A Computer–assisted Proof of the Existence of Solutions to a Boundary Value Problem with an Integral Boundary Condition

Oswald Fogelklou
Department of Mathematics
Uppsala University
P.O. Box 480, SE–751 06 Uppsala, Sweden.
oswald@math.uu.se
Gunilla Kreiss
Department of Information Technology
Uppsala University
Box 337, SE–751 05 Uppsala, Sweden.
gunilla.kreiss@it.uu.se
Warwick Tucker
Department of Mathematics
Bergen University
Johannes Brunsgate 12, 5008 Bergen, Norway.
warwick.tucker@math.uib.no
Malin Siklosi
Department of Information Technology
Uppsala University
Box 337, SE–751 05 Uppsala, Sweden.
malin.siklosi@scania.com

February 21, 2008

Abstract

In this paper, we present a computer–aided method (based on [Ya98]) that establishes the existence and local uniqueness of a stationary solution to the viscous Burgers' equation. The problem formulation involves a left boundary condition and one integral boundary condition, which is a variation of the approach taken in [Si04].

Key words: computer-assisted proof, numerical verification, viscous Burgers' equation, enclosure, existence, two-point boundary value problems, fixed-point problems
AMS subject classification: 65L10, 65G20, 34B15, 47H10

1 Introduction

Many interesting phenomena can be modelled by systems of hyperbolic conservation laws

$$u_t + f(u)_x = 0, \quad -\infty < x < \infty, \, t > 0. \tag{1}$$

Some examples are gas dynamics, magnetohydrodynamics and two phase flow. An important property of these systems is that even with smooth initial data discontinuities may develop. The corresponding parabolic system, where the 0 in the right hand side of (1) is replaced by εu_{xx}, is also important. In most physical situations there is some dissipative mechanism, numerical techniques involve numerical dissipation and taking the limit $\varepsilon \to 0$ is a way of selecting an entropy correct solution of the hyperbolic problem. A fundamental question is when the Riemann problem, (1) together with initial data

$$u(x,0) = \begin{cases} U_L & x < 0 \\ U_R & x > 0 \end{cases},$$

has a solution consisting of simple waves; shocks and rarefactions. For general systems, the results are limited to when the difference $|U_R - U_L|$ is sufficiently small.

We are interested in a related question for the corresponding parabolic system. Can left and right states, that are candidates for being a shock wave solution of the hyperbolic system, be connected by a smooth, travelling wave solution? Such a solution is called a viscous shock wave. As above, for general systems, results are limited to cases when the difference $|U_R - U_L|$ is sufficiently small. For an overview of the theory, see [Br00] or [HR02].

Without restriction, we look for a solution satisfying the steady state system

$$(U^2/2)_x = U_{xx}, \quad -\infty < x < \infty, \tag{2}$$

together with boundary conditions

$$\lim_{x\to\infty} U(x) = U_R, \quad \lim_{x\to-\infty} U(x) = U_L. \tag{3}$$

Classical techniques for proving the presence of solutions of differential equations often consist of constructing a sequence of approximations, and deriving uniform apriori bounds. If these bounds are strong enough, then the approximating sequence can be shown to converge to a solution. In, for example, [Gl65], [KL89], and [BS94], the approximations are constructed by numerical methods. However, since only apriori bounds are used, the approximations do not need to be explicitly computed. As already mentioned, these techniques are not generally applicable.

On the other hand, numerical computations indicate the existence of a viscous shock profile in many cases when existence has not been mathematically proved. From an applied point of view, a grid–converged solution is as much proof as one could wish to have. From a mathematical point of view, however, the numerical solution is a solution to a different problem, and it does not necessarily have anything to do with the solution of the original PDE.

This paper is a step in developing a framework for using a numerically computed approximate solutions with viscous shocks to prove existence of equations with such shocks. The idea of proof is to use a fixed–point formulation. This is achieved by (repeatedly) rewriting the problem in terms of the *defect* – the difference between a (numerically computed) approximate solution and the (unknown) exact solution. The success of the procedure will depend on the truncation error and the norm of the inverse of the linearized differential operator: by carefully choosing the formulation of the fixed–point problem, a sufficiently strong contraction can be achieved.

In principle, all abovementioned computations can be carried out by hand. In practice, however, the amount of work is massive, and a computer must be used. The use of computers as an integral part of a mathematical proof requires the use of auto–validated numerics. Such ideas have previously been used for ODEs ([Tu02], [BM98], [Fa95]), as well as for some strongly dissipative PDEs ([Zg02], [MZ01]), and for two–point boundary value problems ([Ås04], [Pl01], [Na92])

In this paper we successfully apply the abovementioned approach to the viscous Burgers' equation. The aim is the development of the technique, not the result in itself. It is well known that for this scalar problem, existence of solutions can be proven by other means. The viscous Burgers' equation is given by

$$u_t + \left(\frac{1}{2}u^2\right)_x = \varepsilon u_{xx}, \tag{4}$$

where u is the velocity and ε is the viscosity coefficient. This is also referred to as the one–dimensional nonlinear diffusion or heat flow equation, by J.M. Burgers himself [Bu74]. It is a fundamental equation in fluid mechanics, and describes the motion of an infinitely compressible medium without pressure, shear and vortex motion. With the Cole–Hopf substitution

$$u = -2\varepsilon \frac{1}{\phi}\frac{\partial \phi}{\partial x},$$

equation (4) is transformed into

$$\frac{\partial \phi}{\partial x}\left(\frac{\partial \phi}{\partial t} - \varepsilon \frac{\partial^2 \phi}{\partial x^2}\right) = \phi \frac{\partial}{\partial x}\left(\frac{\partial \phi}{\partial t} - \varepsilon \frac{\partial^2 \phi}{\partial x^2}\right),$$

which has the general solution

$$C(t)\phi = D(t)(\frac{\partial \phi}{\partial t} - \varepsilon \frac{\partial^2 \phi}{\partial x^2}).$$

If $C(t) = 0$ and $D(t) \neq 0$ we get

$$\frac{\partial \phi}{\partial t} = \varepsilon \frac{\partial^2 \phi}{\partial x^2},$$

i.e. ϕ satisfies the heat equation.

The paper is organized as follows. The main result is presented as a theorem in the next section. The notation is described in section 3. In section 4 the particular problem that we work with is presented. Yamamoto's method [Ya98] is presented in section 5. This method is applied to the problem presented in section 6. In section 7 an algorithm for using Yamamoto's method is presented. Results other than the main result are shown in section 8. In section 9 we discuss how our method can be improved, and what problems we will study in the future. The appendix consists of all MATLAB codes.

2 Main result

Theorem 2.1. *For* $\varepsilon \geq 0.085$ *there exists a unique solution to the problem*

$$u(x)u'(x) = \varepsilon u''(x), \quad u(-1) = 1, \quad \int_{-1}^{1} u(x)dx = 0. \tag{5}$$

Proof. In this section we show that if a unique solution exists for $\varepsilon = \hat{\varepsilon}$ then a unique solution also exists for all $\varepsilon > \hat{\varepsilon}$. In sections 4–7 and the remark in section 8 we prove uniqueness for $\varepsilon = 0.085$. Therefore the theorem follows. $\square$

2.1 Existence of solutions for larger values of ε

Suppose we have proved existence and local uniqueness of the solution to the equation

$$\hat{u}\hat{u}' = \hat{\varepsilon}\hat{u}'',\ \hat{u}(-1) = 1,\ \hat{u}(1) = -1,$$

which is equivalent (due to anti–symmetry) to the equation

$$\hat{u}\hat{u}' = \hat{\varepsilon}\hat{u}'',\ \hat{u}(-1) = 1,\ \int_{-1}^{1} \hat{u}(x)dx = 0.$$

How do we prove existence of the solution to the equation

$$uu' = \varepsilon u'',\ u(-1) = 1,\ u(1) = -1, \tag{6}$$

where $\varepsilon > \hat{\varepsilon}$? With the ansatz $u(x) = K\hat{u}(ax)$ in $uu' = \varepsilon u''$ we get

$$K^2 a\hat{u}\hat{u}' - \varepsilon K a^2 \hat{u}'' = K^2 a(\hat{u}\hat{u}' - \frac{\varepsilon a}{K}\hat{u}'') = 0, \tag{7}$$

which is true if

$$\frac{\varepsilon a}{K} = \hat{\varepsilon} \Leftrightarrow K = \frac{\varepsilon}{\hat{\varepsilon}}\xi. \tag{8}$$

With the ansatz in $u(\pm 1) = \mp 1$ we get $K\hat{u}(\pm a) = \mp 1$ and using (8) we have

$$a\hat{u}(a) = -\frac{\hat{\varepsilon}}{\varepsilon}. \tag{9}$$

Since $\hat{u}$ is continuous,

$$\lim_{a \to 1-} a\hat{u}(a) = -1 \text{ and } \lim_{a \to 0} a\hat{u}(a) = 0 \tag{10}$$

Therefore the intermediate–value theorem says that (9) has at least one solution if $-\hat{\varepsilon}/\varepsilon \in [0, -1]$ i.e. $\hat{\varepsilon} < \varepsilon$. So existence of solutions for larger values of ε is proved.

