2017-06-01

Stability of Planar Detonations in the Reactive Navier-Stokes Equations

Joshua W. Lytle
Brigham Young University

Stability of Planar Detonations in the Reactive Navier-Stokes Equations

Joshua W. Lytle

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Mathematics

Brigham Young University

ABSTRACT

Stability of Planar Detonations in the Reactive Navier-Stokes Equations

Joshua W. Lytle
Department of Mathematics, BYU
Doctor of Philosophy

This dissertation focuses on the study of spectral stability in traveling waves, with a special interest in planar detonations in the multidimensional reactive Navier-Stokes equations. The chief tool is the Evans function, combined with STABLAB, a numerical library devoted to calculating the Evans function. Properly constructed, the Evans function is an analytic function in the right half-plane whose zeros correspond in multiplicity and location to the spectrum of the traveling wave. Thus the Evans function can be used to verify stability, or to locate precisely any unstable eigenvalues. We introduce a new method that uses numerical continuation to follow unstable eigenvalues as system parameters vary. We also use the Evans function to track instabilities of viscous detonations in the multidimensional reactive Navier-Stokes equations, building on recent results for detonations in one dimension. Finally, we introduce a Python implementation of STABLAB, which we hope will improve the accessibility of STABLAB and aid the future study of large, multidimensional systems by providing easy-to-use parallel processing tools.

Keywords: detonations, Evans function, traveling waves, Navier-Stokes, planar waves, root following

Acknowledgments

I would like to express my gratitude to my advisor Jeffrey Humpherys, for his encouragement and help during my time at graduate school. His friendship and vision of mathematics has been a tremendous influence on me.

I would like to thank my collaborators, Blake Barker, Anna Ghazaryan, Greg Lyng, and Kevin Zumbrun, for the opportunity to work with them and for their contributions to the field of traveling waves. My appreciation extends to my committee for their valuable input and service: Blake Barker, Emily Evans, Vianey Villimizar, and Jared Whitehead.

I would like to thank my parents, Wayne and Marianne Lytle, for providing a foundation for my education and for their support during this work. I would also like to thank Eldon and Rula Lytle for being an early source of inspiration and motivation, and to my family for their encouragement.

I have had a great many teachers to whom I remain indebted; to name a few, Professors David Cardon, Chris Grant, Vianey Villamizar, Kening Lu, and David Wright. I would like to thank the professors I have worked with on the ACME (Applied and Computational Mathematics Emphasis) program at BYU: Jeffrey Humpherys, Tyler Jarvis, Emily Evans, and Jared Whitehead. I would like to thank the mathematics department secretary Lonette Stoddard for her helpfulness and encouragement, and the department staff for their hard work and support.

Contents

List of Tables

List of Figures

Chapter 1. Introduction to stability for traveling waves

A traveling wave arises as a solution of an evolution equation

$$u_t = \mathcal{F}(u, u_x, u_{xx}, \dots) \tag{1.1}$$

which has the form $u(x,t) = \hat{u}(x - st)$, where s is the speed of the traveling wave. Traveling waves are an important subject in applied mathematics. They occur as solutions of physical systems studied in many scientific disciplines, for example in nonlinear optics, materials science, population dynamics, gas dynamics, and combustion processes.

Classical examples of traveling waves include detonations in combustion and solitons in fluid dynamics. Solitons were first discovered in 1834 by John Scott Russel, a naval architect. Russel was looking for the most efficient design for canal boats when he observed the soliton—a peak of water massed at the bow of the boat, which continued down the canal at a constant speed after the boat had stopped moving.

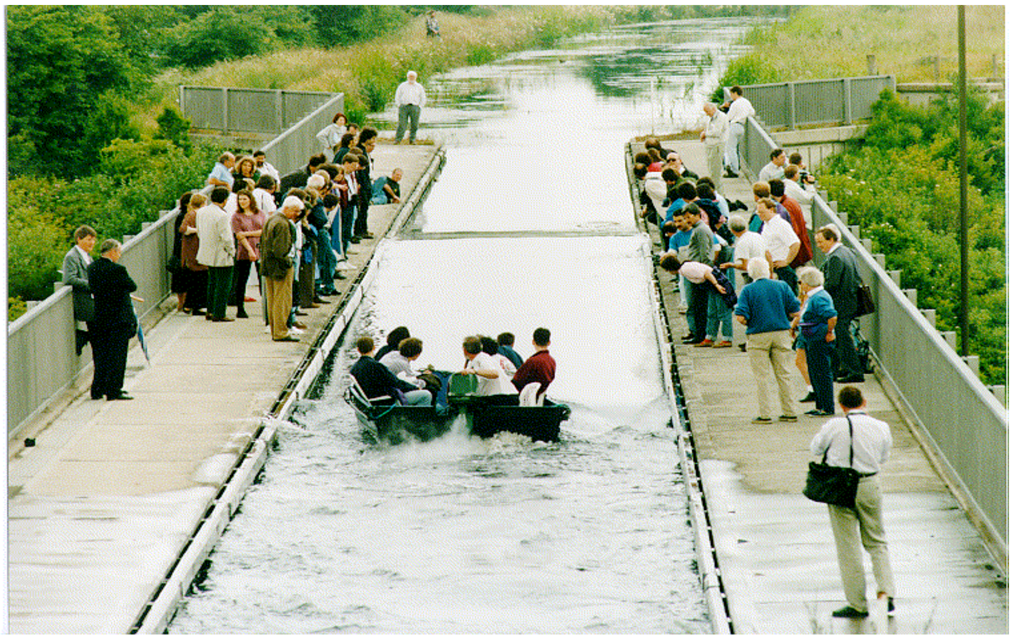


Figure 1.1: Re-creation in 1995 of John Russel's initial sighting of a soliton on the Union Canal, Edinburgh. Thanks to Heriot-Watt University, Scotland.

An important question to resolve in the study of traveling waves is whether a traveling wave is stable. The stability of a traveling wave can determine whether the wave is likely to be observed experimentally, or if the wave is an artifact of a mathematical framework with no real world connection. Verifying the mathematical stability of an unobserved traveling wave suggests possible experiments for scientists.

The spectrum of the operator linearized about the traveling wave has implications for the nonlinear stability of a traveling wave. Zumbrun and collaborators [30, 14, 25, 24] have shown that spectral stability of the linearized operator implies nonlinear stability for certain important classes of (1.1).

To study spectral stability we compute the Evans function, whose zeros correspond to eigenvalues of the linearized operator. The Evans function is analytic in the right half-plane, allowing us to use rootfinding techniques to locate eigenvalues. Energy estimates are often used to find a bounded subset of the right half-plane containing the relevant eigenvalues. A winding number of zero for a contour about the bounded region indicates spectral stability. For those systems that do become spectrally unstable in some parameter regime, we would like to know exactly when and how the instability occurs. We can do this by creating bifurcation diagrams of eigenvalues as they cross into the right half-plane.

1.1 Nonlinear stability

A traveling wave solution $\hat{u}$ of (1.1) solves the profile ODE

$$-su' = \mathcal{F}(u, u', u'', \ldots).$$

This is found by plugging the ansatz $u(x,t) = \hat{u}(x - st)$ into the evolution equation (1.1). The profile ODE can also be found by translating the system into the moving frame: $(x,t) \to$

$(x - st, t)$. In the moving frame the PDE is written

$$u_t = su_x + \mathcal{F}(u, u_x, u_{xx}, \ldots). \tag{1.2}$$

The traveling wave is then a stationary solution of the PDE (1.2) in the moving frame, that is, a solution of the system $s\hat{u}' + \mathcal{F}(\hat{u}) = 0$.

Let X be an appropriate Banach space, and $\mathcal{A} \subset X$ an admissible set of perturbations. The stability problem involves understanding the solutions $u(x,t)$ of the Cauchy problem

$$\begin{aligned} u_t &= su_x + \mathcal{F}(u), \\ u(x,0) &= \hat{u}(x) + v(x,0), \quad v(x,0) \in \mathcal{A}, \end{aligned} \tag{1.3}$$

for admissible perturbations $v \in \mathcal{A}$. We then ask several questions about the solution u: Will the perturbed wave converge to (some translate) $\hat{u}(x+\delta)$ as $t \to \infty$? Or will it lose energy and cohesion, oscillate, or evolve into something else entirely? These questions motivate the definition of nonlinear stability.

Definition 1.1 (*Asymptotic orbital stability* or *nonlinear stability*)**.** A stationary solution $\hat{u}$ of (1.2) is *asymptotically orbitally stable* with regard to the class of perturbations $\mathcal{A}$ if

$$u(\cdot, t) \to \hat{u}(x+\delta) \quad \text{as } t \to \infty$$

for some translation δ whenever $v(x,0) = u(x,0) - \hat{u}(x) \in \mathcal{A}$.

1.2 Spectral stability

The eigenvalue problem comes from linearizing the PDE

$$u_t = su_x + \mathcal{F}(u, u_x, u_{xx}, \ldots) \tag{1.4}$$

about the wave front $\hat{u}$ to get

$$\lambda v = Lv = (s\partial_x + d\mathcal{F}(\hat{u}))v. \tag{1.5}$$

Definition 1.2. Consider the linear operator L defined by (1.5). The spectrum $\sigma(L)$ of L consists of all values $\lambda \in \mathbb{C}$ such that $L - \lambda I$ is not invertible. The spectrum of L can be further decomposed into two sets: the point spectrum $\sigma_p(L)$ of L and the essential spectrum $\sigma_e(L)$ of L. The point spectrum of L are those values $\lambda \in \sigma(L)$ that are isolated eigenvalues of L having finite multiplicity. The remaining points in $\sigma(L)$ are called the essential spectrum.

Definition 1.3. The operator L in (1.5) is spectrally stable if its spectrum does not extend into the closed deleted right half-plane $\sum_+ = \{\lambda \in \mathbb{C}\backslash\{0\} : \mathrm{Re}(\lambda) \geq 0\}$.

The essential spectrum can often be bounded in the left half of the complex plane. The point spectrum can then be considered separately. The Evans function will be our tool of choice as we study the point spectrum. This function, defined in the right half-plane, plays a role similar to the characteristic polynomial. The zeros of the Evans function correspond in both location and multiplicity to the point spectrum of the traveling wave.

A conceptual argument that spectral stability may be used to determine nonlinear stability goes as follows: Any initial state u can be written as $u = \hat{u} + v$, where v is a perturbation of $\hat{u}$. We linearize (1.2) about $\hat{u}$, obtaining

$$\hat{u}_t + v_t = s(\hat{u} + v)_x + \mathcal{F}(\hat{u}) + d\mathcal{F}(\hat{u})v + N(v), \quad N(v) = \mathcal{O}(|v|^2). \tag{1.6}$$

Since $\hat{u}_t = 0$ and $s\hat{u}_x + \mathcal{F}(\hat{u}) = 0$ we obtain

$$\begin{aligned} v_t &= (s\partial_x + d\mathcal{F}(\hat{u}))v + N(v), \\ &= Lv + Nv. \end{aligned}$$

Then using Duhamel's method the solution $v(x,t)$ can be written

$$v = e^{Lt}v_0 + \int_0^t e^{L(t-s)}N(s)ds.$$

Since v is small and $N \sim \mathcal{O}(v^2)$, $\int_0^t e^{L(t-s)}N(s)ds$ is negligible and $v \approx e^{Lt}v_0$.

We note that $\lambda = 0$ is always an eigenvalue of L, and is associated with the translational invariance of the traveling wave.

Lemma 1.4 (Sattinger [28])**.** *The derivative of $\hat{u}$ is an eigenfunction of L with eigenvalue $\lambda = 0$.*

Proof. Due to the translational invariance of the traveling wave, $\mathcal{F}(\hat{u}(x+\delta)) = 0$ for each $\delta \in \mathbb{R}$. Then $\frac{d}{d\delta}\left(\mathcal{F}(\hat{u}(x+\delta))\right)|_{\delta=0} = d\mathcal{F}(\hat{u})\hat{u}' = 0$, so $\hat{u}'$ is an eigenvector of $L = d\mathcal{F}(\hat{u})$ with eigenvalue 0. □

Consider the class of evolution equations

$$u_t + f(u)_x - (B(u)u_x)_x + (C(u)u_{xx})_x + Q(u) = 0 \tag{1.7}$$

where $x \in \mathbb{R}, u, f \in \mathbb{R}^n$, and $B, C, Q \in \mathbb{R}^{n\times n}$ are sufficiently smooth. The function $f(u)_x$ is the flux or convection/advection of u, $(B(u)u_x)_x$ the diffusion of u, $(C(u)u_{xx})_x$ the dispersion term, and $Q(u)$ the reaction term.

Suppose $\hat{u}$ is a traveling wave solution of (1.7). Translating (1.7) with $(x,t) \to (x-st,t)$, we obtain the equation

$$u_t + (f(u) - su)_x - (B(u)u_x)_x + (C(u)u_{xx})_x + Q(u) = 0, \tag{1.8}$$

of which the wave $\hat{u}$ is a stationary solution. Linearizing (1.8) about $\hat{u}$ we obtain

$$v_t = -(Av)_x + (Bv_x)_x - (Cv_{xx})_x - Dv + N(v),$$

where

$$Av := (df(\hat{u}) - s)v - dB(\hat{u})v\hat{u}_x + dC(\hat{u})v\hat{u}_{xx},$$

$$B := B(\hat{u}), \quad C := C(\hat{u}), \quad D := dQ(v),$$

and where the nonlinear portion $N(v)$ consists of higher order terms in v and its derivatives. This yields the linear eigenvalue problem

$$\lambda v = Lv := -(Av)_x + (Bv_x)_x - (Cv_{xx})_x - Dv. \tag{1.9}$$

Following the exposition in [21] and its references, in the next two sections we will describe how to deal with the essential and point spectrums of L.

1.2.1 Essential spectrum.

The following result is useful in dealing with the essential spectrum of L.

Theorem 1.5 (Henry [13]). *Let $L_\pm$ be the operators defined by linearizing* (1.9) *about $\hat{u} = u_\pm$. Then the essential spectrum of L is bounded to the left of*

$$\sigma_e(L_+) \cup \sigma_e(L_-).$$

Proof. We note that linearization about $u_\pm$ yields

$$v_t = L_\pm v = -A_\pm v_x + B_\pm v_{xx} - C_\pm v_{xxx} - D_\pm v,$$

where $A_\pm = A(u_\pm), \ldots, D_\pm = D(u_\pm)$ are constant matrices. Recalling that the constant coefficient linear operators $L_\pm$ do not have a point spectrum, we have $\sigma(L_\pm) = \sigma_e(L_\pm)$.

We use the Fourier transform to make a formal argument concerning $\sigma_e(L_\pm)$. Now

$$\widehat{(L - \lambda I)}^{-1} v = (-i\xi A_\pm - \xi^2 B_\pm + i\xi^3 C_\pm - D_\pm - \lambda I)^{-1} v$$

where $\xi \in \mathbb{R}$. Note that invertibility of $L-\lambda I$ is lost whenever $-i\xi A_\pm - \xi^2 B_\pm + i\xi^3 C_\pm - D_\pm - \lambda I$ is singular, so that $\lambda \in \sigma(L_\pm)$ exactly when $\lambda \in \sigma(-i\xi A_\pm - \xi^2 B_\pm + i\xi^3 C_\pm - D_\pm)$ for some $\xi \in \mathbb{R}$. Thus we have $2n$ curves $\lambda_j^\pm$ defined by the eigenvalues of $-i\xi A_\pm - \xi^2 B_\pm + i\xi^3 C_\pm - D_\pm$, with

$$\sigma_e(L_+) \cup \sigma_e(L_-) = \bigcup_j \lambda_j^+(\xi) \cup \bigcup_j \lambda_j^-(\xi).$$

Thus the eigenvalues of $A_\pm, B_\pm, C_\pm$, and $D_\pm$ describe bounds on the essential spectrum of L. $\square$

1.2.2 Point spectrum.

The problem of obtaining bounds on the point spectrum of L is generally much harder than dealing with the essential spectrum. Energy estimates have proven to be a useful tool in many instances, although their application is often not obvious or intuitive.

One of the difficulties associated with obtaining uniform bounds stems from the presence of $\lambda = 0$ in the point spectrum of L. For the reactionless equation (1.9), $Q(u) = 0$, the eigenvalue problem may instead be viewed in integrated coordinates. Specifically, by integrating both sides of

$$\lambda v = Lv = -(A(\hat{u})v)_x + (B(\hat{u})v_x)_x - (C(\hat{u})v_{xx})_x$$

from $-\infty$ to x, and substituting $w(x) = \int_{-\infty}^x v(y)\,dy$, we obtain the integrated operator

$$\lambda w = \mathcal{L}w := -A(\hat{u})w' + B(\hat{u})w'' - C(\hat{u})w'''.$$

Lemma 1.6. *The point spectrum of $\mathcal{L}$ is the same as L, excluding $\lambda = 0$.*

Proof. Suppose $\lambda v = Lv$ with $\lambda \neq 0$. Integrating both sides from $-\infty$ to x and substituting

$w(x) = \int_{-\infty}^{x} v(y)\,dy$ yields $\lambda w = \mathcal{L}w$. Since

$$\lambda w(+\infty) = -\int_{-\infty}^{\infty} (Av)' + \int_{-\infty}^{\infty} (Bv')' - \int_{-\infty}^{\infty} (Cv'')' = 0,$$

we note that w decays to 0. Similarly $w^{(n)}$ decays to 0 for $n = 1, \ldots,$ so w is an admissible eigenvector and $\sigma(L)\backslash\{0\} \subset \sigma(\mathcal{L})$.

Now suppose $\lambda w = \mathcal{L}w$, $\lambda \neq 0$. Differentiating yields

$$\lambda w' = -(Aw')' + (Bw'')' - (Cw''')',$$

making w' an eigenvalue of L. Thus $\sigma_p(\mathcal{L})\backslash\{0\} = \sigma_p(L)\backslash\{0\}$. □

Example 1.7. Consider Burgers equation

$$u_t + uu_x = u_{xx},$$

together with its traveling wave solution $\hat{u}$. Burgers equation in its moving frame $(x, t) \to (x - st, t)$ is

$$u_t - su_x + uu_x = u_{xx}, \quad x \in \mathbb{R}. \tag{1.10}$$

The wave profile $\hat{u}$ is a stationary solution of (1.10), with $\hat{u}_t = 0$. Thus $\hat{u}$ can be found by solving

$$-su' + \left(\frac{u^2}{2}\right)' = u''. \tag{1.11}$$

Integrating (1.11) from $-\infty$ to x yields

$$-s(u - u_-) + \frac{u^2 - u_-^2}{2} = u'. \tag{1.12}$$

As $x \to \infty$ we obtain the Rankine-Hugoniot condition

$$-s(u_+ - u_-) + \frac{u_+^2 - u_-^2}{2} = 0.$$

This uniquely specifies the wave speed $s = (u_+ + u_-)/2$. Equation (1.12) has the general solution

$$\{\hat{u}(x - st + \delta)\}_{\delta \in \mathbb{R}} = \left\{ s - a \tanh\left(\frac{a(x - st + \delta)}{2} \right) \right\}_{\delta \in \mathbb{R}}, \quad a = \frac{u_- - u_+}{2}. \tag{1.13}$$

Let u be the solution of (1.10) corresponding to the perturbed initial data

$$u(x,t)_{t=0} = \hat{u}(x) + v(x), \quad v \in \mathcal{A}.$$

Substituting u into (1.10), we use $vv_x = o(|v|^2)$ and $-s\hat{u}_x + \hat{u}\hat{u}_x = \hat{u}_{xx}$ to obtain the linearized evolution equation

$$v_t - sv_x + \hat{u}_x v + \hat{u} v_x = v_{xx}.$$

This can be written as $v_t = Lv$, with linear differential operator

$$L := (s - \hat{u})\partial_x - \hat{u}_x + \partial_{xx}$$

and associated eigenvalue problem $\lambda v = Lv$. To put the eigenvalue equation in integrated coordinates, we integrate both sides from $-\infty$ to x and substitute $w = \int_{-\infty}^{x} v$ to get

$$\lambda w = (s - \hat{u})w' + w''. \tag{1.14}$$

Example 1.8. Consider the essential spectrum of the eigenvalue problem for Burgers equation. By taking the Fourier transform of the eigenvalue problem linearized about $u_\pm$, we see that the essential spectrum must be to the left of the curves given by

$$\begin{aligned}\lambda &= -i(u_\pm - s)\xi - \xi^2, \\ &= \pm i\frac{u_- - u_+}{2}\xi - \xi^2, \quad \xi \in \mathbb{R}.\end{aligned}$$

These curves define a parabola in the left half of the complex plane that touches the origin at $\xi = 0$.

