpha} - \alpha\, k\right]^{-\frac{1}{\alpha}}, \\ U^{n+1} &= V + k\,\Lambda(U^{n+1}). \end{aligned} \tag{9}$$

If one chooses to use a second order central finite difference scheme for $\Lambda(v)$, the full discretization becomes

$$\begin{aligned} V_j &= \left[(U_j^n)^{-\alpha} - \alpha\, k\right]^{-\frac{1}{\alpha}}, \\ U^{n+1} &= V_j + \frac{k}{h^2}(U_{j-1}^{n+1} - 2U_j^{n+1} + U_{j+1}^{n+1}), \end{aligned} \tag{10}$$

where h is a uniform spatial step size. Since (10) is constructed using backward Euler it is called the SpBE scheme. Spectral methods to approximate u_{xx} could also be used. Due to time restrictions this was not implemented here. Spectral accuracy in the spatial derivative is not important since we only consider first and second order methods.

Using

$$\mathcal{N}_k(U^n) = \varphi_k^{[F]} \circ \mathcal{N}_k^{[\Lambda]}(U^n),$$

where $\mathcal{N}_k^{[\Lambda]}(v^n) = v^n + k\Lambda(v^n)$ is forward Euler, one can construct the second order method in (7) as

$$\begin{aligned} \mathcal{M}_k(U^n) &= \mathcal{N}_{k/2} \circ \mathcal{N}_{k/2}^*(U^n) \\ &= \varphi_{k/2}^{[F]} \circ \mathcal{N}_{k/2}^{[\Lambda]} \circ \mathcal{N}_{k/2}^{[\Lambda]*} \circ \varphi_{k/2}^{[F]}(U^n). \end{aligned} \tag{11}$$

Once again it is more practical to write this method in the form of (9). If a second order central finite difference scheme is used for $\Lambda(v)$, then $\mathcal{M}_k(U^n)$ becomes

$$\begin{aligned} V_j &= \left[(U_j^n)^{-\alpha} - \alpha\,\frac{k}{2}\right]^{-\frac{1}{\alpha}}, \\ W_j &= V_j + \frac{k}{2h^2}(W_{j-1} - 2W_j + W_{j+1}), \\ T_j &= W_j + \frac{k}{2h^2}(W_{j-1} - 2W_j + W_{j+1}), \\ U_j^{n+1} &= \left[(T_j)^{-\alpha} - \alpha\,\frac{k}{2}\right]^{-\frac{1}{\alpha}}. \end{aligned} \tag{12}$$

It is interesting to note that (12) is the well known Strang splitting scheme [16],

$$\mathcal{M}_k^{[S]} = \varphi_{k/2}^{[F]} \circ \mathcal{S}_k^{[\Lambda]} \circ \varphi_{k/2}^{[F]}.$$

One can write $\mathcal{S}_k^{[\Lambda]} = \mathcal{N}_{k/2}^{[\Lambda]} \circ \mathcal{N}_{k/2}^{[\Lambda]*}$, which is identical to (11). After discretizing in space $\mathcal{S}_k^{[\Lambda]}$ is the Crank-Nicolson scheme, therefore (12) is called the SpCN scheme.

For first and second order splitting schemes (10) and (12) we must consider boundary conditions carefully. Determining boundary conditions for each subproblem can cause difficulty when implementing the splitting approach. For PDE (2) it is simple since there are zero Dirichlet boundary conditions. This means that the solution is only computed on the interior spatial nodes and set to zero for the two end nodes. This is the extent of the splitting methods we introduce here. Higher order methods can be constructed, which basically leads to more stages in the method. The next section investigates the one-step error of SpBE and compares it to standard BE.

2.3 One-Step Error

Since splitting methods are constructed by a composition of flows we are unable to write the methods in the form of LeVeque [11] to examine local truncation error. Therefore the one-step error for splitting methods is calculated. In this section the one-step error for SpBE is calculated and compared to the one-step error of BE.

SpBE applies the exact flow first, then uses a numerical method, i.e. $\mathcal{N}_k^*(U^n) = \mathcal{N}_k^{[\Lambda]*} \circ \varphi_k^{[F]}(U^n)$. Recall we write PDE (2) as

$$u_t = \Lambda(u) + F(u),$$

where $\Lambda(u) \equiv u_{xx}$ and $F(u) \equiv u^{1+\alpha}$. To simplify notation superscripts on the flows are dropped. We also write $\mathcal{N}_k(\cdot) \equiv \mathcal{N}(k;\cdot)$ and $\varphi_k(\cdot) \equiv \varphi(k;\cdot)$ to avoid confusion with subscript notation of derivatives. The following analysis will hold for any PDE of this form where Λ and F are functions or operators.

Applying the exact flow to initial condition U^n, gives $v^0 = \varphi(k; U^n)$. Then we solve

$$\begin{aligned} v_t &= \Lambda(v), \\ v(0) &= v^0, \end{aligned} \tag{13}$$

with a numerical integrator $\mathcal{N}(k; v^0)$. Note that for any numerical flow $\mathcal{N}(k; v^0)$ we have that

$$\mathcal{N}(k; v^0) = v(k) + E(k), \tag{14}$$

where $v(k)$ is the exact solution to (13) and $E(k)$ is the one-step error of the numerical flow. The approximation to $u(x, t^{n+1})$ is given by $V = \mathcal{N}(k; v^0) = \mathcal{N}(k; \varphi(k; U^n))$.

