May 2014

Experiments on Temporal Variable Step BDF2 Algorithms

Anja Katrin Denner
University of Wisconsin-Milwaukee

Experiments on temporal variable step BDF2 algorithms

by

Anja Katrin Denner

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 2014

ABSTRACT

Experiments on temporal variable step BDF2 algorithms

by

Anja Katrin Denner

The University of Wisconsin-Milwaukee, 2014
Under the Supervision of Professor Bruce A. Wade

Efficient algorithms for solving stiff PDEs are of great interest. For developing such an algorithm step sizes should vary in both space and time. We have to understand each separately first before putting it together, and this thesis is dedicated to developing a sharper notion of the performance of a variable step size BDF2 scheme for some examples. We find suitable parameters for the variable step size algorithm proposed by Jannelli and Fazio in their respective paper concerning adaptive stiff solvers at low accuracy and complexity [5]. Finally, we make a short excursion on the stability of BDF2 for the Allen-Cahn Equation.

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGEMENTS

I want to thank Prof. Bruce A. Wade who suggested to work on this topic and helped me in developing the thesis. I also want to thank Prof. Istvan Lauko and Prof. Lei Wang for being on the committee.

Chapter 1
Introduction

Partial differential equations (PDEs) play an important role in modelling real life problems and thus also in research. Unfortunately, PDEs rarely can be solved analytically, which is why there is a lot of effort in developing efficient numerical schemes to solve them. Space and time are discretized separately where the step sizes usually depend on the function. The easiest way of discretizing is to have a uniform mesh. Some of the problem solutions change at some points more drastically than on others, which would enforce a very small step size to assure a good solution. Therefore we desire to use adaptive methods which allow us to use a small step size in areas of rapid changes and a large step size if changes are hardly noticable. This way the program automatically chooses the optimal step size. One question is how the algorithm should choose the step size and remain stable. Jannelli and Fazio proposed an algorithm for BDF2 [5], which involves many parameters that the user has to choose manually in advance. In this thesis we will have a closer look on three test problems and determine suitable parameters for each problem when using this algorithm.

In chapter 2 we will derive the schemes that we will use. Throughout the thesis we will look at *Backward differentiation formulas*, so called BDF schemes, commonly used to solve stiff problems. We will focus on BDF1 and BDF2, two multistep schemes, where the index indicates the number of steps. First uniform versions of BDF1 and BDF2 are being presented which will afterwards be transformed into variable step size schemes. The implementation and a description of the schemes can be found in the appendix.

In chapter 3 we want to further investigate the methods. In a first step we will demonstrate that the methods are working correctly which will be done by conver-

gence tests. Next, we adjust the parameters for the variable step size BDF2 scheme for a problem where an exact solution is known, a Biochemistry problem and the Allen-Cahn Equation. Furthermore, we will have a look at the stability of BDF2. For *ordinary differential equations* (ODEs) it is known that BDF2 remains stable if the amplification factor is chosen no larger than $1 + \sqrt{2}$. This has been proven in 1983 by Grigorieff [3]. We desire a similar result for PDEs. So far Emmrich has proven that stability is guaranteed for an amplification factor of less than or equal to 1.91 [2]. Bruce Wade conjectures that for PDEs the same boundary holds as for ODEs [8]. In section 3.3.1 we will describe and perform some experiments in order to support his theory.

Finally, chapter 4 comprises the conclusions derived from the investigations and the experiments of the chapters before.

Chapter 2
Derivation of the BDF schemes

In this chapter we will derive the schemes of BDF1 and BDF2. Both are A-stable which makes these two BDF schemes so valuable for stiff problems. The BDF schemes loose stability the higher the order of the scheme; schemes of order 3 and higher are not A-stable and those of order 7 and higher are even unstable. Also they are more complicated to implement. BDF2 is just a good compromise: It is of order 2, can be computed relatively easily, and has good stability properties.

Consider a PDE of a reaction-diffusion type:

$$\begin{aligned} u_t(x,t) &= \beta \Delta u(x,t) + f(x,t,u(x,t)), \qquad x \in (a,b) \\ u(x,0) &= u_0(x), \qquad x \in (a,b) \\ u(a,t) &= \alpha_1 \\ u(b,t) &= \alpha_2 \end{aligned} \tag{2.1}$$

Here we focus on one dimension in space in order to advance the numerical experiments. It is sufficient to derive a scheme for homogeneous boundary conditions ($u(a,t) = u(b,t) = 0$) since we can transform any problem with boundary conditions that are not all zero to a homogeneous boundary problem in the following way: We subtract a linear function $w(x)$ to preprocess the problem.

$$w(x) = \alpha_1 + \frac{x-a}{b-a}(\alpha_2 - \alpha_1)$$

This function assures that the boundary conditions

$$u(a,t) = \alpha_1 \qquad \text{and} \qquad u(b,t) = \alpha_2$$

become zero if we define

$$v(x,t) \coloneqq u(x,t) - w(x)$$

Thus

$$v(a,t) = u(a,t) - w(a) = \alpha_1 - \left(\alpha_1 + \frac{a-a}{b-a}(\alpha_2 - \alpha_1)\right) = \alpha_1 - \alpha_1 = 0$$
$$v(b,t) = u(b,t) - w(b) = \alpha_2 - \left(\alpha_1 + \frac{b-a}{b-a}(\alpha_2 - \alpha_1)\right) = \alpha_1 - (\alpha_1 - \alpha_2) = 0$$

Furthermore, since w is independent of t, we get

$$v_t = \frac{\partial}{\partial t}(u(x,t) - w(x)) = u_t \tag{2.2}$$

and also

$$\Delta v = \sum \frac{\partial^2}{\partial x_i^2}(u(x,t) - w(x)) = \sum u_{x_i x_i}(x,t) = \Delta u \tag{2.3}$$

since $w(x)$ is linear in x. Thus the PDE (2.1) turns into

$$v_t(x,t) = \beta \Delta v(x,t) + f(x,t,v(x,t) + w(x)) \tag{2.4}$$

This shows that the only term that changes is the forcing term $f(x,t,v(x,t)+w(x))$. Thus from now on, we assume to have zero boundary conditions.

Let $u(x,t)$ be the exact solution of the PDE. We use the method of lines, that is, we set down a mesh in space and time, but we will consider them separately. The points in space will be denoted with x_i and time with t_n where the maximum number of steps are m and N, respectively.

For the sake of convenience we will denote $u(x_i,t_n)$ with $u_{i,n}$. If we use a computer to solve the problem, we cannot compute these numbers exactly, but only have approximations of them. Thus we use $v_{i,n}$ for the approximation to $u_{i,n}$.

We seek

$$v_{i,n} \text{ for } 1 \leq i \leq m, 0 \leq n \leq N$$

with $v_{i,0} = u(x_i, 0)$ as given.

We want to derive a vectorized form for the discretized version of

$$u_{xx} = \frac{\partial^2}{\partial x^2} u\left(x,t\right)$$

Let h be the spatial step size ($h = \frac{b-a}{m+1}$, where m is the number of x-values that we have). The first derivative can be approximated via

$$\frac{\partial}{\partial x} u\left(x_i,t\right) = \frac{u\left(x_{i+1},t\right) - u\left(x_i,t\right)}{h} = \frac{v_{i+1} - v_i}{h}$$

We evaluate the approximation to the first derivative at $x_i \pm \frac{1}{2}h$ to compute an approximation to the second derivative. This yields

$$\frac{\partial^2}{\partial x^2} u\left(x_i,t\right) = \frac{\frac{v_{i+1}-v_i}{h} - \frac{v_i - v_{i-1}}{h}}{h} = \frac{v_{i+1} - 2v_i + v_{i-1}}{h^2}$$

With this the PDE (2.1) can be written in a semi-discrete form as

$$u_t = \beta \frac{v_{i+1} - 2v_i + v_{i-1}}{h^2} + f\left(x_i,t,v_i\right) \text{ for } 1 \leq i \leq m \tag{2.5}$$

To make it easier we vectorize this result. With the choices

$$v = \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix}, \quad F\left(\boldsymbol{x},t,v\right) = \begin{pmatrix} f\left(x_1,t,v_1\right) \\ f\left(x_2,t,v_2\right) \\ \vdots \\ f\left(x_m,t,v_m\right) \end{pmatrix}, \quad B = \frac{1}{h^2} \begin{pmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{pmatrix},$$

equation (2.5) can be written as

$$u_t = \beta B v + F\left(\boldsymbol{x},t,v\right) \tag{2.6}$$

The approximation for u_t depends on the scheme we use. In section 2.1 we will derive BDF1 and BDF2 for uniform step sizes and afterwards, in section 2.2, for variable temporal step sizes.

2.1 Fixed step size

We derive the BDF schemes for a uniform temporal step size which we denote with $k = \frac{t_f - t_0}{N}$ (with t_f being the final time, t_0 the starting time and N the number of time steps). These schemes are fairly easy since we only have one step size.