Example 1.9. We will use an energy estimate to show stability for Burgers equation. Multiplying both sides of (1.14) by $\overline{w}$ and integrating over the real line, we obtain

$$\lambda \int_{\mathbb{R}} |w|^2 = \int_{\mathbb{R}} (s - \hat{u})\overline{w}w' + \int_{\mathbb{R}} \overline{w}w''.$$

Integration by parts gives

$$\begin{aligned}\int_{\mathbb{R}} \overline{w}w'' &= \overline{w}w'|_{-\infty}^{\infty} - \int_{\mathbb{R}} |w'|^2, \\ &= -\int_{\mathbb{R}} |w'|^2,\end{aligned}$$

so

$$\lambda \int_{\mathbb{R}} |w|^2 = \int_{\mathbb{R}} (s - \hat{u})\overline{w}w' - \int_{\mathbb{R}} |w'|^2.$$

Integrating $(s - \hat{u})\overline{w}w'$ by parts gives

$$\begin{aligned}\int_{\mathbb{R}} (s - \hat{u})\overline{w}w' &= (s - \hat{u})\overline{w}w|_{-\infty}^{\infty} - \int_{\mathbb{R}} (s - \hat{u})\overline{w}'w + \int_{\mathbb{R}} w\overline{w}\hat{u}', \\ 2\Re\left(\int_{\mathbb{R}} (s - \hat{u})\overline{w}w'\right) &= \frac{1}{2}\int_{\mathbb{R}} |w|^2\hat{u}'.\end{aligned}$$

From (1.13) we have $\hat{u}' < 0$. This leads us to

$$\Re(\lambda) < 0.$$

A similar proof can be used to show spectral stability for the general scalar conservation law

$$u_t + f(u)_x = (b(u)u_x)_x;$$

see [21] for further examples of energy estimates.

1.3 The Evans function

The study of spectral stability of a traveling wave begins with bounding its essential spectrum to the left half-plane. After the essential spectrum has been bounded, the point spectrum in the right half-plane can be analyzed using the Evans function. Recall that the point spectra of a traveling wave $\hat{u}$ consists of values λ for which there are nontrivial solutions v of the eigenvalue equation

$$\lambda v = \mathcal{L}v = (s\partial_x + D\mathcal{F}(\hat{u}))v. \tag{1.15}$$

The Evans function is defined as the Wronskian of decaying solutions of (1.15), and is analytic to the right of the essential spectrum. It serves as a "characteristic function" for the linearized operator, analagous to the characteristic polynomial for matrices—its roots correspond in both location and multiplicity to the eigenvalues of the operator. The Evans function can rarely be given as a closed-form, analytic expression. This has been done only for simple systems, such as Burgers equation. Numerical Evans function computation allows us to study physical systems with much greater complexity.

Equation (1.15) can be rewritten as a first order system

$$\begin{aligned}&\frac{d}{dx}W = A(x;\lambda)W, \quad W \in \mathbb{C}^n,\\ &W(\pm\infty) = 0.\end{aligned} \tag{1.16}$$

Thus eigenvalues of (1.15) are those values of λ for which there is a nontrivial solution W of (1.16). The Evans matrix $A(x;\lambda)$ is asymptotically constant since $\hat{u}$ is asymptotically constant; thus we can define

$$A_\pm(\lambda) = \lim_{x\to\pm\infty} A(x;\lambda).$$

We will assume that $A_\pm(\lambda)$ are consistently split, so that

$$\dim U_\pm(\lambda) = k, \quad \dim S_\pm(\lambda) = n-k,$$

where $U_\pm(\lambda)$ and $S_\pm(\lambda)$ are the unstable and stable eigenspaces of $A_\pm(\lambda)$.

Let $U_-(x;\lambda)$ denote the unstable manifold of initial conditions at x whose solutions decay exponentially as $x \to -\infty$, and $S_+(x;\lambda)$ the stable manifold of initial conditions at x whose solutions decay exponentially as $x \to +\infty$. Thus equation (1.16) has a nontrivial bounded solution exactly when $U_-(x;\lambda) \cap S_+(x;\lambda) \neq \emptyset$. If $W_-(x;\lambda)$ and $W_+(x;\lambda)$ are matrices whose columns consist of analytically varying bases for $U_-(x;\lambda)$ and $S_+(x;\lambda)$ respectively, then the Evans function can be defined by

$$D(\lambda) = \det\left(\begin{bmatrix} W_- & W_+ \end{bmatrix}\right)\Big|_{x=0} \tag{1.17}$$

Here $W_+(x;\lambda) = [W_{k+1}^+(x;\lambda), \ldots, W_n^+(x;\lambda)]$ represents the $n-k$ decay modes of $A_+(\lambda)$, and $W_-(x;\lambda) = [W_1^-(x;\lambda), \ldots, W_k^-(x;\lambda)]$ represents the k growth modes of $A_-(\lambda)$. We note that $D(\lambda) = 0$ exactly when $W_1^-, \ldots, W_k^-, W_{k+1}^+, \ldots, W_n^+$ are linearly dependent. Linear dependence results in a solution W of (1.16), an eigenfunction, that connects $U_-(x;\lambda)$ and $S_+(x;\lambda)$ at $x=0$.

Since $W_i^-(x;\lambda)$ is a growth mode of $A_-(\lambda)$, W_i^- is a solution of (1.16) satisfying

$$W_i^-(x;\lambda) \sim e^{\mu_i^- x} r_i^-, \quad x << 0,$$

where r_i^- is the eigenvector associated with positive eigenvalue μ_i^- in the spectrum of $A_-(\lambda)$. Thus for L sufficiently large $W_i^-(x;\lambda)$ may be approximated by the solution of

$$\begin{aligned} \frac{d}{dx} W_i &= A(x;\lambda) W_i, \\ W_i(-L) &= e^{-\mu_i^- L} r_i^-. \end{aligned} \tag{1.18}$$

Similar statements hold for decay modes $W_i^+(x;\lambda)$ of $A_+(\lambda)$. To ensure the analyticity of each $W_i^\pm(x;\lambda)$, and therefore the analyticity of $D(\lambda)$, it is necessary to compute an analytically varying basis $r_i(\lambda)$ of the eigenspace associated with μ_i. This can be done conveniently using the method of Kato.

1.3.1 Adjoint formulation of the Evans function. There is an alternative characterization of the Evans function which is often used in the case that $k < n/2$. Let $\tilde{W}_+(x;\lambda) = [\tilde{W}_1, \ldots, \tilde{W}_k]$ be a matrix whose columns constitute an analytic basis for the stable manifold $S_+^*(x;\lambda)$ of the adjoint equation

$$\frac{d}{dx}\tilde{W} = -A(x;\lambda)^* \tilde{W}$$

near $x = +\infty$. Specifically, let $\tilde{\mu}_1^+, \ldots, \tilde{\mu}_k^+$ be the growth modes of $-A_+(\lambda)^*$, and $\tilde{r}_1^+, \ldots, \tilde{r}_k^+$ the associated analytic eigenvectors. $\tilde{W}_i$ can be found by solving the IVP

$$\begin{aligned} \frac{d}{dx}\tilde{W}_i &= -A(x;\lambda)^* \tilde{W}_i, \\ \tilde{W}(+L) &= e^{\tilde{\mu}_i^+ L} \tilde{r}_i^+. \end{aligned} \tag{1.19}$$

An alternate Evans function can be defined by

$$\det\left(\tilde{W}_+^* W_-\right)\Big|_{x=0}. \tag{1.20}$$

Note that $\det\left(\tilde{W}_+^* W_-\right) = 0$ precisely when there is some choice of constants $c_1, \ldots, c_k$, not all zero, so that

$$\tilde{W}_+^*(c_1 W_1^- + \ldots + c_k W_k^-) = 0.$$

Consider $\tilde{W}_i$ and W_j where $\tilde{W}_i$ is a solution of (1.19) and W_j is a solution of (1.18). It is straightforward to show that $(\tilde{W}_i^* W_j)' = 0$, implying that $\tilde{W}_i^* W_j$ is a constant. Moreover we find that $\tilde{W}_i^*(+\infty)W_j(+\infty) = \tilde{r}_i^* r_j = 0$, since $\tilde{r}_i^*$ is a left eigenvector of $A_+(\lambda)$ with an eigenvalue with positive real part, and thus is orthogonal to r_j. Since the orthogonal complement of the space spanned by the columns of $\tilde{W}_+$ is the space spanned the columns of W_+, this Evans function is zero exactly when W_- and W_+ intersect.

Chapter 2. Numerical computation of the Evans Function

2.1 Computation of the wave profile

Our numerical stability program relies on determining the existence of nontrivial bounded solutions of the Evans system

$$\begin{aligned} &\frac{d}{dx}W = A(x;\lambda)W, \quad W \in \mathbb{C}^n, \\ &W(\pm\infty) = 0. \end{aligned} \tag{2.1}$$

The x-dependence in the Evans matrix $A(x;\lambda)$ typically comes from specific translates of the traveling wave solution and its derivatives. Since there generally is no known explicit formula for the traveling wave solution of a nonlinear PDE, we outline the basics of their numerical solution.

We will restrict our attention to traveling waves with asymptotic boundary conditions. These waves satisfy a boundary value problem on an infinite domain,

$$\begin{aligned} &-su' = f(u, u', u'', \dots), \\ &u(\pm\infty) = u_\pm, \quad u^{(n)}(\pm\infty) = 0,\ n = 1, 2, \dots. \end{aligned} \tag{2.2}$$

This can be written as a first order system

$$\begin{aligned} &U' = F(U), \quad U \in \mathbb{R}^n, \\ &U(\pm\infty) = U_\pm. \end{aligned} \tag{2.3}$$

A traveling wave corresponds to a homoclinic or heteroclinic orbit connecting fixed points in the phase space of (2.3). When $u_- = u_+$ the traveling wave is a pulse, and exists as a homoclinic orbit in phase space. For $u_- \neq u_+$ the traveling wave is a front and corresponds to a heteroclinic orbit. Proving the existence of a connecting orbit between equilibrium points is an important task. There is a standard body of mathematical tools like Lyapunov methods, asymptotic methods, and other topological methods dedicated to demonstrating existence. Here we will content ourselves with discussing their numerical solution.

Boundary conditions. Since the profile ODE (2.3) is autonomous, each solution is translationally invariant in x. Any traveling wave solution $\hat{u}(x)$ is a member of a family of solutions $\{u(x+\delta)\}$, $\delta \in \mathbb{R}$. A phase condition is required to identify a particular translate of the wave. Because the wave corresponds to a connecting orbit in phase space, projective conditions must be imposed to specify how the wave approaches its end states. In any numerical solver, the computational domain must be truncated to some finite interval $[-L, L]$, where $L > 0$ is large enough to capture the asymptotic nature of the wave. The phase condition is typically imposed at $x = 0$, and the projective conditions at $x = \pm L$.

Let $X_S^{\pm}$, $X_U^{\pm}$ be the stable and unstable eigenspaces for $dF(U_{\pm})$, and let $\Pi_S^{\pm}$, $\Pi_U^{\pm}$ be the eigenprojections onto their respective eigenspaces. As $x \to -\infty$, the solution U approaches its end state U_- along the unstable eigenspace of $dF(U_-)$. Similarly, near $x = +\infty$ the solution U will approach U_+ along the stable eigenspace of $dF(U_+)$. We can enforce this by imposing the projective conditions

$$\begin{aligned} \Pi_S^-(U(-L) - U_-) &= 0, \\ \Pi_U^+(U(+L) - U_+) &= 0. \end{aligned} \tag{2.4}$$

Formulation for a BVP solver. This combination of projective conditions and a phase condition leaves us with a condition in the middle of the computational domain. To obtain a two-point boundary value problem, we double the ODE by flipping the solution on one side to the other. This takes us from an n-dimensional ODE

$$U' = F(U), \quad x \in [-L, L], \quad U \in \mathbb{R}^n$$

to the $2n$-dimensional ODE

$$\begin{bmatrix} U \\ V \end{bmatrix}' = \begin{bmatrix} F(U) \\ -F(V) \end{bmatrix}, \quad x \in [0, L], \quad U, V \in \mathbb{R}^n.$$

To this last system we add n "matching" conditions $U(0) = V(0)$. If the boundary value problem has been properly formulated, the number of boundary conditions and the number of ODEs will match up. Sometimes the wave speed s must also be solved numerically—in this case, the ODE can be augmented with the equation $s' = 0$.

Many BVP solvers rely on an advanced implementation of Newtons method; any nonlinear BVP solver will require an initial guess. Tanh and sech functions can often be used to provide satisfactory initial guess functions for fronts and pulses, respectively. However, interesting physical systems usually have one or more tunable parameters with a parameter space over which wave profiles must be computed. In portions of their parameter space, it can be difficult to provide initial guesses that will converge inside a BVP solver. In these cases numerical continuation can be used to construct each initial guess for the wave profile as parameter values vary.

Our stability analysis of traveling waves will depend on the numerical library STABLAB, implemented in MATLAB and more recently in Python. Good BVP solvers used in STABLAB include `bvp4c`, `bvp5c`, and `bvp6c`, which employ fourth, five, and sixth order collocation, respectively. Each of these solvers is implemented in MATLAB. The function `bvp6c` has recently been ported to Python; see [4].

Example 2.1. Slemrod's model for isentropic gas dynamics with capillarity in one spatial dimension has equations

$$\begin{aligned} v_t - u_x &= 0, \\ u_t + p(v)_x &= \left(\frac{u_x}{v}\right)_x - dv_{xxx}. \end{aligned} \tag{2.5}$$

Dependent variables v and u represent specific volume and velocity in Lagrangian coordinates, respectively. The function $p(v)$ is the pressure law for an adiabatic gas and $d \geq 0$ represents capillarity strength.

Traveling wave solutions $(u, v)(x, t) = (\hat{u}, \hat{v})(x - st)$ satisfy the profile equations

$$-sv' - u' = 0,$$
$$-su' + p(v)' = \left(\frac{u'}{v}\right)' - dv'''.$$

Rescaling with $(x, u) \to (-sx, -u/s)$ and substituting the first equation into the second equation yields the single scalar equation

$$v' + ap(v)' = \left(\frac{v'}{v}\right)' - dv'''$$

where $a = 1/s^2$.

Integrating from $-\infty$ to x we obtain a lower-order profile ODE

$$v - v_- + a(p(v) - p(v_-)) = \frac{v'}{v} - dv''. \tag{2.6}$$

The constant a determined by the Rankine-Hugoniot condition is found in limit as $x \to \infty$:

$$a = -\frac{v_+ - v_-}{p(v_+) - p(v_-)}.$$

Without loss of generality, and by a possible rescaling, we can assume that $0 < v_+ < v_- = 1$. The gas law used has the form $p(v) = v^{-\gamma}$, $\gamma \geq 1$ ([5, 15, 16]).

Letting $y = [v, v']^T$, (2.6) can be rewritten as a first order system $y' = F(y)$ where

$$F(y) = \begin{bmatrix} y_2 \\ \frac{1}{d}[y_2/y_1 + (1 - y_1) + a(1 - y_1^{-\gamma})] \end{bmatrix}.$$

Inspecting the eigenvalues of $dF(y_\pm)$ reveals that there is one projective condition at $+\infty$.

The ODE can be transformed to the interval $[0,\infty)$ by letting $z(x) = y(-x)$ represent the left half of the traveling wave. The resulting two-point boundary value problem has four dimensions and four boundary conditions—two matching conditions, one projective condition, and a phase condition:

$$\begin{aligned}
&\begin{bmatrix} y \\ z \end{bmatrix}' = \begin{bmatrix} F(y) \\ -F(z) \end{bmatrix}, \quad x \in [0, L], \quad y, z \in \mathbb{R}^2, \\
&y(0) = z(0), \\
&\Pi_U^+(y(+L) - y_+) = 0, \\
&y_1(0) = \frac{v_- + v_+}{2}.
\end{aligned} \tag{2.7}$$

The traveling wave becomes difficult to compute as capillarity strength d increases and large oscillations develop. Numerical continuation can be used to provide initial guesses in `bvp6c`; see Figures 2.1 and 2.2.

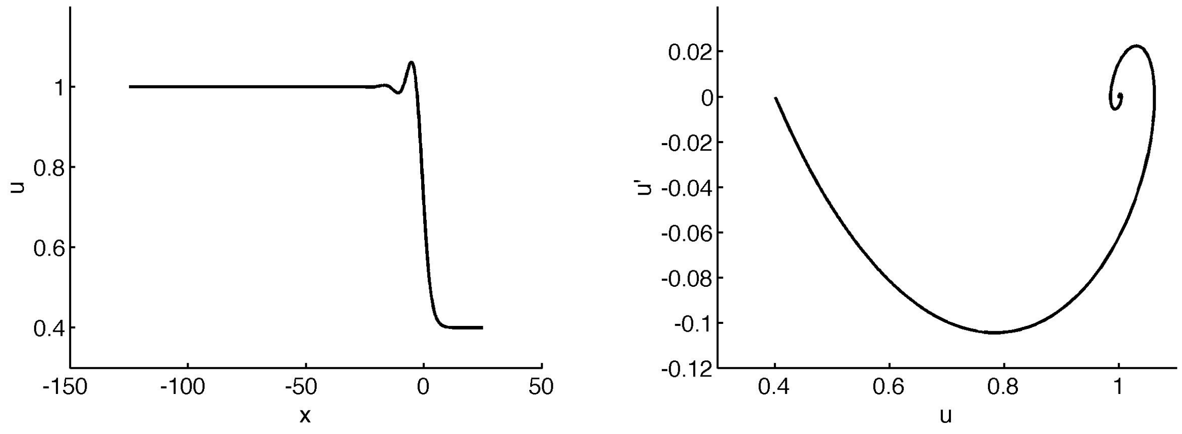


Figure 2.1: The wave profile for Slemrod's capillarity model, also shown in state space. The capillarity strength is $d = 2$.

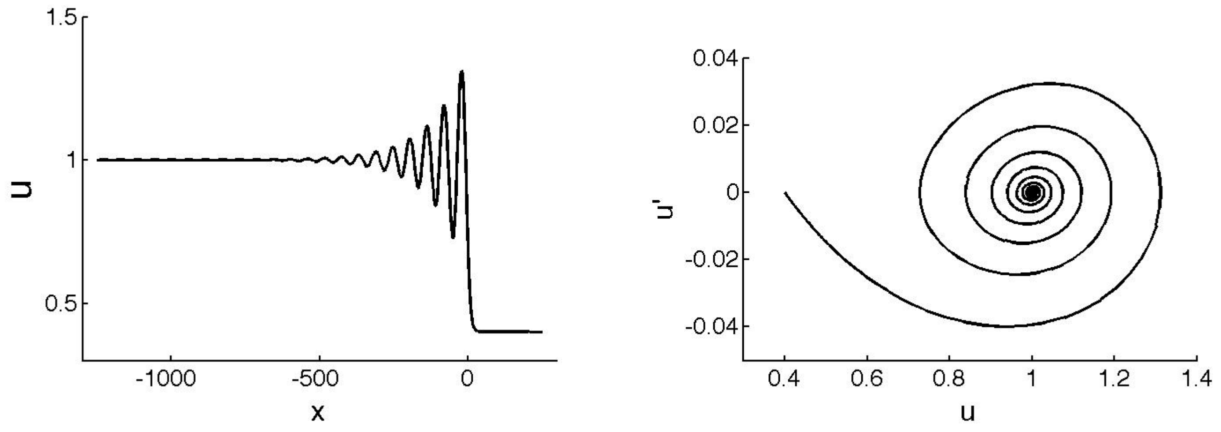


Figure 2.2: The wave profile for Slemrod's capillarity model, with capillarity strength $d = 60$.