To compute the one-step error we expand V as a series of k about $k = 0$. That is,

$$\begin{aligned} V =& \mathcal{N}(0; \varphi(0; U^n)) + k\left[\mathcal{N}_t + \mathcal{N}_v\varphi_t\right] \\ &+ \frac{k^2}{2}\left[\mathcal{N}_{tt} + 2\mathcal{N}_{tv}\varphi_t + \mathcal{N}_{vv}(\varphi_t)^2 + \mathcal{N}_v\varphi_{tt}\right] + \mathcal{O}(k^3), \end{aligned} \tag{15}$$

where all derivatives of $\mathcal{N}$ and φ are evaluated at $(0; \varphi(0; U^n))$ and $(0; U^n)$, respectively. It

is easy to see that $\mathcal{N}(0;\varphi(0;U^n)) = \varphi(0;U^n) = U^n$ and from (14) we have for any ξ

$$\begin{aligned}
&\mathcal{N}(0;\xi) = \xi + E(0),\\
&\mathcal{N}_t(0;\xi) = \Lambda(\xi) + E'(0),\\
&\mathcal{N}_{tt}(0;\xi) = \Lambda'(\xi)\Lambda(\xi) + E''(0),\\
&\mathcal{N}_v(0;\xi) = 1,\\
&\mathcal{N}_{vv}(0;\xi) = 0,\\
&\mathcal{N}_{tv}(0;\xi) = \Lambda'(\xi).
\end{aligned}$$

Substituting these results into the expansion (15) gives

$$\begin{aligned}
V =& U^n + k\left[\Lambda(U^n) + E'(0) + F(U^n)\right] + \frac{k^2}{2}\left[\Lambda'(U^n)\Lambda(U^n)\right.\\
&\left. + E''(0) + 2\Lambda'(U^n)F(U^n) + F'(U^n)F(U^n)\right] + \mathcal{O}(k^3).
\end{aligned}$$

Expanding the exact solution in a series of k about $k = 0$ also gives

$$u(x,k) = U^n + k\left[\Lambda(U^n) + F(U^n)\right] + \frac{k^2}{2}\left[(\Lambda'(U^n) + F'(U^n))(\Lambda(U^n) + F(U^n))\right] + \mathcal{O}(k^3).$$

The one-step error is therefore

$$\mathcal{L}^n = \frac{k^2}{2}\left[F'(U^n)\Lambda(U^n) - \Lambda'(U^n)F(U^n) + E''(0)+\right] + \mathcal{O}(k^3).$$

For backward Euler we have $E''(0) = -\frac{k^2}{2}\Lambda'(u^n)\Lambda(u^n) + \mathcal{O}(k^3)$. The one-step error for $\mathcal{N}_k^* = \mathcal{N}_k^{[\Lambda]*} \circ \varphi_k^{[F]}(U^n)$ is

$$\mathcal{L}^n = \frac{k^2}{2}\left[F'(U^n)\Lambda(U^n) - \Lambda'(U^n)F(U^n) - \Lambda'(U^n)\Lambda(U^n)\right] + \mathcal{O}(k^3). \tag{16}$$

This formula is difficult to compare, for any Λ and F, to the corresponding standard BE one-step error, which is

$$\begin{aligned}
\mathcal{L}^n =& -\frac{k^2}{2}[f_t + f_u f] + \mathcal{O}(k^3),\\
=& -\frac{k^2}{2}\left[F'(U^n)\Lambda(U^n) + \Lambda'(U^n)F(U^n)\right.\\
&\left. + \Lambda'(U^n)\Lambda(U^n) + F'(U^n)F(U^n)\right] + \mathcal{O}(k^3).
\end{aligned} \tag{17}$$

For the blow-up PDE (2) it is likely that the term $F'(U^n)F(U^n)$, which appears in the one-step error for BE and not SpBE, is a large contribution as $t \to T_b$. Intuitively, one would expect the one-step error of the standard BE to be larger than SpBE. One can approximate the one-step errors numerically. Substituting the operator and function $\Lambda(u)$ and $F(u)$ into (16) and (17) we have

$$\mathcal{L}^n = \frac{k^2}{2}\left[(U^n)^{1+\alpha}\Delta U^n - \Delta(U^n)^{1+\alpha} - \Delta(\Delta(U^n))\right] + \mathcal{O}(k^3),$$

for SpBE and

$$\mathcal{L}^n = -\frac{k^2}{2}\left[(U^n)^{1+\alpha}\Delta U^n + \Delta(U^n)^{1+\alpha} + \Delta(\Delta(U^n)) + (U^n)^{2+2\alpha}\right] + \mathcal{O}(k^3),$$

for the standard BE. The solution U^n to PDE (2) at time $t = 0.493$ ($T_b \approx 0.49445$) is approximated using ode15s. The spatial derivative Δv, in one dimension, is approximated using second order central finite differences with 51 uniform spatial nodes. With these evaluations the maximum norm of the one-step error for SpBE is

$$\|\mathcal{L}^n\|_\infty \approx 1.5681,$$

while for BE we have

$$\|\mathcal{L}^n\|_\infty \approx 20.6365.$$

Clearly the one-step error for BE is worse and will only increase as $U^n \to \infty$.