2.1.1 BDF1

BDF1 is a first order backward differentiation formula, also known as *Backward Euler*. A recursive formula using centered differences in space at the time step t_{n+1}, assuming that we are marching forward and already know all $v_{i,n}, 1 \leq i \leq m$, can be obtained via

$$\frac{v^{n+1} - v^n}{k} = Bv^{n+1} + F\left(\boldsymbol{x}, t_{n+1}, v^{n+1}\right)$$

Solving for v^{n+1} yields

$$v^{n+1} = (I - kB)^{-1}\left(v^n + kF\left(\boldsymbol{x}, t_{n+1}, v^{n+1}\right)\right) \tag{2.7}$$

where the matrix $(I - kB)$ is invertible since it is strictly diagonally dominant and v^n denotes the vector:

$$v^n = \begin{pmatrix} v_{1,n} \\ \vdots \\ v_{m,n} \end{pmatrix}$$

This is an implicit, recursive formula and the algorithm to solve a stiff problem using BDF1 is the following:

Algorithm 2.1: BDF1

$v^0 = u(x, 0)$
`for n:=0 to N-1 do`
  $v^{n+1} = (I - kB)^{-1}\left(v^n + kF\left(\boldsymbol{x}, t_{n+1}, v^{n+1}\right)\right)$
`end`

Since we do not know v^{n+1} inside the nonlinearity on the right-hand side of equation (2.7) we use a predictor-corrector method. As a predictor we use the *Forward Euler* method, which is an explicit scheme. This method is preferred over Newton's

method because the latter needs to compute the Jacobian at each time step and thus is expensive. Additionally, it is proven that only one step of the corrector is necessary (see [6], pp. 105-107). However, better results can be achieved if the correction is done 3 or 4 times. As can be seen in Algorithm 2.2, we correct the prediction until the difference between two consecutive corrections is small enough or the maximum number of iterations is reached.

Algorithm 2.2: BDF1 as a predictor corrector

$v^0 = u(x, 0)$
`for n:=0 to N-1 do`
  `comment: predictor`
  $\widetilde{v}^{n+1} = Bv^n + kF(\boldsymbol{x}, t_{n+1}, v^n)$
  `comment: corrector`
  $v_{old} = \widetilde{v}^{n+1}$
  $err = \infty$
  $p = 0$
  `while` $err > tol$ `and` $p < maxIterations$ `do`
    $\widetilde{v}^{n+1} = (I - kB)^{-1}(v^n + kF(\boldsymbol{x}, t_{n+1}, \widetilde{v}^{n+1}))$
    $err = \|v_{old} - \widetilde{v}^{n+1}\|$
    $v_{old} = \widetilde{v}^{n+1}$
    $p = p + 1$
  `end`
  $v^{n+1} = \widetilde{v}^{n+1}$
`end`

The respective implementation in MATLAB can be found in the appendix (see MATLAB-Function 2).

2.1.2 BDF2

BDF2 is a second order backward differentiation formula. Since it is a two-step scheme we need two inital values, necessarily computed by some other formula. We can use the BDF1 scheme to obtain a second starting value.

This time

$$u_t(\boldsymbol{x}, t_n) \approx \frac{\frac{3}{2}v^{n+1} - 2v^n + \frac{1}{2}v^{n-1}}{k} \tag{2.8}$$

The equation can be derived by interpolation, e. g. using a Newton's polynomial where the error is $\mathcal{O}(k^3)$. As before this is best put into vector form. If we plug the second order backward differentiation formula into equation (2.6) we obtain the following:

$$\frac{\frac{3}{2}v^{n+1} - 2v^n + \frac{1}{2}v^{n-1}}{k} = Bv^{n+1} + F\left(\boldsymbol{x}, t_{n+1}, v^{n+1}\right) \tag{2.9}$$

If we solve for v^{n+1} equation (2.9) becomes

$$v^{n+1} = \left(I - \frac{2}{3}kB\right)^{-1}\left(\frac{4}{3}v^n - \frac{1}{3}v^{n-1} + \frac{2}{3}kF\left(\boldsymbol{x}, t_{n+1}, v^{n+1}\right)\right)$$

since $\left(I - \frac{2}{3}kB\right)$ is strictly diagonally dominant and thus invertible.

Now we can formulate an algorithm for BDF2. This algorithm already includes the predictor, Forward Euler, and the first order scheme, BDF1, to get a second initial value.

Algorithm 2.3: BDF2

$v^0 = u(x, 0)$

```
comment:    do one step of BDF1 to obtain second initial value
for  n:=1  to  N-1  do
```

comment: predictor

$\widetilde{v}^{n+1} = Bv^n + kF\left(\boldsymbol{x}, t_{n+1}, v^n\right)$

comment: corrector

$v_{old} = \widetilde{v}^{n+1}$

$err = \infty$

$p = 0$

while $err > tol$ **and** $p < maxIterations$ **do**

$\widetilde{v}^{n+1} = \left(I - \frac{2}{3}kB\right)^{-1}\left(\frac{4}{3}v^n - \frac{1}{3}v^{n-1} + \frac{2}{3}kF\left(\boldsymbol{x}, t_{n+1}, \widetilde{v}^{n+1}\right)\right)$

$err = \|v_{old} - \widetilde{v}^{n+1}\|$

$v_{old} = \widetilde{v}^{n+1}$

$p = p + 1$

end

$v^{n+1} = \widetilde{v}^{n+1}$

end

In the appendix, there is also a MATLAB code of this algorithm available (see MATLAB-Function 3).

2.2 Variable temporal step size

For our experiments we also need a variable temporal step size version of the schemes derived in the previous section. For BDF1 there is no big change in the scheme: it is first order and a simple one step scheme. The changes for BDF2 are significant. In the following, we will derive a variable step size scheme for BDF2 (VS-BDF2).

Assume the approximations $v^0, v^1, \ldots, v^n$ are already computed. Now we have to find a formula to compute v^{n+1}. We use the data points v^{n+1}, v^n, v^{n-1} to interpolate with a quadratic function. Using the Newton polynomials we can find the interpolant. The polynomials can be computed using the Neville-Aitken scheme for divided differences.

		$v\left[t_i, t_{i+1}\right]$	$v\left[t_i, t_{i+1}, t_{i+2}\right]$
t_{n-1}	v^{n-1}		
t_n	v^n	$\frac{v^n - v^{n-1}}{t_n - t_{n-1}}$	
t_{n+1}	v^{n+1}	$\frac{v^{n+1} - v^n}{t_{n+1} - t_n}$	$\frac{\frac{v^{n+1} - v^n}{t_{n+1} - t_n} - \frac{v^n - v^{n-1}}{t_n - t_{n-1}}}{t_{n+1} - t_{n-1}}$

Table 2.1: Neville-Aitken scheme for divided differences

With the help of Table 2.1 we get:

$$p\left(t\right) = v^{n-1} + \left(\frac{v^n - v^{n-1}}{t_n - t_{n-1}}\right)\left(t - t_{n-1}\right) + \left(\frac{\frac{v^{n+1} - v^n}{t_{n+1} - t_n} - \frac{v^n - v^{n-1}}{t_n - t_{n-1}}}{t_{n+1} - t_{n-1}}\right)\left(t - t_{n-1}\right)\left(t - t_n\right) \tag{2.10}$$

Suppose we have a characteristic step size, τ, such that there exist positive δ_0, δ_1 satisfying

$$\delta_0 \tau \leq \tau_n \leq \delta_1 \tau \ , \ n = 1, \ldots, N$$

where we define $\tau_n \coloneqq t_{n+1} - t_n$ and $N \geq 1$. Then the step size ratios are $r_n \coloneqq \frac{\tau_n}{\tau_{n-1}}$. This is called a "quasi-uniform" mesh. Thus using τ_n, equation (2.10) becomes

$$p\left(t\right) = v^{n-1} + \left(\frac{v^n - v^{n-1}}{\tau_{n-1}}\right)\left(t - t_{n-1}\right) + \left(\frac{\frac{v^{n+1} - v^n}{\tau_n} - \frac{v^n - v^{n-1}}{\tau_{n-1}}}{\tau_{n-1} + \tau_n}\right)\left(t - t_{n-1}\right)\left(t - t_n\right) \tag{2.11}$$

Now the VS-BDF2 scheme is derived through the following collocation at t_{n+1}

$$p'\left(t_{n+1}\right) = f\left(t_{n+1}, v^{n+1}\right)$$

Thus equation (2.11) yields

$$p'\left(t\right) = \frac{v^n - v^{n-1}}{\tau_{n-1}} + \left(\frac{\frac{v^{n+1} - v^n}{\tau_n} - \frac{v^n - v^{n-1}}{\tau_{n-1}}}{\tau_{n-1} + \tau_n}\right)\left(2t - t_{n-1} - t_n\right)$$

and the scheme is

$$f\left(t_{n+1}, v^{n+1}\right) = \frac{v^n - v^{n-1}}{\tau_{n-1}} + \left(\frac{\frac{v^{n+1} - v^n}{\tau_n} - \frac{v^n - v^{n-1}}{\tau_{n-1}}}{\tau_{n-1} + \tau_n}\right)\left(2t_{n+1} - t_{n-1} - t_n\right)$$

$$
\begin{aligned}
&= \frac{v^n - v^{n-1}}{\tau_{n-1}} + \left(\frac{\frac{v^{n+1}-v^n}{\tau_n} - \frac{v^n - v^{n-1}}{\tau_{n-1}}}{\tau_{n-1}+\tau_n} \right) (\tau_{n-1} + 2\tau_n) \\
&= \frac{1}{\tau_{n-1}} v^n - \frac{1}{\tau_{n-1}} v^{n-1} \\
&\quad + \frac{\tau_{n-1}+2\tau_n}{\tau_{n-1}+\tau_n} \left(\frac{\tau_{n-1}v^{n+1} - \tau_{n-1}v^n - \tau_n v^n + \tau_n v^{n-1}}{\tau_{n-1}\tau_n} \right) \\
&= \frac{\tau_{n-1}+2\tau_n}{\tau_n(\tau_{n-1}+\tau_n)} v^{n+1} + \left(\frac{1}{\tau_{n-1}} - \frac{(\tau_{n-1}+2\tau_n)\left(\frac{1}{\tau_{n-1}} + \frac{1}{\tau_n}\right)}{\tau_{n-1}+\tau_n} \right) v^n \\
&\quad + \left(-\frac{1}{\tau_{n-1}} + \frac{\tau_{n-1}+2\tau_n}{\tau_{n-1}(\tau_{n-1}+\tau_n)} \right) v^{n-1}
\end{aligned}
$$

Now we consider the coefficients of v^{n+1}, v^n, v^{n-1} separately. We want to simplify each coefficient as much as possible to get an easy representation for VS-BDF2. For our simplifications we define $r_n \coloneqq \frac{\tau_n}{\tau_{n-1}}$.