Example 2.2. Here we describe the numerical solution of traveling waves in a model describing one-dimensional combustion of a fuel with a high Lewis number [11]. In nondimensional coordinates its equations are

$$\begin{aligned} u_t &= u_{xx} + y\Omega(u), \\ y_t &= \epsilon y_{xx} - \beta y\Omega(u). \end{aligned} \tag{2.8}$$

Dependent variables u and y are scaled temperatures and fuel concentration, respectively. The inverse Lewis number and exothermicity are represented by ϵ and β. The reaction rate is described by the function

$$\Omega(u) = \begin{cases} e^{-1/u} \text{ for } u > 0 \\ 0 \text{ otherwise.} \end{cases}$$

Equation (2.8) has traveling wave solutions

$$(u, y)(x, t) = (\hat{u}, \hat{y})(x - st) \tag{2.9}$$

with asymptotically constant end states $(u_-, y_-) = (1/\beta, 0)$, $(u_+, y_+) = (0, 1)$. At $x = -\infty$ the fuel has been burned and the heat maximized, while at $+\infty$ the fuel is unburned and we are at ambient temperature.

Substituting (2.9) into (2.8) leads to the profile ODEs

$$\begin{aligned} u'' + su' + y\Omega(u) &= 0, \\ \epsilon y'' + sy' - \beta y\Omega(u) &= 0. \end{aligned} \tag{2.10}$$

This can be rewritten as a first order system

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}' = \begin{bmatrix} y_2 \\ -sy_2 - y_3\Omega(y_1) \\ y_4 \\ \frac{1}{\epsilon}(-sy_4 + \beta y_3\Omega(y_1)) \end{bmatrix} \tag{2.11}$$

with end states

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}(-\infty) = \begin{bmatrix} \frac{1}{\beta} \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}(+\infty) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}. \tag{2.12}$$

where $[y_1, y_2, y_3, y_4]^T = [u, u', y, y']^T$. This system has four dimensions and five projective conditions, leading us to look for a further simplification.

From equations (2.10) it follows that

$$\beta u'' + \beta su' + \epsilon y'' + sy' = 0.$$

We then integrate from $-\infty$ to x find the conserved quantity

$$\beta u' + \beta su + \epsilon y' + sy = s,$$

or written differently,

$$y_3 = \frac{1}{s}\left[s - \beta s y_1 - \beta y_2 - \epsilon y_4\right]. \tag{2.13}$$

This conserved quantity allows to define $z_1 = y_1, z_2 = y_2, z_3 = y_4$, and reformulate the profile ODE as

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}' = \begin{bmatrix} z_2 \\ -sz_2 - \frac{1}{s}\left[s - \beta s z_1 - \beta z_2 - \epsilon z_3\right]\Omega(z_1) \\ \frac{1}{\epsilon}(-sz_3 + \frac{\beta}{s}\left[s - \beta s z_1 - \beta z_2 - \epsilon z_3\right]\Omega(z_1)) \end{bmatrix}, \tag{2.14}$$

which we may refer to simply as $z' = F(z)$. This new system has three projective conditions at $\pm\infty$, and one phase condition at $x = 0$. Doubling the dimension of the system leads to a dimension six BVP

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}' = \begin{bmatrix} F(Z_1) \\ -F(Z_2) \end{bmatrix}, \quad x \in [0, L], \quad Z_1, Z_2 \in \mathbb{R}^3, \tag{2.15}$$

with an additional three matching conditions $Z_1(0) = Z_2(0)$. Since the wave speed $s > 0$ is still unknown, the system is supplemented with an additional equation $s' = 0$. This results in a system with seven dimensions and seven boundary conditions.

Remark. This system highlights several possible difficulties involved with obtaining traveling wave profiles. First, it is important to verify that the number of boundary conditions matches the dimension of the system. The solution of traveling waves u and y depended on discovering a conserved quantity and reducing the dimension of the ODE. Second, the wave speed s is unknown, and must be found numerically by adding a supplementary ODE $s' = 0$. Third, the size of the computational domain $[-L, L]$ grows considerably as the exothermicity parameter β increases. It is helpful to rescale (2.15) to the interval $[0, 1]$ and to treat L as a tunable

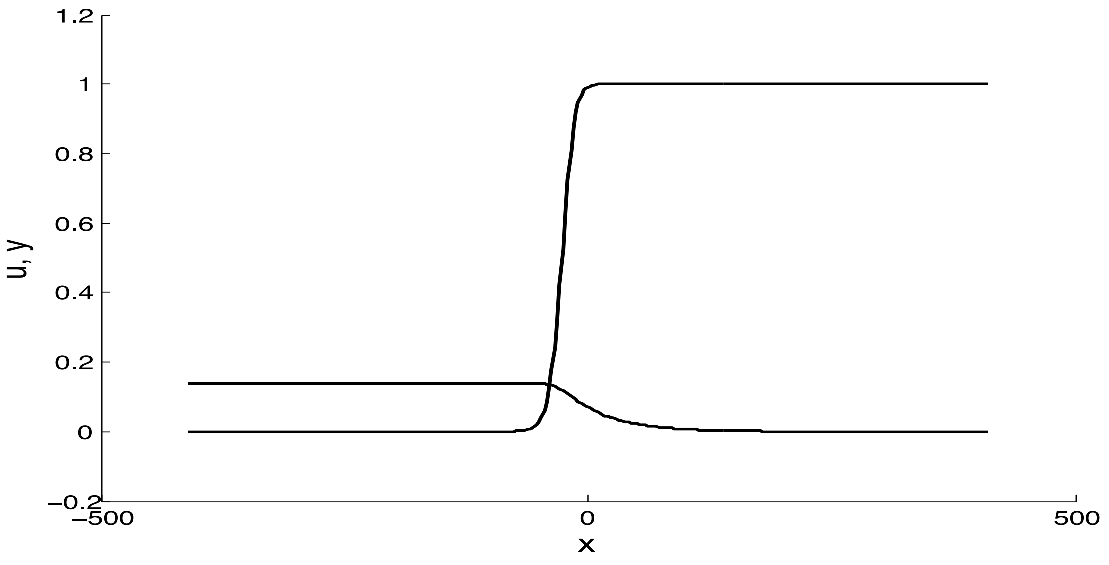


Figure 2.3: Traveling wave solutions $\hat{u}$ and $\hat{y}$.

parameter. In full, the BVP can be written

$$\begin{bmatrix} Z_1 \\ Z_2 \\ s \end{bmatrix}' = \begin{bmatrix} LF(Z_1) \\ -LF(Z_2) \\ 0 \end{bmatrix}, \quad x \in [0,1], \quad Z_1, Z_2 \in \mathbb{R}^3,$$
$$Z_1(0) - Z_2(0) = 0, \quad u(0) - \frac{u_- + u_+}{2} = 0,$$
$$\Pi_S^+(Z_1(1) - [0,0,0]^T) = 0, \quad \Pi_U^+(Z_2(1) - [1/\beta, 0, 0]^T) = 0. \tag{2.16}$$

2.2 Polar coordinate method

Let $U_-(\lambda)$ be the unstable manifold of $A_-(\lambda)$ and $S_+(\lambda)$ the stable manifold of $A_+(\lambda)$. Let $W_-(\lambda)$ and $W_+(\lambda)$ be $n \times k$ and $n \times (n-k)$ matrices whose columns form analytic bases for $U_-(\lambda)$ and $S_+(\lambda)$ at $x = \pm\infty$, respectively. We can find orthonormal matrices $k \times n$ and $n - k \times n$ matrices $\Omega_-(\lambda)$ and $\Omega_+(\lambda)$ with change-of-basis matrices $\alpha_-(\lambda)$ and $\alpha_+(\lambda)$ satisfying

$$W_\pm = \Omega_\pm \alpha_\pm.$$

This can be written in block matrix form as

$$\begin{bmatrix} W_- & W_+ \end{bmatrix} = \begin{bmatrix} \Omega_- & \Omega_+ \end{bmatrix} \begin{bmatrix} \alpha_- & 0 \\ 0 & \alpha_+ \end{bmatrix}.$$

Using this decomposition, the Evans function can be written

$$D(\lambda) = \det\left(\begin{bmatrix} W_- & W_+ \end{bmatrix}\right)\Big|_{x=0} = \gamma_-\ \gamma_+\ \det\left(\begin{bmatrix} \Omega_- & \Omega_+ \end{bmatrix}\right)\Big|_{x=0}, \tag{2.17}$$

where $\gamma_\pm = \det(\alpha_\pm)$.

2.2.1 Continuous orthogonalization.

Suppose $W = \Omega\alpha$ where $W, \Omega \in \mathbb{C}^{n\times k}$ and Ω is orthonormal. Using the Evans system we obtain

$$\Omega'\alpha + \Omega\alpha' = A\Omega\alpha.$$

Letting $B = \alpha'\alpha^{-1}$ leads to the system of equations

$$\begin{aligned} \Omega' &= A\Omega - \Omega B, \\ \alpha' &= B\alpha. \end{aligned} \tag{2.18}$$

The dimension of (2.18) is greater than the original Evans system, and requires additional constraints. These can be found by using the orthonormality of Ω,

$$\begin{aligned} 0 = I' = (\Omega^*\Omega)' &= (\Omega^*)'\Omega + \Omega^*\Omega', \\ &= (\Omega^* A^* - B^*\Omega^*)\Omega + \Omega^*(A\Omega - \Omega B), \\ &= \Omega^*(A^* + A)\Omega - B^*\Omega^*\Omega - \Omega^*\Omega B, \\ &= \Omega^*(A^* + A)\Omega - B^* - B. \end{aligned}$$

When system (2.18) is initialized with $\Omega_\infty^*\Omega_\infty = I$, it turns out that $\Omega^*(A^* + A)\Omega - B^* -$

$B = 0$ is a necessary and sufficient condition for orthonormality, since $(\Omega^*\Omega)' = 0$ implies that $\Omega^*\Omega$ is constant. This allows a variety of choices for B. Notable choices include Drury's method ($B = \Omega^* A\Omega$) and Davey's method ($B = (\Omega^*\Omega)^{-1}\Omega^* A\Omega$). Drury's method can also be derived by setting $\Omega^*\Omega' = 0$, so that the change in Ω is orthogonal to the space spanned by Ω.

Substituting $B = \Omega^* A\Omega$ in (2.18), we arrive at the equations

$$\begin{aligned} \Omega' &= (I - \Omega\Omega^*)A\Omega, \\ \alpha' &= (\Omega^* A\Omega)\alpha. \end{aligned} \tag{2.19}$$

Using Abel's equation, (2.19) results in the scalar equation

$$\gamma' = \operatorname{trace}(\Omega^* A\Omega)\gamma$$

where $\gamma = \det(\alpha)$.

2.2.2 Computation of $\gamma(x)$.

The condition that $\Omega^*\Omega' = 0$ states that all change in Ω must occur orthogonal to the span of Ω; the exponential growth once seen in the evolution of W is now seen in the evolution of γ. To deal with this, we replace the exponentially growing γ with the rescaled

$$\tilde{\gamma}_\pm(x) := \gamma_\pm e^{-\operatorname{trace}(\Omega^* A\Omega)_\pm x}.$$

This results in

$$\tilde{\gamma}_\pm'(x) = \operatorname{trace}(\Omega^* A\Omega - (\Omega^* A\Omega)_\pm)\tilde{\gamma}_\pm. \tag{2.20}$$

We note that $\gamma(0)$ can be replaced with $\tilde{\gamma}(0)$ in (2.17). The rescaled variable $\tilde{\gamma}$ is essential for good numerical results, and has the form

$$\begin{aligned}\tilde{\gamma}_-(x) &= e^{\int_{-\infty}^{x} \operatorname{trace}(\Omega^* A \Omega(x) - \Omega_-^* A_- \Omega_-)\, ds} \tilde{\gamma}_-(-\infty),\\ \tilde{\gamma}_+(x) &= e^{\int_{x}^{+\infty} -\operatorname{trace}(\Omega^* A \Omega(x) - \Omega_+^* A_+ \Omega_+)\, ds} \tilde{\gamma}_+(+\infty).\end{aligned}$$

After making the substitution $\theta_-(x) = \int_{-\infty}^{x} \operatorname{trace}(\Omega^* A\Omega(x) - \Omega_-^* A_- \Omega_-)\, ds$, and defining $\theta_+(x)$ similarly, we see that

$$\tilde{\gamma}_\pm(0) = e^{\theta_\pm(0)} \tilde{\gamma}_\pm(\pm\infty), \tag{2.21}$$

where $\theta_\pm(x)$ satisfies the IVP

$$\begin{aligned}&\theta_\pm' = \operatorname{trace}(\Omega^* A \Omega - (\Omega^* A \Omega)_\pm),\\ &\theta_\pm(\pm\infty) = 0.\end{aligned} \tag{2.22}$$

If a different initial condition is used in (2.22), then the resulting Evans output will differ by a constant multiplicative factor.

2.2.3 Post-processing with Kato. The method of Kato allows us to compute an analytically varying basis $R_\pm(\lambda)$ for the growth manifold of $A_-(\lambda)$ and the decay manifold of $A_+(\lambda)$. Given any orthogonal decomposition $\Omega(\pm\infty, \lambda)$ of the linear span of the column vectors of $R_\pm(\lambda)$, there exists a change of basis matrix $\tilde{\alpha}(\pm\infty, \lambda)$, so that

$$R_\pm(\lambda) = \Omega(\pm\infty, \lambda)\tilde{\alpha}(\pm\infty, \lambda).$$

Thus the appropriate initial value of $\tilde{\gamma}$ is

$$\begin{aligned}\tilde{\gamma}(\pm\infty) &= \det(\tilde{\alpha}(\pm\infty,\lambda)),\\ &= \det\left(\Omega^*(\pm\infty,\lambda)R_\pm(\lambda)\right).\end{aligned} \tag{2.23}$$

We note that $\Omega(\pm\infty,\lambda)$ is not analytic in λ; the burden of analyticity is carried by $R_\pm(\lambda)$.

The Evans function $D(\lambda)$ arises from the numerical solution of the initial value problem

$$\begin{aligned}&\Omega'(x) = (I - \Omega(x)\Omega^*(x))A(x,\lambda)\Omega(x),\\ &\theta'(x) = \mathrm{trace}(\Omega^*(x)A(x,\lambda)\Omega(x) - (\Omega^* A(\lambda)\Omega)_\pm),\\ &\theta(\pm L) = 0, \quad \Omega(\pm L) = \Omega(\pm\infty,\lambda),\end{aligned} \tag{2.24}$$

defined on $[-L,0]$, $[0,L]$ for $L>0$ sufficiently large. The Evans function is then given by

$$D(\lambda) = \tilde{\gamma}(-\infty)\tilde{\gamma}(+\infty)e^{\theta_-(0)}e^{\theta_+(0)}\det\left(\begin{bmatrix}\Omega_- & \Omega_+\end{bmatrix}\right)\Big|_{x=0}, \tag{2.25}$$

where $\tilde{\gamma}(\pm\infty)$ is computed by post-processing with Kato; see equation (2.23).

2.3 The method of Kato

Recall that the Evans function is defined as

$$D(\lambda) = \det\left(\begin{bmatrix}W_-(x;\lambda) & W_+(x;\lambda)\end{bmatrix}\right)$$

where $W_\pm(x;\lambda)$ are matrices whose columns form analytically varying bases for the unstable and stable manifolds $U_-(\lambda)$ and $S_+(\lambda)$ of $A_-(x;\lambda)$ and $A_+(x;\lambda)$, respectively. Furthermore, $W_\pm(x;\lambda)$ can be made to vary analytically by computing analytic bases $\{r_1,\ldots,r_k\}$ and $\{r_{k+1},\ldots,r_n\}$ for the unstable and stable eigenspaces of $A_-(\lambda)$ and $A_+(\lambda)$. This problem can be solved using the method of Kato; we will describe several numerical schemes introduced by Zumbrun [29] that are based on Kato's reduced ODE.

For a projection P that is analytic in a domain D of the complex plane, it is a well known result in ODE theory that one can find a basis $\{r_1(\lambda), \dots, r_k(\lambda)\}$ for the range of $P(\lambda)$ that is analytic throughout D. The matrix $R(\lambda) = [r_1(\lambda), \dots, r_k(\lambda)]$ is a solution of Kato's ODE

$$\begin{aligned} R' &= (P'P - PP')R, \quad ' = \frac{d}{d\lambda} \\ R(\lambda_0) &= R_0, \end{aligned} \tag{2.26}$$

where the columns of $R_0 = [r_1(\lambda_0), \dots, r_k(\lambda_0)]$ form some initial basis for Range $P(\lambda_0)$; see [23].

Proposition 2.3. *Let $' := \frac{d}{d\lambda}$ and $[A, B] := AB - BA$ be the commutator of A and B. Consider the ODE*

$$\begin{aligned} S' &= [P', P]S, \\ S(\lambda_0) &= I. \end{aligned} \tag{2.27}$$

From standard linear ODE theory, we note that there is a solution S of (2.27) *and it exists throughout the simply connected domain D. Moreover S satisfies*

$$(S^{-1}PS)(\lambda_0) = P(\lambda_0),$$

thus defining a globally analytic coordinate change that takes the range of $P(\lambda)$ to that of $P(\lambda_0)$.

Proof. Differentiating $P^2 = P$, and multiplying on the right and on the left by P, shows that $PP'P = 0$. Differentiating $I = SS^{-1}$ and substituting (2.27) in the resulting expression, we see that S^{-1} satisfies $(S^{-1})' = -S^{-1}[P', P]$. Finally, differentiating $S^{-1}PS$ we obtain

$$\begin{aligned}(S^{-1}PS)' &= (S^{-1})'PS + S^{-1}P'S + S^{-1}PS', \\ &= -S^{-1}[P', P]PS + S^{-1}P'S + S^{-1}P[P', P]S, \\ &= S^{-1}[-(P'P - PP')P + P' + P(P'P - PP')]S, \\ &= S^{-1}[-P'P + P' - PP']S, \\ &= 0\end{aligned}$$

□

Remark. An analytically varying basis $\{r_j(\lambda)\}$ for $P(\lambda)$ may then be taken from the columns of $R(\lambda) = S(\lambda)R_0$, where $R_0 =$ range P_0. Thus the analytic basis vectors $r_j(\lambda)$ satisfy

$$r_j'(\lambda) = [P'(\lambda), P(\lambda)]r_j(\lambda), \quad r_j(\lambda_0) = r_j^0. \tag{2.28}$$

Let $R(\lambda) = [r_1(\lambda) \dots r_k(\lambda)]$, and note that by the previous results R satisfies $R' = [P', P]R$. Then a first order approximation of this differential equation is

$$\frac{R(\lambda_{j+1}) - R(\lambda_j)}{\lambda_{j+1} - \lambda_j} \approx \left[\frac{P(\lambda_{j+1}) - P(\lambda_j)}{\lambda_{j+1} - \lambda_j}, P(\lambda_j)\right] R(\lambda_j)$$

Rearranging, we obtain the first order scheme

$$\begin{aligned}R_{j+1} &= R_j + [(P_{j+1} - P_j)P_j - P_j(P_{j+1} - P_j)]R_j, \\ R_{j+1} &= [I + P_{j+1}P_j - P_jP_{j+1}]R_j.\end{aligned}$$

Finally, to stabilize the scheme we evaluate the RHS with P_{j+1}, ensuring that R_{j+1} is in the range of P_{j+1}:

$$\begin{aligned}R_{j+1} &= P_{j+1}[I + P_{j+1}P_j - P_jP_{j+1}]R_j, \\ &= P_{j+1}[I + P_j(I - P_{j+1})]R_j.\end{aligned}$$

A reduced version of Kato's ODE is the initial value problem

$$\begin{aligned} R' &= P'R, \\ R(\lambda_0) &= R_0. \end{aligned} \tag{2.29}$$

By standard linear ODE theory, (2.29) has a unique solution that exists throughout the simply connected domain D. The solution $R(\lambda)$ can then be found numerically. An interesting property of Kato's ODE is that $PR' = 0$, with the direction of change in the basis R is orthogonal to its span.