2.4 Growth Rate of Numerical Method

The concept of stability in a numerical method does not apply for blow-up problems. For a numerical method to be stable we want to ensure that the numerical solution does not become unbounded. For blow-up problems however the true solution becomes unbounded as $t \to T_b$. We therefore want the numerical scheme to also produce an unbounded solution. The restriction needed on the numerical scheme is for the solution to tend to infinity at the same rate as the true solution.

Delightfully, Friedman and McLeod [13] approximated the growth rate of the exact solution to PDE (2). The solution to $v_t = v^{1+\alpha}$ is $v = [\alpha(T_b - t)]^{-1/\alpha}$, when using the condition $\lim_{t\to T_b} v = \infty$ in determining the integration constant. Since this reaction part of PDE (2) dominants as $t \to T_b$ we expect

$$u(t) \sim [\alpha(T_b - t)]^{-\frac{1}{\alpha}}, \tag{18}$$

near blow-up. Using this Friedman and McLeod [13] proved that the solution to PDE (2) satisfies

$$(T_b - t)^{\frac{1}{\alpha}} u(x,t) \to \frac{1}{\alpha}^{\frac{1}{\alpha}} \qquad \text{as } t \to T_b^-, \tag{19}$$

provided $|x| \le C(T_b - t)^{1/2}$ for some $C > 0$. It is desired that the numerical solution $U^n(x)$ satisfies the same growth rate near blow-up. For some time $\hat{T}_b$ near blow-up and values of x near the blow-up point, we want

$$U^n(x) \sim \left[\alpha\left(\hat{T}_b - t^n\right)\right]^{-\frac{1}{\alpha}}.$$

Consider the case when $\alpha = 1$, which satisfies

$$\frac{U^{n+1}}{U^n} \sim \frac{\hat{T}_b - t^n}{\hat{T}_b - t^{n+1}}. \tag{20}$$

The following theorem from [7] proves that SpBE computes a numerical solution that satisfies (20).

Theorem 1. *Suppose there exists a constant C_0 that satisfies*

$$C_0 \geq \|U^0\|_\infty \qquad \text{and} \qquad -\Lambda(U^0) - (U^0)^2 + C_0 U^0 \geq 0. \tag{21}$$

If $t^{n+1} < T^*$, where $T^* \equiv 1/C_0$, then the function $U^{n+1}(x)$ given by

$$U^{n+1} - k\,\Lambda(U^{n+1}) = \left[\frac{1}{U^n} - k\right]^{-1}, \tag{22}$$

satisfies for all x

$$U^{n+1}(x) \leq \frac{T^* - t^n}{T^* - t^{n+1}} U^n(x). \tag{23}$$

Even though the proof was given in [7] we reproduce it here to give greater understanding. Note that (22) is the SpBE method given in (9) written as one formula and taking $\alpha = 1$. Therefore this proof is only for the SpBE method and does not prove anything about the second order method SpCN. The proof uses induction, first showing the result for $n = 0$ and extending for all n.

Proof. For the initial step, $n = 0$, we want

$$U^1(x) \leq \frac{T^*}{T^* - k} U^0(x),$$

from (23). Substituting this into SpBE (22) we have

$$U^0 - k\Lambda(U^0) \geq \left(\frac{T^* - k}{T^*}\right)\left[\frac{1}{U^0} - k\right]^{-1},$$

which simplifies to

$$-\Lambda(U^0) \geq \frac{T^*(U^0)^2 - U^0}{T^*(1 - kU^0)}. \tag{24}$$

Let $x = kU^0 < 1$, which gives the right hand side as

$$f(x) \equiv \frac{T^*(U^0)^2 - U^0}{T^*(1 - x)}.$$

This function is decreasing if $T^*(U^0)^2 - U^0 < 0$ since

$$f'(x) = \frac{T^*(U^0)^2 - U^0}{T^*(1 - x)^2}.$$

Since $T^* \equiv 1/C_0 \leq 1/\|U^0\|_\infty$ indeed we have $T^*(U^0)^2 - U^0 < 0$ and the function f is decreasing. Therefore we just need to ensure that $-\Lambda(U^0) \geq f(x)$ when x approaches zero. That is,

$$-\Lambda(U^0) \geq \lim_{x \to 0} \frac{T^*(U^0)^2 - U^0}{T^*(1 - x)} = \frac{T^*(U^0)^2 - U^0}{T^*}, \tag{25}$$

which is satisfied by condition (21), completing the proof for $n = 0$.

For induction we assume that $\lambda^n U^n \leq U^{n-1}$, where

$$\lambda^n \equiv \frac{T^* - t^n}{T^* - t^{n-1}} \in (0, 1).$$

The desired result is that

$$U^{n+1}(x) \leq \frac{1}{\lambda^{n+1}} U^n(x).$$

So we begin like $n = 0$ by substituting this into SpBE (22), to give

$$U^n - k\,\Lambda(U^n) \geq \lambda^{n+1} \left[\frac{1}{U^n} - k\right]^{-1}. \tag{26}$$

From SpBE (22) it is also known that U^n satisfies

$$U^n - k\,\Lambda(U^n) = \left[\frac{1}{U^{n-1}} - k\right]^{-1},$$

which simplifies (26) to

$$\left[\frac{1}{U^{n-1}} - k\right]^{-1} \geq \lambda^{n+1} \left[\frac{1}{U^n} - k\right]^{-1}. \tag{27}$$

By the assumption of induction, $\lambda^n U^n \leq U^{n-1}$, the inequality (27) can be written solely in terms of U^n as

$$\left[\frac{1}{\lambda^n U^n} - k\right]^{-1} \geq \lambda^{n+1} \left[\frac{1}{U^n} - k\right]^{-1}.$$

Expanding and collecting powers of U^n we can write this as $P(U^n) \geq 0$, where

$$P(U^n) = -k\lambda^n(1 - \lambda^{n+1})(U^n)^2 + (\lambda^n - \lambda^{n+1})U^n.$$

Since $\lambda^n < 1$ for all $n \in \mathbb{Z}$ the leading coefficient of $P(U^n)$ is negative. Note that $P(0) = 0$ so if $P \geq 0$ at $U^n = [T^* - t^n]^{-1}$ then we know that $P \geq 0$ over the entire interval $[0, [T^* - t^n]^{-1}]$. With positive P over $[0, [T^* - t^n]^{-1}]$ we know that

$$U^{n+1}(x) \leq \frac{T^* - t^n}{T^* - t^{n+1}} U^n(x)$$

for all x and $t^{n+1} < T^*$, which completes the proof. Substituting $U^n = [T^* - t^n]^{-1}$ into P and doing some simplifications gives

$$P\left([T^* - t^n]^{-1}\right) = 0.$$

Therefore $U^n = [T^* - t^n]^{-1}$ is the other root of P and thus $P > 0$ for $(0, [T^* - t^n]^{-1})$ as desired. □

The above proof shows that SpBE follows the same approximate growth rate as $t \to T_b$ as the exact solution to PDE (2) when $\alpha = 1$. Beck [7] conjectures that a similar result holds for all $\alpha > 0$. The numerics explored in the next section will however only be for $\alpha = 1$.

3 Numerical Results

Numerical results for each of the splitting methods, SpBE and SpCN, are given in this section. The effectiveness of the splitting methods are compared to the standard BE and CN methods, respectively. Section 3.1 explores the fixed time step methods given in section 2.2. While section 3.2 details an adaptive time stepping method using both SpBE and SpCN. The adaptive time stepping splitting method is compared to an adaptive scheme created with the standard BE and CN. The code for SpBE and BE was created during a USRA with initial and boundary conditions adjusted. The methods SpCN, CN, SpBE–SpCN and BE–CN were however coded for this project.

3.1 Fixed Time Stepping

Here we give results for the methods SpBE (10) and SpCN (12). They are compared to the standard BE and CN with identical spatial discretization. Details for the standard BE and CN methods can be found in [11]. All results are numerical solutions to the PDE (2) with $\alpha = 1$. The initial condition is taken to be $u(x, 0) = 4\cos(\pi x/2)$ to ensure blow-up occurs. Numerical convergence studies are carried out to verify the order of each method. This reveals that splitting methods obtain a slightly lower order than the standard methods. The errors from the splitting methods are however more than a magnitude smaller than errors from standard methods. The growth rate of the numerical solutions are then compared numerically. It becomes obvious that the splitting methods follow the same profile as the exact solution, while the standard methods grow too quickly.

The methods SpBE (10) and SpCN (12) are implemented using 51 uniform spatial nodes on $[-1, 1]$, i.e. $h = 1/25$. The odd number of spatial nodes is used to ensure that the blow-up point, $x_j = 0$, is one of the nodes. All methods are implicit and therefore a scheme such as Newton's method must be used to solve the system of equations at each time step. For the implementation here, the built-in MATLAB function fsolve is used to solve the implicit system of equations. The tolerance of fsolve is set at 10^{-10}. To determine the error in the numerical solutions an exact solution is desired. There is no exact solution to PDE (2), therefore a higher order numerical approximation is considered. The built-in MATLAB solver ode15s is used to compute this pseudo exact solution. The solver ode15s is an ODE integrator and therefore the PDE (2) is discretized in space using the method of lines (MOL). The MOL is implemented with the same second order finite difference scheme with 51 uniform spatial nodes.

The error for each method is then computed as the max norm of the difference from the pseudo exact solution. Note that ode15s is an adaptive time stepping method. Therefore ode15s should return a superior solution not only because it is higher order, but also because it reduces k as $t \to T_b$. The numerical solution is computed to a time near T_b. An estimate on the blow-up time is calculated by letting ode15s compute a solution until it returns a warning. The warning returned declares that a step size is smaller than the minimum allowed due to machine precision. MATLAB stops the computation and returns the final time of integration. This time we take to be an approximation to the blow-up time T_b.

Each method is used to compute an approximate solution to the PDE (2) with time steps $k = 0.0001$, 0.00005, 0.000025, and 0.0000125. For the particular problem described here the

approximate blow-up time is calculated to be $T_b \approx 0.494452$. Each method calculates the numerical solution up to the final time $t_f = 0.493$, which is approximately 99.7% of T_b. Figure 1 gives an error convergence plot for the first order methods BE and SpBE. Notice

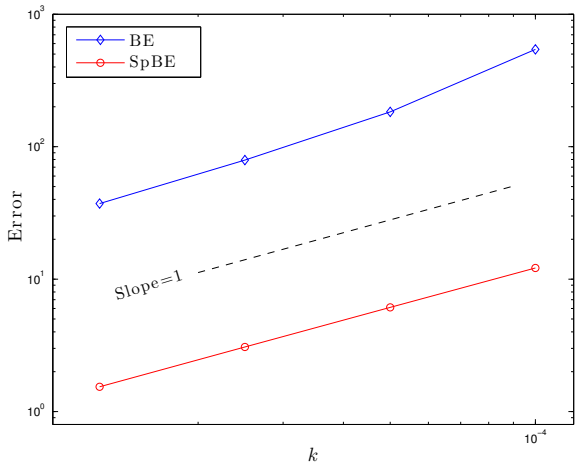

Figure 1: Convergence plot for first order methods BE and SpBE for four different time steps. The dashed line shows the slope one, which verifies the correct order of the methods.

that the error for the BE method is much larger than SpBE for all time steps k. This confirms numerically that the one-step error for SpBE derived in section 2.3 is in fact smaller than that of BE. The average order of convergence, p_{avg}, over the four time steps is also calculated. The average order of convergence for SpBE is $p_{avg} = 0.9931$, which is closer to one than the $p_{avg} = 1.2895$ for BE. The order of convergence for BE is possibly greater since the errors are larger than expected for the bigger step sizes.

The numerical solution with second order methods are also computed to $t_f = 0.493$ using the same time steps. The error convergence plot in Figure 2 shows the numerical error for CN and SpCN. The plot shows that the standard CN method has larger errors than SpCN. The calculation was not given in this paper, however, this gives evidence that the one-step error for SpCN is smaller than CN. The average order of convergence for SpCN is $p_{avg} = 1.8810$. The average order of convergence for the standard CN is actually better as $p_{avg} = 2.0104$. Figure 2 also shows that the error from the second order methods are orders of magnitude smaller than the first order methods.

The growth rate of each scheme can be examined numerically also. Recall from section 1 that blow-up occurs only at $x = 0$. The solution profile along $x = 0$ is given in Figure 3 using $k = 10^{-4}$ for the first order methods. Notice that the solution only increases from $u(0, 0) = 4$ to $u(0, 0.48) \approx 75$ in $[0, 0.48]$. Then the solution increases more rapidly in the remaining interval close to T_b. It can be seen that SpBE follows the same profile of ode15s. The standard BE grows at the wrong rate as it becomes almost double the size. This confirms the results given in section 2.4.

The second order methods show the same behaviour. Figure 4 shows the profile at $x = 0$ for $0.485 \leq t \leq 0.493$ for the second order methods. The same step size of $k = 10^{-4}$ is used for these second order numerical solutions. The solution increases from $u(0, 0) = 4$ to just

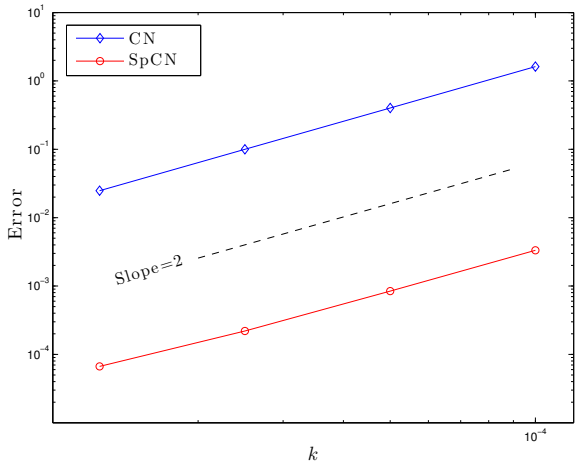

Figure 2: Convergence plot for second order methods CN and SpCN for four different time steps. The dashed line shows the slope of two, which verifies the correct order of the methods.

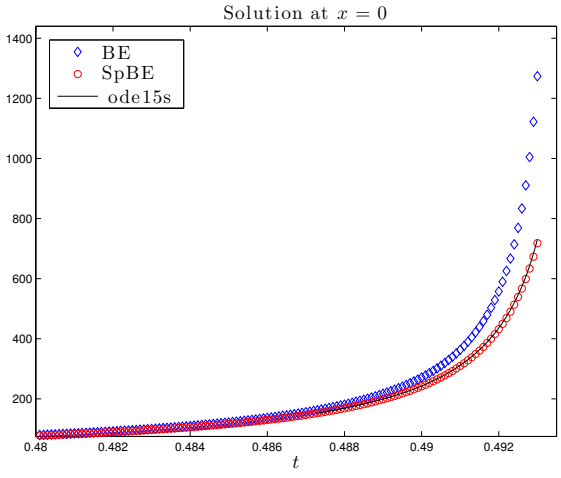

Figure 3: First order solutions to PDE (2) along $x = 0$ for $0.48 \leq t \leq 0.493$.

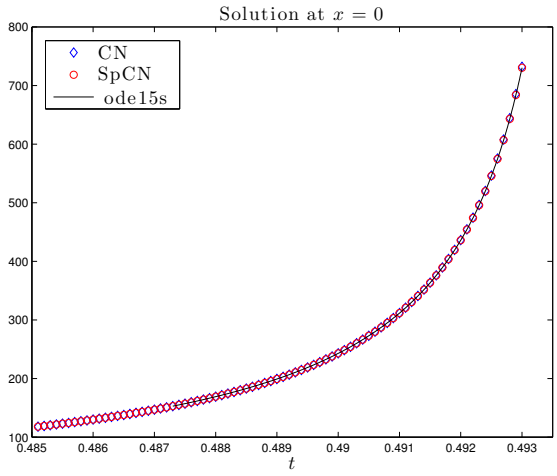

Figure 4: Second order solutions to PDE (2) along $x = 0$ for $0.485 \leq t \leq 0.493$.

over 100 in $[0, 0.485]$. The solution then increases drastically to approximately 750 in just $[0.485, 0.493]$. It is more difficult to see from Figure 4 that the solution from SpCN follows the growth rate better than CN. Upon further inspection it can be seen that the solution from CN actually grows too quickly like BE.

3.2 Adaptive Time Stepping

The rapid increase of $u(x, t)$ means some type of adaptive time stepping scheme must be used for blow-up problems. As $t \to T_b$ the step size required to resolve the solution decreases. Therefore it is extremely inefficient to use one fixed time step for the entire time interval. Here we investigate two types of adaptive time stepping schemes. The first is a local error control scheme called local extrapolation. This method uses a local error estimate to determine the next appropriate step size. The other is called global reintegration, which tries to control the global error. These methods were investigated in the thesis [14]. The code for these methods applied to the equations in [14] were borrowed from other researchers. Thus the simpler implementation here of BE–CN and SpBE–SpCN was coded for this project.

Local extrapolation considers a solution of a higher order method to be the exact solution. Here we use first and second order methods to implement this adaptive time stepping scheme. Let first and second order methods have solutions denoted y^n and z^n at time t^n, respectively. The methods have associated local errors E^n and e^n, that is $y^n = u(t^n) + E^n$ and $z^n = u(t^n) + e^n$. Of course the exact solution, $u(t^n)$, and numerical solutions, y^n and z^n, are functions of x also for PDE (2). The step size k is controlled through the formula

$$k^{n+1} = \sigma \left\| \frac{\delta^n}{\text{tol}} \right\|_\infty^{-\frac{1}{2}} k^n, \tag{28}$$

where δ^n is a local error estimate and $\sigma < 1$ is a safety factor [17]. The safety factor is used to decrease k slightly and hopefully increase the number of successful steps.

In the implementation here $\sigma = 0.9$. The estimate of local error in y^n is given by

$$\delta^n = y^n - z^n = E^n - e^n = \mathcal{O}(k),$$

since $E^n = \mathcal{O}(k)$ and $e^n = \mathcal{O}(k^2)$. The new step size is determined from the last k^n, a local error estimate δ^n and an error tolerance tol. If the local error estimate is greater than tol we recompute the step with a new step size. The tolerance is specified by the user in hope that the numerical solution returned has local error less than tol. There are many ways of defining tol, possibly the most common is

$$\text{tol} = \text{atol} + \text{rtol}\, \|y^n\|_\infty, \tag{29}$$

where atol and rtol are absolute and relative error tolerances respectively. The idea of (29) is when $\|y^n\|_\infty \approx 1$ we control the absolute error and when $\|y^n\|_\infty \gg 1$ we are controlling the relative error.

We also ensure that k from one step to another does not change drastically. The code does not permit the step size to decrease by more than a factor of 1/10 or increase more than a factor of 5. Furthermore, after a failed step the step size is not allowed to increase. Pseudo code for local extrapolation is given in Algorithm 1, which shows the main components of the scheme. Local extrapolation is implemented using standard methods BE and CN as the first

Algorithm 1 Local extrapolation

Define: y^0, atol, rtol, t^0, t^f, k^0
while $t^n \leq t^f$ **do**
  $\text{tol}^n = \text{atol} + \text{rtol}\, \|y^n\|_\infty$
  compute y^n and z^n at t^n using k^n
  compute the local error estimate $\delta^n = y^n - z^n$
  if $\|\delta^n\|_\infty < \text{tol}^n$ **then**
    step is accepted and we set $t^{n+1} = t^n + k^n$
    determine next step size by $k^{n+1} = \sigma \|\delta^n / \text{tol}^n\|_\infty^{-\frac{1}{2}} k^n$
  else if $\|\delta^n\|_\infty \geq \text{tol}^n$ **then**
    step is rejected
    step redone with $k^n = \sigma \|\delta^n / \text{tol}^n\|_\infty^{-\frac{1}{2}} k^n$
  end if
end while
return t^f, y^f

and second order methods (BE-CN). We compare this implementation to local extrapolation using SpBE and SpCN instead (SpBE-SpCN).

The methods are compared by computing the solution until the global error is greater than tol. The built-in MATLAB ode15s is used like in section 3.1, to compute the global error, ϵ. The initial time step is taken as $k^0 = 10^{-4}$. The absolute and relative error tolerances are set as $\text{atol} = 10^{-6}$ and $\text{rtol} = 10^{-6}$.

The schemes BE-CN and SpBE-SpCN compute solutions until $\epsilon \geq 10^{-5}$. It is found that the standard method BE-CN can compute to $t_f \approx 0.671\, T_b$. Whereas the splitting method

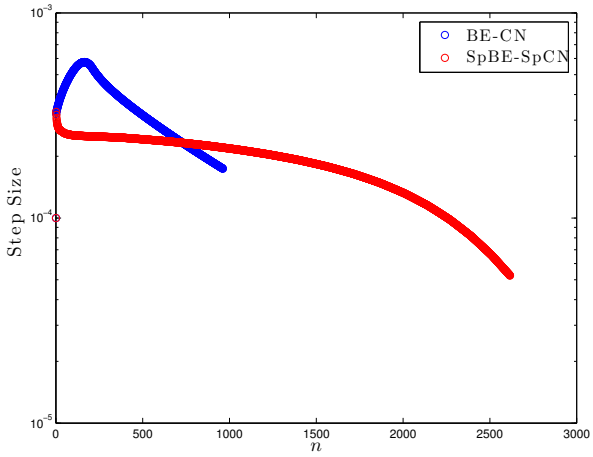

Figure 5: Step sizes of adaptive time stepping schemes BE-CN and SpBE-SPCN.

SpBE-SPCN computes much further until $t_f \approx 0.969 T_b$. Splitting methods are more accurate and therefore the method can compute much closer to T_b. It can be seen in Figure 5 that the step sizes used by BE-CN start off larger to that of SpBE-SpCN. The BE-CN step sizes then decrease smaller than SpBE-SpCN before it finishes. The local error for BE-CN is larger than that of SpBE-SpCN. This causes the local error estimate to be unreliable and therefore BE-CN can not compute as far. Figure 6 shows the global error of both BE-CN

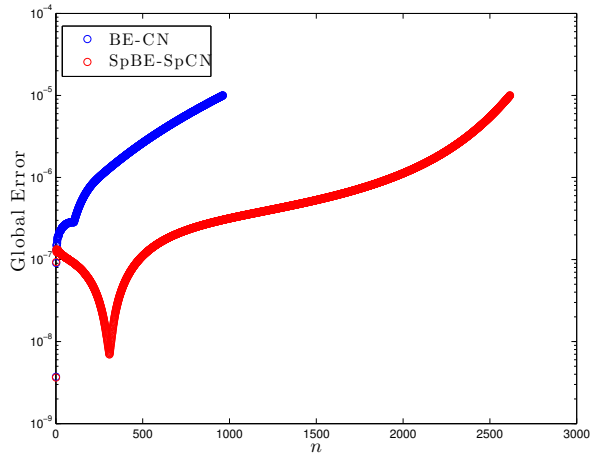

Figure 6: Global error of adaptive time stepping schemes BE-CN and SpBE-SPCN.

and SpBE-SpCN. The global error of BE-CN is always larger than that of SpBE-SpCN.

Another type of adaptive time stepping called global reintegration was studied. This method was first examined while writing the thesis [14], however a greater understanding was achieved with this project. The implementation of the method was unable to be completed in time, however, a brief description of the method is provided here. For a full description see [18]. The basic idea is to compute a global error estimate, ϵ, while you compute a

numerical solution with some local error control scheme. The global error estimate, ϵ, is then used to determine if the solution is acceptable. If ϵ is too large the solution is recomputed using smaller error tolerances atol and rtol.

Any local error control method can be used to integrate PDE (2) in time, e.g. local extrapolation. Simultaneously, an initial value problem for the global error is solved to obtain the global error estimate ϵ^{n+1} at time step t^{n+1}. This initial value problem is derived in [18] and one approach to simultaneously solving it is described. The global error estimate is then used to determine if the solution, U^f, at t^f is acceptable. If

$$\|\epsilon^f\|_\infty \leq C_{\text{control}}\ \text{tol}^f, \tag{30}$$

where $C_{\text{control}} \approx 1$, then the solution is deemed reliable. If ϵ^f does not satisfy the latter, we repeat the integration with the following adjusted tolerances:

$$\text{atol} = \text{atol}\ \times\ \text{fac}, \qquad \text{rtol} = \text{rtol}\ \times\ \text{fac}, \qquad \text{fac} = \text{tol}^f/\|\epsilon^f\|_\infty. \tag{31}$$

This method hinges on the global error estimate and tolerance proportionality.

Tolerance proportionality refers to the assumption that decreasing the tolerance by a factor fac decreases the global error by the same factor. In equation (30) the reliability of ϵ^f is essential for obtaining a numerical solution with error less than tol_f. The success of equations in (31) is based on the expectation that reducing the local error estimates by a factor of fac will reduce the true global error, ϵ_f, by fac. The global reintegration method is summarized in Algorithm 2.

Algorithm 2 Global reintegration.

Define: y^0, atol, rtol, t^0, t^f, k^0
while $t^n \leq t^f$ **do**
  $\text{tol}^n = \text{atol} + \text{rtol}\ \|y^n\|_\infty$
  compute y^{n+1} and ϵ^{n+1}, simultaneously, at t^n using k^n
  use local error control scheme to determine k^{n+1}
  determine global error estimate ϵ^{n+1} simultaneously
end while
at t^f check size of ϵ^f
if $\|\epsilon^f\|_\infty > C_{\text{control}}\ \text{tol}^f$ **then**
  solution is unreliable and integration is redone over $[t^0, t^f]$
  set $\text{fac} = \text{tol}^f/\|\epsilon^f\|_\infty$
  integration redone with $\text{atol} = \text{atol}\ \times\ \text{fac}$ and $\text{rtol} = \text{rtol}\ \times\ \text{fac}$
else if $\|\epsilon^f\|_\infty \leq C_{\text{control}}\ \text{tol}^f$ **then**
  solution is accepted
end if
return t^f, y^f

4 Conclusion

Blow-up problems are interesting and difficult numerical problems. Standard numerical methods are obviously not well suited as show through this paper. Methods introduced here

are not the most innovative for solving blow-up problems. The fixed time step studies did however show superiority of splitting methods over standard methods. The one-step error of SpBE was shown to be less than that of BE. The growth rate of SpBE was also shown to match that of the true growth rate. Numerically, the error in the SpBE and SpCN were at least an order of magnitude smaller than that of BE and CN. The implementation of local extrapolation also favored the splitting methods. The SpBE-SpCN method kept the error less than 10^{-5} substantially closer to T_b than BE-CN.

Obvious disadvantages of SpBE and SpCN is the need of an exact solution to $u_t = F(u)$. There are many blow-up problems where $u_t = F(u)$ does indeed have an exact solution. For numerous examples see Beck [7]. If an exact solution is not known, one could use approximations for both parts of the PDE. Using numerical solutions for both parts may however create inappropriate growth rates.

It is eminent that adaptive time and space methods are needed for blow-up problems. More novel approaches than the ones introduced here are desirable. Since k and h decrease as the solution $u(x,t) \to \infty$ we are ultimately restricted by machine precision. Therefore adaptive methods which scale time and space are probably more suited. The purpose of this paper was however to show that specific splitting methods for this difficult numerical problem are superior to standard generic numerical methods.

References

[1] D. A. Frank-Kamenetskii, "Towards temperature distributions in a reaction vessel and the stationary theory of thermal explosion," in *Doklady Akad. Nauk SSSR*, vol. 18, 1938, pp. 411–412.

[2] H. Fujita, "On the blowing up of solutions of the Cauchy problem for $u_t = \Delta u + u^{1+\alpha}$," *J. Fac. Sci. Univ. Tokyo Sect. I*, vol. 13, pp. 109–124, 1966.

[3] ——, "On the nonlinear equations $\delta u + e^u = 0$ and $\partial v/\partial t = \delta v + e^v$," *Bulletin of the American Mathematical Society*, vol. 75, no. 1, pp. 132–135, 1969.

[4] ——, "On some nonexistence and nonuniqueness theorems for nonlinear parabolic equations," in *Nonlinear Functional Analysis (Proc. Sympos. Pure Math., Vol. XVIII, Part 1, Chicago, Ill., 1968)*. Providence, R.I.: Amer. Math. Soc., 1970, pp. 105–113.

[5] ——, "On the asymptotic stability of solutions of the equation $v_t = \delta v + e^v$," in *Proc. Internat. Conf. Functional Analysis and Related Topics (Tokyo, 1969)*, 1970, pp. 252–259.

[6] F. B. Weissler, "Single point blow-up for a semilinear initial value problem," *J. Differential Equations*, vol. 55, no. 2, pp. 204–224, 1984.

[7] M. Beck, "B-methods: Special time–integrators for differential equations with blow-up solutions," Ph.D. dissertation, McGill University, Montréal, Québec, Canada, August 2009.

[8] T. Nakagawa, "Blowing up of a finite difference solution to $u_t = u_{xx} + u^2$.," *Appl. Math. Optim.*, vol. 2, no. 4, pp. 337–350, 1975/76.

[9] C. Bandle and H. Brunner, "Blowup in diffusion equations: a survey," *J. Comput. Appl. Math.*, vol. 97, no. 1-2, pp. 3–22, 1998.

[10] E. Hairer, C. Lubich, and D. Gerhard Wanner, *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*. Springer, 2006, vol. 31.

[11] R. J. LeVeque, *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

[12] R. I. McLachlan and G. R. W. Quispel, "Splitting methods," *Acta Numer.*, vol. 11, pp. 341–434, 2002. [Online]. Available: http://dx.doi.org/10.1017/S0962492902000053

[13] A. Friedman and B. McLeod, "Blow-up of positive solutions of semilinear heat equations," *Indiana Univ. Math. J.*, vol. 34, no. 2, pp. 425–447, 1985.

[14] N. King, "A numerical investigation of local and global error control with the method of lines applied to a blow-up semilinear parabolic equation," St. John's, Newfoundland, Canada, April 2013.

[15] R. Glowinski, L. Shiau, and M. Sheppard, "Numerical methods for a class of nonlinear integro-differential equations," *Calcolo*, vol. 50, no. 1, pp. 17–33, 2013.

[16] G. Strang, "On the construction and comparison of difference schemes," *SIAM J. Numer. Anal.*, vol. 5, pp. 506–517, 1968.

[17] L. R. Petzold and U. M. Ascher, *Computer methods for ordinary differential equations and differential-algebraic equations*. Siam, 1998, vol. 61.

[18] J. Lang and J. G. Verwer, "On global error estimation and control for initial value problems," *SIAM J. Sci. Comput.*, vol. 29, no. 4, pp. 1460–1475, 2007.