For the coefficient of v^{n+1} we obtain

$$
\begin{aligned}
\frac{\tau_{n-1}+2\tau_n}{\tau_n(\tau_{n-1}+\tau_n)} &= \frac{1}{\tau_{n-1}+\tau_n}\left(\frac{\tau_{n-1}}{\tau_n} + \frac{2\tau_n}{\tau_n}\right) \\
&= \frac{1}{\tau_{n-1}+\tau_n}\left(\frac{1}{r_n} + 2\right) \\
&= \frac{\frac{1}{r_n}+2}{\tau_n\left(\frac{\tau_{n-1}}{\tau_n}+1\right)} \\
&= \frac{\frac{1}{r_n}+2}{\tau_n\left(\frac{1}{r_n}+1\right)} \\
&= \frac{1+2r_n}{\tau_n(1+r_n)},
\end{aligned}
$$

where we try to replace as much as possible by r_n. Similar manipulations for the coefficient of v^n lead to

$$
\frac{1}{\tau_{n-1}} - \frac{(\tau_{n-1}+2\tau_n)\left(\frac{1}{\tau_{n-1}}+\frac{1}{\tau_n}\right)}{\tau_{n-1}+\tau_n} = \frac{1}{\tau_{n-1}} - \frac{\tau_{n-1}+2\tau_n}{\tau_{n-1}(\tau_{n-1}+\tau_n)} - \frac{\tau_{n-1}+2\tau_n}{\tau_n(\tau_{n-1}+\tau_n)}
$$

$$
\begin{aligned}
&= \frac{\frac{\tau_n}{\tau_{n-1}}}{\tau_n} - \frac{\frac{\tau_n}{\tau_{n-1}}\left(\tau_{n-1}+2\tau_n\right)}{\tau_n\left(\tau_{n-1}+\tau_n\right)} - \frac{\left(\tau_{n-1}+\tau_n\right)+\tau_n}{\tau_n\left(\tau_{n-1}+\tau_n\right)} \\
&= \frac{r_n}{\tau_n} - \frac{r_n\left(\tau_{n-1}+2\tau_n\right)}{\tau_n\left(\tau_{n-1}+\tau_n\right)} - \frac{1}{\tau_n} - \frac{1}{\tau_{n-1}+\tau_n} \\
&= \frac{r_n}{\tau_n} - \frac{r_n\left(\tau_{n-1}+\tau_n\right)+r_n\tau_n}{\tau_n\left(\tau_{n-1}+\tau_n\right)} - \frac{1}{\tau_n} - \frac{1}{\tau_{n-1}+\tau_n} \\
&= \frac{r_n}{\tau_n} - \frac{r_n}{\tau_n} - \frac{r_n}{\tau_{n-1}+\tau_n} - \frac{1}{\tau_n} - \frac{1}{\tau_{n-1}+\tau_n} \\
&= -\frac{1}{\tau_n}\left(\frac{\tau_n r_n}{\tau_{n-1}+\tau_n} + 1 + \frac{\tau_n}{\tau_{n-1}+\tau_n}\right) \\
&= -\frac{1}{\tau_n}\left(1 + \frac{\tau_n\left(r_n+1\right)}{\tau_{n-1}+\tau_n}\right) \\
&= -\frac{1}{\tau_n}\left(1 + \frac{r_n+1}{\frac{1}{\tau_n}\left(\tau_{n-1}+\tau_n\right)}\right) \\
&= -\frac{1}{\tau_n}\left(1 + \frac{r_n+1}{\frac{1}{r_n}+1}\right) \\
&= -\frac{1}{\tau_n}\left(1 + \frac{r_n\left(r_n+1\right)}{1+r_n}\right) \\
&= -\frac{1}{\tau_n}\left(1+r_n\right)
\end{aligned}
$$

Lastly we need to simplify the coefficient of v^{n-1} using the results that we got for the second coefficient. We have $\frac{\tau_{n-1}+2\tau_n}{\tau_{n-1}(\tau_{n-1}+\tau_n)} = \frac{r_n(\tau_{n-1}+2\tau_n)}{\tau_n(\tau_{n-1}+\tau_n)}$, therefore

$$
\begin{aligned}
-\frac{1}{\tau_{n-1}} + \frac{\tau_{n-1}+2\tau_n}{\tau_{n-1}\left(\tau_{n-1}+\tau_n\right)} &= -\frac{\frac{\tau_n}{\tau_{n-1}}}{\tau_n} + \frac{r_n\left(\tau_{n-1}+2\tau_n\right)}{\tau_n\left(\tau_{n-1}+\tau_n\right)} \\
&= \frac{1}{\tau_n}\left(-r_n + \frac{r_n\left(\tau_{n-1}+2\tau_n\right)}{\tau_{n-1}+\tau_n}\right) \\
&= \frac{1}{\tau_n}\left(-r_n + \frac{r_n\left(\tau_{n-1}+\tau_n\right)+r_n\tau_n}{\tau_{n-1}+\tau_n}\right) \\
&= \frac{1}{\tau_n}\left(-r_n + r_n + \frac{r_n\tau_n}{\tau_{n-1}+\tau_n}\right) \\
&= \frac{1}{\tau_n}\frac{r_n\frac{\tau_n}{\tau_{n-1}}}{\frac{1}{\tau_{n-1}}\left(\tau_{n-1}+\tau_n\right)}
\end{aligned}
$$

$$= \frac{1}{\tau_n} \frac{r_n^2}{1+r_n}$$

Finally, putting all these simplifications together, we obtain a compact version for VS-BDF2:

$$\frac{1+2r_n}{1+r_n} v^{n+1} - (1+r_n)\, v^n + \frac{r_n^2}{1+r_n} v^{n-1} = \tau_n f\left(t_{n+1}, v^{n+1}\right) \tag{2.12}$$

2.2.1 Step size selection for BDF2

Now that we derived a variable step size version of BDF2 one may wonder how to adapt the time step. Our goal is to design an automatic procedure that works effectively and remains stable. Jannelli and Fazio suggest such an algorithm in their paper [5] for stiff ODEs. The adaption can be done by using the monitoring function

$$\eta^n = \frac{\|v^{n+1} - v^n\|}{\|v^n\| + \varepsilon_M} \tag{2.13}$$

where $\varepsilon_M > 0$ is a constant which assures that η^n stays within the given limits, so $\eta_{min} \leq \eta^n \leq \eta_{max}$. The monitoring function is then used to control the step size selection. If $\eta^n > \eta_{max}$ the chosen step size was too large. As a consequence the step size is being rejected and reduced by the chosen reduction factor, σ, before the procedure is being repeated. If $\eta^n < \eta_{min}$ the step size is accepted but for the next time step it is increased by the amplification factor, ρ.

Jannelli and Fazio not only introduce this monitoring function but also the following algorithm to select the next step size:

Algorithm 2.4: Step size selection [5]

Given: k_n, v^n, $time$

calculate v^{n+1} using the current step size k_n and the monitoring function η^n

IF $\eta_{min} \leq \eta^n \leq \eta_{max}$ THEN
  $time = time + k_n$
  IF $time > t_f$ THEN
    $time = t_f$
    $k_{n+1} = t_f - (time - k_n)$
  END
ELSE IF $\eta^n < \eta_{min}$
  $time = time + k_n$
  $k_{n+1} = \rho k_n$ with $\rho > 1$
  IF $k_{n+1} > k_{max}$ THEN
    $k_{n+1} = k_{max}$
  ELSE IF $k_{n+1} < k_{min}$ THEN
    $k_{n+1} = k_{min}$
  END
  IF $time > t_f$ THEN
    $time = t_f$
    $k_{n+1} = t_f - (time - k_n)$
  END
  continue with next time step
ELSE IF $\eta^n > \eta_{max}$
  $k_n = \sigma k_n$ with $0 < \sigma < 1$
  IF $k_n > k_{max}$ THEN
    $k_n = k_{max}$
  ELSE IF $k_n < k_{min}$ THEN

$k_n = k_{min}$

END

start the procedure again with the smaller time step

END

This algorithm requires the user to choose the parameters appropriately, which might be difficult especially since the behaviour of the function is unknown. The parameters represent the following:

$$
\begin{aligned}
k_n &= \text{current step size} \\
k_{min} &= \text{minimum allowed step size} \\
k_{max} &= \text{maximum allowed step size} \\
\eta^n &= \text{current value of the monitoring function} \\
\eta_{min} &= \text{minimum value of the monitoring function} \\
\eta_{max} &= \text{maximum value of the monitoring function} \\
v^n &= \text{solution vector at time step } n \\
\rho &= \text{amplification factor} \\
\sigma &= \text{reduction factor}
\end{aligned}
$$

The problem which arises when using this algorithm is that the range for η^n has to be large enough, otherwise the algorithm can be stuck in a loop of decreasing the stepsize until the limiting conditions for η^n are met. On the other hand, if the range for η^n is too wide the step size remains the same for most cases. In that case, if the step size is very small it stays small and the advantage in comparison with the uniform step size version no longer exists. Also, the amplification and reduction factors have to be chosen wisely.