2.2 Uniqueness

Suppose that we have two solutions of (5) say

$$(u^2(x))'/2 = \varepsilon u''(x) \quad u(-1) = 1, \quad \int_{-1}^{1} u(x)dx = 0 \tag{11}$$

and

$$(v^2(x))'/2 = \varepsilon v''(x) \quad v(-1) = 1, \quad \int_{-1}^{1} v(x)dx = 0. \tag{12}$$

Define $f = u - v$. If we subtract (12) from (11) we get

$$(u^2(x))'/2 - (v^2(x))'/2 = \varepsilon f''(x) \quad f(-1) = 0, \quad \int_{-1}^{1} f(x)dx = 0 \tag{13}$$

so that

$$\frac{1}{2}((u+v)f)' = \varepsilon f''$$

and after integration we get

$$\frac{1}{2}(u+v)f = \varepsilon f' + C.$$

Using integrating factor we achieve

$$\frac{d}{dx}(e^{-\frac{1}{2\varepsilon}\int_{-1}^{x}(u(\xi)+v(\xi))d\xi} f(x)) = -\frac{C}{\varepsilon}.$$

With $f(-1) = 0$ from (13) we have

$$e^{-\frac{1}{2\varepsilon}\int_{-1}^{x}(u(\xi)+v(\xi))d\xi} f(x) = -\int_{-1}^{x} \frac{C}{\varepsilon} dy$$

or

$$f(x) = -\frac{C}{\varepsilon}(x+1)e^{\frac{1}{2\varepsilon}\int_{-1}^{x}(u(\xi)+v(\xi))d\xi}.$$

If $C > 0$ then $f < 0$ so that $\int_{-1}^{1} f(x)dx = 0$ in (13) is violated. If $C < 0$ then $f > 0$ and again $\int_{-1}^{1} f(x)dx = 0$ is violated. Hence $f = C = 0$ so that $u = v$. Therefore the solution of (14) is unique.

3 Notation

In this paper we will work with meshes on the interval $[-1, 1]$. A *mesh on the interval* $[-1, 1]$ is a set of *mesh points* $x_0 = -1 < x_1 < \ldots < x_{N-1} < x_N = 1$. The set $\{[x_0, x_1], [x_1, x_2], \ldots, [x_{N-1}, x_N]\}$ is called a *partition of the interval* $[-1, 1]$. Since we only work with the interval $[-1, 1]$ we will from now on omit "on the interval $[-1, 1]$" and "of the interval $[-1, 1]$". A *uniform partition* is a partition whose elements have the same length. It is clear what is meant by a *uniform mesh*. Given a partition and a function $u\colon [-1, 1] \to \mathbb{R}$, we define $u_i = u(x_i)$, and let u^h be the function which is linear on each element of the partition with $u^h(x_i) = u_i$. Since u^h is uniquely defined by the mesh points we can represent u^h by the vector $(u_0, u_1, \ldots, u_N)^T$ called $\overline{u^h}$. Throughout this text vectors are denoted by strokes. Finally the projection Π^h is the mapping defined by $u \mapsto u^h$.

4 Burgers' Equation with an Integral Boundary Condition

We study stationary solutions of (4) i.e. with $u_t = 0$, together with a boundary/integral condition:

$$u(x)u'(x) = \varepsilon u''(x) \quad u(-1) = 1, \quad \int_{-1}^{1} u(x)dx = 0. \tag{14}$$

This is a reformulation of the system studied by Siklosi and Åsén [Si04], where $u(1) = -1$ instead of the integral condition was used. The reason for studying the modification (14) is that we expect this to improve the numerics. Note, however, that the problems are mathematically equivalent. Both have the unique solution

$$u(x) = -\frac{\tanh(rx)}{\tanh(r)}, \text{ where } r \text{ satisfies } r\tanh(r) = \frac{1}{2\varepsilon}.$$

The corresponding integral equation formulation of (14) is

$$u(x) = \frac{1}{2\varepsilon}\int_{-1}^{x} u(\tau)^2 d\tau - \frac{1+x}{4\varepsilon}\int_{-1}^{1}\int_{-1}^{x} u(\tau)^2 d\tau dx - x \overset{\text{def}}{=} F(u). \tag{15}$$

Let V_1 be the space $V_1 = \{v \in C[-1,1] | v(-1) = 1, \int_{-1}^{1} v(x)dx = 0\}$. Then the solution of (14) is in V_1. Let S_1^h be the intersection of V_1 and the space of all piecewise linear functions on some partition of $[-1,1]$ into N intervals (note that S_1^h has dimension $N-1$). Let u^h be a piecewise linear approximate solution of (15) and assume $u^h \in V_1$. Then u^h is in S_1^h. Let V_2 be the space defined by

$$V_2 = \left\{ v \in C[-1,1] | v(-1) = 0, \int_{-1}^{1} v(x)dx = 0 \right\}. \tag{16}$$

Then the error $w = u - u^h$ is in V_2. Let S_2^h be the space of all piecewise linear functions in V_2 on the same partition of $[-1,1]$ which was used for S_1^h. Then w^h, a piecewise linear approximate solution of the error w, is in S_2^h. We now define the Banach space $X = S_2^h \times X^\infty$ where $X^\infty = (I - \Pi^h)V_2$. This gives the fixed point equation

$$w = T(w), \quad w \in X, \tag{17}$$

where $w = (w^h, w^\infty) \in X$ and $T(w) = (T_h(w), T_\infty(w))$ with

$$\begin{aligned} T_h(w) &= (I - A^h)^{-1}(\Pi^h F(u^h + w^h + w^\infty) - (u^h + A^h w^h)) \text{ and} && (18)\\ T_\infty(w) &= (I - \Pi^h)F(u^h + w^h + w^\infty), && (19) \end{aligned}$$

where A^h is an approximation of the Fréchet derivative described in subsection 6.2. See also [Si04].

5 General Statement of Convergence Conditions

For $w = (w^h, w^\infty) \in X$ we will use the notation

$$\begin{aligned} (w)_i &= |w^h(x_i)| \geq 0 \ \forall i \in \{1, \ldots, N-1\} \text{ and} \\ (w)_N &= ||w^\infty||_\infty \geq 0. \end{aligned}$$

Suppose we want to prove the existence of a unique solution of (17) in a set W, referred to as a candidate set. Taking a vector $\overline{W} \in \mathbb{R}^N$ with positive components

$$\overline{W} = (W_1, \ldots, W_N)^T, \tag{20}$$

a candidate set W is defined by

$$W = \{w \in X | (w)_i \leq W_i, \ \forall i \in \{1, \ldots, N\}\}. \tag{21}$$

In [Ya98], sufficient conditions on W to prove the existence and local uniqueness of a solution to (17) are derived. The proof involves assumptions on bounds of $T(0)$ and T' respectively, where T' is the Frechét derivative of T. For the reader's convenience, we give the assumptions and the theorem from [Ya98].

Assumption 1. *There is a vector $\overline{Y} = (Y_1, \ldots, Y_N)^T \in \mathbb{R}^N$ with positive components, such that the conditions*

$$(T(0))_i \leq Y_i \ \forall i \in \{1, \ldots, N\}$$

hold.

Assumption 2. *The operator T has a Fréchet derivative T' with the following property. For any $\overline{W}$ there exists a vector $\overline{Z} = (Z_1, \ldots, Z_N)^T \in \mathbb{R}^N$ with nonnegative components such that the conditions*

$$(T'(\tilde{w})w)_i \leq Z_i \ \forall i \in \{1, \ldots, N\}$$

hold for any w, $\tilde{w} \in W$. Since the Z_i's satisfying the above inequality depend on $\overline{W}$ in general, we write them as $Z_i(\overline{W})$.

We define the set K in X by

$$K = \{v \in X | (v)_i \leq Y_i + Z_i(\overline{W}) \ \forall i \in \{1, \ldots, N\}\}$$

Theorem 5.1. *If $K \subset W$ holds for the candidate set W defined by (21), then there exists a solution to (20) in K. Moreover, the solution is unique within the set W.*

The straightforward proof is based on Banach's fixed point theorem. In the proof, it is shown that the set K includes the image $T(W)$. In the next section, we will outline in detail how these bounds can be rigorously verified in computations and show one approach on how to obtain a suitable W.

6 Convergence Conditions for Burgers' Equation

In this section, we derive Y and Z such that Assumption 1 and Assumption 2 hold for Burgers' equation. In order to facilitate comparison with the numerical implementation, key expressions are labelled with names corresponding to the names used in the code.

We first consider the derivation of Y, which requires bounds on $(T(0))_i$. From (18), we have

$$T_h(0) = \left(I - A^h\right)^{-1} \left(\Pi^h F(u^h) - u^h\right). \tag{22}$$

6.1 Estimation of the Operator F

Let us first consider the evaluation of $\Pi^h F(u^h)$. We have

$$\Pi^h F(u^h)(x_i) = \frac{1}{2\varepsilon}\int_{-1}^{x_i} u^h(\tau)^2 d\tau - \frac{1+x_i}{4\varepsilon}\int_{-1}^{1}\int_{-1}^{x} u^h(\tau)^2 d\tau dx - x_i. \tag{23}$$

We have

$$u^h(\tau) = u_i + \frac{u_{i+1}-u_i}{h_{i+1}}(\tau - x_i) \quad \text{for } x_i \le x \le x_{i+1}, \tag{24}$$

where $h_{i+1} = x_{i+1} - x_i$ and we will use the notation $h = x_{i+1} - x_i$ for uniform meshes. From (24) we get

$$\int_{x_i}^{x} u^h(\tau)^2 d\tau = \frac{h_{i+1}}{3(u_{i+1}-u_i)}\left((u_i + \frac{u_{i+1}-u_i}{h_{i+1}}(x-x_i))^3 - u_i^3\right)$$

for $x_i \le x \le x_{i+1}$, especially

$$\int_{x_i}^{x_{i+1}} u^h(\tau)^2 d\tau = \frac{h_{i+1}}{3}\left(u_i^2 + u_i u_{i+1} + u_{i+1}^2\right) \overset{\text{def}}{=} a_i, \text{ if } i \in \{0, 1, \ldots N-1\}. \tag{25}$$

Hence, for $x_i \le x \le x_{i+1}$

$$\int_{-1}^{x} u^h(\tau)^2 d\tau = \sum_{k=0}^{i-1} a_k + \frac{h_{i+1}}{3(u_{i+1}-u_i)}\left(\left(u_i + \frac{u_{i+1}-u_i}{h}(x-x_i)\right)^3 - u_i^3\right).$$

We then obtain

$$\int_{x_i}^{x_{i+1}}\int_{-1}^{x} u^h(\tau)^2 d\tau dx = h_{i+1}\left(\frac{h_{i+1}}{12}\left(3u_i^2 + 2u_i u_{i+1} + u_{i+1}^2\right) + \sum_{k=0}^{i-1} a_k\right) \overset{\text{def}}{=} b_i \text{ if } i \in \{0, 1, \ldots, N-1\}.$$

Therefore

$$\int_{-1}^{1}\int_{-1}^{x} u^h(\tau)^2 d\tau dx = \sum_{k=0}^{N-1} b_k.$$

We can now express (23) as

$$\Pi^h F(u^h)(x_i) = \frac{1}{2\varepsilon}\sum_{k=0}^{i-1} a_k - \frac{1+x_i}{4\varepsilon}\sum_{k=0}^{N-1} b_k - x_i.$$

6.2 Estimation of the Fréchet Derivative of F

In the rest of the paper we will compute function values at the points $x_1, x_2, \ldots, x_N$ although the dimension of our function spaces is $N-1$ and it would be enough to study the points $x_1, x_2, \ldots, x_{N-1}$. The reason is that we want to control the error on the right boundary $x_N = 1$. We need to compute A^h, an approximate Fréchet derivative of $\Pi^h F(v)$, at $v = u^h$. Since A^h is a linear operator on the finite dimensional space S^h, there is a matrix $\tilde{B}$ such that the coefficient vector of $A^h v^h$ can be expressed as $\tilde{B}\overline{v^h}$. The Fréchet derivative of the operator F is

$$F'(u)v = \frac{1}{\varepsilon}\int_{-1}^{x} u(\tau)v(\tau)d\tau - \frac{1+x}{2\varepsilon}\int_{-1}^{1}\int_{-1}^{x} u(\tau)v(\tau)d\tau dx. \tag{26}$$

Hence, the elements $\tilde{B}_{ij}$ of $\tilde{B}$ are given by

$$\tilde{B}_{ij} = \frac{1}{\varepsilon}\int_{-1}^{x_i} u^h(\tau)\psi_j(\tau)d\tau - \frac{1+x_i}{2\varepsilon}\int_{-1}^{1}\int_{-1}^{x} u^h(\tau)\psi_j(\tau)d\tau dx, \text{ if } i,j \in \{1,2,\dots,N\}. \tag{27}$$

We have

$$u^h(\tau) = \begin{cases} u_{i-1} + \frac{u_i - u_{i-1}}{h_i}(\tau - x_{i-1}) & x_{i-1} \le \tau \le x_i \\ u_i + \frac{u_{i+1}-u_i}{h_{i+1}}(\tau - x_i) & x_i \le \tau \le x_{i+1} \end{cases}$$

and for $i = 1, \dots, N-1$, we have

$$\psi_i(\tau) = \begin{cases} (\tau - x_{i-1})/h_i & x_{i-1} \le \tau \le x_i \\ (x_{i+1} - \tau)/h_{i+1} & x_i < \tau \le x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

together with

$$\psi_N(\tau) = \begin{cases} (\tau - x_{N-1})/h_N & x_{N-1} \le \tau \le x_N = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The integral appearing in the first term of (27) can be evaluated via

$$\int_{-1}^{x} u^h(\tau)\psi_j(\tau)d\tau =$$
$$\begin{cases} 0 & 0 \le x \le x_{j-1} \\ u_{j-1}(x - x_{j-1})^2/2h_j + (u_j - u_{j-1})(x-x_{j-1})^3/3h_j^2 & x_{j-1} < x \le x_j \\ \frac{u_j}{2}(h_{j+1} - \frac{(x-x_{j+1})^2}{h_{j+1}}) + \frac{(u_{j+1}-u_j)(x-x_j)^2}{h_{j+1}^2}(\frac{h_{j+1}}{2} - \frac{x-x_j}{3}) + h_j(\frac{u_{j-1}}{6} + \frac{u_j}{3}) & x_j < x < x_{j+1} \\ h_j(\frac{u_{j-1}}{6} + \frac{u_j}{3}) + h_{j+1}(\frac{u_{j+1}}{6} + \frac{u_j}{3}) & x_{j+1} \le x \le 1. \end{cases} \tag{28}$$

At the nodes, this specialises to

$$c_{ij} \overset{\text{def}}{=} \int_{-1}^{x_i} u^h(\tau)\psi_j(\tau)d\tau = \begin{cases} 0 & i < j \\ h_j(\frac{u_j}{3} + \frac{u_{j-1}}{6}) & i = j \\ h_j(\frac{u_{j-1}}{6} + \frac{u_j}{3}) + h_{j+1}(\frac{u_{j+1}}{6} + \frac{u_j}{3}) & i > j. \end{cases} \tag{29}$$

Using (28) above, we can also evaluate the second integral of (27):

$$\begin{aligned} d_j &\overset{\text{def}}{=} \int_{-1}^{1}\int_{-1}^{x} u^h(\tau)\psi_j(\tau)d\tau dx \\ &= \int_{x_{j-1}}^{x_j}\int_{-1}^{x} u^h(\tau)\psi_j(\tau)d\tau dx + \int_{x_j}^{x_{j+1}}\int_{-1}^{x} u^h(\tau)\psi_j(\tau)d\tau dx + \int_{x_{j+1}}^{1}\int_{-1}^{x} u^h(\tau)\psi_j(\tau)d\tau dx \\ &= \frac{h_j^2}{12}(u_{j-1} + u_j) + h_{j+1}^2(\frac{u_{j+1}}{12} + \frac{u_j}{4}) + h_{j+1}h_j(\frac{u_{j-1}}{6} + \frac{u_j}{3}) \\ &\quad +(1 - x_{j+1})\left(h_j(\frac{u_{j-1}}{6} + \frac{u_j}{3}) + h_{j+1}(\frac{u_{j+1}}{6} + \frac{u_j}{3})\right) \text{ if } j \in \{1,2,\dots,N-1\} \end{aligned}$$

together with

$$d_N \stackrel{\text{def}}{=} \int_{x_{N-1}}^{1} \int_{-1}^{x} u^h(\tau)\psi(\tau)d\tau dx = \frac{h_N^2}{12}(u_{N-1} + u_N).$$

Hence, we can write

$$\tilde{B}_{ij} = \frac{1}{\varepsilon}c_{ij} - \frac{1+x_i}{2\varepsilon}d_j \text{ for } i,j \in \{1,2,\ldots,N\}. \tag{30}$$

6.3 Candidate Vector for Satisfying Assumption 1

From (22), we can now obtain bounds on $(T(0))_i$ by solving

$$(I - \tilde{B})\overline{T_h(0)} = \overline{\Pi^h F(u^h)} - \overline{u^h}. \tag{31}$$

From (19) we have

$$T_\infty(0) = \left(I - \Pi^h\right) F\left(u^h\right).$$

We can get a bound on $T_\infty(0)$ via the following proposition.

Theorem 6.1. *Suppose $v \in C^1([x_i, x_{i+1}])$, and let v^h be the straight line between $(x_i, v(x_i))$ and $(x_{i+1}, v(x_{i+1}))$. Then*

$$\max_{x\in[x_i,x_{i+1}]} |v^h(x) - v(x)| \le \frac{x_{i+1}-x_i}{2} \max_{x\in[x_i,x_{i+1}]} |v'(x)|.$$

Now suppose that v belongs to $C^2([x_i, x_{i+1}])$. Then

$$\max_{x\in[x_i,x_{i+1}]} |v^h(x) - v(x)| \le \frac{(x_{i+1}-x_i)^2}{8} \max_{x\in[x_i,x_{i+1}]} |v''(x)|.$$

Proof. In both cases, we have

$$v^h(x) = v(x_i) + \frac{x - x_i}{x_{i+1}-x_i}(v(x_{i+1}) - v(x_i)).$$

In the first case we have

$$v(x_i) = v(x) + (x_i - x)v'(\xi_1) \text{ and } v(x_{i+1}) = v(x) + (x_{i+1} - x)v'(\xi_2),$$

where $\{x, \xi_1, \xi_2\} \subset [x_i, x_{i+1}]$, so that

$$\begin{aligned}
|v^h(x) - v(x)| &= \left|(x_i - x)v'(\xi_1) + \frac{x-x_i}{x_{i+1}-x_i}\Big((x_{i+1}-x)v'(\xi_2) - (x_i - x)v'(\xi_1)\Big)\right| \\
&\le \max_{x\in[x_i,x_{i+1}]} |v'(x)| \frac{|(x_i - x)(x_{i+1}-x_i) + (x-x_i)^2| + |(x_{i+1}-x)(x-x_i)|}{x_{i+1}-x_i} \\
&\le \max_{x\in[x_i,x_{i+1}]} \frac{2|v'(x)|}{x_{i+1}-x_i} \max_{x\in[x_i,x_{i+1}]} (x-x_i)(x_{i+1}-x) = \frac{x_{i+1}-x_i}{2} \max_{x\in[x_i,x_{i+1}]} |v'(x)|.
\end{aligned}$$

In the second case we have

$$\begin{aligned} v(x_i) &= v(x) + (x_i - x)v'(x) + (x_i - x)^2 \frac{v''(\eta_1)}{2} \text{ and} \\ v(x_{i+1}) &= v(x) + (x_{i+1} - x)v'(x) + (x_{i+1} - x)^2 \frac{v''(\eta_2)}{2}, \end{aligned}$$

where $\{x, \eta_1, \eta_2\} \subset [x_i, x_{i+1}]$, so that

$$\begin{aligned} |v^h(x) - v(x)| = \Bigg| &(x_i - x)v'(x) + (x_i - x)^2 \frac{v''(\eta_1)}{2} \\ &+ \frac{x - x_i}{x_{i+1} - x_i} \left((x_{i+1} - x_i)v'(x) + \frac{(x_{i+1} - x)^2 v''(\eta_2) - (x_i - x)^2 v''(\eta_1)}{2} \right) \Bigg| \\ \leq &\max_{x \in [x_i, x_{i+1}]} \frac{|v''(x)|}{2} \frac{|(x_i - x)^2 (x_{i+1} - x_i) - (x - x_i)^3| + |(x_{i+1} - x)^2 (x - x_i)|}{x_{i+1} - x_i} \\ \leq &\max_{x \in [x_i, x_{i+1}]} \frac{|v''(x)|}{2} \max_{x \in [x_i, x_{i+1}]} (x - x_i)(x_{i+1} - x) = \frac{(x_{i+1} - x_i)^2}{8} \max_{x \in [x_i, x_{i+1}]} |v''(x)|. \end{aligned}$$

□

Hence from the strong (second) case we get

$$||T_\infty(0)||_\infty \leq \frac{h^2}{8} || \frac{d^2 F(u^h)}{dx^2} ||_\infty. \tag{32}$$

Using (23) we get

$$\|T_\infty(0)\|_\infty \leq \max_{i \in \{1,\dots,N\}} \frac{h_i^2}{16\varepsilon} \left| \frac{d}{dx}(u^h(x)^2) \right|_{x \in [x_{i-1}, x_i]} \tag{33}$$

We now have on each interval $x \in [x_{i-1}, x_i]$

$$\frac{d}{dx} \left(u^h(x)^2 \right) = 2 \left(u_{i-1} + \frac{u_i - u_{i-1}}{h_i}(x - x_{i-1}) \right) \frac{u_i - u_{i-1}}{h_i}$$

Since this is linear in x, the maximum modulus occurs at $x = x_i$ or $x = x_{i-1}$. Considering all intervals gives

$$\|T_\infty(0)\|_\infty = \max_{i \in \{1,\dots,N\}} \frac{h_i}{8\varepsilon} |u_i - u_{i-1}| \max\{|u_{i-1}|, |u_i|\} \tag{34}$$

Using (31) and (34), we obtain a vector

$$\overline{Y} = \begin{pmatrix} |(I - \tilde{B})^{-1} (\overline{\Pi^h F(u^h)} - \overline{u^h})| \\ \max_{i \in \{1,\dots,N\}} \frac{h_i}{8\varepsilon} |u_i - u_{i-1}| \max\{|u_{i-1}|, |u_i|\} \end{pmatrix} \tag{35}$$

that satisfies Assumption 1.

6.4 Candidate Vector for Satisfying Assumption 2

We now derive Z, i.e. we bound $(T'(\tilde{w})w)_i$. From (18) and (19) we have

$$\begin{aligned} T'_h(\tilde{w})w &= (I - A^h)^{-1}(\Pi^h F'(u^h + \tilde{w}^h + \tilde{w}^\infty)(w^h + w^\infty) - A^h w^h), \\ T'_\infty(\tilde{w})w &= (I - \Pi^h)F'(u^h + \tilde{w}^h + \tilde{w}^\infty)(w^h + w^\infty). \end{aligned}$$

For A^h, we have already derived the corresponding matrix $\tilde{B}$. From (26) and from using $A^h w^h = \Pi^h F'(u^h)w^h$, $\{w^h, \tilde{w}^h\} \subset S_2^h$ and $(w)_i, (\tilde{w})_i \leq W_i$ we have

$$\begin{aligned} &\varepsilon\left(\Pi^h F'(u^h + \tilde{w}^h + \tilde{w}^\infty)(w^h + w^\infty) - A^h w^h\right)_i \overset{\text{def}}{=} \varepsilon\left|\Pi^h F'(u^h + \tilde{w}^h + \tilde{w}^\infty)(w^h + w^\infty) - A^h w^h\right|(x_i) \\ &= \left|\int_{-1}^{x_i}\left(u^h w^\infty + (\tilde{w}^h + \tilde{w}^\infty)(w^h + w^\infty)\right)d\tau - \tfrac{1+x_i}{2}\int_{-1}^{1}\int_{-1}^{x}\left(u^h w^\infty + (\tilde{w}^h + \tilde{w}^\infty)(w^h + w^\infty)\right)d\tau dx\right| \\ &\leq \int_{-1}^{x_i}(|\tilde{w}^h| + |\tilde{w}^\infty|)(|w^h| + |w^\infty|)d\tau + \frac{1+x_i}{2}\int_{-1}^{1}\int_{-1}^{x}(|\tilde{w}^h| + |\tilde{w}^\infty|)(|w^h| + |w^\infty|)d\tau dx \\ &\qquad + \frac{1+x_i}{2}\int_{-1}^{1}\int_{-1}^{x}|w^\infty u^h|d\tau dx + \int_{-1}^{x_i}|u^h w^\infty|d\tau \qquad (36) \\ &\leq \sum_{j=1}^{i} D_j + \sum_{j=1}^{i} C_j + (x_i + 1)W_{N+1}^2 + \frac{1+x_i}{2}\Big(\sum_{j=1}^{N} h_j(E_j + \sum_{k=1}^{j-1} D_k) + \sum_{j=1}^{N} h_j(F_j + \sum_{k=1}^{j-1} C_k) + 2W_{N+1}^2\Big) \\ &\qquad + \frac{1+x_i}{2}\sum_{j=1}^{N} h_j(B_j + \sum_{k=1}^{j-1} A_k) + \sum_{j=1}^{i} A_j \\ &= \sum_{j=1}^{i}(A_j + C_j + D_j) + 2(x_i + 1)W_{N+1}^2 + \frac{1+x_i}{2}\sum_{j=1}^{N} h_j(B_j + E_j + F_j + \sum_{k=1}^{j-1} A_k + C_k + D_k) \overset{\text{def}}{=} \varepsilon q_i, \end{aligned}$$

where $i \in \{1, \ldots, N\}$, $W_0 = 0$ and we have introduced

$$\begin{aligned} A_j &\overset{\text{def}}{=} W_{N+1}\int_{x_{j-1}}^{x_j}|u^h(\tau)|d\tau = \begin{cases} \frac{h_j W_{N+1}}{2}\frac{u_{j-1}^2 + u_j^2}{|u_{j-1}| + |u_j|} & u_j u_{j-1} < 0 \\ \frac{h_j W_{N+1}}{2}(|u_{j-1}| + |u_j|) & \text{otherwise} \end{cases}, \\ B_j &\overset{\text{def}}{=} \frac{W_{N+1}}{h_j}\int_{x_{j-1}}^{x_j}\int_{x_{j-1}}^{x}|u^h(\tau)|d\tau dx = \begin{cases} \frac{h_j W_{N+1}}{6}\frac{|u_j|^3 + 3u_{j-1}^2|u_j| + 2|u_{j-1}|^3}{(|u_{j-1}| + |u_j|)^2} & u_j u_{j-1} < 0 \\ \frac{h_j W_{N+1}}{6}(2|u_{j-1}| + |u_j|) & \text{otherwise} \end{cases}, \\ C_j &\overset{\text{def}}{=} h_j W_{N+1}(W_{j-1} + W_j) \geq W_{N+1}\int_{xj-1}^{x_j}(|w^h(\tau)| + |\tilde{w}^h(\tau)|)d\tau, \\ D_j &\overset{\text{def}}{=} \frac{h_j}{3}(W_j^2 + W_j W_{j-1} + W_{j-1}^2) \geq \int_{xj-1}^{x_j}|w^h(\tau)\tilde{w}^h(\tau)|d\tau, \\ E_j &\overset{\text{def}}{=} \frac{h_j}{12}(3W_{j-1}^2 + 2W_j W_{j-1} + W_j^2) \geq \frac{1}{h_j}\int_{xj-1}^{x_j}\int_{xj-1}^{x}|w^h(\tau)\tilde{w}^h(\tau)|d\tau dx \text{ and} \\ F_j &\overset{\text{def}}{=} \frac{h_j W_{N+1}}{3}(2W_{j-1} + W_j) \geq \frac{W_{N+1}}{h_j}\int_{xj-1}^{x_j}\int_{xj-1}^{x}(|w^h(\tau)| + |\tilde{w}^h(\tau)|)d\tau dx. \end{aligned}$$

The last component of Z is again bounded by the use of the interpolation error. However, using (32) as previously requires differentiating (26) twice with respect to x. Since we only have a bound of $\|w^\infty\|_\infty$, such a bound would not be useful. Instead, we use the the weaker (first) case of Proposition 1. From (26), we have

$$
\begin{aligned}
&\|(I-\Pi^h)F'(u^h+\tilde{w})(w)\|_\infty \le \\
&\quad \max_{i\in\{1,\dots,N\}} \frac{h_i}{2}\left|\frac{1}{\varepsilon}(u^h+\tilde{w}^h+\tilde{w}^\infty)(w^h+w^\infty)-\frac{1}{2\varepsilon}\int_{-1}^{1}\int_{-1}^{x}(u^h+\tilde{w}^h+\tilde{w}^\infty)(w^h+w^\infty)d\tau dx\right| \\
&\le \max_{i\in\{1,\dots,N\}} \frac{h_i}{2\varepsilon}\Bigg((W_{N+1}+\max\{W_i,W_{i-1}\})(W_{N+1}+\max\{W_i,W_{i-1}\}+\max\{|u_{i-1}|,|u_i|\})+ \\
&\frac{1}{2}\Big(\sum_{j=1}^{N}h_j(B_j+\sum_{k=1}^{j-1}A_k)+\sum_{j=1}^{N}h_j(E_j+\sum_{k=1}^{j-1}D_k)+\sum_{j=1}^{N}h_j(F_j+\sum_{k=1}^{j-1}C_k)+2W_{N+1}^2+\sum_{j=1}^{N}h_j(H_j+\sum_{k=1}^{j-1}G_k)\Big)\Bigg) \\
&= \max_{i\in\{1,\dots,N\}} \frac{h_i}{2\varepsilon}\Bigg((W_{N+1}+\max\{W_i,W_{i-1}\})(W_{N+1}+\max\{W_i,W_{i-1}\}+\max\{|u_{i-1}|,|u_i|\})+ \\
&\qquad W_{N+1}^2+\frac{1}{2}\sum_{j=1}^{N}h_j(B_j+E_j+F_j+H_j+\sum_{k=1}^{j-1}A_k+C_k+D_k+G_k)\Bigg) \overset{\text{def}}{=} Z_{N+1},
\end{aligned}
$$

where

$$
\begin{aligned}
G_j &\overset{\text{def}}{=} \frac{h_j}{6}(2W_{j-1}|u_{j-1}|+W_j|u_{j-1}|+W_{j-1}|u_j|+2W_j|u_j|) \ge \int_{x_{j-1}}^{x_j}|u^h(\tau)w^h(\tau)|d\tau \text{ and} \\
H_j &\overset{\text{def}}{=} \frac{h_j}{12}(3W_{j-1}|u_{j-1}|+W_j|u_{j-1}|+W_{j-1}|u_j|+W_j|u_j|) \ge \frac{1}{h_j}\int_{x_{j-1}}^{x_j}\int_{x_{j-1}}^{x}|u^h(\tau)w^h(\tau)|d\tau dx.
\end{aligned}
$$

Hence, we obtain a vector

$$
\overline{Z}(\overline{W}) = \begin{pmatrix} |(I-\tilde{B})^{-1}|\overline{q} \\ Z_{N+1} \end{pmatrix},
$$

that satisfies Assumption 2.

7 Algorithm

The algorithm consists of several steps. First we get an approximate solution u^h of (14) using some boundary value solver. Then we can compute $\overline{Y}$ using (35). This part we need to fulfil Assumption 1. Eventually we fulfil Assumption 2 with the algorithm part

```
$k \Leftarrow 0$
$W_i^{(k)} \Leftarrow Z_i(\overline{0})$, $i \in \{1,\dots,N+1\}$
while $Y_i + Z_i(\overline{W}^{(k)}) \ge W_i^{(k)}$ for some $i \in \{1,2,\dots,N+1\}$
    $k \Leftarrow k+1$
    $W_i^{(k)} \Leftarrow (1+\delta)(Y_i + Z_i(\overline{W}^{(k-1)}))$, $i \in \{1,2,\dots,N+1\}$
end.
```

If the while–loop terminates, existence and local uniqueness are proved. The main function for the whole algorithm is called `adspline`. It starts by calling the MATLAB boundary value problem solver `bvp4c`, which computes an approximate solution u^h as our approximate candidate. (Note that in order to use the solver `bvp4c`, the equation must be written as a first order system.) Next, the values of u^h over the user–defined mesh, are computed using splines. Following this, the program enters an outer loop over mesh modifications, computes Y for the mesh, and enters an inner loop where Z and W are repeatedly computed. If the convergence condition $K \subset W$ is satisfied, the iterations stop. If not, W is increased by a factor $1+\delta$, which is typically about 1.01. If convergence is not reached within the loop, one or two points are added to the mesh where W is maximal, u^h and Y are computed again, and the inner loop starts again. If successful, the program outputs the maximal values of K and W on the final mesh, as well as their infinite parts are displayed; otherwise an error message appears. All interval computations are performed using the free MATLAB package INTLAB [In].

8 Result

The solutions of (14) for different values of ε are shown in Figure 1. We see that for large values of ε, the solutions are almost linear. When ε approaches zero, the viscous shock becomes more pronounced, and also increasingly hard to compute.

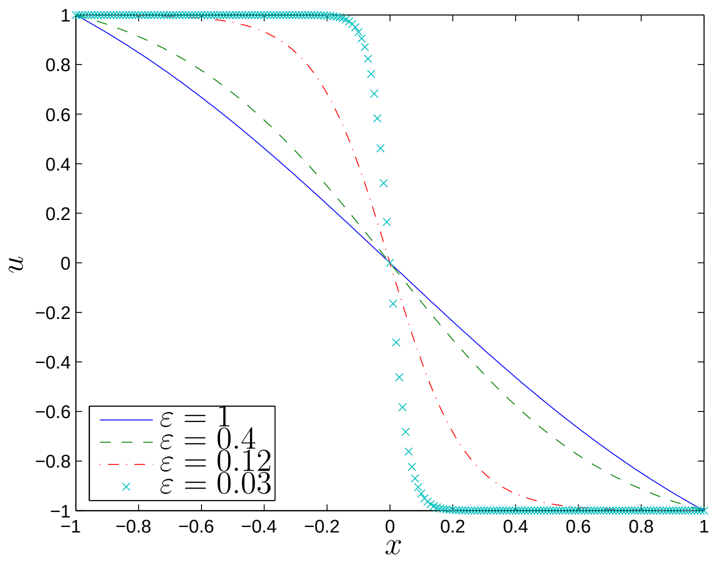

Figure 1: The solutions of (14) for different values of ε.

If, for a given resolution h, ε is too small, the algorithm described in the previous section does not converge – instead W and K grow without bounds. On the other hand, if we choose h too small, our computations will consume too much time or memory. It is therefore interesting to study how small ε can be chosen, given a fixed resolution h. To establish where the limit between convergence and divergence lies, we first choose some values of h. For every such value we run the algorithm for varying ε. If we get convergence, we decrease ε, and restart the algorithm. If we do not get convergence, we try a larger value and so on, until the difference between the smallest value of ε for which we have have convergence and the largest value of ε for which we have divergence is sufficiently small. For example, when $h = 0.02$, we get convergence for $\varepsilon = 0.295$, but not for $\varepsilon = 0.285$. Hence, rounded off to two decimals, $\varepsilon = 0.29$. When computing u^h, we use a uniform starting mesh with $h = 0.001$.

The result is shown in Table 1 and in Figure 2, which is a plot of $\log(h)$ as a function of $\log(\varepsilon)$. The estimated least square line slope is approximately 2.20.

Remark 1. *The smallest value of ε for which we manage to establish convergence is* **0.085**. *In this case $h = 0.001$ is used.*

ε	0.08	0.11	0.15	0.22	0.29	0.38	0.58	0.85	1.33
h	0.001	0.002	0.004	0.01	0.02	0.04	0.1	0.2	0.4

Table 1: The minimum value of ε for which the algorithm converges for given values of h.

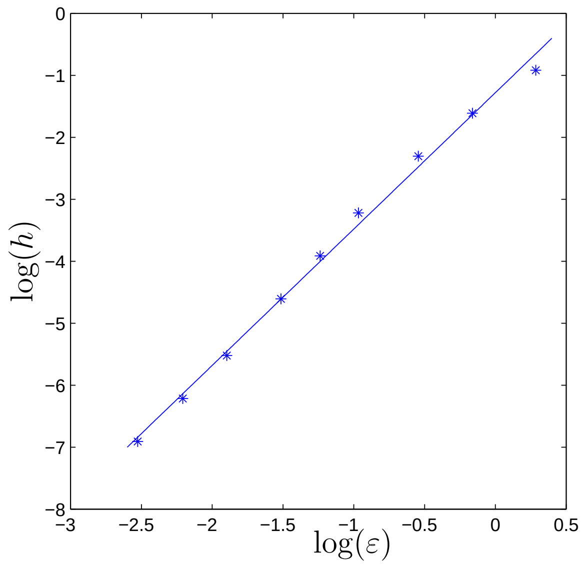

Figure 2: The logarithm of the minimum value of ε for which the algorithm converges for given values of the logarithm of h. A least square straight line estimation of $\log(\varepsilon)$ versus $\log(h)$ is also shown.

We have also computed the infinite matrix norm of the inverse of $I - \tilde{B}$ for a uniform mesh with $h = 0.01$, and with $\varepsilon \in \{0.01, 0.02, 0.03, \ldots, 0.10\}$. It is interesting to see that $I - \tilde{B}$ approaches a singular matrix when ε approaches zero. The result is displayed in Table 2, and the logarithmic version is displayed in Figure 3. The least square straight line

ε	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
$\lVert(I - \tilde{B})^{-1}\rVert_\infty$	49.68	24.96	16.65	12.52	10.08	8.43	7.27	6.40	5.74	5.21

Table 2: The infinite matrix norm of the inverse of $I - \tilde{B}$ for different values of ε.

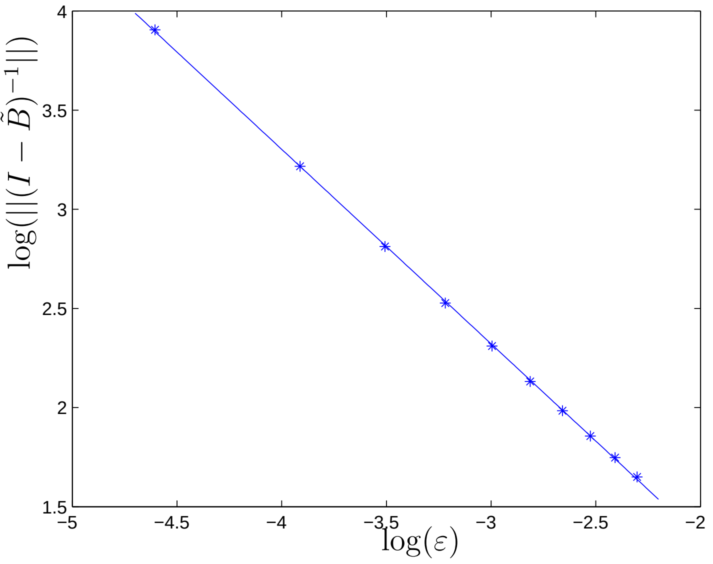

Figure 3: The logarithm of the infinite matrix norm of the inverse of $I - \tilde{B}$ for different values of the logarithm of ε and the least square straight line estimation.

h	$\max W$	W_∞
0.04	3.27×10^{-3}	4.56×10^{-4}
0.02	6.21×10^{-4}	8.51×10^{-5}
0.01	1.38×10^{-4}	1.87×10^{-5}
0.005	3.29×10^{-5}	4.43×10^{-6}
0.004	2.09×10^{-5}	2.81×10^{-6}
0.002	5.12×10^{-6}	6.86×10^{-7}
0.001	1.26×10^{-6}	1.68×10^{-7}

Table 3: The maximal W component, and W_∞ for different values of the resolution h. $\varepsilon = 0.5$.

approximation for the logarithms is also computed. The estimated least square straight line slope is approximately -0.98.

Finally, we have computed the maximal W-components as well as W_∞ for $\varepsilon = 0.5$ and uniform meshes with different resolutions. The logarithms of the values are shown in Table 3 and in Figure 4 together with the least square approximation lines. The slopes of the lines are 2.12 and 2.13 respectively, illustrating that the maximal W-component and W_∞ are almost proportional to each other.

9 Discussion

Since $||\tilde{B}||_2 << 1 = ||I||_2$ when we have convergence, and q_i (36) is non–decreasing in i, Z_i and therefore also W_i are typically also non–decreasing in i. Hence the mesh is updated with

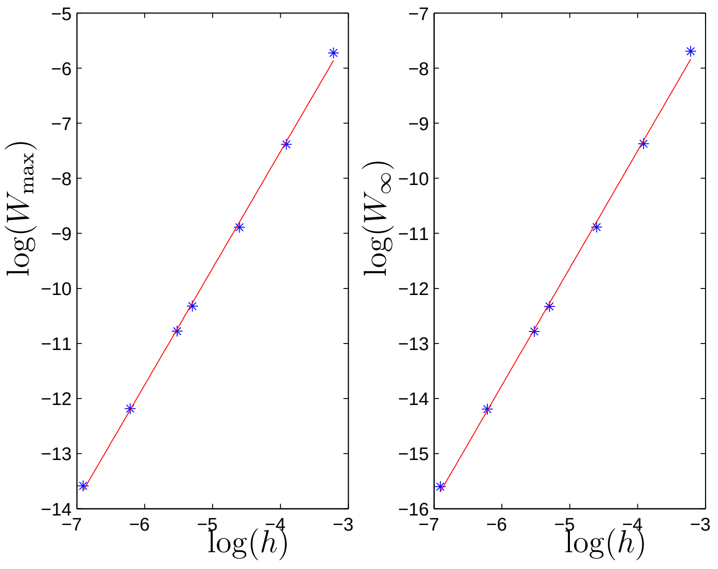

Figure 4: The logarithm of the resolution h versus the logarithm of the maximal W component and W_∞. The least square straight line estimation for the logarithms are also shown. Here $\varepsilon = 0.5$.

more points on the right boundary, where the error estimate is maximal. But this does not necessarily mean that the error $w = u - u^h$ is maximal there. It is simply due to the lack of cancellation in the inequalities for computing q_i and Z_{N+1}.

There are several ways to improve the algorithm. The function spaces do not necessarily have to be spanned by hat functions; perhaps other basis functions are more appropriate. This change would probably increase the complexity of each step of the calculations, but it should also increase speed and accuracy. The bottleneck of our computations is the inversion of the matrix $I - \tilde{B}$. According to (30), this matrix is a sum of a lower triangular matrix and an outer product. Utilizing the Sherman–Morrison formula [Pr02], we may thus increase the speed of the inversion.

In future work we will study time–dependent problems or systems of equations. Such problems are more interesting since they can describe more complicated physical models, and they may not be analytically solvable. As an example one could consider the time–dependent version of this problem with the method described in [Zg01].

References

[BM98] Berz, M., Makino, K., *Verified Integration of ODEs and Flows using Differential Algebraic Methods on High–Order Taylor Models*, Reliable Computing **4**, 361–369, 1998.

[Br00] Bressan, A. *Hyperbolic Systems of Conservation Laws.* Oxford University Press, 2000.

[BS94] Brenner, S. and Scott, R. *The Mathematical Theory of Finite Element Methods.* Springer–Verlag, 1994.

[Bu74] Burgers, J.M., *The Nonlinear Diffusion Equation: Asymptotic Solutions and Statistical Problems.* Reidel, 1974.

[Fa95] Farzaneh, R. *A Computer Generated Proof for the Existence of Periodic Orbits for Three–Dimensional Vector Fields.* Ph.D. Thesis, Cornell, 1995.

[Gl65] Glimm, J., *Solutions in the large for nonlinear hyperbolic systems of equations.* Comm. Pure Appl. Math, **18**, 679–715, 1965.

[HR02] Holden, H. and Risebro, N. H. *Front Tracking for Hyperbolic Conservation Laws.* Springer–Verlag, 2002.

[In] INTLAB – INTerval LABoratory Version 5.3.
Available from `http://www.ti3.tu-harburg.de/rump/intlab/`.

[KL89] Kreiss, H–O. and Lorenz, J. *Initial–boundary value problems and the Navier–Stokes equations.* Pure and applied mathematics, vol. 136, Academic Press, 1989.

[MZ01] Mischaikow, K. and Zgliczyński, P. *Rigorous Numerics for Partial Differential Equations: the Kuramoto–Sivashinsky Equation.* Found. Comput. Math. **1** 255–288, 2001

[Na92] Nakao, M. T. *A numerical verification method for the existence of weak solutions for nonlinear boundary value problems.* J. Math. Anal. and Appl. **164**, 489–507, 1992.

[Pl01] Plum, M. *Computer–assisted enclosure methods for elliptic differential equations.* Linear Algebra and its Applications **324**, 147–187, 2001.

[Pr02] Press, W.H., et. al. *Numerical Recipes in C++: The Art of Scientific Computing.* Second Edition, Cambridge University Press, 2002.

[Si04] Siklosi, M. and Åsén, P.O., *On a Computer–Assisted Method for Proving Existence of Solutions of Boundary Value Problems.* KTH–Numerical Analysis and Computer Science, October, 2004.

[Tu02] Tucker, W. *A Rigorous ODE Solver and Smale's 14th Problem.* Foundations of Computational Mathematics, 2:1, 53–117, 2002.

[Ya98] Yamamoto, N., *A Numerical Verification Nethod for Solutions of Boundary Value Problems with Local Uniqueness by Banach's Fixed Point Theorem.* SIAM Journal on Numerical Analysis, 35(5), 2004–2013, 1998.

[Zg01] Mischaikow, K. and Zgliczyński, P. *Rigorous Numerics for Partial Differential Equations: the Kuramoto–Sivashinsky Equation.* Found. Comput. Math. **1** 255–288, 2001.

[Zg02] Zgliczynski, P. *Attracting Fixed Points for the Kuramoto–Sivashinsky Equation: A Computer Assisted Proof* SIAM Journal on Applied Dynamical Systems, Volume 1, Number 2, 215–235, 2002.

[Ås04] Åsen, P–O. *A proof of a Resolvent Estimate for Plane Couette flow by new analytical and Numerical Techniques.* TRITA–NA–0427, KTH, 2004.

10 Appendix

Here we list all MATLAB code used in implementing the algoritm. The tilde sign means that a row is terminated in the listing, but not in the corresponding M–file.

adspline.m

```
function [K,W]=adspline(eps,xguess,maxmesh,tol,x,delta,iter1,iter2)
% xguess is the starting mesh for MATLAB's boundary value solver bvp4c.
% maxmesh is the maximal number of mesh points.
% tol is the maximal relative tolerance in the bvp4c process.
% x is the starting mesh for the main algorithm.
% delta is an algorithm parameter describing how much the search domain should
% be increased in every step. Usually it is set to 0.01.
% iter1 is the number of mesh refinements.
% iter2 is the number of iterations per refinement.
format long
global eps1
eps1=eps;
eps1int=intval(eps1);
delta=intval(delta);
convergence=0;
s=size(x);
if s(1)<s(2)
  x=x';
end
s=size(xguess);
if s(1)<s(2)
  xguess=xguess';
end
l=0; % Loop counter.
for k=1:iter1 % Loops over the number of partition refinements.
  disp('Solving the boundary value differential equation...')
  options=bvpset('FJacobian',@jacob,'Nmax',maxmesh,'Stats','on','RelTol',tol);
  % The solver will compute with a jacobian, which is defined in jacob.m.
  solinit=bvpinit(xguess,[1;-1]); % Starting guess u=-x.
  sol=bvp4c(@odefun,@bcfun,solinit,options); % The ODE is written as a first order system.
  u=sol.y(1,:); % The system has two unknowns: u and u'. Only the first is needed.
  error=max(abs(u-exactu(sol.x,eps1)))
  disp('Computing splines...')
  u=spline(sol.x,u,x); % The value of u in the points x are computed using splines.
  uint=intval(u(2:end));
  N=length(x)
  disp('Computing Y...')
  x=intval(x);
  Y=makeYint(eps1int,x,uint); % For Assumption 1.
  Z=intval(zeros(N,1));
  for m=1:iter2 % Iterations over some fix resolution.
    l=l+1;
    disp(['Iteration' int2str(l) '...'])
    W=(1+delta)*(Y+Z); % W is increased.
    Z=Zh(eps1int,W,uint,x);
    mean_of_W=mean(W.sup)
    if ~sum((Y.sup+Z.sup)>W.inf) % Is Assumption 2 fulfilled?
      convergence=1;
      break
    end
  end
  if(convergence)
    break
  end
  disp('Updating the partition...')
  [M,j]=max(W(1:end-1).inf);
  % Checks where W is maximal. The infinite part i.e. the last part is not considered.
  Maximal_point=x(j+1)
  if j==length(W)-1 %Maximun on the boundary. Only one point is added.
    x=[x(1:j);(x(j)+x(j+1))/2;x(j+1)];
  else
    x=[x(1:j);(x(j)+x(j+1))/2;x(j+1);(x(j+2)+x(j+1))/2;x(j+2:end)];
    % Doubles the resolution around this maximum.
  end
  plot(x.mid,zeros(length(x),1),'.') % Plots the partition.
  hold on
  plot(x.mid(j+1),0,'r*') % Plots a red star where the maximum is.
  hold off
  x=mid(x);
  drawnow
end
if(convergence)
  K=Y.sup+Z.sup;
  W=W.sup;
  K=[max(K(1:end-1));K(end)];
  W=[max(W(1:end-1));W(end)];
else
  error('No convergence')
end
```

jacob.m

```
function dydx=jacob(x,y)
% The equation eps1*u_xx=(u^2/2)_x is rewritten as (u_x,v_x)=(v,uv/eps1).
% Therefore the jacobian is [0 1;v/eps1 u/eps1].
global eps1
dydx=[0 1;y(2)/eps1 y(1)/eps1];
```

odefun.m

```
function dydx=odefun(x,y)
% The equation eps1*u_xx=(u^2/2)_x is written as a first order system.
global eps1
dydx=[y(2); y(1)*y(2)/eps1];
```

bcfun.m

```
function z=bcfun(ya,yb)
% The equivalent boundary conditions u(-1)-1=0 and u(1)+1=0 are set.
  z=[ya(1)-1
   yb(1)+1];
```

exactu.m

```
function y=exactu(x,eps1)
% Returns the "almost" exact solution y=-tanh(rx)/tanh(r).
% where r can only be computed numerically.
if e>0.5
r=fzero(@(r) r*tanh(r)-1/2/eps1,1/sqrt(2*eps1)); % For large epsilons rtanh(r) is small
else % so rtanhr\approx r^2. Use 1/sqrt(2*eps1) as a starting guess.
  r=fzero(@(r) r*tanh(r)-1/2/eps1,1/(2*eps1));
% For small epsilons rtanh(r) is large
end % so rtanhr\approx r. Use 1/(2*eps1) as a starting guess.
y=-tanh(r*x)/tanh(r);
```

makeYint.m

```
function Yint=makeYint(eps1,xint,uint)
% Computes Y for Assumption 1.
n=length(uint);
uint=[1;uint]; % The left boundary is added.
h=xint(2:end)-xint(1:end-1);
d=abs(uint(2:end)-uint(1:end-1)); % Needed for the maximal derivative.
m1=abs(uint(1:end-1))';
m2=abs(uint(2:end))';
m=max([m1.sup;m2.sup]);
Yinf=intval(max(h.sup.*d.sup.*m'))/8/eps1 % The infinite dimensional part.
uint=uint(2:end); % The left boundary is removed.
Yint=abs((speye(n)-makeBint(eps1,xint,uint))\(Fintint(eps1,xint,uint)-uint));
Yint=[Yint;Yinf];
```

makeBint.m

```
function Bint=makeBint(eps1,xint,uint)
% Computes the matrix \tilde{B}.
n=length(uint);
uint=[1;uint]; % The left boundary is added.
hint=xint(2:end)-xint(1:end-1);
Bint=spdiags(hint.*(uint(2:end)/3+uint(1:end-1)/6),0,n,n); % Diagonal elements.
for j=1:n-1
  Bint(j+1:n,j)=hint(j)*(uint(j)/6+uint(j+1)/3)+hint(j+1)*(uint(j+2)/6+uint(j+1)/3);~
% Below the diagonal.
end
d=hint.^2.*(uint(1:end-1)+uint(2:end))/12; % Double integral contribution.
d(1:n-1)=d(1:n-1)+hint(2:end).^2.*(uint(3:end)/12+uint(2:end-1)/4)+hint(2:end).*~
hint(1:end-1).*(uint(1:end-2)/6+uint(2:end-1)/3)+(1-xint(3:end)).*(hint(1:end-1).*~
(uint(1:end-2)/6+uint(2:end-1)/3)+hint(2:end).*(uint(3:end)/6+uint(2:end-1)/3));
Bint=(Bint-(1+xint(2:end))*d'/2)/eps1;
```

Fintint.m

```
function PiF=Fintint(eps1,xint,uint)
% Computes F(u^h)(x_i)
n=length(uint);
uint=[1;uint];
hint=xint(2:end)-xint(1:end-1);
s(1)=hint(1)*(uint(1)^2+uint(1)*uint(2)+uint(2)^2);
for k=2:n
```

```
  s(k)=s(k-1)+hint(k)*(uint(k)^2+uint(k)*uint(k+1)+uint(k+1)^2);
end
s=s'; % The single integral contribution.
t=hint.*(uint(2:end).^2+2*uint(2:end).*uint(1:end-1)+3*uint(1:end-1).^2)/4;~
% The double integral contribution.
PiF=-xint(2:end)+(2*s-(1+xint(2:end))*sum(hint.*(t+[0;s(1:end-1)])))/eps1/12
```

Zh.m

```
function Zint=Zh(eps1,W,uint,xint)
% Computes Z for Assumption 2.
N=length(uint);
hint=xint(2:end)-xint(1:end-1);
b=[1;uint];
a=abs(b);
W=[0;W]; %The error vanishes on the left boundary.

% The first N components
%-------------------------------------------------------
% Simple integral contribution.
r=b(1)*b(2);
if r.sup<0
AC(1)=hint(1)*((a(1)^2+a(2)^2)/(a(1)+a(2))/2+W(1)+W(2));
% AC=(The cumulative sum of the sum of A and C)/W(end)
else
  AC(1)=hint(1)*((a(1)+a(2))/2+W(1)+W(2));
end
  D(1)=hint(1)*(W(1)^2+W(1)*W(2)+W(2)^2)/3;
for j=2:N
  r=b(j)*b(j+1);
  if r.sup<0
    AC(j)=AC(j-1)+hint(j)*((a(j)^2+a(j+1)^2)/(a(j)+a(j+1))/2+W(j)+W(j+1));
  else
    AC(j)=AC(j-1)+hint(j)*((a(j)+a(j+1))/2+W(j)+W(j+1));
  end
  D(j)=D(j-1)+hint(j)*(W(j)^2+W(j)*W(j+1)+W(j+1)^2)/3;
end
  ACD=W(end)*AC+D; % ACD=The cumulative sum of the sum of A, C and D.

% Double integral contribution.
r=b(1)*b(2);
if r.sup<0
  s=hint(1)^2*(W(end)*((a(2)^3+3*a(2)*a(1)^2+2*a(1)^3)/(a(1)+a(2))^2/6+(2*W(1)+W(2))/3)+~
(3*W(1)^2+2*W(1)*W(2)+W(2)^2)/12);
 else
   s=hint(1)^2*(W(end)*((2*a(1)+a(2))/6+(2*W(1)+W(2))/3)+(3*W(1)^2+2*W(1)*W(2)+~
W(2)^2)/12);
end
for j=2:N
  r=b(j)*b(j+1);
  if r.sup<0
    s=s+hint(j)*((W(end)*((a(j+1)^3+3*a(j+1)*a(j)^2+2*a(j)^3)/(a(j)+a(j+1))^2/6+~
(2*W(j)+W(j+1))/3)+(3*W(j)^2+2*W(j)*W(j+1)+W(j+1)^2)/12)*hint(j)+ACD(j-1));~
% ACD is the double sum contribution.
  else
    s=s+hint(j)*((W(end)*((2*a(j)+a(j+1))/6+(2*W(j)+W(j+1))/3)+(3*W(j)^2+2*W(j)*W(j+1)~
+W(j+1)^2)/12)*hint(j)+ACD(j-1)); % ACD is the double sum contribution.
  end
end
q=ACD'+(xint(2:end)+1)*(2*W(end)^2+s/2);
Zshort=abs(inv(speye(N)-makeBint(eps1,xint,uint)))*q;
%-------------------------------------------------------
% Last component of Z
G(1)=hint(1)*(2*(W(1)*a(1)+W(2)*a(2))+W(1)*a(2)+W(2)*a(1));
for j=2:N
  G(j)=G(j-1)+hint(j)*(2*(W(j)*a(j)+W(j+1)*a(j+1))+W(j)*a(j+1)+W(j+1)*a(j));
end
G=G/6;
s=s+hint(1)^2*(3*W(1)*a(1)+W(1)*a(2)+W(2)*a(1)+W(2)*a(2))/12;
% Using that s is already summed over A, B, C, D, E and F.
for j=2:N
  s=s+hint(j)*((3*W(j)*a(j)+W(j)*a(j+1)+W(j+1)*a(j)+W(j+1)*a(j+1))*hint(j)/12+G(j-1));~
% G is the double sum contribution.
end
for j=1:N
  M(j)=hint(j)*((W(end)+max(W(j).sup,W(j+1).sup))*(W(end)+max(W(j).sup,W(j+1).sup)~
+max(a(j).sup,a(j+1).sup))+W(end)^2+s/2);
end
Zinf=intval(max(M.sup))/2

Zint=[Zshort;Zinf]/eps1;
```