Proposition 2.4. *The unique solution R of (2.29) satisfies*

(i) $PR = R$,

(ii) $PR' = 0$,

(iii) $R' = (P'P - PP')R$.

Proof.

(i)

$$\begin{aligned} (PR - R)' &= P'R + PR' - R', \\ &= P'R + PP'R - P'R, \\ &= PP'R - PP'PR, \\ &= -PP'(PR - R). \end{aligned}$$

Since $(PR - R)(\lambda_0) = 0$, by uniqueness of solutions $PR - R = 0$.

(ii) $PR' = PP'R = PP'PR = 0$.

(iii) $R' = P'R = P'PR = (P' - PP')R = (P'P - PP')R$.

□

2.3.1 The reduced Kato ODE. The reduced Kato ODE is given by

$$\begin{aligned} R' &= P'R, \\ R(\lambda_0) &= R_0. \end{aligned} \tag{2.30}$$

This equation is a simplification of (2.27) that is particularly useful in constructing difference schemes for $R(\lambda)$.

A first order approximation to (2.30) is

$$\frac{R(\lambda_{j+1}) - R(\lambda_j)}{\lambda_{j+1} - \lambda_j} = \frac{P(\lambda_{j+1}) - P(\lambda_j)}{\lambda_{j+1} - \lambda_j} R(\lambda_j).$$

This approximation leads to a simple first order explicit difference scheme

$$\begin{aligned} R_{j+1} &= R_j + P_{j+1}R_j - P_jR_j, \\ &= P_{j+1}R_j. \end{aligned}$$

We now consider a second order discretization of (2.30). Let $\triangle\lambda_j := \lambda_{j+1} - \lambda_j$. We can use second order approximations

$$\begin{aligned} R_{j+1} - R_j &\approx \triangle\lambda_j P'_{j+1/2} R_{j+1/2}, \\ R_{j+1/2} &\approx P_{j+1/2} R_j, \\ P'_{j+1/2} &\approx (P_{j+1} - P_j)/\triangle\lambda_j, \\ P_{j+1/2} &\approx \frac{1}{2}(P_{j+1} + P_j), \end{aligned}$$

to build the scheme

$$\begin{aligned}
\frac{R_{j+1}-R_j}{\triangle\lambda_j} &= P'_{j+1/2}R_{j+1/2}, \\
\frac{R_{j+1}-R_j}{\triangle\lambda_j} &= \frac{P_{j+1}-P_j}{\triangle\lambda_j}P_{j+1/2}R_j, \\
R_{j+1}-R_j &= \frac{1}{2}(P_{j+1}-P_j)(P_{j+1}+P_j)R_j, \\
R_{j+1} &= \frac{1}{2}(P_{j+1}+P_j-P_jP_{j+1}+P_{j+1}P_j)R_j.
\end{aligned}$$

After stabilizing this scheme by evaluating the right hand side at P_{j+1}, we obtain

$$R_{j+1} = P_{j+1}\left(I+\frac{1}{2}P_j(I-P_{j+1})\right)R_j. \tag{2.31}$$

Note that this second order method from the reduced Kato ODE is a relaxation of the first order method from the original ODE; this method is commonly used for serious computations (see [29]).

Chapter 3. Root following techniques

Consider an evolutionary system

$$u_t = \mathcal{F}(u, u_x, u_{xx}, \ldots) \tag{3.1}$$

with traveling wave solutions $\hat{u}$ depending on a system parameter μ. As μ varies the eigenvalues of the linearized system $\lambda v = \mathcal{L}v$ can travel from the left half of the complex plane to the right half-plane and back again. Since the spectral stability of $\hat{u}$ is determined by the presence or absence of the point spectrum of $\mathcal{L}$ in the right half-plane, we will discuss how to detect when an eigenvalue has moved into the right half-plane, and how to follow it as it varies with system parameter μ.

The Evans function $D(\lambda)$ is an analytic function whose zeros match the eigenvalues of the linearized eigenvalue problem in both location and multiplicity. We can leverage the analyticity of the Evans function in rootfinding methods such as Newton's method or the secant method. Section 3.1 discusses how the argument principle and winding number calculations can be used to locate roots of the Evans function.

Eigenvalues of $\mathcal{L}$ correspond to nontrivial bounded solutions W of the Evans system augmented with an equation for λ,

$$\begin{pmatrix} W \\ \lambda \end{pmatrix}' = \begin{pmatrix} A(x;\lambda,\mu) & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} W \\ \lambda \end{pmatrix}, \quad W \in \mathbb{C}^n. \tag{3.2}$$

For an eigenvalue λ its corresponding 'eigenfunction' W is a homoclinic orbit in the phase space of the Evans system. Section 3.2 describes a root following technique that formulates λ and W as functions of μ, and uses continuation to compute (λ, W) as μ varies; see [19]. Throughout this chapter we will assume that the traveling wave solution U has already been computed numerically.

3.1 The argument principle

Roots of the Evans function can be detected using the Argument Principle from complex analysis. Suppose f is an analytic function defined on a simple domain containing a closed positively oriented curve C. If f is nonzero on C, then the number of zeros of f inside C is the winding number of $f(C)$ about 0, given by

$$W = \frac{1}{2\pi i}\int_C \frac{f'(z)}{f(z)}\,dz.$$

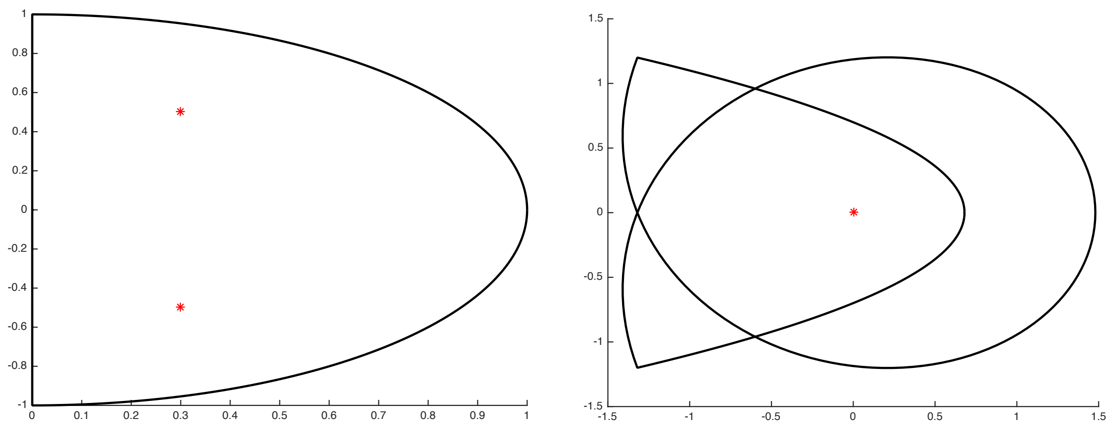
The Argument Principle can be further generalized to the Method of Moments; see [7, 11, 2]. If f is nonzero on C with roots $z_1, \ldots, z_n$ inside C, then the *p-th moment of f about* z^* is given by

$$M_p(z^*) = \frac{1}{2\pi i}\oint_\Gamma \frac{(z-z^*)^p f'(z)}{f(z)} = \sum_{k=1}^{n}(z_k - z^*)^p. \tag{3.3}$$

The zeroth moment of f about 0 is its winding number W, while the first and second moments give the sum and the sum of squares of the roots, respectively. By calculating the moments of f with a reliable quadrature rule, we can calculate the position of any roots within C. This can only be done with low order moments, since calculating the roots of a polynomial from its coefficients is an ill-conditioned problem.

3.1.1 An efficient algorithm for computing the winding number. Suppose f is an analytic function, and nonzero on a contour C. If $f(C)$ does not intersect the branch cut of $\log(z)$, then $\int_C f'(z)/f(z)\,dz = \log(f(z_1)) - \log(f(z_0))$ where z_0 is the initial point of C and z_1 the terminal point. If $f(C)$ does pass through the branch cut of $\log(z)$, then $\int_C f'/f$ can be approximated by integrating over subcontours C_j for which $f(C_j)$ does not intersect the branch cut of $\log(z)$. As the images $f(C_j)$ approach the branch cut, the sum of their integrals tends toward $\int_C f'/f$.

Since $\log(z) = \log|z| + i\arg(z)$, for a closed contour C the integral $\int_C f'/f$ counts how



(a) A contour C in the right half-plane. **(b)** The image contour $f(C)$.

Figure 3.1: Roots of a function f in some region can be found by calculating the winding number of a contour about that region. Here we consider the function $f(z) = 2(z - .3 - .5i)(z - .3 + .5i)$ about a semicircular contour with radius 1 in the right half-plane.

often and with what orientation $f(C)$ crosses the branch cut. As an example, consider the function $f(z) = 2(z - .3 - .5i)(z - .3 + .5i)$ about a semicircular contour C with radius 1 in the right half-plane, with C centered at the origin. Since $f(C)$ crosses the branch cut twice in the counterclockwise direction, we have $\int_C f'/f = 2\pi i + 2\pi i$ with winding number two; see Figure 3.1.

This algorithm for computing the winding number is more accurate and efficient than using integration. Once an eigenvalue has been detected, a form of the bisection method can be used to get additional accuracy. Essentially the winding number is computed for rectangular regions, which are further subdivided to obtain the required accuracy.

3.2 Continuation of the eigenvalue and eigenfunction λ, W

3.2.1 The boundary value problem. Consider the BVP (3.2) defined on $(-\infty, \infty)$. The eigenfunction W satisfies $n-k$ projective conditions at $-\infty$ and k projective conditions at $+\infty$. Because of the scaling invariance of the eigenfunction, an additional scaling condition is required to fix the size of W at $x = 0$. This leads to a $n+1$ dimensional BVP with $n+1$ boundary conditions. To continue in (λ, W) as μ varies, we begin by reformulating the

$n+1$ dimensional BVP (3.2) defined on $(-\infty,\infty)$, into a $2n+1$ dimensional two-point BVP defined on $[0,\infty)$. Letting $Y(x)=W(-x)$, we obtain

$$\begin{pmatrix} W \\ Y \\ \lambda \end{pmatrix}' = \begin{pmatrix} A(x;\lambda,\mu) & 0 & 0 \\ 0 & -A(-x;\lambda,\mu) & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} W \\ Y \\ \lambda \end{pmatrix}, \quad x \in [0,\infty). \tag{3.4}$$

The boundary conditions corresponding to (3.4) consist of n matching conditions, n projective conditions at $x=\infty$, and one phase condition to eliminate the scaling invariance in the eigenfunction solution. Letting $P_+(\lambda,\mu)$ be the projection onto the unstable manifold of $A_+(\lambda,\mu)$ and $P_-(\lambda,\mu)$ the projection onto the stable manifold of $A_-(\lambda,\mu)$, the matching and projective conditions can be implemented on a truncated interval $[0,L]$ by

$$\begin{pmatrix} W(0) \\ P_+(\lambda,\mu)W(L) \\ P_-(\lambda,\mu)Y(L) \end{pmatrix} = \begin{pmatrix} Y(0) \\ 0 \\ 0 \end{pmatrix}, \tag{3.5}$$

where $L>0$ is large enough to capture the asymptotic behavior of the eigenfunction. The phase condition can be imposed at $x=0$ by fixing one coordinate of W or Y.

3.2.2 Constructing an initial guess.

For an initial system parameter μ, suppose an eigenvalue $\lambda \in \sigma(\mathcal{L})$ and bases $W_-(x;\lambda)$ and $W_+(x;\lambda)$ have been computed for the unstable and stable manifolds of the Evans system at $x=-\infty$ and $x=+\infty$. Since the Evans function $D(\lambda)=0$ we can find a vector $\mathbf{c}$ where

$$\mathbf{c} = \begin{pmatrix} c_- \\ c_+ \end{pmatrix} \in \text{Null}\left(\begin{bmatrix} W_-(x=0;\lambda) & W_+(x=0;\lambda) \end{bmatrix}\right) \tag{3.6}$$

This can be done in several ways, such as least squares or the singular-value decomposition. From (3.6) we have $W_-(0;\lambda)c_- + W_+(0;\lambda)c_+ = 0$, which allows us to construct the eigenfunction $W(x)$ and obtain the following initial guess for the BVP:

$$\begin{pmatrix} -W_+(x;\lambda)c_+ \\ W_-(x;\lambda)c_- \\ \lambda \end{pmatrix}. \tag{3.7}$$

Once an initial guess is constructed for μ and the BVP is solved, the parameter can be updated to $\mu + \Delta\mu$ and the most recent BVP solution used as an initial guess in the new BVP.

Evolving $W_\pm(x;\lambda)$ from $x = \pm\infty$ to $x = 0$ is a numerically stiff problem, so we employ the polar-coordinate method to find our eigenfunction W; see [20]. Let $\Omega_-(x;\lambda)$ and $\Omega_+(x;\lambda)$ be matrices of orthonormal vectors spanning $W_-(x;\lambda)$ and $W_+(x;\lambda)$, respectively. Then there are coordinate frames $\alpha_-(x;\lambda)$ and $\alpha_+(x;\lambda)$ such that

$$W_\pm(x;\lambda) = \Omega_\pm(x;\lambda)\alpha_\pm(x;\lambda). \tag{3.8}$$

Thus manifolds $W_\pm(x;\lambda)$ are constructed by evolving $(\Omega_\pm(\lambda), \alpha_\pm(\lambda))$ from $x = \pm\infty$ to 0 according to the ODE

$$\begin{aligned} \Omega' &= (I - \Omega\Omega^*)A\Omega, \\ \alpha' &= (\Omega^* A\Omega)\alpha, \end{aligned} \tag{3.9}$$

described more fully in Section 2.2.1. Plugging $W_\pm(x;\lambda)$ into (3.7) gives us our initial guess.

Prior to continuing (λ, W) in μ, it is good practice to solve (3.4) with the initial guess (3.7) for the initial parameter μ to refine the estimate on the (λ, W) pair. Because errors in computing (3.6) result in a discontinuity in the estimated eigenfunction, this fine-tuning provides a better initial guess.

3.3 Experiments

Here we apply our root following method to three different models. The first two models are canonical examples within the Evans function literature, because their traveling wave profiles have analytic expressions. We then apply our method to a combustion model whose traveling wave profile is difficult to compute—in fact, it is necessary to use numerical continuation to compute both the wave profile and the eigenvalue-eigenfunction pair.

Our computations have been carried out using the numerical library STABLAB. STABLAB was developed by Humpherys et al [4] in MATLAB, and implements the Evans function utilities described in this text. There are several two-point BVP solvers implemented in MATLAB that can be used for computing wave profiles and eigenfunctions. We have used `bvp6c`, a sixth order extension of the Lobatto method implemented in `bvp4c`; see [12] for details. Our absolute and relative tolerances have been set to 10^{-8} and 10^{-6}, respectively. The basic functionality of STABLAB and `bvp6c` has recently been ported to Python, where these experiments have also been verified.

For each of these systems, as the traveling wave becomes unstable we compute the eigenvalue-eigenfunction pair using the Drury method of continuous orthogonalization; see Section 2.2.1. We have used `ode45`, a MATLAB-implementation of the Dormand–Prince Runge–Kutta algorithm [8], with the same settings for the absolute and relative tolerances as in `bvp6c`. After an estimate of the eigenvalue-eigenfunction pair has been found, the estimate is refined using the BVP formulation (3.4). The BVP then allows us to continue in (λ, W) as system parameters vary.

It is interesting to note that this system of root tracking allows us to follow the roots of the combustion model beyond where they were able to be followed in [11] using regular Evans function computation.

3.3.1 The "good" Boussinesq equation.

Here we examine solitary wave solutions of the "good" Boussinesq equation

$$u_{tt} = u_{xx} - u_{xxxx} - (u^2)_{xx}. \tag{3.10}$$

Making the ansatz $u(x,t) = \hat{u}(x - st)$, where s is the wave speed, it can be shown that these solutions of Boussinesq equation have the form

$$\hat{u}(x,t) = \frac{3}{2}(1-s^2)\,\mathrm{sech}^2\left(\frac{\sqrt{1-s^2}}{2}(x-st)\right). \tag{3.11}$$

These waves are stable for $\frac{1}{2} \le |s| < 1$ and unstable for $0 < |s| < 1/2$; see [1, 20].

Translating (3.10) into the moving frame $(x,t) \to (x - st, t)$ the solitary waves (3.11) become stationary. By linearizing the PDE about its steady-state solution $\hat{u}$ we obtain the eigenvalue problem

$$\lambda^2 u - 2s\lambda u' = (1-s^2)u'' - u'''' - (2\hat{u}u)''. \tag{3.12}$$

This can be written as a first-order system $W'(x) = A(x;\lambda,s)W(x)$ where

$$A(x;\lambda,s) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\lambda^2 - 2\hat{u}_{xx} & 2\lambda s - 4\hat{u}_x & (1-s^2) - 2\hat{u} & 0 \end{pmatrix} \quad \text{and} \quad W = \begin{pmatrix} u \\ u' \\ u'' \\ u''' \end{pmatrix}. \tag{3.13}$$

Our procedure begins at $s = 0.48$ with locating the unstable eigenvalue λ after it has crossed into the right half-plane. Once λ has been found, the eigenfunction W is found using the polar coordinate method introduced in (2.19).

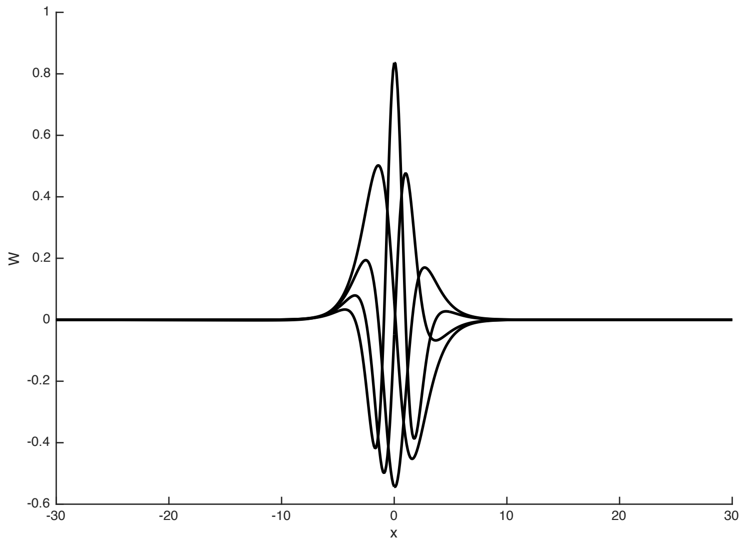


Figure 3.2: The eigenfunction W associated with the eigenvalue λ of the Boussinesq equation for $s = 0.48$. W was first approximated using the polar coordinate method, which provides an initial guess that BVP (3.4) refines further.

3.3.2 The generalized Korteweg-de Vries equation.

Here we examine solitary wave solutions of the generalized Korteweg-de Vries equation (gKdV)

$$u_t + u_{xxx} + (1/p)(u^p)_x = 0, \quad p \geq 2. \tag{3.14}$$

Making the substitution $u(x,t) = \hat{u}(x - st)$ into the PDE, we can find wave solutions of the form

$$\hat{u}(x,t) = \left(\frac{p(p+1)}{2} \operatorname{sech}^2 \left(\frac{1-p}{2}(x - st) \right) \right)^{1/(p-1)}. \tag{3.15}$$

These solutions are known to be stable when $p < 5$ and unstable when $p > 5$; see [26].

Translating (3.14) into the moving frame $(x,t) \to (x - st, t)$, the solitary waves become stationary. The eigenvalue problem is then obtained by linearizing the PDE about the steady state $\hat{u}$:

$$\lambda u - su' + u''' + (\hat{u}^{p-1}u)' = 0. \tag{3.16}$$

This problem can be further simplified by making the substitution $u \to u'$, and integrating

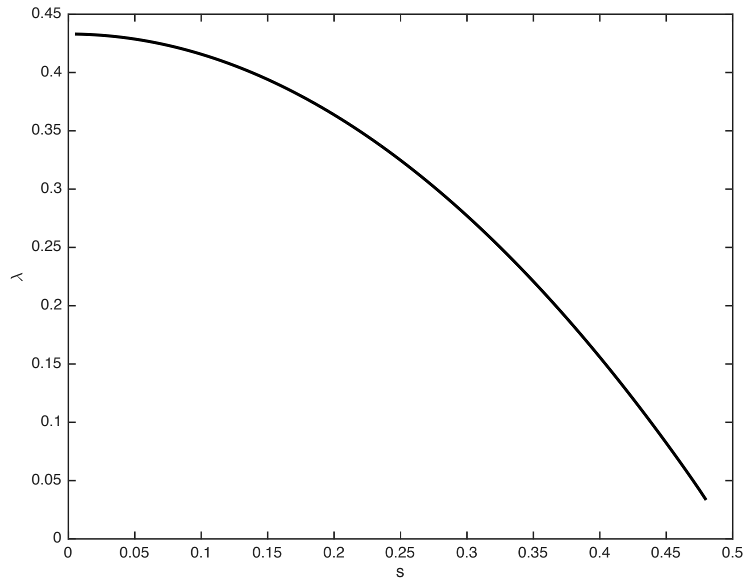


Figure 3.3: The unstable eigenvalue λ of the "good" Boussinesq equation, graphed as a function of the wave speed s for $0.005 \leq s \leq 0.48$. These eigenvalues were followed numerically using the continuation approach developed in Section 3.2.

from $-\infty$ to x to get

$$\lambda u - su' + u''' + \hat{u}^{p-1}u' = 0. \tag{3.17}$$

In these integrated coordinates the eigenvalue problem has the exact same spectrum, with the sole exception that the eigenvalue at the origin has been removed; see [5, 18] and their references for more information. This has the effect that the Evans function can be computed on contours that pass through the origin. The eigenvalue problem can be rewritten as a first order linear system $W'(x) = A(x; \lambda, p)W(x)$ where

$$A(x; \lambda, p) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\lambda & s - \hat{u}^{p-1} & 0 \end{pmatrix} \quad \text{and} \quad W = \begin{pmatrix} u \\ u' \\ u'' \end{pmatrix}. \tag{3.18}$$

As p increases and passes through 5, a single real eigenvalue crosses from the left half plane into the right half plane thus signaling the onset of instability. At $p = 5.2$ we find the unstable eigenvalue at $\lambda = 0.098$ by plotting $D(\lambda)$ as a function of $\lambda > 0$ and inspecting the

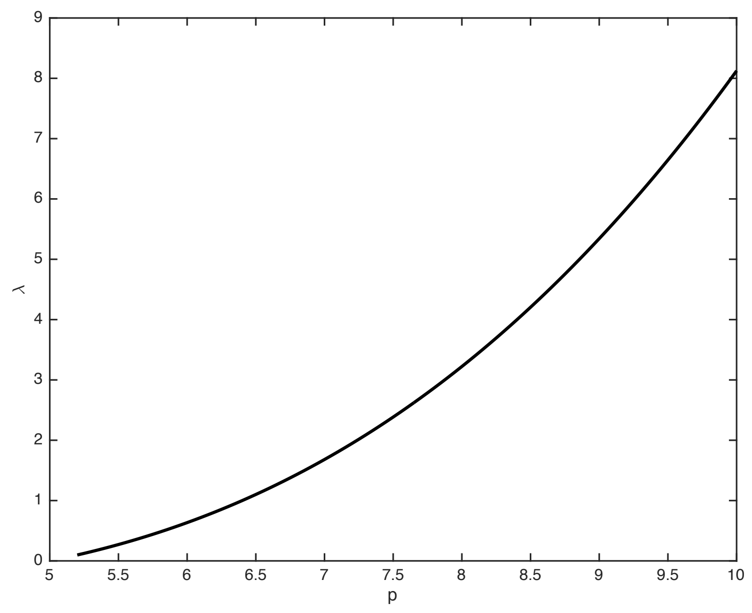


Figure 3.4: The unstable eigenvalues λ of the gKdV equation, graphed as a function of p for $5.2 \leq p \leq 10$. These eigenvalues were followed numerically using the continuation approach developed in Section 3.2.

root on the positive real axis. We find the eigenvalue-eigenfunction pair at $p = 5.2$ as before by using the polar-coordinate method and then shoring up the eigenvalue-eigenfunction pair, by solving (3.4) with the previous solution as the initial guess. We then continue in λ via (3.4) and trace out the resulting values of λ as p varies from 5.2 to 10; see Figure 3.4 for a graph of the output.

Finally, we compare the speed of the root following method by comparing it with root-finding the Evans function via the secant method. In Table 3.1, we show that the root following method is faster when compared to the same accuracy.

3.3.3 High Lewis number combustion. To test this method on a more challenging problem, we examine a reaction diffusion model that describes the evolution of combustion

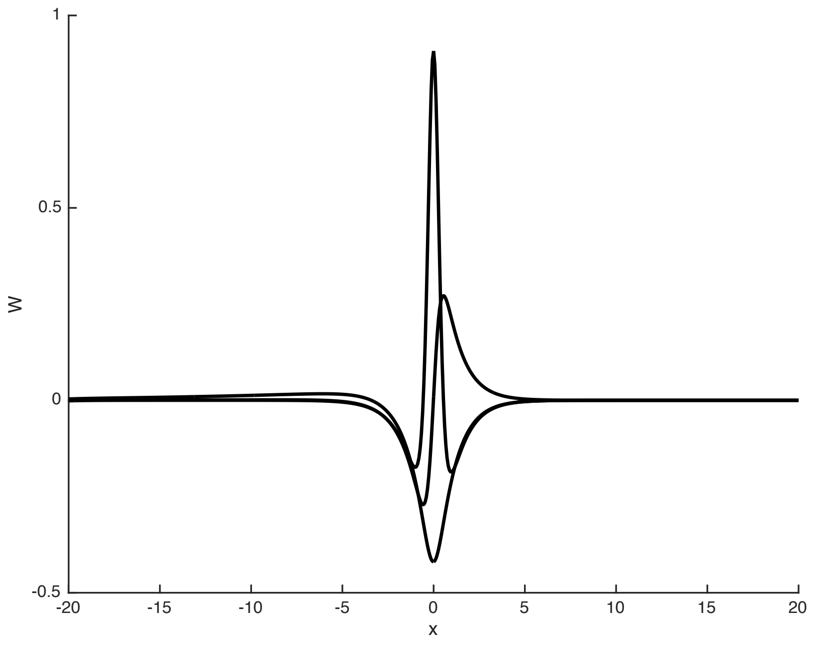


Figure 3.5: The eigenfunction W associated with the eigenvalue λ of the gKdV equation for $p = 5.2$. The eigenfunction W was first approximated using the polar coordinate method, to obtain an initial guess for BVP (3.4) to refine the solution.

waves in one spatial dimension. Consider the partial differential equation

$$\begin{aligned} u_t &= u_{xx} + y\Omega(u), \\ y_t &= \varepsilon y_{xx} - \beta y \Omega(u), \end{aligned} \tag{3.19}$$

where $u = u(x,t)$ and $y = y(x,t)$ denote, respectively, the scaled temperature and concentration of the fuel. We denote $\varepsilon = 1/\mathrm{Le} \geq 0$ as the reciprocal of the Lewis number, which represents the ratio of the fuel diffusivity to the heat diffusivity, $\beta > 0$ as the exothermicity, which is the ratio of the activation energy to the heat of the reaction, and $\Omega(u)$ as the ignition function

$$\Omega(u) = \begin{cases} e^{-1/u} \text{ for } u > 0, \\ 0 \text{ otherwise}, \end{cases}$$

which is a smooth Arrhenius law, where the reaction starts at the ambient temperature $u = 0$.

	secant method		continuation
Δp	ave	steps	ave
0.1	0.94 sec	8.9	0.59 sec
0.05	0.97 sec	8.3	0.41 sec
0.01	0.72 sec	7.3	0.27 sec
0.005	0.71 sec	7.2	0.25 sec

Table 3.1: We compare average computation time for two methods, Newton's method (secant) and our continuation method, of tracking the roots of the Evans function for the gKdV equation throughout the parameter space $5.2 \leq p \leq 10$. We also include the average number of secant iterations required for each value of p.

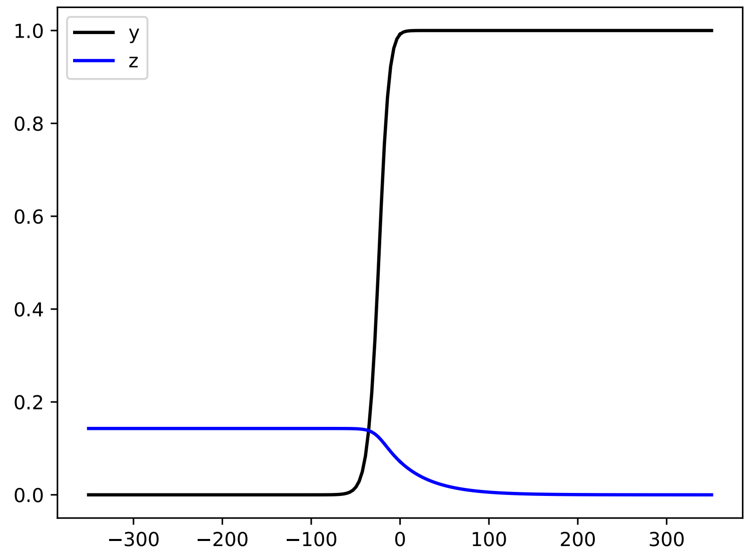


Figure 3.6: Wave profiles for u and y, the scaled temperature and concentration of the fuel, for parameter values $\beta = 7$, $\epsilon = 0.1$.

Traveling wave solutions of (3.19) may be found by substituting $(u, y)(x, t) = (\hat{u}, \hat{y})(x-st)$ into (3.19) and solving the resulting system of equations

$$\begin{aligned} \hat{u}'' + s\hat{u}' + \hat{y}\Omega(\hat{u}) &= 0, \\ \varepsilon\hat{y}'' + s\hat{y}' - \beta\hat{y}\Omega(\hat{u}) &= 0. \end{aligned} \tag{3.20}$$

We note that s represents the (unknown) speed of the traveling wave, and traveling wave solutions must satisfy boundary conditions $(\hat{u}, \hat{y})(-\infty) = (1/\beta, 0)$, $(\hat{u}, \hat{y})(\infty) = (0, 1)$, and $(\hat{u}_x, \hat{y}_x)(\pm\infty) = (0, 0)$.

By substituting $z_1 = u$, $z_2 = u'$, $z_3 = y$, and $z_4 = y'$ and using the conserved quantity $\beta u' + \beta s u + \varepsilon y' + s y = s$, we simplify (3.20) to the three dimensional first-order system given by $U' = f(U)$, where

$$U = \begin{pmatrix} z_1 \\ z_2 \\ z_4 \end{pmatrix} \quad \text{and} \quad f(U) = \begin{pmatrix} z_2 \\ -sz_2 - \left(1 - \beta z_1 - \frac{\beta}{s} z_2 - \frac{\varepsilon}{s} z_4\right) \Omega(z_1) \\ \frac{1}{\varepsilon}\left[-sz_4 + \beta\left(1 - \beta z_1 - \frac{\beta}{s} z_2 - \frac{\varepsilon}{s} z_4\right)\Omega(z_1)\right] \end{pmatrix}, \tag{3.21}$$

with boundary conditions

$$U_- := U(-\infty) = \begin{pmatrix} \frac{1}{\beta} \\ 0 \\ 0 \end{pmatrix} \qquad \text{and} \qquad U_+ := U(+\infty) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

We remark that it is non-trivial to compute the traveling wave profiles for large values of exothermicity. The typical approach is to compute the profile in a less challenging parameter regime and then continue the profile to the desired parameters; see [11] for details.

By transforming (3.19) into a moving frame coinciding with (3.20) and (3.21), the traveling wave becomes stationary, and thus we can linearize about the steady-state solution, thus arriving at the eigenvalue problem

$$\begin{aligned} \lambda u &= u_{xx} + s u_x + e^{-1/\hat{u}} y + \hat{y}\hat{u}^{-2} e^{-1/\hat{u}} u, \\ \lambda y &= \varepsilon y_{xx} + s y_x - \beta e^{-1/\hat{u}} y - \beta \hat{y}\hat{u}^{-2} e^{-1/\hat{u}} u, \end{aligned} \tag{3.22}$$

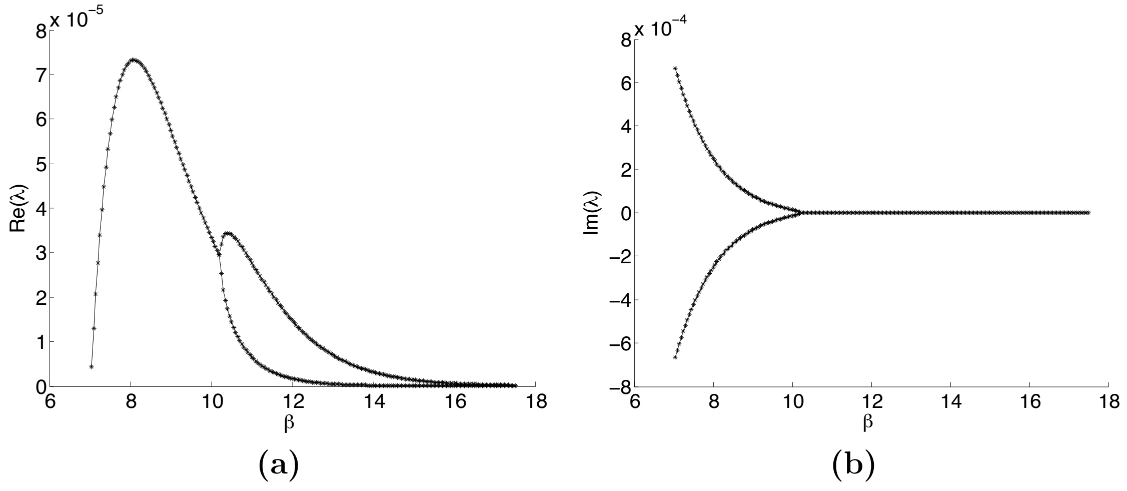


Figure 3.7: The unstable eigenvalues found using the continuation method for fronts in the system (3.19) for $7.05 \leq \beta \leq 17.5$ and $\varepsilon = 0.1$.

which can be written as a first-order system $W'(x) = A(x;\lambda,\beta)W(x)$, where

$$A(x;\lambda,\beta) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \lambda + \hat{y}\hat{u}^{-2}e^{-1/\hat{u}} & -s & -e^{-1/\hat{u}} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{\beta}{\varepsilon}\hat{y}\hat{u}^{-2}e^{-1/\hat{u}} & 0 & \frac{1}{\varepsilon}(\lambda + \beta e^{-1/\hat{u}}) & -\frac{s}{\varepsilon} \end{pmatrix} \quad \text{and} \quad W = \begin{pmatrix} u \\ u' \\ y \\ y' \end{pmatrix}.$$

A spectral analysis of wave fronts in this system has been recently studied in [11]. The wave fronts are known to be unstable as the exothermicity parameter $\beta > 0$ gets large. In this case a complex conjugate pair of eigenvalues has been viewed crossing into the right-half plane at about $\beta = 7.03$. Thus we locate the eigenvalues using the method of moments given in Equation (3.3) and described in [7, 2, 11]. We then find the corresponding eigenfunctions using the method described in Section 3.2. Finally we shore up the eigenvalue-eigenfunction pair by recomputing via (3.4). We are then ready to continue the pair in β to trace out the path of the root of the Evans function and thus the corresponding unstable eigenvalues of the problem. As β continues to increase from $\beta = 7.03$ to $\beta = 17.5$, the eigenvalues join and then split along the reals, and begin heading toward the origin; see Figure 3.7 and Table 3.2 for details.

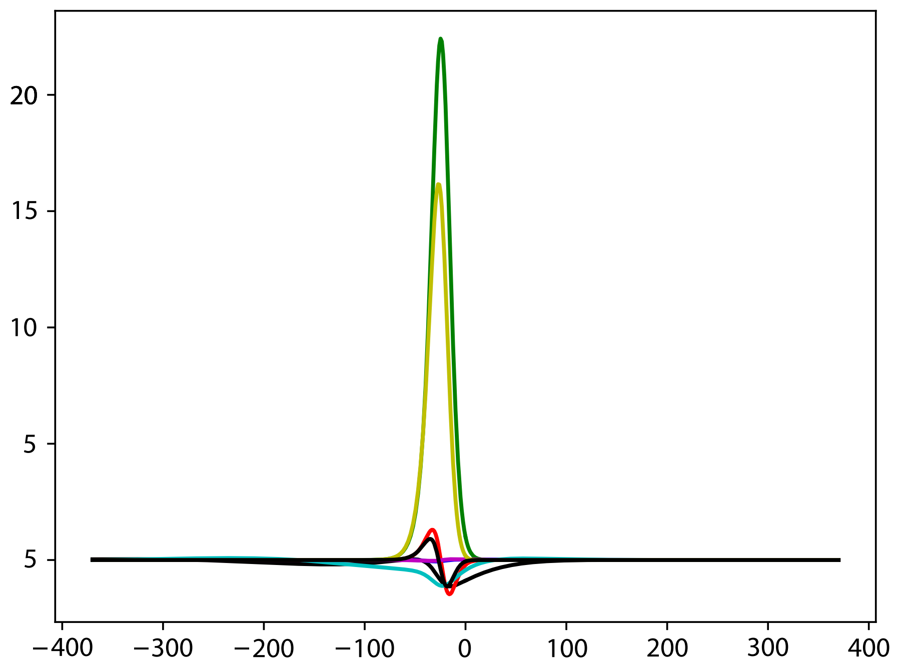


Figure 3.8: This is a graph of the eigenfunction $W(x)$ associated with the unstable eigenvalue $\lambda = (1.3\text{e-}5 + i6.32\text{e-}4)$ for $\beta = 7.1$, $\epsilon = 0.1$.

In [11], the eigenvalues were tracked in β by partitioning the admissible region of the right-half plane into increasingly small cells and integrating along their boundaries. With the continuation method, root following is quick and essentially automatic with only a little effort at the point where the eigenvalues collide. This is remedied by perturbing the initial guesses with positive and negative bump functions to put the initial guesses into different basins of attraction.

We note that in [11], there was difficulty resolving the roots of the Evans function beyond $\beta = 14.1$ using root location methods. By contrast, with the continuation method presented here, we were able to go past $\beta = 17$. This suggests that the continuation method presented here may be a good general approach for exploring the unstable spectrum in parameter regions that are too extreme for Evans function computation.

β	λ_1	λ_2	L	Max Residual
8	7.3e-5 + i2.5e-4	7.3e-5 - i2.5e-4	631	2.5e-10
9	5.8e-5 + i7.9e-5	5.8e-5 - i7.9e-5	1084	6.2e-10
10	3.3e-5 + i1.4e-5	3.3e-5 - i1.4e-5	1860	1.4e-9
11	2.8e-5	6.5e-6	3191	3.6e-9
12	1.5e-5	1.7e-6	5477	7.3e-9
13	7.0e-6	4.8e-7	9398	9.8e-9
14	3.2e-6	1.5e-7	16127	9.2e-9
15	1.4e-6	4.5e-8	27674	9.7e-9
16	6.0e-7	1.4e-8	47488	4.5e-9
17	2.5e-7	4.6e-9	81490	8.9e-9

Table 3.2: Eigenvalues for the combustion equation, with the error returned from `bvp6c`. The exponentially growing numerical domain $[-L, L]$ illustrates the difficulty in computing the wave profiles.

Chapter 4. Detonations in the multi-D reactive Navier-Stokes equations

Figure 4.1: The first of a series of three detonations during Operation Sailor Hat in 1965. 500 tons of TNT were detonated to simulate the effects of a nuclear explosion. Public domain image.

4.1 The reactive Navier-Stokes equations in one dimension

Most models for detonations regard viscous effects as negligible during the combustion process. Thus detonations are usually investigated mathematically with the Zel'ldovich-von Neumann-Döring (ZND) or the inviscid reactive Euler equations. The program of determining the spectral stability of inviscid detonation waves was begun by Erpenbeck in the 1960s [10, 9]. Here we review recent work done by Barker, Humpherys, Lyng, and Zumbrun [3] on the stability of viscous detonations in the reactive Navier-Stokes equations in one spatial dimension. They found important differences caused by introducing viscosity. Indeed, when viscosity is present unstable eigenvalues return to the left half of the complex plane as activation energy increases.

The equations for a one dimensional reaction following Navier-Stokes are

$$\begin{aligned} \tau_t - u_x &= 0, \\ u_t + p_x &= \left(\frac{\mu u_x}{\tau}\right)_x, \\ \left(e + \frac{u^2}{2}\right)_t + (pu)_x &= \left(\frac{\mu u u_x}{\tau} + \frac{\kappa T_x}{\tau}\right)_x + qk\phi(T)z, \\ z_t &= -k\phi(T)z + \left(\frac{\beta z_x}{\tau^2}\right)_x. \end{aligned} \tag{4.1}$$

Dependent variables $\tau, u, e,$ and z represent specific volume, velocity, internal energy, and the mass fraction of the reactant. The "viscous effects" modeled by the RNS system, and ignored by standard ZND studies of detonations, are controlled with parameters $\mu, \kappa,$ and β. These constants signify viscosity, heat conductivity, and species diffusivity, respectively. Constants q and $k > 0$ represent collision frequency and the energy difference between the reactant and the product. Energy e satisfies the ideal gas law, $e = c_v T$. The ignition function ϕ serves as an on/off switch for the reaction, and has the Arrhenius form

$$\phi(T) = \begin{cases} \exp(-E_A/[c_v(T - T_{\text{ig}})]) & \text{if } T \geq T_{\text{ig}}, \\ 0 & \text{otherwise.} \end{cases}$$

A traveling wave solution $(\tau, u, e, z)(x,t) = (\hat{\tau}, \hat{u}, \hat{e}, \hat{z})(x - st)$ of (4.1) is a steady state solution of

$$\begin{aligned} \tau_t - s\tau_x - u_x &= 0, \\ u_t - su_x + p_x &= \left(\frac{\mu u_x}{\tau}\right)_x, \\ \left(e + \frac{u^2}{2}\right)_t - s\left(e + \frac{u^2}{2}\right)_x + (pu)_x &= \left(\frac{\mu u u_x}{\tau} + \frac{\kappa T_x}{\tau}\right)_x + qk\phi(T)z, \\ z_t - sz_x &= -k\phi(T)z + \left(\frac{\beta z_x}{\tau^2}\right)_x. \end{aligned} \tag{4.2}$$

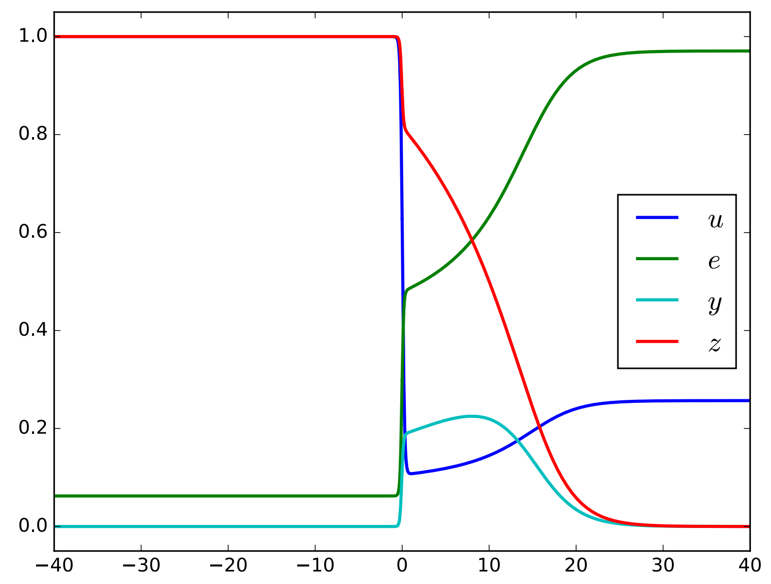


Figure 4.2: RNS wave profiles in Lagrangian coordinates with $E_A = 7.1$.

Substituting $\nu = \kappa/c_v$ and rescaling by

$$\begin{aligned}
&(x,t) \to \left(\frac{\tau_+ s x}{L}, \frac{\tau_+ s^2 t}{L}\right),\\
&(\tau, u, e) \to \left(\frac{\tau L}{\tau_+}, \frac{uL}{\tau_+ s}, \frac{eL^2}{\tau_+^2 s^2}\right),\\
&(q, k, \beta, E_A) \to \left(\frac{qL^2}{\tau_+^2 s^2}, \frac{kL}{\tau_+ s^2}, \frac{\beta L}{\tau_+}, \frac{E_A L^2}{\tau_+^2 s^2}\right),
\end{aligned} \tag{4.3}$$

leads to the PDE

$$\begin{aligned}
\tau_t - \tau_x - u_x &= 0,\\
u_t - u_x + p_x &= \left(\frac{\mu u_x}{\tau}\right)_x,\\
\left(e + \frac{u^2}{2}\right)_t - \left(e + \frac{u^2}{2}\right)_x + (pu)_x &= \left(\frac{\mu u u_x}{\tau} + \frac{\nu e_x}{\tau}\right)_x + qk\phi(e)z,\\
z_t - z_x &= -k\phi(e)z + \left(\frac{\beta z_x}{\tau^2}\right)_x.
\end{aligned} \tag{4.4}$$

By rescaling and using Galilean invariance, we may assume that $s = 1$ and $\tau_+ = 1$. Then

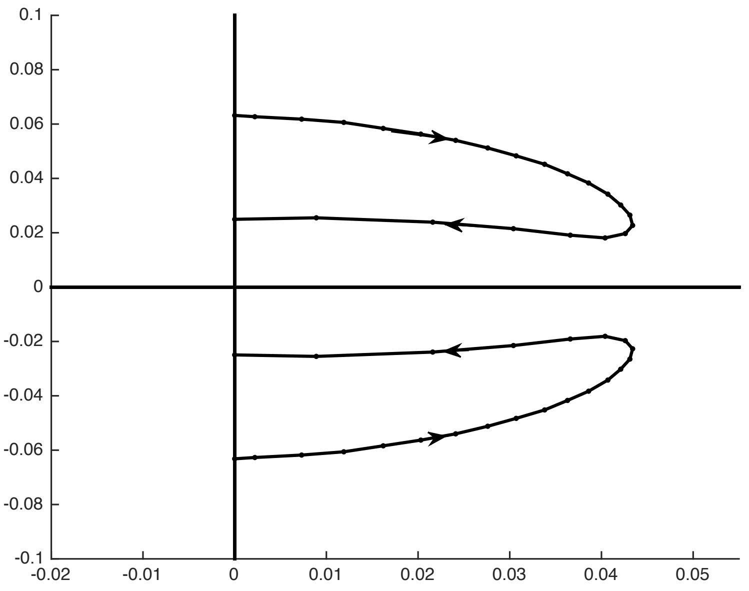


Figure 4.3: The path of the smallest pair of eigenvalues in the right half-plane, as activation energy E_A increases from 2.8 to 6.8. The eigenvalues enter the right half-plane at about $\Im(\lambda) = 0.063i$, and then eventually return and restabilize with increasing activation energy.

the traveling wave profiles satisfy

$$\begin{aligned} \tau' &= -u', \\ -u' + p' &= \mu \left(\frac{u'}{\tau} \right)', \\ -(e + u^2/2)' + (pu)' &= \left(\frac{\mu u u'}{\tau} + \frac{\nu e'}{\tau} \right)' + qk\phi(e)z, \\ -z' &= -k\phi(e)z + \left(\frac{\beta z'}{\tau^2} \right)'. \end{aligned} \tag{4.5}$$

We can also assume that $u_+ = 0$. We then proceed by integrating from x to $+\infty$, obtaining

$$\begin{aligned} \tau' &= -\frac{1}{\mu} \left[\tau(\tau - 1) + \Gamma(e - e_+\tau) \right], \\ e' &= -\frac{\tau}{\nu} \left[-\frac{1}{2}(\tau - 1)^2 + (e - e_+) + \Gamma e_+(\tau - 1) + q(y + z - 1) \right], \\ y' &= k\phi(e)z - \beta^{-1} y \tau^2, \\ z' &= \beta^{-1} y \tau^2, \end{aligned} \tag{4.6}$$

where we have used $\tau = 1 - u$ and introduced the flux variable $y = \frac{\beta z'}{\tau^2}$. These equations can be solved numerically to produce the wave profiles graphed in Figure 4.2.

4.2 The reactive Navier-Stokes equations in multiple dimensions

The one dimensional scalar equations consider instabilities in the direction of the motion of the wave. Spectral analysis in multiple dimensions must also consider instabilities in directions transverse to the direction of motion. We begin by introducing the general dimension d reactive Navier-Stokes (RNS) equations, then reduce to two dimensions to work in a concrete setting. Since Lagrangian coordinates involves adding variables to the multidimensional RNS equations, with a concomitant array of unnecessary pure imaginary essential spectra, our computations will use the Eulerian coordinates and notation used in [27]; see Table 4.1.

Our spatial directions will be given by $(y_1, \ldots, y_d)$. For a reacting fluid with a one-step exothermic reaction in d dimensions, the RNS equations are given in Eulerian coordinates as

$$\rho_t + \operatorname{div}(\rho \mathbf{u}) = 0, \tag{4.7a}$$

$$(\rho u_j)_t + \operatorname{div}(\rho u_j \mathbf{u}) + p_{y_j} = \delta \Delta u_j + \omega \operatorname{div}\big(\mathbf{u}_{y_j}\big), \quad j = 1, \ldots d, \tag{4.7b}$$

$$\begin{aligned}(\rho \tilde{E})_t + \operatorname{div}\left[(\rho \tilde{E} + p)\mathbf{u}\right] =& \Delta\big(\kappa T + \delta \frac{|\mathbf{u}|^2}{2}\big) + \delta \operatorname{div}\big((\nabla \mathbf{u})\mathbf{u}\big) \\ &+ (\omega - \delta) \operatorname{div}((\operatorname{div} \mathbf{u})\mathbf{u}) + \operatorname{div}\left(q\rho\beta\nabla z\right),\end{aligned} \tag{4.7c}$$

$$(\rho z)_t + \operatorname{div}(\rho z \mathbf{u}) = \operatorname{div}\left(\rho\beta\nabla z\right) - k\rho z \varphi(T). \tag{4.7d}$$

This system has dimension $(d+3) \times (d+3)$ and dependent variables $(\rho, \mathbf{u}, T, z)$. In (4.7c) $\nabla \mathbf{u}$ refers to the Jacobian matrix of the velocity vector with respect to the spatial variables.

ρ	density
p	pressure
$\mathbf{u} = (u_1, \ldots, u_d)^{\mathrm{tr}}$	fluid velocity
T	temperature
$\tilde{e}$	specific internal energy
z	mass fraction of reactant
ω, δ	viscosity coefficients
κ	heat conductivity
β	species diffusion
k	reaction rate
q	heat release

Table 4.1: RNS variables and parameters

Our variables and parameters are labelled in Table 4.1. We write

$$\tilde{E} = \tilde{e} + \frac{|\mathbf{u}|^2}{2}, \quad \tilde{e} = e + qz.$$

Letting the pressure p and energy e be given by the ideal gas law,

$$p = p_0(\rho, T) = R\rho T, \quad e = e_0(T) = c_v T,$$

the reactive Navier-Stokes equations in two spatial dimensions can be written as

$$\rho_t + (\rho u)_{x_1} + (\rho v)_{x_2} = 0, \tag{4.8a}$$

$$(\rho u)_t + (\rho u^2 + p)_{x_1} + (\rho uv)_{x_2} = (2\mu + \eta)u_{x_1x_1} + \mu u_{x_2x_2} + (\mu + \eta)v_{x_1x_2}, \tag{4.8b}$$

$$(\rho v)_t + (\rho uv)_{x_1} + (\rho v^2 + p)_{x_2} = \mu v_{x_1x_1} + (2\mu + \eta)v_{x_2x_2} + (\mu + \eta)u_{x_2x_1}, \tag{4.8c}$$

$$\begin{aligned}(\rho\tilde{E})_t &+ (\rho u\tilde{E} + up)_{x_1} + (\rho v\tilde{E} + vp)_{x_2} \\ &= \left(\kappa T_{x_1} + (2\mu + \eta)uu_{x_1} + \mu v(v_{x_1} + u_{x_2}) + \eta uv_{x_2}\right)_{x_1} \\ &\quad + \left(\kappa T_{x_2} + (2\mu + \eta)vv_{x_2} + \mu u(v_{x_1} + u_{x_2}) + \eta vu_{x_1}\right)_{x_2} \\ &\quad + (q\rho\beta z_{x_1})_{x_1} + (q\rho\beta z_{x_2})_{x_2},\end{aligned} \tag{4.8d}$$

$$(\rho z)_t + (\rho u z)_{x_1} + (\rho v z)_{x_2} = (\rho\beta z_{x_1})_{x_1} + (\rho\beta z_{x_2})_{x_2} - k\rho z\varphi(T), \tag{4.8e}$$

where $\mathbf{u} = (u, v)$, and $\delta = \mu$, $\omega = \mu + \eta$. We then subtract the reaction equation from the energy equation to obtain

$$\rho_t + (\rho u)_{x_1} + (\rho v)_{x_2} = 0, \tag{4.9a}$$

$$(\rho u)_t + (\rho u^2 + p)_{x_1} + (\rho uv)_{x_2} = (2\mu + \eta)u_{x_1x_1} + \mu u_{x_2x_2} + (\mu + \eta)v_{x_1x_2}, \tag{4.9b}$$

$$(\rho v)_t + (\rho uv)_{x_1} + (\rho v^2 + p)_{x_2} = \mu v_{x_1x_1} + (2\mu + \eta)v_{x_2x_2} + (\mu + \eta)u_{x_2x_1}, \tag{4.9c}$$

$$\begin{aligned}(\rho E)_t &+ (\rho uE + up)_{x_1} + (\rho vE + vp)_{x_2} \\ &= (\kappa T_{x_1} + (2\mu + \eta)uu_{x_1} + \mu v(v_{x_1} + u_{x_2}) + \eta uv_{x_2})_{x_1} \\ &\quad + (\kappa T_{x_2} + (2\mu + \eta)vv_{x_2} + \mu u(v_{x_1} + u_{x_2}) + \eta vu_{x_1})_{x_2} \\ &\quad + qk\rho z\varphi(T),\end{aligned} \tag{4.9d}$$

$$(\rho z)_t + (\rho u z)_{x_1} + (\rho v z)_{x_2} = (\rho\beta z_{x_1})_{x_1} + (\rho\beta z_{x_2})_{x_2} - k\rho z\varphi(T), \tag{4.9e}$$

where $E = \tilde{E} - qz = e + u^2/2 + v^2/2$.

This system of PDEs is invariant under the rescaling

$$\begin{aligned}&(x_1, x_2, t; \rho, u, v, T, z) \to \Big(mx_1, mx_2, \epsilon m^2 t; \epsilon\rho, \frac{u}{\epsilon m}, \frac{v}{\epsilon m}, \frac{T}{\epsilon^2 m^2}, z\Big), \\ &\beta \to \frac{\beta}{\epsilon}, \quad q \to \frac{q}{\epsilon^2 m^2}, \quad k \to \frac{k}{\epsilon m^2}, \quad E_A \to \frac{E_A}{\epsilon^2 m^2}.\end{aligned} \tag{4.10}$$

We note that the pressure and energy laws $p = R\rho T$ and $e = c_v T$ are preserved after rescaling. Our ignition function will have Arrhenius form

$$\varphi(T) = \begin{cases} \exp(-E_A/[c_v(T - T_{\text{ig}})]) & \text{if } T \geq T_{\text{ig}}, \\ 0 & \text{otherwise,} \end{cases}$$

where E_A is the activation energy and T_{ig} is the ignition temperature. We will also write the ignition function as a function of e, with $\check{\varphi}(e) = \varphi(T)$.

4.3 Wave profiles

We now proceed to look for a traveling wave solution

$$(\rho, u, v, e, z)(x_1, x_2, t) = (\hat{\rho}, \hat{u}, \hat{v}, \hat{e}, \hat{z})(x_1 - st).$$

By Galilean invariance, we can consider standing profiles ($s = 0$), which satisfy

$$(\rho u)' = 0, \tag{4.11a}$$

$$(\rho u^2)' + p' = (2\mu + \eta)u'', \tag{4.11b}$$

$$(\rho uv)' = \mu v'', \tag{4.11c}$$

$$(\rho uE + up)' = \kappa e''/c_v + (2\mu + \eta)(uu')' + \mu(vv')' + qk\rho z\check{\varphi}(e), \tag{4.11d}$$

$$(\rho uz)' = (\rho\beta z')' - k\rho z\check{\varphi}(e), \tag{4.11e}$$

obtained by setting t and x_2 derivatives in (4.9) equal to zero. Note that ρu is constant, and let $m := \rho u$. The third equation simplies to $mv' = \mu v''$, whose only bounded solutions are constant v. By a possible coordinate change we may assume $v = 0$.

Let $\epsilon = 1/\rho_-$; then in the rescaled coordinates $\rho_- = u_- = 1$, $\rho u = 1$, and (4.11) still holds by invariance of (4.9). Thus we obtain the system

$$u' + p' = (2\mu + \eta)u'', \tag{4.12a}$$

$$E' + (up)' = \kappa e''/c_v + (2\mu + \eta)(uu')' + qk\rho z\check{\varphi}(e), \tag{4.12b}$$

$$z' = (\rho\beta z')' - k\rho z\check{\varphi}(e). \tag{4.12c}$$

Substitute (4.12c) into (4.12b) to get

$$u' + p' = (2\mu + \eta)u'',$$

$$E' + (up)' = \kappa e''/c_v + (2\mu + \eta)(uu')' + q(\rho\beta z')' - qz',$$

$$z' = (\rho\beta z')' - k\rho z\check{\varphi}(e).$$

Integrating from $-\infty$ to x yields

$$(u - u_-) + (p - p_-) = (2\mu + \eta)u', \tag{4.14a}$$

$$(E - E_-) + (up - u_-p_-) = \kappa e'/c_v + (2\mu + \eta)uu' + q\rho\beta z' - q(z - z_-), \tag{4.14b}$$

$$z' = (\rho\beta z')' - k\rho z\check{\varphi}(e). \tag{4.14c}$$

Setting $\nu = \kappa/c_v$ and using $E = e + u^2/2$, we can substitute the first equation into the second to obtain

$$u' = (2\mu + \eta)^{-1}((u - u_-) + (p - p_-)),$$

$$(e - e_-) - (u - u_-)^2/2 + p_-(u - u_-) = \nu e' + q\rho\beta z' - q(z - z_-),$$

$$z' = (\rho\beta z')' - k\rho z\check{\varphi}(e).$$

Since $u_- = \rho_- = z_- = 1$, after applying the pressure law $p = \Gamma\rho e$ the system becomes

$$u' = (2\mu + \eta)^{-1}\Big((u - 1) + \Gamma(\rho e - e_-)\Big),$$

$$e' = \nu^{-1}\Big((e - e_-) - (u - 1)^2/2 + \Gamma e_-(u - 1) - q\rho\beta z' + q(z - 1)\Big),$$

$$z' = (\rho\beta z')' - k\rho z\check{\varphi}(e).$$

By setting $y = -\rho\beta z'$ and using $\rho = 1/u$, we obtain the first order system

$$u' = (2\mu + \eta)^{-1}\Big((u-1) + \Gamma(u^{-1}e - e_-)\Big), \tag{4.17a}$$

$$e' = \nu^{-1}\Big((e - e_-) - (u-1)^2/2 + \Gamma e_-(u-1) + q(y + z - 1)\Big), \tag{4.17b}$$

$$y' = \beta^{-1}uy - ku^{-1}z\check{\varphi}(e), \tag{4.17c}$$

$$z' = -\beta^{-1}uy. \tag{4.17d}$$

4.4 Rankine-Hugoniot conditions

The Rankine-Hugoniot conditions differ slightly from those found in the dimension one case because our multidimensional wave has been oriented toward the left. We note that when $q = 0$ the profile equations (4.17) reduce to the Navier-Stokes profile equations, with $e_- < e_+$; see [17]. This corresponds to end states $z_- = 1$, $z_+ = 0$, and a leftward moving traveling wave. This reorientation of the wave has been done to better match up the construction of the Evans function for the multidimensional RNS case with the work done on the multidimensional nonreactive Navier-Stokes equations in [17]. Taking the limit of (4.17) as $x \to +\infty$, we obtain equations

$$(u_+ - 1) + \Gamma(u_+^{-1}e_+ - e_-) = 0, \tag{4.18a}$$

$$(e_+ - e_-) - (u_+ - 1)^2/2 + \Gamma e_-(u_+ - 1) + q(y_+ + z_+ - 1) = 0, \tag{4.18b}$$

$$\beta^{-1}u_+y_+ - ku_+^{-1}z_+\check{\varphi}(e_+) = 0, \tag{4.18c}$$

$$-u_+y_+ = 0. \tag{4.18d}$$

Since $y_+ = -\beta z_+'/u_+ = 0 = z_+$, equations (4.18c) and (4.18d) are trivially satisfied.

Solving (4.18a) for e_+ and substituting into (4.18b) allows us to parametrize e_+ and u_+

as functions of e_-, Γ, and q, where

$$e_+ = u_+ e_- + \Gamma^{-1} u_+ (1 - u_+), \tag{4.19a}$$

$$u_+ = \frac{(\Gamma+1)(\Gamma e_- + 1) - \sqrt{(\Gamma+1)^2(\Gamma e_- + 1)^2 - \Gamma(\Gamma+2)(1 + 2e_-(\Gamma+1) + 2q)}}{\Gamma + 2}. \tag{4.19b}$$

The negative square root above is the parameter regime for strong detonations.

4.5 Computation of the traveling wave

We numerically compute the wave profiles on the interval $[0,1]$. The detonation can be rescaled from $[-L, L]$ to $[0,1]$ by first flipping the left half of the wave to $[0, L]$ on the positive x-axis, and then rescaling to the interval $[0,1]$. If we rewrite (4.17) as

$$Y' = f(Y),$$

then the system we solve on $[0,1]$ is

$$Z' = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = L \begin{bmatrix} f(Y_1) \\ -f(Y_2) \end{bmatrix} = F(Z),$$

with an 8×8 analytic Jacobian

$$DF(Z) = L \begin{pmatrix} Df(Y1) & 0 \\ 0 & -Df(Y_2) \end{pmatrix}$$

where

$$Df = \begin{pmatrix} (2\mu+\eta)^{-1}(1-\Gamma u^{-2}e) & (2\mu+\eta)^{-1}\Gamma u^{-1} & 0 & 0 \\ \nu^{-1}(1-u+\Gamma e_-) & \nu^{-1} & \nu^{-1}q & \nu^{-1}q \\ \beta^{-1}y + ku^{-2}z\check{\varphi}(e) & -ku^{-1}z\check{\varphi}'(e) & \beta^{-1}u & -ku^{-1}\check{\varphi}(e) \\ -\beta^{-1}y & 0 & -\beta^{-1}u & 0 \end{pmatrix}. \tag{4.20}$$

To use this numerical device, we impose the matching conditions $Y_1(0) = Y_2(0)$. Because of the translational invariance of the wave, we impose an additional phase condition specifying the value of a coordinate of the solution at $x = 0$, for example, $Y_{11}(0) = c$.

Projective conditions must be satisfied at $\pm\infty$, which will depend on the end states of the profiles. Specifically, $z_- = 1$ and $z_+ = 0$ are the end states for the mass fraction of the reactant, $u_- = 1$ from the rescaling, e_+ and u_+ are determined by e_-, Γ, and q using the Rankine-Hugoniot conditions, and $y_\pm = 0$ because z has constant end states and $u_\pm \neq 0$.

Substituting $(u_\pm, e_\pm, y_\pm, z_\pm)$ into the Jacobian (4.20), we numerically solve for two growth and two decay modes at $x = +\infty$ and three growth and one center mode at $x = -\infty$. Requiring the detonation to approach its end states orthogonal to the center mode at $x = -\infty$ and to the growth modes at $x = +\infty$ leads to three projective conditions at $x = \pm\infty$. Let Π_g and Π_c be matrices whose columns are bases for the growth eigenspace at $x = +\infty$ and the centered eigenspace at $x = -\infty$. Numerically the boundary conditions for the detonation profiles are

$$\Pi_g^*(Y_1(1) - (u_+, e_+, y_+, z_+)^T) = 0, \tag{4.21}$$

$$\Pi_c^*(Y_2(1) - (u_-, e_-, y_-, z_-)^T) = 0, \tag{4.22}$$

$$Y_1(0) - Y_2(0) = 0, \tag{4.23}$$

$$Y_{11}(0) = c. \tag{4.24}$$

The system parameters for the profile are $\Gamma, (2\mu+\eta), \nu, k, q, \beta, E_A$, and e_-. We note

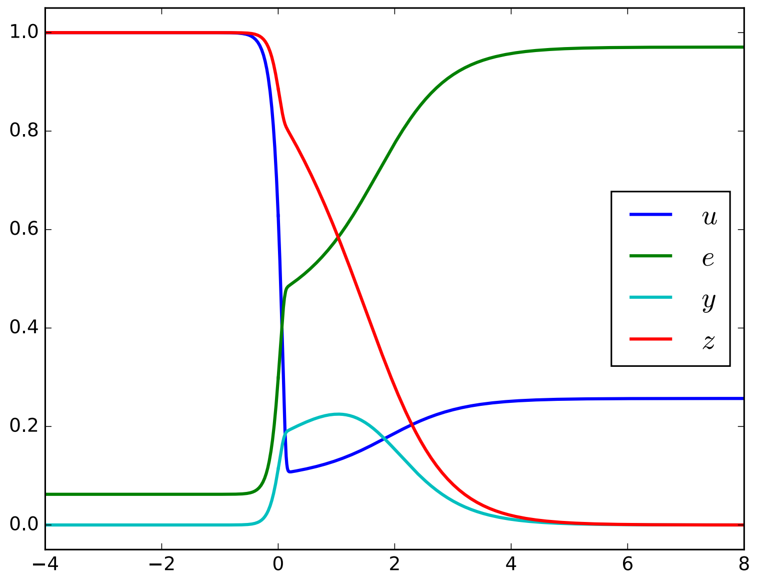


Figure 4.4: A leftward moving traveling wave solution (strong detonation) of the reactive Navier Stokes equations in Eulerian coordinates, with activation energy $E_A = 2.7$ and $(2\mu + \eta) = 0.1$. This is the same profile computed in [3], although computed in Eulerian coordinates and oriented toward the left.

that Evans function computation requires both viscous parameters μ and η, rather than the single parameter $(2\mu + \eta)$ required for the wave profile.

4.6 The linearized eigenvalue equations

To be more concise, let

$$\tilde{\mu} = (2\mu + \eta), \quad \tilde{\eta} = (\mu + \eta), \quad \hat{p} = \Gamma\hat{\rho}\hat{e}, \quad \gamma = \Gamma + 1.$$

The linearized eigenvalue problem comes from linearizing (4.9) about the traveling wave and taking the Fourier transform in the x_2 direction, resulting in

$$\lambda\rho + (\hat{\rho}u + \hat{u}\rho)' + i\xi\hat{\rho}v = 0, \tag{4.25a}$$

$$\lambda(\hat{\rho}u + \hat{u}\rho) + (2u + \hat{u}^2\rho + \Gamma(\hat{e}\rho + \hat{\rho}e))' = \tilde{\mu}u'' - \xi^2\mu u + i\xi(\tilde{\eta}v' - v), \tag{4.25b}$$

$$\lambda\hat{\rho}v + v' + i\xi\Gamma(\hat{e}\rho + \hat{\rho}e) = \mu v'' - \xi^2\tilde{\mu}v + i\xi\tilde{\eta}u', \tag{4.25c}$$

$$\begin{aligned}
&\lambda\left(\hat{\rho}e + u + \rho(\hat{e} + \frac{\hat{u}^2}{2})\right) + i\xi v\left(\gamma\hat{\rho}\hat{e} + \hat{u}/2\right) + \\
&\left[\gamma(\hat{e}\hat{u}\rho + e + \hat{e}\hat{\rho}u) + \frac{1}{2}\rho\hat{u}^3 + \frac{3}{2}\hat{u}u\right]' = \\
&\qquad (\nu e' + \tilde{\mu}(\hat{u}'u + \hat{u}u') + i\xi\eta\hat{u}v)' - \xi^2\nu e \\
&\qquad + \mu(i\xi\hat{u}v' - \xi^2\hat{u}u) + i\xi\eta\hat{u}_{x_1}v + qk\left(\hat{\rho}\hat{z}\check{\varphi}'(\hat{e})e + \hat{\rho}\check{\varphi}(\hat{e})z + \hat{z}\check{\varphi}(\hat{e})\rho\right)
\end{aligned} \tag{4.25d}$$

$$\begin{aligned}
&\lambda(\hat{z}\rho + \hat{\rho}z) + (\hat{u}\hat{z}\rho + \hat{\rho}\hat{z}u + z)' + i\xi\hat{\rho}\hat{z}v = \\
&\qquad \beta\left((\hat{\rho}z' + \hat{z}'\rho)' - \xi^2\hat{\rho}z\right) - k\left(\hat{\rho}\hat{z}\check{\varphi}'(\hat{e})e + \hat{\rho}\check{\varphi}(\hat{e})z + \hat{z}\check{\varphi}(\hat{e})\rho\right)
\end{aligned} \tag{4.25e}$$

We rewrite the system in flux coordinates, building off the work done in [17]. Defining flux variables

$$w_1 := -\hat{\rho}u - \hat{u}\rho, \tag{4.26a}$$

$$w_2 := \tilde{\mu}u' - (2u + \hat{u}^2\rho) - \Gamma(\hat{e}\rho + \hat{\rho}e) + i\xi\tilde{\eta}v, \tag{4.26b}$$

$$w_3 := \mu v' - v + i\xi\tilde{\eta}u, \tag{4.26c}$$

$$w_4 := \tilde{\mu}(\hat{u}_{x_1}u + \hat{u}u') + \nu e' - \gamma(e + \hat{u}\hat{e}\rho + \hat{e}\hat{\rho}u) - \left(\frac{3}{2}\hat{u}u + \frac{1}{2}\hat{u}^3\rho\right) + i\xi\tilde{\eta}\hat{u}v, \tag{4.26d}$$

$$w_5 := \beta(\hat{\rho}z' + \hat{z}_{x_1}\rho) - (\hat{u}\hat{z}\rho + \hat{\rho}\hat{z}u + z), \tag{4.26e}$$

we then adjust with

$$\tilde{w}_2 := w_2 - \hat{u}w_1 = \tilde{\mu}u' - u - \Gamma(\hat{e}\rho + \hat{\rho}e) + i\xi\tilde{\eta}v, \tag{4.27a}$$

$$\tilde{w}_4 := w_4 - \hat{u}\tilde{w}_2 - \hat{E}w_1 = \nu e' + \tilde{\mu}\hat{u}_{x_1}u - e - \hat{p}u. \tag{4.27b}$$

In this coordinate system the eigenvalue equations become

$$w_1' = -\lambda\hat{\rho}w_1 - \lambda\hat{\rho}^2u + i\xi\hat{\rho}v, \tag{4.28a}$$

$$\tilde{w}_2' = -\hat{u}_{x_1} w_1 + (\lambda\hat{\rho} + \mu\xi^2)u, \tag{4.28b}$$

$$w_3' = -i\xi\hat{p}w_1 - i\xi\hat{p}\hat{\rho}u + (\lambda\hat{\rho} + \xi^2\tilde{\mu})v + i\xi\Gamma\hat{\rho}e, \tag{4.28c}$$

$$\begin{aligned} \tilde{w}_4' = (-\hat{e}_{x_1} + \hat{z}f_2)w_1 - \hat{u}_{x_1}\tilde{w}_2 + i\xi f_1(\hat{u}, \hat{u}_{x_1})v \\ + f_4 e + \hat{\rho}\hat{z}f_2 u - f_2 z, \end{aligned} \tag{4.28d}$$

$$w_5' = i\xi\hat{\rho}\hat{z}v - \hat{\rho}\hat{z}f_3 w_1 - \hat{\rho}^2\hat{z}f_3 u + \hat{\rho}(\beta\xi^2 + f_3)z + k\hat{\rho}\hat{z}\check{\varphi}'(\hat{e})e, \tag{4.28e}$$

$$\tilde{\mu}u' = -\hat{p}w_1 + \tilde{w}_2 + (1 - \hat{p}\hat{\rho})u - i\xi\tilde{\eta}v + \Gamma\hat{\rho}e, \tag{4.28f}$$

$$\mu v' = w_3 - i\xi\tilde{\eta}u + v, \tag{4.28g}$$

$$\nu e' = \tilde{w}_4 + (\hat{p} - \tilde{\mu}\hat{u}_{x_1})u + e, \tag{4.28h}$$

$$\beta z' = \hat{u}w_5 + \hat{u}z + \beta\hat{\rho}\hat{z}_{x_1}u + (\beta\hat{z}_{x_1} - \hat{u}\hat{z})w_1, \tag{4.28i}$$

where

$$\begin{aligned} f_1 &= \hat{p} + (\mu - \eta)\hat{u}_{x_1}, \\ f_2 &= qk\hat{\rho}\check{\varphi}(\hat{e}), \\ f_3 &= \lambda + k\check{\varphi}(\hat{e}), \\ f_4 &= \lambda\hat{\rho} + \nu\xi^2 - qk\hat{\rho}\hat{z}\check{\varphi}'(\hat{e}). \end{aligned}$$

4.6.1 The Evans system. The linearized eigenvalue problem may be written in the form $W' = A(x;\lambda,\xi)W$ where $W = [w_1, \tilde{w}_2, w_3, \tilde{w}_4, w_5, u, v, e, z]^T$ and the matrix A is given by

$$\begin{pmatrix}
-\lambda\hat\rho & 0 & 0 & 0 & 0 & -\lambda\hat\rho^2 & i\xi\hat\rho & 0 & 0 \\
-\hat u_{x_1} & 0 & 0 & 0 & 0 & \lambda\hat\rho+\mu\xi^2 & 0 & 0 & 0 \\
-i\xi\hat p & 0 & 0 & 0 & 0 & -i\xi\hat p\hat\rho & \lambda\hat\rho+\tilde\mu\xi^2 & i\xi\Gamma\hat\rho & 0 \\
-\hat e_{x_1}+\hat z f_2 & -\hat u_{x_1} & 0 & 0 & 0 & \hat\rho\hat z f_2 & i\xi f_1 & f_4 & -f_2 \\
-\hat\rho\hat z f_3 & 0 & 0 & 0 & 0 & -\hat\rho^2\hat z f_3 & i\xi\hat\rho\hat z & k\hat\rho\hat z\check\varphi'(\hat e) & \hat\rho(\beta\xi^2+f_3) \\
-\tilde\mu^{-1}\hat p & \tilde\mu^{-1} & 0 & 0 & 0 & \tilde\mu^{-1}(1-\hat p\hat\rho) & -i\tilde\mu^{-1}\xi\tilde\eta & \tilde\mu^{-1}\Gamma\hat\rho & 0 \\
0 & 0 & \mu^{-1} & 0 & 0 & -i\mu^{-1}\xi\tilde\eta & \mu^{-1} & 0 & 0 \\
0 & 0 & 0 & \nu^{-1} & 0 & (\hat p-\tilde\mu\hat u_{x_1})/\nu & 0 & \nu^{-1} & 0 \\
\hat z_{x_1}-\hat u\hat z/\beta & 0 & 0 & 0 & \beta^{-1}\hat u & \hat\rho\hat z_{x_1} & 0 & 0 & \beta^{-1}\hat u
\end{pmatrix}. \tag{4.29}$$

4.7 Numerical results

4.7.1 Evans function formulation. The dimension nine Evans matrix (4.29) provides a first order formulation of the linearized eigenvalue problem (4.28). Unstable detonations in the multidimensional RNS equations correspond to values $\{\xi, \lambda \mid \xi \geq 0, \Re(\lambda) \geq 0\}$ for which there are nontrivial solutions W of

$$W' = A(x;\lambda,\xi)W. \tag{4.30}$$

Intuitively, λ describes instabilities occurring in the longitudinal direction, and ξ describes instabilities in the transverse direction.

A nontrivial solution W must grow along the four dimensional unstable eigenspace of $A_(\lambda,\xi)$ near $x=-\infty$, and decay along the five dimensional stable eigenspace of $A_+(\lambda,\xi)$ near $x=+\infty$. Let $W_-(x;\lambda,\xi)$ be an analytic basis in λ for the growth manifold of (4.30), and $W_+(x;\lambda,\xi)$ an analytic basis for the decay manifold. The separation between the growth

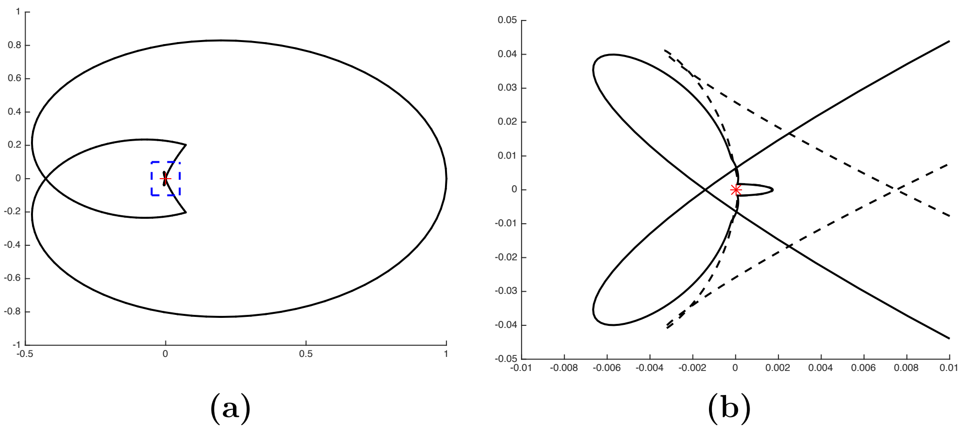


Figure 4.5: In **(a)** we show the Evans function output for $E_A = 2.7$ on a semicircular contour of radius $R = 0.1$. The inset in **(a)** is magnified in **(b)**, where it is easy to see that the winding number changes between $E_A = 2.7$ and $E_A = 2.8$ as a Hopf bifurcation occurs.

manifold and the decay manifold at $x = 0$ is captured by the Evans function

$$D(\lambda, \xi) = \det \left[W_-(x; \lambda, \xi); W_+(x; \lambda, \xi)\right]_{x=0}.$$

To compute analytic bases $W_\pm(x; \lambda, \xi)$, we begin by initializing with analytic bases at infinity using the method of Kato. $W_\pm$ are then evolved from $x = \pm\infty$ to 0 using the method of Drury. The methods of Dury and Kato are described in detail by Sections 2.3 and 2.2 and the references contained therein. Our calculations use an alternative formulation of the Evans function near $x = +\infty$, which allows us to evolve a four dimensional adjoint subspace $\tilde{W}_+(x; \lambda, \xi)$ from $+\infty$ to 0 instead of the standard five dimensional decay manifold W_+. The alternative Evans function is given by

$$\det \left[\tilde{W}_+^*(x; \lambda, \xi) W_-(x; \lambda, \xi)\right]_{x=0};$$

see Section 1.3.1 for further details on the adjoint formulation.

4.7.2 Parameter values. Our numerical computations have set $\mu = \eta = 1/30$, $\beta = 0.1$, $T_{ig} = 0.06641$, and $c_v = 1$. To better compare our results with those in [3], we have chosen

k as a function of activation energy E_A. This choice of k regulates the scale length of the reaction, thereby simplifying the solution of the detonation profiles. After setting $\Gamma = 0.2$, $q = 0.623$, and $e_- = 0.0623$, parameters $e_+ = 0.9706, u_+ = 0.2569$ are determined by the Rankine-Hugoniot condition.

Detonations are known to experience a cascade of Hopf-like bifurcations at higher values of $\Im(\lambda)$ as activation energy increases. Our experiments have focused on the smallest pair of eigenvalues as they enter the right half-plane, and eventually return and restabilize. Our parameters of interest are activation energy E_A and heat conductivity ν. The first experiment describes these instabilities as E_A varies between 1.6 and 7.1 with ν fixed. The second experiment fixes E_A and allows ν to vary. This helps us observe the nature of the unstable manifold of eigenvalues present in the multidimensional RNS system. These results should be compared to those found in [3].

4.7.3 Fixed heat conductivity with varying activation energy. For $\nu = 0.1$ we allowed E_A to vary from 1.6 to 7.1 and tracked the first pair of eigenvalues seen crossing into the right half-plane. Since $\xi = 0$ corresponds to the standard 1D RNS Evans function, we were able to verify our results by comparing with those found in [3]. In both systems ($\xi = 0$) instability occurs around $E_A = 2.7$ and the system restabilizes around $E_A = 7.1$. Those eigenvalues also restabilize as ξ increases. Because instabilities in planar detonations correspond to a location λ in the right half-plane and a Fourier frequency ξ, varying activation energy E_A defines a manifold of instabilities $(\Re(\lambda), \Im(\lambda), \xi)$; see Figure 4.6.

Perhaps a more interesting way to view the unstable eigenvalue pair is to consider it as an object in the three dimensional space of triples (E_A, ν, ξ). Cross-sections of this object along the ν axis allow us to view the neutral stability boundary in (E_A, ξ)-space. Higher values of heat conductivity correspond to a faster return to stability as activation energy E_A increases; see Figure 4.7.

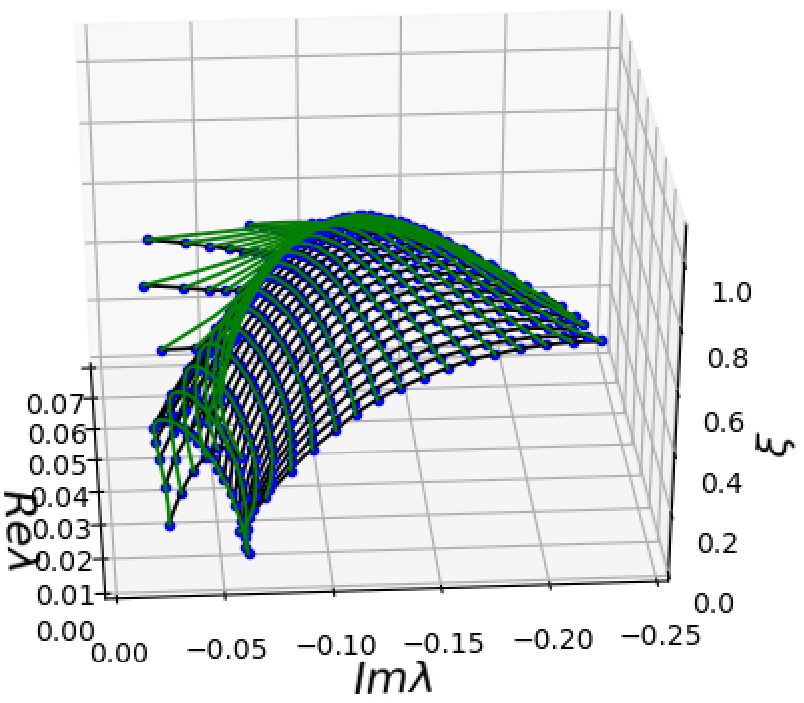
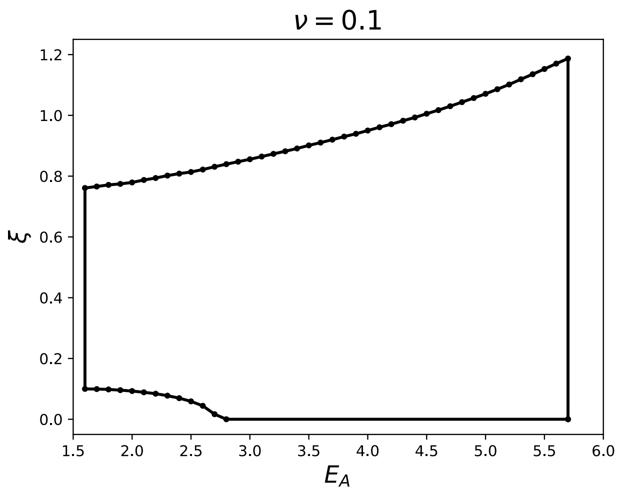


Figure 4.6: We plot a portion of the unstable eigenvalues of strong detonations in the multidimensional RNS equations with $E_A \in [1.6, 6.6]$. The location of the unstable eigenvalue λ with negative imaginary part is recorded together with its Fourier frequency ξ in the transverse direction. For fixed ξ, as E_A increases the unstable eigenvalue turns around and heads back to the left half-plane.

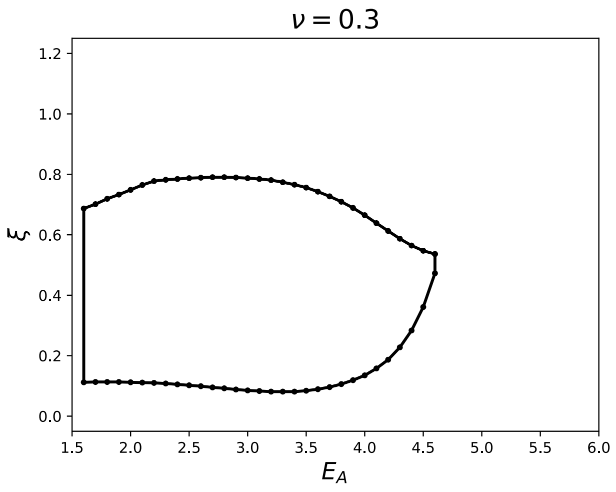
4.7.4 Fixed activation energy with varying heat conductivity.

These experiments have focused on the effect of increasing heat conductivity on the smallest pair of unstable eigenvalues, for several fixed values of activation energy. In general, as heat conductivity increases the unstable pair of eigenvalues returns to the left half-plane and restabilize. This effect is illustrated in Figures 4.9 and 4.10 by plotting the Evans function output on a semi-circular contour with radius 0.4 in the right half-plane. As the heat conductivity increases from $\nu = 0.1$ to 0.8, the Evans contours can be seen to unwind from the origin, indicating that the eigenvalue pair has restabilized.

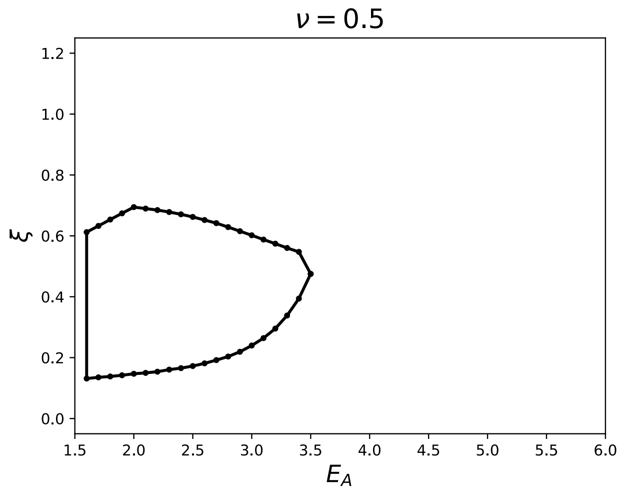
Figure 4.11 graphs the neutral stability boundary in (ν, ξ) space for $E_A = 2, 3$, and 4. The boundary demonstrates a return to stability as ν increases—with a quicker return to stability for the larger values of E_A.



(a)



(b)



(c)

Figure 4.7: Neutral stability boundary in (E_A, ξ) space for $\nu = 0.1, 0.3$, and 0.5. For higher values of heat conductivity, we see a faster return to stability as activation energy ν increases. We restricted our analysis to $E_A \in [1.6, 5.8]$.

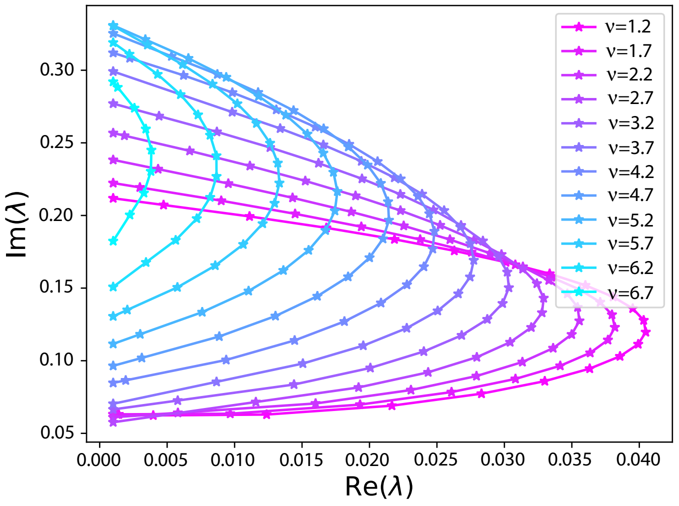
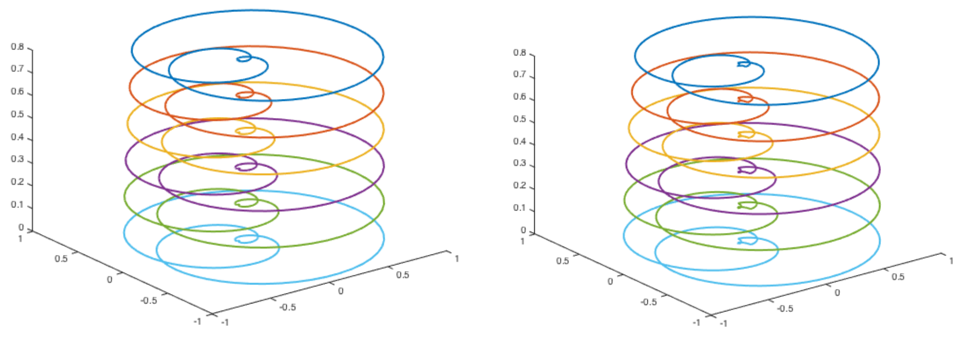
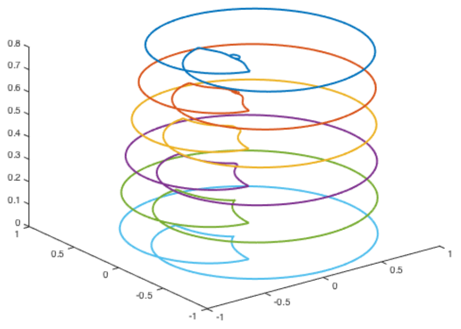


Figure 4.8: Unstable eigenvalues as heat conductivity ν varies. Each line is parameterized by Fourier frequency ξ. Eigenvalues return to the left half-plane and restabilize as ν increases.

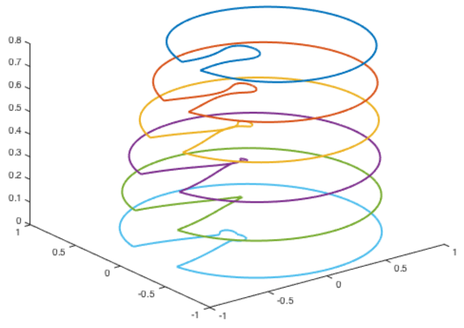


(a) **(b)**

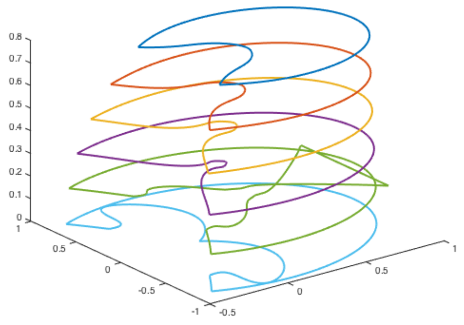
Figure 4.9: Evans function output for several Fourier frequencies $\xi \in [0, 0.8]$ on a semicircular contour with radius 0.4. **(a):** $\nu = 1/10$. **(b):** $\nu = 1/5$.



(a)

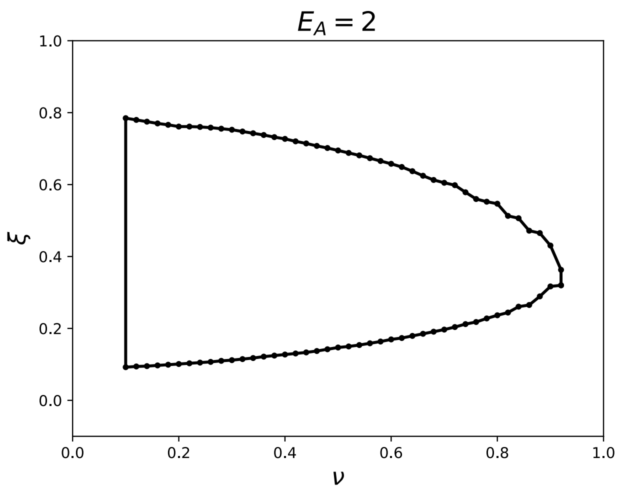


(b)

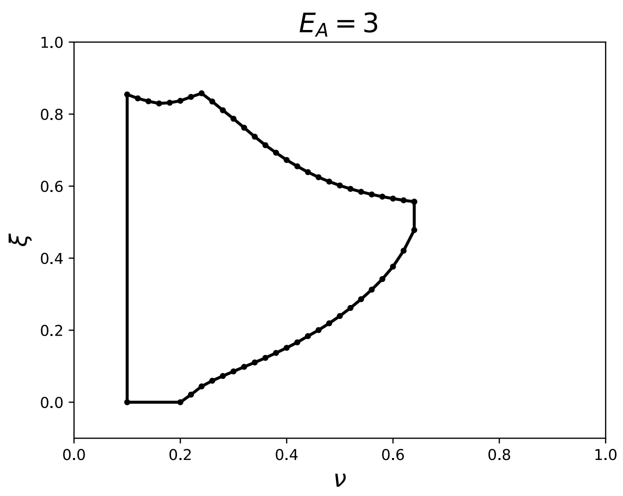


(c)

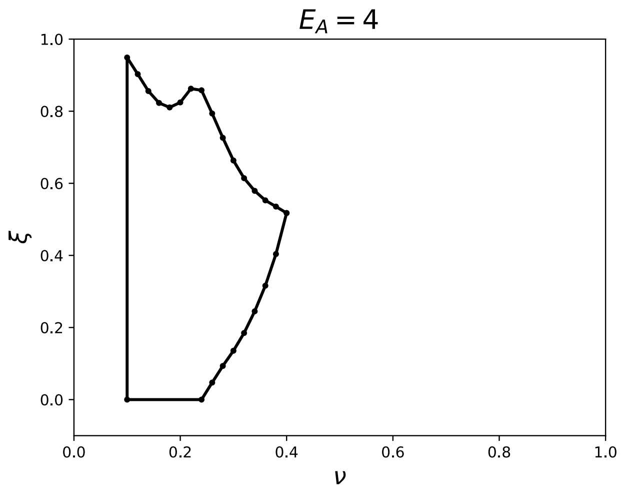
Figure 4.10: Evans function output for several Fourier frequencies $\xi \in [0, 0.8]$ on a semicircular contour with radius 0.4. **(a):** $\nu = 2/5$. **(b):** $\nu = 4/5$. **(c):** $\nu = 8/5$.



(a)



(b)



(c)

Figure 4.11: Neutral stability boundary in (ν, ξ) space for $E_A = 2, 3$, and 4. For higher values of activation energy, we see a quicker return to stability as heat conductivity ν increases. We restricted our analysis to values of $\nu \geq 0.1$.

Chapter 5. A Python implementation of STABLAB

Numerical Evans function computation has become a mature technology; see [20, 4, 6]. Our numerical computations rely on STABLAB, a MATLAB-based library developed by Humpherys et al. STABLAB provides core functionality for computing wave profiles, initializing analytic bases, and calculating the Evans function. It is able to set up batch processes, and has some capabilities for multi-core and distributed computing in MATLAB. STABLAB has recently been ported to Python, with its open source scientific computing environment and parallel processing libraries. Our work with detonations in the reactive Navier-Stokes equations used Python's parallel processing tools to locate the instabilities graphed in Figure 4.6. The computations used half a day and several hundred processors on Mary Lou, BYU's supercomputer.

5.1 STABLAB: Stability Laboratory for Evans function computation

MATLAB provides several robust functions for solving initial and boundary value problems that are used extensively in STABLAB. The IVP solvers are used to integrate the growth and decay manifolds of the Evans system from $x = \pm\infty$ to $x = 0$. MATLAB function `ode45` uses a Dormand-Prince Runge-Kutta algorithm. This algorithm constructs fourth and fifth order solutions which allow the routine to adaptively control the step size to bound the numerical error. The variable-order solver `ode15s` can be used to solve stiff IVPs.

STABLAB also relies heavily on robust BVP solvers to compute traveling wave profiles. Collocation and shooting are common approaches to solving BVPs. Good MATLAB solvers include `bvp4c` and `bvp5c`, which use 4th and 5th order collocation. We use the non-native MATLAB function `bvp6c`, which extends `bvp4c` with a 6th order Lobatto collocation method; see [12].

Today's Evans function studies are often highly demanding, involving systems with a large number of relevant physical parameters. The most intense computations are ODE integration routines, which evolve manifolds from $x = \pm\infty$ to $x = 0$. These computations are embarrassingly parallel and are an easy source of potential speed gains. As more physical, multi-D systems are encountered with larger parameter spaces, parallel processing tools will make complicated Evans function computations feasible. Unfortunately, MATLAB's Parallel Computing Toolbox limits users to 12 cores on a single machine.

5.2 Migration to Python

The core STABLAB routines have recently been implemented in Python's scientific computing environment. