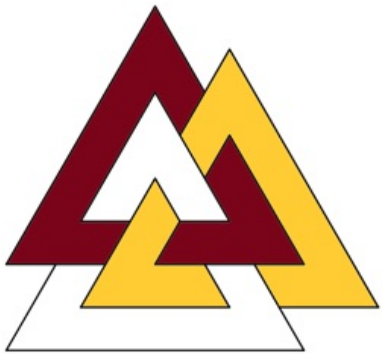
The Minnesota Journal of Undergraduate Mathematics

The automated generation of time evolution code for conservation and reaction-diffusion laws

Jalen Morgan and Ray David Todd

Department of Mathematics, Brigham Young University



The Minnesota Journal of Undergraduate Mathematics

Volume 4 (2018-2019 Academic Year)

Sponsored by
School of Mathematics
University of Minnesota
Minneapolis, MN 55455

The Minnesota Journal of Undergraduate Mathematics

Volume 4 (2018-2019 Academic Year)

The automated generation of time evolution code for conservation and reaction-diffusion laws

Jalen Morgan and Ray David Todd*

Department of Mathematics, Brigham Young University

Abstract. Time evolution of initial data in Partial Differential Equations (PDEs) plays an important role in understanding physical phenomena, and is of particular interest in determining the long term dynamics of perturbed unstable waves. In this paper, we describe the Python package we have developed for carrying out time evolution studies. This package allows the user to input a conservation law or reaction-diffusion equation in one spatial dimension in system form. The Python program then generates the MATLAB driver and system specific files to be used in carrying out time evolution using the Crank-Nicolson scheme. We demonstrate the performance of this package in three example systems: Burgers' equation, nonisentropic Navier-Stokes, and reactive Navier-Stokes (rNS). The rNS study suggests the way in which instability of traveling waves is manifested in that system.

1. Introduction

Partial Differential Equations (PDEs) are used to model a variety of occurrences in nature, such as roll waves in inclined water flow [1] or shock waves in a combustion process [10]. When these models are accurate, they allow practitioners to gain valuable insight into a mathematical system before conducting physical experiments. Consequently, determining how well a model captures natural behavior is important in model verification. When considering equations and systems of equations, not all give rise to traveling wave solutions. For those that do, however, it is particularly important that the stability properties of a model's traveling wave solution are similar to those exhibited physically. Indeed, stable, or at least metastable, waves should exist in models that correspond to physically observed waves. In this work, we present a tool we have developed to aid in the understanding and study of partial differential equations. This tool automatically generates the time evolution code for conservation and reaction-diffusion laws in one spatial dimension. Specifically, we consider conservation or reaction-diffusion laws taking the modified flux form

$$f^0(\vec{u})_t + f^1(\vec{u})_x + G(\vec{u}) = (B(\vec{u})\,\vec{u}_x)_x, \tag{1}$$

* *Corresponding author*

where $\vec{u}$ is a vector of n system variables, and f^0, f^1, G, and B are functions of the same system variables with f^0,f^1,$G : \mathbb{R}^n \to \mathbb{R}^n$, and $B : \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}^n$; see [3] for details. It is not necessary that the functions be linear with respect to our system variables. Our software only requires a user to input the equations f^0, f^1, G, B, and the boundary conditions, then everything else needed to carryout time evolution code is automatically generated. Specifically, we use the Crank-Nicolson finite difference scheme for the time evolution because it is well-known for balancing accuracy and speed [11]. We implement code in Python, using the SymPy package, which takes as input the functions f^0, f^1, G and B and writes the MATLAB files needed to carry out time evolution. In sections 3, 4, and 5, we describe our code for time evolution of Burger's Equation, nonisentropic Navier-Stokes, and reactive Navier-Stokes (rNS) respectively. The latter two examples demonstrate that our code supports systems of PDEs, including reaction-diffusion equations.

2. The code

Our Python program takes minimal input about a mathematical system and writes MATLAB code for carrying out time evolution studies using the Crank-Nicolson finite difference scheme. To use our program, the user must first represent the system parameters and variables as SymPy symbols. The user then creates arrays f^0, f^1, G and B corresponding to the same named functions in (1). The user must next provide a path for the generated MATLAB files to be saved, and at this point the user can run our program by calling the function, *create_code*.

Our program then proceeds to generate the finite difference code by first distributing the derivatives of (1), so that the Crank-Nicolson finite differences can be easily substituted into the equation, which yields

$$Df^0(\vec{u})\vec{u}_t + Df^1(\vec{u})\vec{u}_x + G(\vec{u}) - \left(\frac{\partial}{\partial x}B(\vec{u})\right)\vec{u}_x - B(\vec{u})\vec{u}_{xx} = 0, \tag{2}$$

where Df is the Jacobian of f with respect to $\vec{u}$. Our Python program creates code for $Df^0(\vec{u})$, $Df^1(\vec{u})$, and $\frac{\partial}{\partial x}B(\vec{u})$ by using SymPy's *diff* function. Specifically, our program expands the ith, jth entry of $\frac{\partial}{\partial x}B(\vec{u})$ as $\sum_{k=0}^{n} \frac{\partial B_{i,j}(\vec{u})}{\partial \vec{u}_k}\frac{\partial \vec{u}_k}{\partial x}$. It then performs matrix-vector multiplication between the terms $Df^0(\vec{u})$, $Df^1(\vec{u})$, $\frac{\partial}{\partial x}B(\vec{u})$, $B(\vec{u})$ and $\vec{u}_t$, $\vec{u}_x$, $\vec{u}_x$, $\vec{u}_{xx}$ of (2) respectively. This results in n SymPy equations. Next, it substitutes in for $\vec{u}_t$, $\vec{u}_x$, and $\vec{u}_{xx}$ the finite differences corresponding to the Crank-Nicolson scheme.

Because we wish to allow the user to change the values of Δx and Δt without having to again generate the MATLAB code, our program leaves them as parameters. Next, our Python program converts the n SymPy equations to a string and writes it as a MATLAB function, *fd_F*, to the file path specified by the user. The inputs for *fd_F* are arrays corresponding to the system variables (rows) evaluated at the nodes (columns) at the future and current time steps, Δx, and Δt, and a MATLAB structure p that holds system parameters. Because our MATLAB program uses Newton's method to solve the nonlinear finite difference equations, we need the Jacobian of *fd_F* with respect to the unknown variables, that is the system variables evaluated at the grid points corresponding to the forward

time step. Again using the *diff* function in SymPy, our Python program calculates the Jacobian of our symbolic function *fd_F*, converts it to a string, and writes it to the file path, specified previously by the user, as a MATLAB function named *fd_jac*. The function *fd_jac* takes the same inputs as *fd_F*.

At this point, the user can write a driver function, specify system parameters, input an initial function, define boundary conditions, assign values to Δx and Δt, and then evolve the PDE forward in time by passing these parameters to our MATLAB function *finite_diff_advance*.

The function *finite_diff_advance* forms a function F by attaching the generated *fd_F* function and the user specified boundary conditions. It forms the Jacobian of F by putting together the *fd_jac* function and the user specified boundary condition derivatives. Then, by using the multivariable Newton method, it solves the finite difference equations to advance the solution forward in time.

The source code is available on GitHub under the repository `github.com/finitediff/finite_difference_matlab`.

3. Example: Viscous Burgers' Equation

Burgers' equation is given by

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

where u represents a conserved quantity. We look for a traveling wave solution of the form $u(\xi)$ where $\xi = x - ct$. We then have an ODE of the form,

$$-cu' + uu' = u''. \tag{3}$$

Taking $u_- := \lim_{x\to-\infty} u(x) = 2$, $u_+ := \lim_{x\to+\infty} u(x) = 0$, yields $c = 1$ due the well known Rankine-Hugoniot condition for conservation laws. In the modified flux form, Burgers' equation is thus written

$$\vec{u} = \begin{pmatrix} u \end{pmatrix}, \quad f^0(\vec{u}) = \begin{pmatrix} u \end{pmatrix}, \quad f^1(\vec{u}) = \begin{pmatrix} \frac{u^2}{2} - u \end{pmatrix}, \quad G(\vec{u}) = \begin{pmatrix} 0 \end{pmatrix}, \quad B(\vec{u}) = \begin{pmatrix} u \end{pmatrix}.$$

An explicit traveling wave solution to (3) is given by $u_0(x) := 1 - \tanh(\frac{x}{2})$. We display snapshots of the evolution of the perturbation $v(x) = u_0(x) + \sin(x)\exp(-x^2/5)$ in Figure 1. To create the figure, we use $x \in [-30, 30]$, $\Delta t = 0.1$, and $\Delta x = 0.1$, and Dirichlet boundary conditions $u(-30) = 2$, $u(30) = 0$. In addition, we perform a convergence study keeping $\frac{\Delta t}{(\Delta x)^2}$ constant. To measure the rate of convergence of the finite difference scheme, we take $\Delta t_0 := 0.1$, and $\Delta x_0 := 0.1$. We then evolve the PDE until time $T = 12$ for $\Delta t = \epsilon^2 \Delta t_0$ and $\Delta x = \epsilon \Delta x_0$ with $\epsilon \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}\}$. We consider the solution corresponding to $\epsilon = \frac{1}{8}$ to be the true solution, name it u_*, and measure the convergence of solutions to u_* in the infinity norm as ϵ decreases. We report the trial statistics in Table 1. We note that the convergence error reduces by a little more than $1/4$ each time we half the size of ϵ, which is consistent with the second order accuracy one would expect when using the Crank-Nicolson finite difference scheme for a parabolic system; see [11].

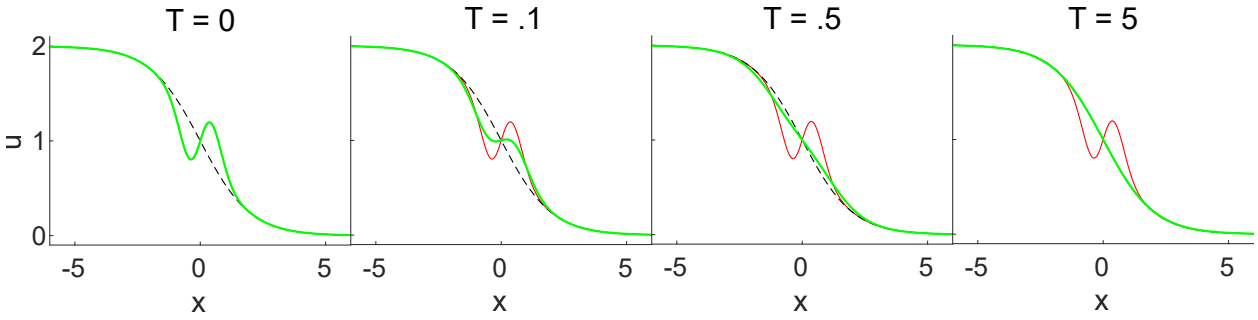


FIGURE 1. Plot of the time evolution of the perturbation $v(x) = u_0(x) + \sin(x)\exp(-x^2/5)$ in Burgers' equation. A thin red curve indicates the initial perturbation v, a black dotted line corresponds to the traveling wave solution u_0, and a thick green line indicates the evolution of v at time $T = 0$, $T = 0.1$, $T = .5$, $T = 5$. The stability of this wave is manifest as the evolution of the perturbed solution approaches the traveling wave.

The Python code specific to this system is available on GitHub at `finite_difference_matlab` in the examples folder under Viscous Burgers. As a concrete example of the code our Python program generates, we note that the finite difference equations at interior nodes take the following form for Burgers' equation using the Crank-Nicolson scheme,

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + u_j^{n+1}\left(\frac{u_{j+1}^{n+1} - u_{j-1}^{n+1} + u_{j+1}^n - u_{j-1}^n}{4\Delta x}\right) - \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1} + u_{j+1}^n - 2u_j^n + u_{j-1}^n}{2(\Delta x)^2} = 0.$$

Each term with a superscript n is known and each term with a superscript $n+1$ is an unknown variable when solving the finite difference equations. Because the equations are clearly nonlinear, we use a multidimensional Newton solver. In this example and the ones that follow, we consider the Newton solver to have converged when the infinity norm between successive iterations in the solver varies no more than 1e-8.

ϵ	1	1/2	1/4
$\|u - u_*\|_\infty$	5.17e-4	1.23e-4	2.46e-5

TABLE 1. Table displaying for Burgers' equation, the convergence of the solution u of the finite difference scheme to u_* as step size decreases, $\Delta t = \epsilon^2/10$, $\Delta x = \epsilon/10$.

4. Example: Nonisentropic Navier-Stokes

The one-dimensional compressible Navier-Stokes equations in Eulerian coordinates are given by

$$\begin{aligned} \rho_t + (\rho u)_x &= 0, \\ (\rho u)_t + (\rho u^2 + p)_x &= (2\mu + \eta)u_{xx}, \\ (\rho(e + u^2/2))_t + (\rho u(e + u^2/2) + up)_x &= (2\mu + \eta)uu_x + \kappa e_{xx}, \end{aligned} \tag{4}$$

where ρ represents density, u the fluid velocity, and e the specific internal energy of the system. The coefficient ν corresponds to the dynamic viscosity, μ to the second viscosity, and κ to the heat conductivity of the system. The coefficient Γ relates the pressure of the system in terms of the density and specific internal energy according to the pressure law given by $p = \Gamma\rho e$. See [8] for details. Due to Galilean invariance, we may take the wave speed to be zero. In modified flux coordinates, equation (4) is

$$\vec{u} = \begin{pmatrix} \rho \\ u \\ e \end{pmatrix}, \quad f^0(\vec{u}) = \begin{pmatrix} \rho \\ \rho u \\ \rho(e + u^2/2) \end{pmatrix}, \quad f^1(\vec{u}) = \begin{pmatrix} \rho u \\ \rho u^2 + \Gamma\rho e \\ \rho u(e + u^2/2) + \Gamma\rho u e \end{pmatrix},$$
$$G(\vec{u}) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad B(\vec{u}) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2\mu + \eta & 0 \\ 0 & (2\mu + \eta)u & \kappa \end{pmatrix}.$$

Using code provided courtesy of the authors of STABLAB, see [4], we obtain a numerical approximation of the traveling wave solution, which we label u_0. Gilbarg showed existence of traveling waves solutions to the system in [7]. We form a perturbation of the traveling wave, $v(x) = u_0(x) + \sin(x)\exp(-2x^2)$, for the initial condition of the time evolution. We apply Dirichlet boundary conditions to ρ, u, and e at $x = -10$, and to u and e at $x = 10$. We utilize a no flux boundary condition for ρ at $x = 10$. Details about why these are the appropriate choice of boundary conditions are given in [9]. For parameter values, we use $\Gamma = 2/3$, $\mu = 1$, $\eta = -2\mu/3$, and $\kappa = 2\mu$. The boundary condition of u at $x = 10$ is chosen to be 0.3. All other parameters are are given via the Rankine-Hugoniot conditions, as described in [8]. We display the time evolution of this perturbation in Figure 2. The time evolution study is consistent with stability of u_0.

We also perform a convergence study for this model, following the exact same protocol as for Burgers' equation. See Table 2 for the related data.

ϵ	1	1/2	1/4
$\|u - u_*\|_\infty$	2.19e-3	2.74e-4	5.44e-5

TABLE 2. Table displaying for nonisentropic Navier-Stokes, the convergence of the solution u of the finite difference scheme to u_* as step size decreases, $\Delta t = \epsilon^2/10$, $\Delta x = \epsilon/10$.

5. Example: Reactive Navier-Stokes

We demonstrate how our code performs when a reaction term is present by carrying out time evolution studies in the reactive Navier-Stokes (rNS) equations in Lagrangian

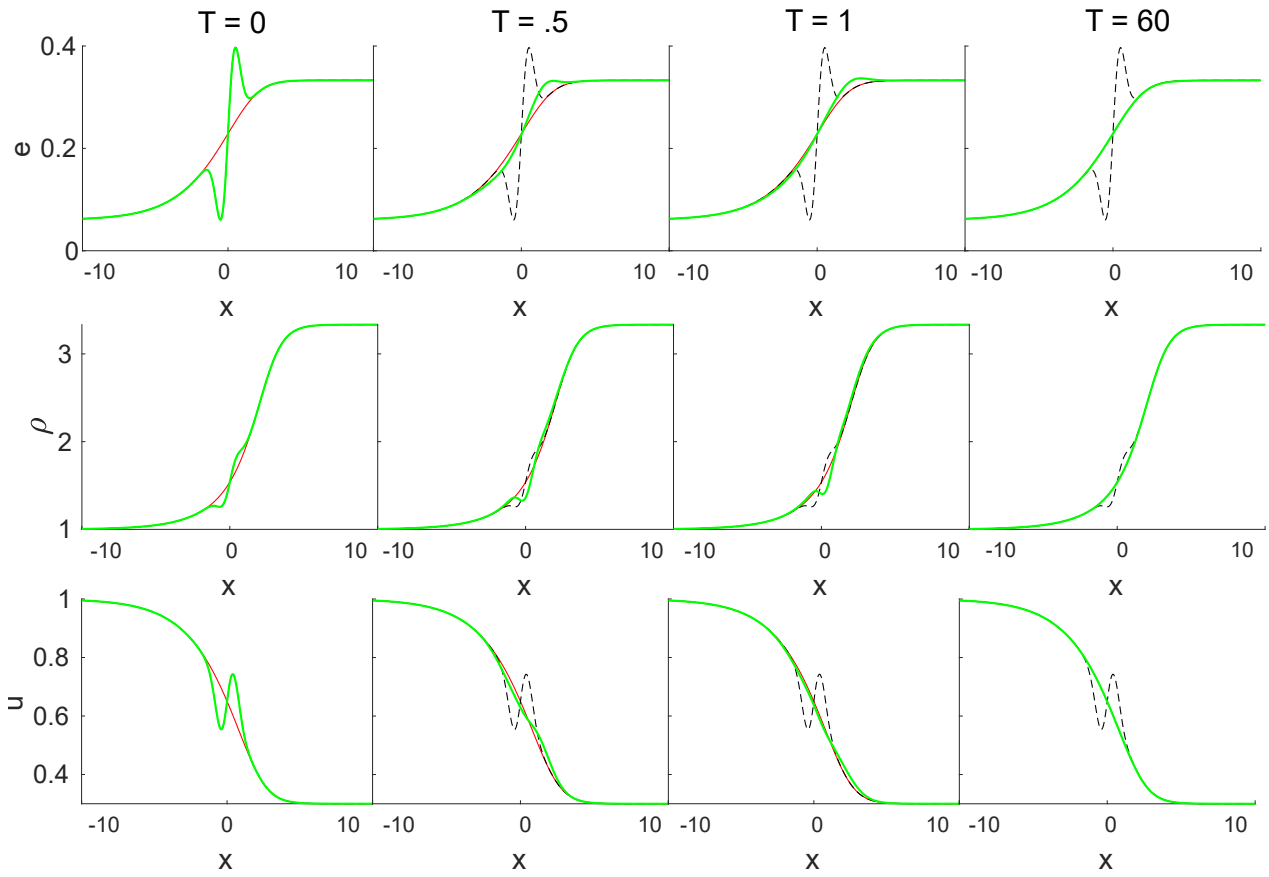


FIGURE 2. Plot of the time evolution of the perturbation $v(x) = u_0(x) + \sin(x)\exp(-2x^2)$ in the nonisentropic Navier-Stokes equations. Black dotted lines indicate the initial perturbation v, a red curve corresponds to the traveling wave solution u_0, and thick green lines indicate the evolution of v at time $T = 0$, $T = 0.5$, $T = 1$, $T = 60$.

coordinates; see [2], for example. The equations are given by

$$\begin{aligned} \tau_t - \tau_x - u_x &= 0, \\ u_t - u_x + p_x &= \left(\frac{\nu u_x}{\tau}\right)_x, \\ \left(e + \frac{u^2}{2}\right)_t - \left(e + \frac{u^2}{2}\right)_x + (pu)_x - qk\phi(T)z &= \left(\frac{\nu u u_x}{\tau} + \frac{\kappa T_x}{\tau}\right)_x, \\ z_t - z_x + k\phi(T)z &= \left(\frac{D z_x}{\tau^2}\right)_x, \end{aligned} \tag{5}$$

where volume, velocity, energy, and mass fraction of reactant of the gas are given by τ, u, e, and z, respectively. Viscosity coefficients are given by ν, κ, and D, and the difference in the heat of formation of the reactant and the product is given by q. We specifically consider an ideal, polytropic gas, hence, $p = \frac{RT}{\tau}$ and $e = c_v T$ where temperature is given by T, and R and c_v are constants that characterize the gas. In our study, we take $c_v = 1$, so that $e = T$. We use an Arrhenius type ignition function, $\phi(T) = \exp(-A/(c_v(T - T_{ig})))$ if $T > T_{ig}$

and 0 otherwise. Here A is the activation energy and T_{ig} is the ignition temperature. In flux coordinates, equation (5) is

$$\vec{u} = \begin{pmatrix} \tau \\ u \\ e \\ z \end{pmatrix}, \quad f^0(\vec{u}) = \begin{pmatrix} \tau \\ u \\ e + u^2/2 \\ z \end{pmatrix}, \quad f^1(\vec{u}) = \begin{pmatrix} -u - \tau \\ Re/\tau - u \\ Reu/\tau - (e + u^2/2) \\ -z \end{pmatrix},$$

$$R(\vec{u}) = \begin{pmatrix} 0 \\ 0 \\ -qk\phi(T)z \\ k\phi(T)z \end{pmatrix}, \quad B(\vec{u}) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \nu/\tau & 0 & 0 \\ 0 & \nu u/\tau & \kappa/(c_\nu \tau) & 0 \\ 0 & 0 & 0 & D/\tau^2 \end{pmatrix}.$$

Because the ignition function is piecewise constant, our code does not support these equations directly. However, by running the Python code with the non-zero portion of the ignition function, we were able to then manually insert the piecewise ignition function into the generated MATLAB files so it would be defined properly.

We experimented with various boundary conditions until we found ones that did not lead to finite blow up of the numerical solution. We used Dirichlet boundary conditions on the left for all the components, and on the right for τ, u, and e. We used a no flux boundary condition for z on the right.

We carried out a time evolution study for a stable and unstable wave solution provided to us courtesy of the authors of [2]. Existence of traveling wave solutions to the system was examined in [5], [6], and [12].

In Figure 3, we plot snapshots of the time evolution of a perturbed traveling wave with a profile solution determined by the parameter values, $A = 2.7$, $R = 0.2$, $s = 1$, $q = 0.6231$, $k = 12.3609$, $T_{ig} = 0.0664$, $c_\nu = 1$, $\nu = 0.1$, $\kappa = 0.1$. We note that in [2], it was found that this traveling wave is stable. Figure 4 shows a time evolution of all four system variables overlaid on a single graph to demonstrate the overall behavior of the system.

In addition to the stable wave, we demonstrate how the time evolution code behaves with the unstable wave, where $A = 7$. We use $R = 0.2$, $s = 1$, $q = 0.6231$, $k = 1.4723 \times 10^5$, $T_{ig} = .0664$, $c_\nu = 1$, $\nu = 0.1$, $\kappa = 0.1$. All other parameters are given via the Rankine-Hugoniot conditions, as described in [2]. The time evolution of this perturbed unstable wave is shown in Figure 5 and suggests the way in which the instability is manifest. It seems to be traveling and diffusing to the left. In Table 3 we display the results of our convergence study.

ϵ	1	1/2	1/4
$\|u - u_*\|_\infty$	4.90e-3	1.00e-3	2.05e-4

TABLE 3. Table displaying for reactive Navier-Stokes the convergence of the solution u of the finite difference scheme to u_* as step size decreases, $\Delta t = \epsilon^2/10$, $\Delta x = \epsilon/10$.

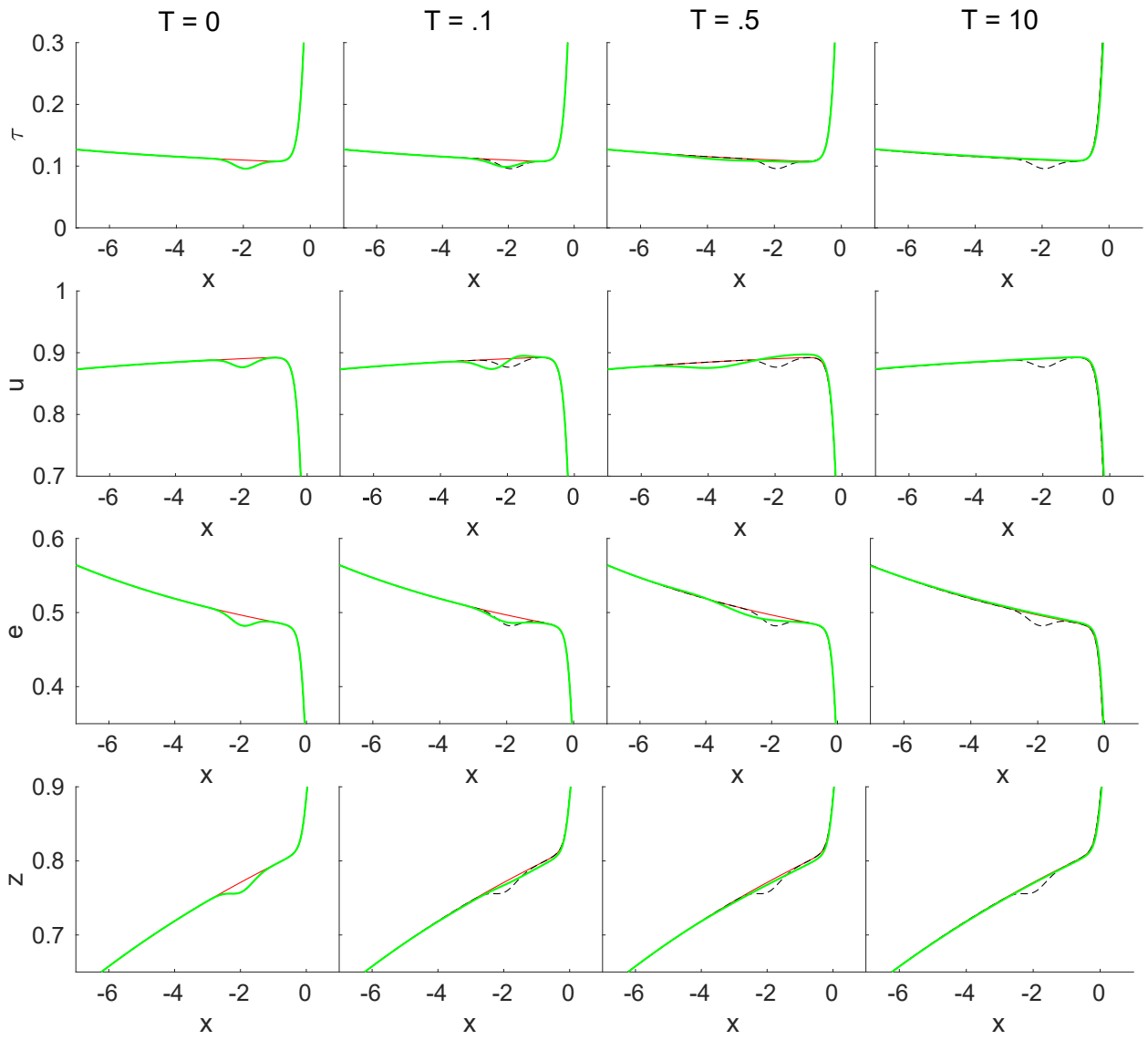


FIGURE 3. Plot of the time evolution of the perturbation $v(x) = u_0(x) + 0.015\sin(x)\exp(-4(x+2)^2)$ in the reactive Navier Stokes equations for $A = 2.7$. Black dotted lines indicate the initial perturbation v, a red curve corresponds to the traveling wave solution u_0, and thick green lines indicate the evolution of v at time $T = 0$, $T = 0.1$, $T = .5$, $T = 10$. The graph is zoomed in to make it more visible. This wave was found to be stable in [2], and that is what we observe.

6. Conclusion

We carry out the first time evolution study in rNS for some of the specific traveling waves considered in [2]. In particular, we evolve a perturbation of a stable and unstable wave. Time evolution of the perturbed stable wave approaches a translate of the traveling wave consistent with stability. Time evolution of the unstable wave suggests the manner in which the instability manifests itself. In particular, it appears that a small perturbation

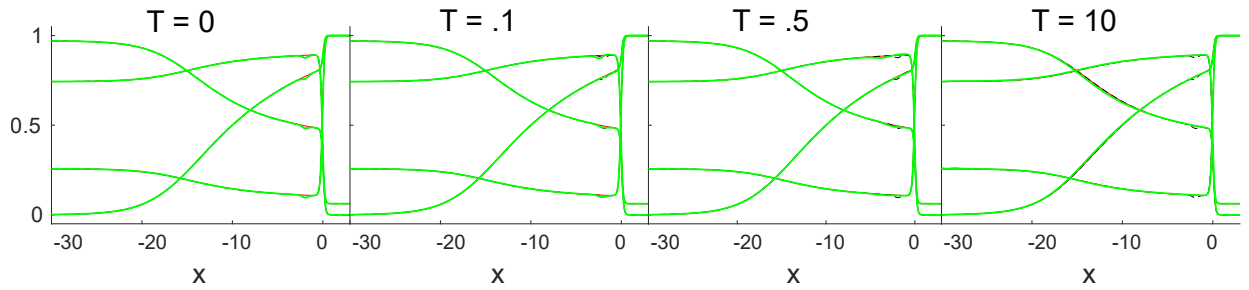


FIGURE 4. Plot of rNS with a stable value of $A = 2.7$ showing all four system variables in one graph. Black dotted lines indicate the initial perturbation v, a red curve corresponds to the traveling wave solution u_0, and thick green lines indicate the evolution of v at time steps, $T = 0$, $T = .1$, $T = .5$, and $T = 10$.

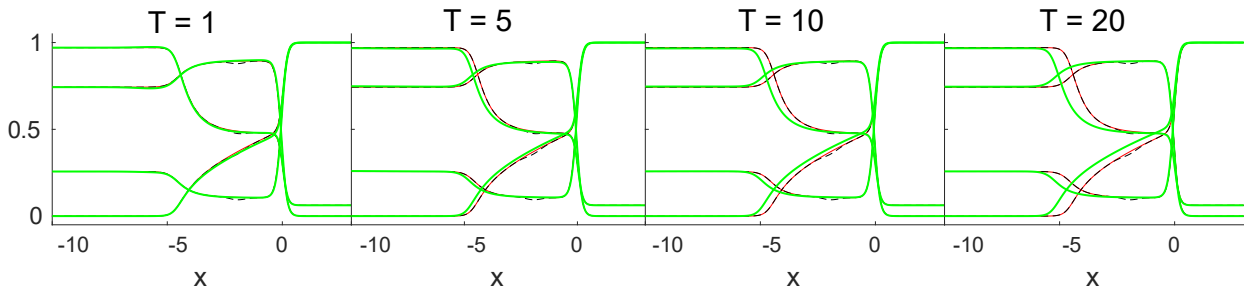


FIGURE 5. Plot of rNS with an unstable value of $A = 7$, showing all four system variables in one graph. Black dotted lines indicate the initial perturbation v, a red curve corresponds to the traveling wave solution u_0, and thick green lines indicate the evolution of v at time steps, $T = 0$, $T = 5$, $T = 10$, and $T = 20$.

of the traveling wave diffuses to the left while maintaining its original structure to the right.

We provide Python code and supporting MATLAB code that makes it so a researcher can input a conservation or reaction-diffusion law of the form given in (1) and all of the system specific code needed for time-evolution using the Crank-Nicolson scheme is automatically derived and written to MATLAB files. This code saves the user a significant amount of time since the derivations are automated, and also has the potential to save time since the automated code circumvents the opportunity to introduce human error into the derivation. This code is freely available at GitHub under finited-iff/finite_difference_matlab.

We demonstrate our code with application to Burgers' equation, the nonisentropic Navier-Stokes equations, and the reactive Navier-Stokes equations to show that it is versatile and can work with systems of various dimensions.

In the future, we plan to build functionality into the code allowing the user to choose various finite difference schemes. In particular, we would like the user to be able to choose a different finite difference scheme for the hyperbolic equations and parabolic

equations in a hyperbolic-parabolic system, like isentropic Navier-Stokes. We also plan to improve the efficiency of the code. We believe this code will provide a useful tool to researchers studying the stability of traveling waves.

Solving for the unstable manifold in the rNS system to verify the way in which instability manifests itself would be an interesting direction, which we plan to pursue. Our time evolution code will be useful in verifying the manifold for which we solve.

7. Acknowledgments

We thank Blake Barker, whose research is partially supported by the National Science Foundation under Award No. 1400872, for significant help on this project in the capacity of mentor. We thank the Brigham Young University math department and college of physical and mathematical sciences for support in completing this project. Research of J Morgan was supported by the BYU Math Department and the BYU College of Physical and Mathematical Sciences. Research of RD Todd was supported by an ORCA grant, the BYU Math Department, and the BYU College of Physical and Mathematical Sciences.

References

[1] N. J. Balmforth and J. J. Liu. Roll waves in mud. *Journal of Fluid Mechanics*, 519:33–54, 2004.

[2] B. Barker, J. Humpherys, G. Lyng, and K. Zumbrun. Viscous hyperstabilization of detonation waves in one space dimension. *SIAM Journal on Applied Mathematics*, 75(3):885–906, 2015.

[3] B. Barker, J. Humpherys, G. Lyng, and K. Zumbrun. Balanced flux formulations for multidimensional Evans-function computations for viscous shocks. *Quarterly of Applied Mathematics*, 76(3):531–545, 2017.

[4] B. Barker, J. Humpherys, J. Lytle, and K. Zumbrun. *STABLAB: A MATLAB-Based Numerical Library for Evans Function Computation*, 2015. http://github.com/nonlinear-waves/stablab/.

[5] R. A. Gardner. On the detonation of a combustible gas. *Transactions of the American Mathematical Society*, 277(2):431–431, 1983.

[6] I. Gasser and P. Szmolyan. A geometric singular perturbation analysis of detonation and deflagration waves. *SIAM Journal on Mathematical Analysis*, 24(4):968–986, 1993.

[7] D. Gilbarg. The existence and limit behavior of the one-dimensional shock layer. *American Journal of Mathematics*, 73(2):256, 1951.

[8] J. Humpherys, G. Lyng, and K. Zumbrun. Multidimensional stability of large-amplitude Navier–Stokes shocks. *Archive for Rational Mechanics and Analysis*, 226(3):923–973, 2017.

[9] B. Melinand and K. Zumbrun. Existence and stability of steady compressible Navier-Stokes solutions on a finite interval with noncharacteristic boundary conditions. 2017.

[10] E. E. Meshkov. Instability of the interface of two gases accelerated by a shock wave. *Fluid Dynamics*, 4(5):101–104, 1969.

[11] J. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2nd edition, 2004.

[12] M. Williams. Heteroclinic orbits with fast transitions: A new construction of detonation profiles. *Indiana University Mathematics Journal*, 59(3):1145–1209, 2010.

Student biographies

Jalen Morgan: (*Corresponding author:* jalen.morgan000@gmail.com) Jalen Morgan is an undergraduate student at Brigham Young University (BYU) with a strong background in mathematics and computer science. Currently, Jalen is studying Applied Mathematics. He has worked with the Math department as a Research Assistant since December of 2017, working closely with professor Blake Barker. In 2018, Jalen served as the president of the BYU student chapter for SIAM (Society for Industrial and Applied Mathematics). He especially enjoys working on problems that offer new challenges and creative thinking. To learn more, visit his profile at linkedin.com/in/jalen-morgan.

Ray David Todd: (*Corresponding author:* rd.todd25@gmail.com) Ray David Todd graduated from Brigham Young University in 2018 with a B.S. in Mathematics and a minor in Global Studies. He is currently employed as a secondary mathematics teacher and academic coach, where he seeks to share his passion for mathematics with the next generation of students.