For ODEs, Grigorieff proved that the amplification factor for a two step scheme (BDF2) can be chosen no larger than $1 + \sqrt{2}$ without potentially causing instability

([3], p. 405). However, stability for ODEs and PDEs is not the same. For ODEs we assume to have only one variable whereas for PDEs we have another parameter which changes. Becker has proven that for PDEs stability is guaranteed for an amplification factor of less than $\frac{2+\sqrt{13}}{3} \approx 1.86$ [1]. Emmrich improved this upper boundary to 1.91 [2].

Since we use the method of lines we have a fixed spatial grid and thus are in the case of an ODE. This means we can choose our amplification factor not larger than $1+\sqrt{2}$. However, since the problem is a PDE we will decrease the step size in space as well to get a sense of how the algorithm reacts on step size changes for both variables.

Besides the amplification factor all the other parameters can be chosen relatively arbitrarily. An investigation on how to choose them for some test problems will be done in the next chapter. Also, the results of a stability analysis for the Allen-Cahn Equation will be presented.

Chapter 3
Test Problems

The test problems that are being used in this thesis are three types:

i) A problem which is exponentially decaying with a known exact solution,

ii) A problem from biochemistry [7] and

iii) The Allen-Cahn Equation, a reaction-diffusion problem from mathematical Physics [7]

In this chapter VS-BDF2, which was introduced in section 2.2, is used to solve the problems mentioned above. The chapter is devided into three sections, one on each problem. In each section the problem itself will be presented as well as the parameters for Algorithm 2.4 which lead to satisfying results. For the first problem there will also be a numerical proof of the correctness of the fixed step size methods and for the Allen-Cahn Equation the results of some experiments on the stability of the BDF2 scheme are added.

3.1 A First Test Problem

This problem is an exponentially decaying problem which is designed so that we know the exact solution. We use this problem to verify that all the methods run correctly. The given PDE is

$$\begin{aligned} \frac{\partial}{\partial t}u &= \frac{\partial^2}{\partial x^2}u - 2u + 2e^{-2t} \qquad (3.1)\\ u(x,0) &= x(1-x)\\ u(0,t) &= u(1,t) = 0\\ x &\in [0,1] \ , \quad t > 0 \end{aligned}$$

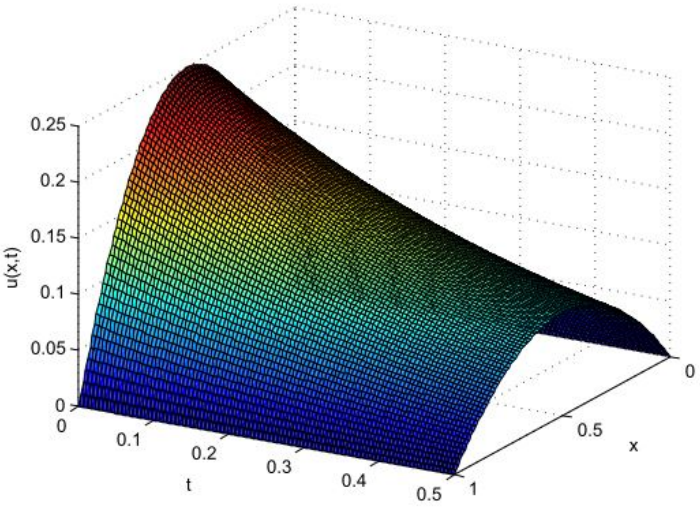

Figure 3.1: Exact solution to equation (3.1)

The exact solution for this problem is

$$u(x,t) = e^{-2t}x(1-x)$$

with $x \in [0,1]$ and $t > 0$.

First we test the BDF1 method. The method is run with different step sizes for space, h, and time, k. For the spatial step size a geometric sequence is chosen. The temporal step size is chosen accordingly using the ratio $k = \frac{1}{2}h$. Each time we compute the error between the computed solution and the exact solution at the final time step. This is done by using the maximum norm of the difference of the computed and the exact solution. The result of this error computation is being stored in a vector, err, with an entry for each step size. Finally, log ratio is the logarithmic ratio between two consecutive errors, thus

$$\text{log ratio} = \frac{\log\left(\frac{err_{i-1}}{err_i}\right)}{\log\left(\frac{h_{i-1}^2+k_{i-1}}{h_i^2+k_i}\right)}$$

The results of this experiment can be seen in Table 3.1.

h	k	err	log ratio
0.125000	0.062500	0.001251	–
0.062500	0.031250	0.000608	0.90483
0.031250	0.015625	0.000300	0.93938
0.015625	0.007813	0.000149	0.96588
0.007813	0.003906	0.000074	0.98193
0.003906	0.001953	0.000037	0.99071
0.001953	0.000977	0.000019	0.99529

Table 3.1: Convergence of BDF1. Log ratios appear to approach indicating first order convergence.

We observe that the last column converges to 1 as we reduce the step sizes. This means we can be sure that the method BDF1 works correctly. It is a first order scheme and thus the logarithmic ratio of the error should approximately be 1.

We do the same test for BDF2. The results we get can be seen in Table 3.2.

h	k	err	log ratio
0.125000	0.062500	0.000096	–
0.062500	0.031250	0.000023	1.76916
0.031250	0.015625	0.000006	1.86225
0.015625	0.007813	0.000001	1.92424
0.007813	0.003906	0.000000	1.96097
0.003906	0.001953	0.000000	1.98027
0.001953	0.000977	0.000000	1.99007

Table 3.2: Convergence of BDF2. Log ratios indicate second order convergence.

Here we can see that the logarithmic ratio converges to 2. Since BDF2 is a second order scheme this implies that BDF2 is also working correctly.

Now that we know that the schemes for uniform step sizes work as we expect, we switch to the variable step size scheme. We apply Algorithm 2.4 described in section 2.2.1. Each problem needs different parameters thus we have to adjust them for each

problem separately.

It is not obvious how to choose the parameters. While the range for η^n should be small to maintain accuracy it has to be large enough to not be caught in a loop of reducing the step size. Regarding the step size we also have to consider which jumps we allow and how small or how large the step size can be. Also it is not known precisely what effect on stability results from such step size changes [1, 2, 3].

By choosing different values for the parameters and constantly comparing the results obtained by the adaptive VS-BDF2 the following parameters led to satisfying results:

$$\begin{aligned} k_{min} &= 2^{-9} \cdot k \\ k_{max} &= t_1 - t_0 \\ \rho &= 2 \\ \sigma &= 0.5 \\ \eta_{max} &= 10^{-3} \\ \eta_{min} &= 0.8 \cdot \eta_{max} \\ tol &= k^4 \end{aligned}$$

Here "satisfying" is meant in terms of the achieved accuracy as well as the needed CPU-time.

The initial step size for time was chosen as $\frac{1}{32}$ and the spatial step size as $\frac{1}{1024}$. This configuration led to an accuracy which is reached if the uniform step size is $\frac{1}{2048}$ but was twice as fast. During this computation the smallest step size which the algorithm chose was $4.8828e-04$ and the largest step size was also $4.8828e-04$ which is the same as $\frac{1}{2048}$.

Since we are working on a PDE we need to have a look at the variable step size algorithm where both, spatial and temporal step size, change. However, the algorithm which we are studying is just adaptive in time. To still get a chance to find out how

the algorithm behaves if the spatial step size goes to zero, we use different spatial step sizes but we still consider a uniform mesh in space. Experimenting with the parameters we realize that for this problem it does not matter which step size for space is used. Also the error stays about the same. The only thing that changes is the time which the algorithm needs. While VS-BDF2 is slightly slower (5.85 seconds vs. 5.62 seconds) for a spatial step size $h = \frac{1}{512}$, it is twice as fast (32.46 seconds vs. 69.02 seconds) if $h = \frac{1}{1024}$ and for $h = \frac{1}{2048}$ it is faster by a factor of 3.7 (172.15 seconds vs. 641.26 seconds). Further results can be seen in Table 3.3.

$h = \frac{1}{512}$	variable scheme	uniform scheme
rejections/total	6/1025	
minimum k	4.8828e-04	9.7656e-04
maximum k	4.8828e-04	9.7656e-04
CPU-time (in sec)	5.851239	5.618539
error	5.6243e-09	2.2256e-08
$h = \frac{1}{1024}$	variable scheme	uniform scheme
rejections/total	6/1025	
minimum k	4.8828e-04	4.8828e-04
maximum k	4.8828e-04	4.8828e-04
CPU-time	32.462305	69.021668
error	5.6260e-09	5.5642e-09
$h = \frac{1}{2048}$	variable scheme	uniform scheme
rejections/total	6/1025	
minimum k	4.8828e-04	2.4414e-04
maximum k	4.8828e-04	2.4414e-04
CPU-time	172.147816	641.259316
error	5.6327e-09	1.4006e-09

Table 3.3: VS-BDF2 vs. BDF2 for the first test problem

If we compare the results for $h = \frac{1}{1024}$ and $h = \frac{1}{2048}$ we can see that the accuracy did not decrease as much as the CPU-time increased. So we realize that a smaller step size does not improve the results for this problem. Also the algorithm always chooses the same temporal step size. This might be a cause of the simplicity of the problem. Since it is exponentially decaying, it might not be very reasonable to change the step size.

3.2 A Problem from Biochemistry

The problem which we consider in this section is a problem which arises in biochemistry.

$$\begin{aligned} \frac{\partial}{\partial t}u &= \beta\frac{\partial^2}{\partial x^2}u - \frac{u}{1+u} \qquad (3.2)\\ u(x,0) &= 1\\ u(0,t) &= u(1,t) = 0\\ x &\in [0,1]\\ t &> 0 \end{aligned}$$

For our calculations β is chosen to be 1. If we just have a look at the graph this problem looks similar to the one presented in the previous section. However, it seems to be a more challenging problem, as it is difficult to find suitable parameters.

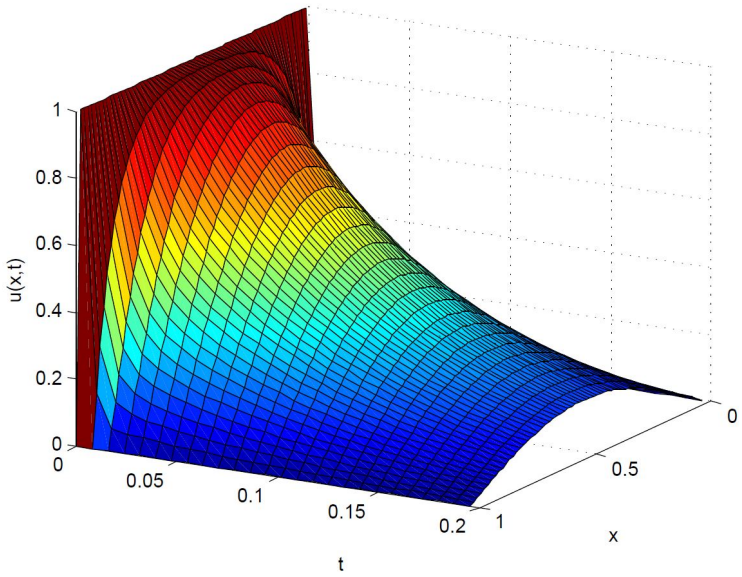

Figure 3.2: Solution for the Biochemistry Problem with h = 1/64 and k = 0.5h

Unlike the other problems the amplification factor influences the accuracy of the problem a lot. If we choose this factor to be larger than 1.91, which is the upper

bound for guaranteed stability proposed in [2], the solution lacks in accuracy. If we keep the amplification factor at 1.91 we obtain results of about the same accuracy as we get for a uniform time discretization (see Table 3.4). Nevertheless, the CPU time of the variable step size scheme is higher than for the uniform step size version.

$h = \frac{1}{128}$	variable scheme	uniform scheme
rejections/total	10/521	
minimum k	1.9073e-06	3.9064e-03
maximum k	6.4525e-04	3.9064e-03
CPU-time	0.239829	0.035578
error	0.0011	0.0012
$h = \frac{1}{256}$	variable scheme	uniform scheme
rejections/total	12/598	
minimum k	4.7684e-07	1.9531e-03
maximum k	5.8849e-04	1.9531e-03
CPU-time	0.885438	0.140825
error	0.0014	0.0012
$h = \frac{1}{512}$	variable scheme	uniform scheme
rejections/total	14/532	
minimum k	1.1921e-07	9.7656e-04
maximum k	0.0010	9.7656e-04
CPU-time	3.742556	1.464874
error	0.0011	0.0012
$h = \frac{1}{1024}$	variable scheme	uniform scheme
rejections/total	16/596	
minimum k	2.9802e-08	4.8828e-04
maximum k	9.3494e-04	
CPU-time (in sec)	23.172954	15.039555
error	8.9112e-04	4.0384e-04

Table 3.4: VS-BDF2 vs. BDF2 for the Biochemistry Problem

3.3 The Allen-Cahn Equation

The Allen-Cahn Equation is a problem involving a high gradient. This makes it a hard problem for any solver.

$$\begin{aligned} \frac{\partial}{\partial t}u &= \beta\frac{\partial^2}{\partial x^2}u + u - u^2 \\ u(x,0) &= 0.53x + 0.47\sin(-1.5\pi x) \\ u(-1,t) &= -1 \\ u(1,t) &= 1 \\ x &\in [0,1] \\ t &> 0 \end{aligned} \tag{3.3}$$

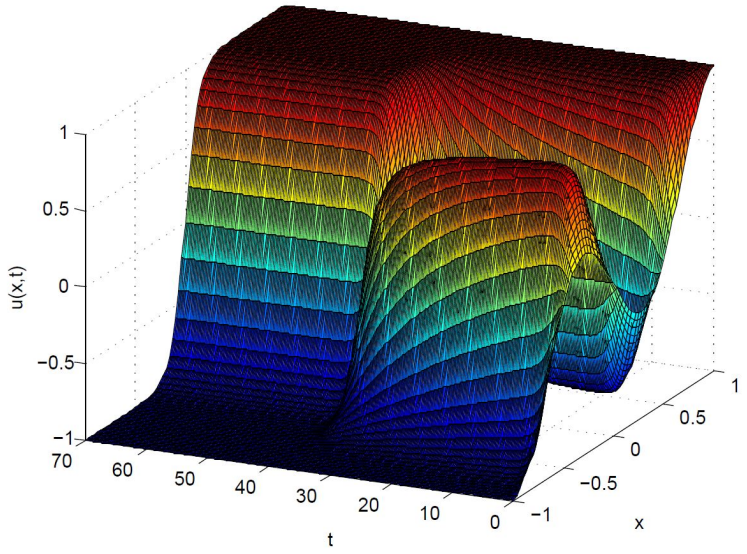

Figure 3.3: Solution for the Allen-Cahn Equation with h = 1/32 and k = 8h

Here we chose β to be 0.01. The idea is to use VS-BDF2 for this problem. Theoretically, the following should happen: In the region of the high gradient, the step sizes should be chosen very small while they should be bigger in other regions. With this choice we should be able to compute an accurate solution without having a high

CPU time. In practice we used VS-BDF2 and observed that the algorithm chose the step sizes exactly as we predicted. While experimenting, we found that the following choice of parameters yields a satisfying result:

$$\begin{aligned} k_{min} &= 2^{-11} \\ k_{max} &= t_1 - t_0 \\ \rho &= 1 + 1.4142 \\ \sigma &= 0.5 \\ \eta_{max} &= 10^{-3} \\ \eta_{min} &= 0.1 \cdot \eta_{max} \\ tol &= k^4 \end{aligned}$$

For a start, the spatial discretization was chosen to be $h = \frac{1}{256}$. For this configuration only 19 step sizes were rejected of 11685 in total; a relatively small amount. The accuracy is about the same, with an error of 4.8622e-05 for the variable step size scheme and 4.2344e-05 for the uniform step size scheme which uses the temporal step size $k = \frac{1}{256} = 3.90625e-03$. Both times the error is computed in relation to a uniform time step solution with a step size of $k = \frac{1}{4096}$. The variable step size scheme used a minimum step size of 8.5882e-04 and a maximum step size of 2.3925. The advantage for the CPU time was 8.2% since the variable step size scheme needs 71.9 seconds with this configuration whereas the uniform step size scheme needs 78.3 seconds. Further results can be seen in Table 3.5

Still the variable step size scheme is probably more accurate in the region of the turn since it uses a very small step size in that area. The smallest step size is used for $t = 36.318$ which is right in the area of the sharp gradient. Afterwards the step size is increased and the largest is used for $t = 45.497$. This can be seen in Figure 3.4.

To investigate the accuracy further we will now compare the error in the area of the sharp gradient. According to the results in Table 3.6 it seems like the uniform

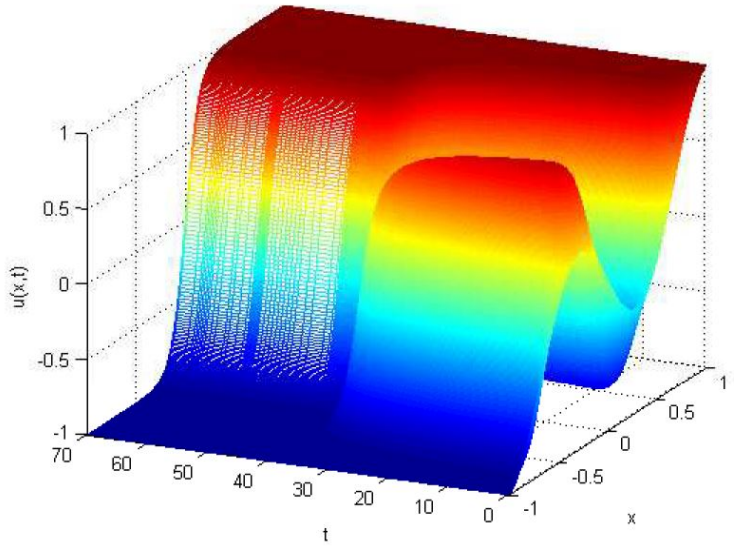

Figure 3.4: Solution to the Allen-Cahn Equation using VS-BDF2 with h = 1/256 and the parameters as mentioned above.

scheme would create a higher accuracy but if we take a look at the minimum used step size we realize that this is close to the step size which we used to compute the exact solution. It is still very likely that in this area the variable step size scheme is of higher accuracy than our sample solution. Since we do not have an exact solution it would be necessary to compute a solution with an even smaller uniform step size to compute a meaningful error. For such a computation a high performance computer would be needed since a usual laptop does not allow to work with matrices as large as they would have to be. Another way of solving this dilemma would be to use another programming language.

One might wonder why an amplification factor of $1 + 1.4142$ was used although so far stability for PDEs is only guaranteed for an amplification factor of ≤ 1.91. We will discuss that in the next section.

$h = \frac{1}{64}$	variable scheme	uniform scheme
rejections/total	18/11688	
minimum k	8.5882e-04	1.5625e-02
maximum k	2.3925	1.5625e-02
CPU-time (in sec)	6.537504	2.048366
error	7.1284e-04	6.9014e-04
$h = \frac{1}{128}$	variable scheme	uniform scheme
rejections/total	18/11685	
minimum k	8.5882e-04	7.8125e-03
maximum k	2.3925	7.8125e-03
CPU-time	16.217164	10.248174
error	1.5571e-04	1.7149e-04
$h = \frac{1}{256}$	variable scheme	uniform scheme
rejections/total	19/11685	
minimum k	8.5882e-04	3.9063e-03
maximum k	2.3925	3.9063e-03
CPU-time	71.916994	78.284406
error	4.8622e-05	4.2344e-05
$h = \frac{1}{512}$	variable scheme	uniform scheme
rejections/total	19/11682	
minimum k	8.5882e-04	1.9531e-03
maximum k	2.3925	1.9531e-03
CPU-time	538.930810	1036.027920
error	4.8219e-05	1.0085e-05

Table 3.5: VS-BDF2 vs. BDF2 for the Allen-Cahn Equation

$h = \frac{1}{64}$	variable scheme	uniform scheme
rejections/total	9/7376	
minimum k	4.6674e-04	1.5625e-02
maximum k	0.0137	1.5625e-02
CPU-time (in sec)	3.462705	1.048271
error	0.0256	0.0199
$h = \frac{1}{128}$	variable scheme	uniform scheme
rejections/total	9/7429	
minimum k	4.5623e-05	7.8125e-03
maximum k	0.0137	7.8125e-03
CPU-time	9.581836	6.045913
error	0.0106	0.0050
$h = \frac{1}{256}$	variable scheme	uniform scheme
rejections/total	9/7444	
minimum k	4.5623e-05	3.9063e-03
maximum k	0.0137	3.9063e-03
CPU-time	47.324385	48.033946
error	0.0069	0.0012
$h = \frac{1}{512}$	variable scheme	uniform scheme
rejections/total	9/7445	
minimum k	4.5623e-05	1.9531e-03
maximum k	0.0137	1.9531e-03
CPU-time	348.108156	652.074548
error	0.0059	2.9466e-04

Table 3.6: VS-BDF2 vs. BDF2 for the Allen-Cahn Equation in the critical area

3.3.1 Experiments on the stability of BDF2

From our experiments we found out that an amplification factor of $1 + 1.4142$ led to satisfying results and that the scheme seemed to be stable. If we just consider one fixed spatial step size we are in the case of solving a stiff ODE and as stated previously, for ODEs stability is guaranteed for a step size of less than or equal to $1 + \sqrt{2}$. But this thesis concerns experiments on the stability of the BDF2 scheme for PDEs in the method of lines. We do a sequence of experiments where both step sizes go to zero. The results of Emmrich's research [2] are useful but an amplification factor which is greater than 2 would be more practical. Wade conjectures that for PDEs the same boundary holds as for ODEs [8].

It is difficult to show instability in experiments since the claim is just an implication, not an if and only if condition. For the experiments instability means that the method has difficulties to compute an accurate solution. It is hard to find evidence for these difficulties. One either has to find abnormalities in the numbers or in purely looking at the plot and spotting areas where the function does not behave as one would expect. The result of a series of experiments with VS-BDF2 for the Allen-Cahn Equation can be seen in Table 3.7.

h	starting k	2	2.2	2.41421	2.75
0.03125	0.03125	2.71654E-03	NaN	NaN	NaN
0.01563	0.01563	6.90136E-04	3.22435E-01	NaN	NaN
0.00781	0.00781	1.71488E-04	1.71488E-04	NaN	NaN
0.00391	0.00391	4.23444E-05	4.23443E-05	4.22324E-05	NaN
0.00195	0.00195	1.00853E-05	1.00947E-05	1.00863E-05	6.99309E+181
0.00098	0.00098	2.05341E-06	2.05911E-06	2.05064E-06	2.05871E-06

Table 3.7: Results for the Allen-Cahn Equation

The first column is the spatial discretization, the second is the initial temporal discretization and in the other four columns the results for the different amplification factors which were used are listed. For this problem the whole time interval was

equally divided into 20 intervals. During these intervals the step size is kept the same. At the end of each interval the step size is either increased (by the chosen amplification factor) or reduced (by the factor 0.4). The strategy is to decrease once after two times of increasing. This assures that the step size does not become too large after a short time.

The NaN entries in this table are blow ups which means the numbers were too large for MATLAB. As can be seen in the table the problem seems to be stable even for an amplification factor of 2.41421. The smaller h and k get the more accurate the scheme seems to become, even for large amplification factors. However, these blow ups show us that there is something going on which should be investigated theoretically in the future.

Besides the difficulty to show instability, we do not know the exact solution, thus an approximated solution with a small step size is computed. And the question arises how we can measure the error. For all these experiments the error is the maximum norm of the difference of the solutions at the last time step.

Since it is difficult to show instability for the Allen-Cahn Equation with the initial data given the above initial conditions, we have chosen to drive the scheme unstable by more challenging initial conditions. The original initial data is smooth, and in fact in C^∞. To create more difficulties for the scheme the initial data was changed into

$$u(x,0) = \begin{cases} 2.33333 \cdot x + 1.33333, & \text{if } x \leq -0.5 \\ -1.2 \cdot x, & \text{if } -0.5 \leq x \leq 0.5 \\ 1.73333 \cdot x - 0.733333, & \text{if } x > 0.5 \end{cases} \tag{3.4}$$

This has a similar shape but two corners. Not even being C^1 should cause a bit more trouble for the scheme than a smooth initial function. However, it seems to be even stable for an amplification factor greater than $1 + \sqrt{2}$ as can be seen in Table 3.8 but at the same time the error is much larger than in Table 3.7. This implies

that the scheme has some difficulties solving the problem.

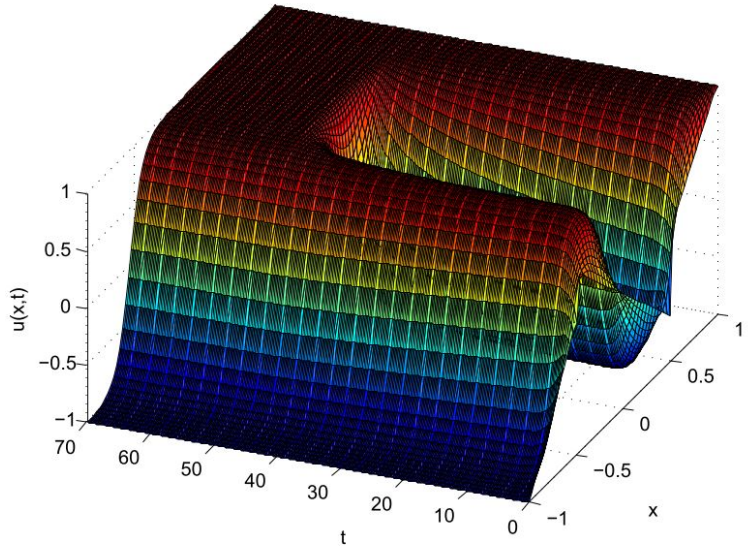

Figure 3.5: Solution for the Allen-Cahn Equation with new initial data and h = 1/32 and k = 8h

h	starting k	2	2.2	2.41421	2.75
0.03125	0.03125	1.06344E-01	NaN	NaN	NaN
0.01563	0.01563	4.59888E-02	4.59192E-02	NaN	NaN
0.00781	0.00781	2.14673E-02	2.14801E-02	NaN	NaN
0.00391	0.00391	9.85246E-03	9.85310E-03	9.86729E-03	NaN
0.00195	0.00195	4.18946E-03	4.18940E-03	4.18972E-03	5.32344E-02
0.00098	0.00098	1.39108E-03	1.39106E-03	1.39102E-03	1.39209E-03

Table 3.8: Results for the Allen-Cahn Equation with new initial data

Chapter 4
Conclusion

Based on the variable step size algorithm published by Jannelli and Fazio [5] we have adjusted the parameters for three test problems. For the first test problem, an exponentially decaying problem, the algorithm chose a uniform step size which was chosen independently of the initial step size. This indicates that this problem is not suitable for a variable step size algorithm. For the Biochemistry problem it was difficult to find any parameters for the adaptive VS-BDF2 to improve the performance of the uniform BDF2 scheme. A reason for that might be the similar shape to the first problem. For the Allen-Cahn Equation it turned out that an amplification factor of $\approx 1 + \sqrt{2}$ was a good choice. This is higher than the upper bound which guarantees stability that has been proven so far [2], yet the same as the limit for the ODE case [3]. The Allen-Cahn Equation seems to be a problem which is robust and big step size changes do not affect the solution as the initial step sizes for space and time go to zero. This suggests, that the bound of 1.91 is not sharp and that a greater amplification factor might still guarantee stability. It is worth a try to mimic the proof for ODEs in the PDE case and see whether similar results can be achieved.

In the future a fully adaptive scheme, which means adaptive in space and time together is desired. In our case, where only the temporal step size was varying, we had a vector of a fixed size whose values were changing. We could easily alter this to a variable spatial step size scheme, but applying variable step sizes in both time and space would be challenging. One of the biggest hurdles that would have to be taken in practice is the question about an appropriate data structure. Despite the effort, that a solution of this problem would bring, it is definitely worth a try as this approach could lead to undreamed-of possibilities.

Bibliography

[1] J. Becker, *A second order backward difference method with variable steps for a parabolic problem*, 1998

[2] E. Emmrich, *Stability and error of the variable two-step BDF for semilinear parabolic problems*, 2005

[3] E. Hairer, S.P. Nørsett, G. Wanner, *Solving Ordinary Differential Equations I - Nonstiff Problems*, Springer Verlag, 2008

[4] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems*, Springer-Verlag, 1991

[5] A. Jannelli, R. Fazio, *Adaptive stiff solvers at low accuracy and complexity*, 2006

[6] J.D. Lambert, *Numerical Methods For Ordinary Differential Systems*, Wiley, 1991

[7] A.Q.M. Khaliq, J. Martín-Vaquero, B.A. Wade, M. Yousuf, *Smoothing schemes for reaction-diffusion systems with nonsmooth data*, 2009

[8] B.A. Wade, *Private communication*

Appendix

In this appendix all methods that were used to obtain the results in this thesis are listed as well as a description for each method.

MATLAB-Function 1: forwardEuler.m

```
1  function u = forwardEuler(u,k,B,f,t,w)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %
4  % DESCRIPTION
5  % This function computes a solution of an ODE using forward euler.
6  %
7  % IN
8  % u = solution at the current time step
9  % k = stepsize for time
10 % B = matrix for calculating u_n+1
11 % f = pertubation function
12 % t = current time
13 % w = value of the preprocessing function (optional)
14 %
15 % OUT
16 % u = approximation of u for the next time step
17 %
18 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
19
20 if nargin<6
21     w = zeros(size(u));
22 end
23 % forward euler for obtaining approximation for u_n+1
24 u = u + k*(B*u+f(t,u+w));
```

Line 20-22: If no preprocessing values are defined we set $w = 0$.

Line 24: Forward Euler scheme: $u_{n+1} = u_n + k\left(Bu + f\left(t, u_n + w\right)\right)$

MATLAB-Function 2: BDF1_uniform.m

```
function [S u] = BDF1_uniform(x,h,t0,t1,k,beta,f,u0,iter,eps,w)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION
% This function computes a solution of a PDE of a reaction
% diffusion type using the BDF1 scheme.
%
% IN
% x = spatial discretization
% h = spatial stepsize
% t0 = start time
% t1 = final time
% k = stepsize for time
% beta = constant
% f = forcing function
% u0 = initial condition
% iter = number of iterations for corrector
% eps = error that is allowed (for stopping fix point iteration)
% w = preprocessing function (optional)
%
% OUT
% S = value of u at each time step
% u = solution of the partial differential equation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin<11
    w = @(y) 0.*y;
end

% Number of entries in u
N = size(x,1);
time = t0;
% preparation for the loop
u = u0;

% matrix in which all results are being stored
S = zeros(size(u0,1),(t1-t0)/k+1);
i = 1;
```

```
40 S(:,i) = u0;
41
42 B = beta/h^2*(diag(ones(N-1, 1), -1) ...
43     + diag(-2*ones(N, 1), 0) ...
44     + diag(ones(N-1, 1), 1));
45
46 w_value = w(x);
47
48 A = (eye(size(B))-k*B);
49 for t=t0+k:k:t1
50     time = time+k;
51     % predictor
52     u_new = forwardEuler(u,k,B,f,time,w_value);
53
54     it = 0;
55     err = inf;
56     % u is fixed for this loop
57     u_old = u;
58     % corrector
59     while err>eps && it<iter
60         u_new = A\(u + k*f(time,u_new + w_value));
61         err = norm(u_old-u_new,inf);
62         u_old = u_new;
63         it = it+1;
64     end
65     u = u_new;
66     % store new result
67     i = i+1;
68     S(:,i) = u;
69 end
```

Lines 27-29: If no preprocessing function is passed on we define $w(x) := 0 \cdot x$.

Line 35: Storing the initial values in u.

Line 38: Initializing the matrix S in which the vector u will be stored for each time step.

Line 42-44: Initializing the matrix B as defined in chapter 2.

Line 48: A is the matrix which we need for BDF1, so $A = I - kB$ where I is the identity matrix.

Lines 52: We use the Forward Euler method as a predictor for the next time step.

Line 59-64: We use BDF1 to correct the prediction. The correction is repeated until either the maximum norm of two consecutive corrections is smaller than a certain tolerance or until the maximum number of iterations is reached.

Line 65-68: Storing the new value.

MATLAB-Function 3: BDF2_uniform.m

```
1  function [S u] = BDF2_uniform(x,h,t0,t1,k,beta,f,u0,iter,eps,w)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %
4  % DESCRIPTION
5  % This function computes a solution of a PDE of a reaction
6  % diffusion type using the BDF2 scheme.
7  %
8  % IN
9  % x = spatial discretization
10 % h = spatial stepsize
11 % t0 = start time
12 % t1 = final time
13 % k = stepsize for time
14 % beta = constant
15 % f = forcing function
16 % u0 = initial condition
17 % iter = number of iterations to approximate u at the next timestep
18 % eps = error that is allowed (for stopping fix point iteration)
19 % w = preprocessing function (optional)
20 %
21 % OUT
22 % S = value of u at each time step
```

```
% u = solution of the partial differential equation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin<11
    w = @(y) 0.*y;
end

% Number of entries in u
N = size(x,1);
time = t0;
% preparation for the loop
u = u0;

% matrix in which all results are being stored
S = zeros(size(u0,1),(t1-t0)/k+1);
i = 1;
S(:,i) = u0;

B = beta/h^2*(diag(ones(N-1, 1), -1) ...
    + diag(-2*ones(N, 1), 0) ...
    + diag(ones(N-1, 1), 1));

% 2 initial values needed for BDF2
% -> use BDF1 to get the second value
w_value = w(x);
time = time + k;

% predictor
u_new = forwardEuler(u,k,B,f,time,w_value);

it = 0;
err = inf;
% u is fixed for this loop
u_old = u;
A = (eye(size(B))-k*B);
% corrector
while err>eps && it<iter
    u_new = A\(u + k*f(time,u_new + w_value));
    err = norm(u_old-u_new,inf);
    u_old = u_new;
    it = it+1;
```

```
65 end
66 u = u_new;
67 % store new result
68 i = i+1;
69 S(:,i) = u;
70 % loop for BDF2
71 for t=t0+2*k:k:t1
72     time = time + k;
73     % predictror
74     u_new = forwardEuler(u,k,B,f,time,w_value);
75
76     it = 0;
77     err = inf;
78     % u is fixed for this loop
79     u_old = u;
80     A = (eye(size(B))-2/3*k*B);
81     % corrector
82     while err>eps && it<iter
83         u_new = A\( 4/3*u-1/3*S(:,i-1)+2/3*k*f(time,u_new+w_value) );
84         err = norm(u_old-u_new,inf);
85         u_old = u_new;
86         it = it+1;
87     end
88     u = u_new;
89     % store new result
90     i = i+1;
91     S(:,i) = u;
92 end
```

Lines 27-29: If no preprocessing function is passed on we define $w(x) \coloneqq 0 \cdot x$.

Line 35: Storing the initial values in u.

Line 38: Initializing the matrix S in which the vector u will be stored for each time step.

Line 42-44: Initializing the matrix B as defined in chapter 2.

Line 52: Use Forward Euler to get a predictor for u_1.

Line 60-65: Use BDF1 to calculate u_1 which is needed to use BDF2.

Lines 74: Using Forward Euler as predictor.

Lines 82-87: Iterations for BDF2 which is used as the corrector. Like for BDF1 the correction is repeated until either the maximum norm of two consecutive corrections is smaller than a certain tolerance or until the maximum number of iterations is reached.

MATLAB-Function 4: BDF1_vspre.m

```
function [S u] = BDF1_vspre(x,h,t0,t1,kt,beta,f,u0,iter,eps,w)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION
% This function computes a solution of a PDE of a reaction
% diffusion type using a variable step size form of the BDF1 scheme.
%
% IN
% x = spatial discretization
% h = spatial stepsize
% t0 = start time
% t1 = final time
% kt = stepsize for time (vector)
% beta = constant
% f = forcing function
% u0 = initial condition
% iter = number of iterations for corrector
% eps = error that is allowed (for stopping fix point iteration)
% w = preprocessing function (optional)
%
% OUT
% S = value of u at each time step
% u = solution of the partial differential equation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin<11
    w = @(y) 0.*y;
end

```

```
% Number of entries in u
N = size(x,1);
time = t0;
% preparation for the loop
u = u0;

% matrix in which all results are being stored
S = zeros(size(u0,1),length(kt));
i = 1;
S(:,i) = u0;

B = beta/h^2*(diag(ones(N-1, 1), -1) ...
    + diag(-2*ones(N, 1), 0) ...
    + diag(ones(N-1, 1), 1));

% forward euler for obtaining approximation for u_n+1
w_value = w(x);

for t=1:length(kt)
    % set k to the current temporal step size
    k = kt(t);
    % update current time
    time = time+k;

    % predictor
    u_new = forwardEuler(u,k,B,f,time,w_value);

    it = 0;
    err = inf;
    % u is fixed for this loop
    u_old = u;
    A = (eye(size(B))-k*B);
    % corrector
    while err>eps && it<iter
        u_new = A\(u + k*f(time,u_new + w_value));
        err = norm(u_old-u_new,inf);
        u_old = u_new;
        it = it+1;
    end
    u = u_new;
    % store new result
    i = i+1;
```

```
    S(:,i) = u;
end
```

The only difference to the uniform version of BDF1 is that `k` is no longer a scalar but a vector in which the step sizes are being stored.

MATLAB-Function 5: BDF2_vspre.m

```
function [S u] = BDF2_vspre(x,h,t0,t1,k,beta,f,u0,iter,eps,w)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION
% This function computes a solution of a PDE of a reaction
% diffusion type using a variable step size form of the BDF2 scheme.
%
% IN
% x = spatial discretization
% h = spatial stepsize
% t0 = start time
% t1 = final time
% kt = stepsize for time (vector)
% beta = constant
% f = forcing function
% u0 = initial condition
% iter = number of iterations to approximate u at the next timestep
% eps = error that is allowed (for stopping fix point iteration)
% w = value of the preprocessing function (optional)
%
% OUT
% S = value of u at each time step
% u = solution of the partial differential equation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin<11
    w = @(y) 0.*y;
end

% Number of entries in u
N = size(x,1);
```

```
time = t0;
% preparation for the loop
u = u0;

% matrix in which all results are being stored
S = zeros(size(u0,1),length(k));
i = 1;
S(:,i) = u0;

B = beta/h^2*(diag(ones(N-1, 1), -1) ...
    + diag(-2*ones(N, 1), 0) ...
    + diag(ones(N-1, 1), 1));

% 2 initial values needed for BDF2
% -> use BDF1 to get the second value

w_value = w(x);
time = time + k(1);
% predictor
u_new = forwardEuler(u,k(1),B,f,time,w_value);

it = 0;
err = inf;
% u is fixed for this loop
u_old = u;
A = (eye(size(B))-k(1)*B);
% corrector
while err>eps && it<iter
    u_new = A\(u + k(1)*f(time,u_new + w_value));
    err = norm(u_old-u_new,inf);
    u_old = u_new;
    it = it+1;
end
u = u_new;

% store new result
i = i+1;
S(:,i) = u;

%loop for BDF2
for t=2:length(k)
    time = time + k(t);
```

```
75      rt = k(t)/k(t-1);
76      % predictor
77      u_new = forwardEuler(u,k(t),B,f,time,w_value);
78
79      it = 0;
80      err = inf;
81      % u is fixed for this loop
82      u_old = u;
83      A = ((1+2*rt)/(1+rt)*eye(size(B))-k(t)*B);
84      % corrector
85      while err>eps && it<iter
86          u_new = A\( (1+rt)*u-(rt)^2/(1+rt)*S(:,i-1)...
87                      +k(t)*f(time,u_new+w_value) );
88          err = norm(u_old-u_new,inf);
89          u_old = u_new;
90          it = it+1;
91      end
92      u = u_new;
93      % store new result
94      i = i+1;
95      S(:,i) = u;
96  end
```

For the variable step size BDF2 scheme `k` is no longer a scalar but a vector in which the step sizes are being stored. Thus in lines 86-87 the equation for VS-BDF2 is realized.

MATLAB-Function 6: BDF2_vs_adaptive.m

```
1   function [S u timedisc] = BDF2_vs_adaptive(x,h,t0,t1,k,beta,f,u0,...
2                                              iter,eps,kmin,kmax,rho,...
3                                              sigma,eta_min,eta_max,tol,w)
4   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5   %
6   % DESCRIPTION
7   % This function computes a solution of a PDE of a reaction diffusion
8   % type using a variable step size form of the BDF2 scheme and the
9   % step size selection algorithm suggested by Jannelli and Fazio.
10  %
11  % IN
```

```
12 % x = spatial discretization
13 % h = spatial stepsize
14 % t0 = start time
15 % t1 = final time
16 % k = initial temporal step size
17 % beta = constant
18 % f = forcing function
19 % u0 = initial condition
20 % iter = number of iterations to approximate u at the next timestep
21 % eps = error that is allowed (for stopping fix point iteration)
22 % kmin = minimal temporal stepsize
23 % kmax = maximal temporal stepsize
24 % rho = step size amplification factor
25 % sigma = step size reduction factor
26 % eta_min = lower bound for the tolerance
27 % eta_max = upper bound for the tollerance
28 % tol = of order of the rounding unit so that eta is modified as
29 % needed
30 % w = preprocessing function (optional)
31 %
32 % OUT
33 % S = value of u at each time step
34 % u = solution of the partial differential equation
35 %
36 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
37
38 if nargin<18
39     w = @(y) 0.*y;
40 end
41
42 % Number of entries in u
43 N = size(x,1);
44 time = t0;
45 % preparation for the loop
46 u = u0;
47
48 % matrix in which all results are being stored
49 i = 1;
50 S(:,i) = u0;
51
52 B = beta/h^2*(diag(ones(N-1, 1), -1) ...
53     + diag(-2*ones(N, 1), 0) ...
```

```
54      + diag(ones(N-1, 1), 1));
55
56  % 2 initial values needed for BDF2
57  % -> use BDF1 to get the second value
58
59  % forward euler for obtaining approximation for u_n+1
60  w_value = w(x);
61  status = 'again';
62  time_old = time;
63  timedisc(1) = time_old;
64  counter = 0;
65
66  while strcmp(status,'again')
67      time = time_old + k;
68      % predictor
69      u_new = forwardEuler(u,k,B,f,time,w_value);
70
71      it = 0;
72      err = inf;
73      % u is fixed for this loop
74      u_old = u;
75      A = (eye(size(B))-k*B);
76      % corrector
77      while err>eps && it<iter
78          u_new = A\(u + k*f(time,u_new + w_value));
79          err = norm(u_old-u_new,inf);
80          u_old = u_new;
81          it = it+1;
82      end
83      % get new temporal step size
84      [k, status, counter] = adaptive(k,kmin,kmax,rho,sigma,eta_min,...
85                                  eta_max,u,u_new,tol,time-k,t1,counter);
86  end
87  status = 'again';
88  u = u_new;
89
90  % store new result
91  i = i+1;
92  S(:,i) = u;
93  time_old = time_old + k;
94  timedisc(i) = timedisc(i-1)+k;
95
```

```
%loop for BDF2
while time_old<t1

    while strcmp(status,'again')
        % update current time
        time = time_old+k;
        rt = k/(timedisc(i)-timedisc(i-1));
        % predictor
        u_new = forwardEuler(u,k,B,f,time,w_value);

        it = 0;
        err = inf;
        % u is fixed for this loop
        u_old = u;
        A = ((1+2*rt)/(1+rt)*eye(size(B))-k*B);
        % corrector
        while err>eps && it<iter
            u_new = A\( (1+rt)*u-(rt)^2/(1+rt)*S(:,i-1)...
                        +k*f(time,u_new+w_value) );
            err = norm(u_old-u_new,inf);
            u_old = u_new;
            it = it+1;
        end
        % get new temporal step size
        [k, status, counter] = adaptive(k,kmin,kmax,rho,sigma,...
                                        eta_min,eta_max,u,u_new,...
                                        tol,time-k,t1,counter);
    end
    status = 'again';
    u = u_new;

    % store new result
    i = i+1;
    S(:,i) = u;
    time_old = time_old + k;
    timedisc(i) = timedisc(i-1)+k;
end
counter
```

The only difference to the preassigned version is that `k` is chosen adaptively.

MATLAB-Function 7: adaptive.m

```
function [k, status, counter] = adaptive(k,kmin,kmax,rho,sigma,
    eta_min,eta_max,un,un1,tol,time,tmax,counter)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION
% This function the next step size using a monitoring function.
%
% IN
% k = temporal discretization
% kmin = minimal temporal stepsize
% kmax = maximal temporal stepsize
% rho = step amplification factor
% sigma = step reduction factor
% eta_min = lower bound for the tolerance
% eta_max = upper bound for the tollerance
% un = approximation at time step n
% un1 = approximation at time step n+1
% tol = of order of the rounding unit so that eta is modified as
% needed
% time = current time
% tmax = maximal time
%
% OUT
% k = new temporal time step
% status = string that indicates whether this time step needs to be
% repeated or not
% counter = counts the number of rejected step sizes
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% monitor function
eta = norm(un-un1,inf)/(norm(un,inf)+tol);

%% adapt step size
% eta too large
if eta>eta_max
    k = sigma*k;
    status = 'again';
```

```
39      counter = counter+1;
40      % eta too small
41  elseif eta<eta_min
42      k = rho*k;
43      status = 'proceed';
44      % eta ok
45  else
46      status = 'proceed';
47  end
48
49  %% check step size
50  % step size is too small
51  if k<kmin
52      k = kmin;
53  % step size is too large
54  elseif k>kmax
55      k = kmax;
56  end
57
58  %% check maximal time
59  if time+k > tmax
60      k = tmax - time;
61  end
```

The function is an implementation of the step size selection algorithm proposed in [5].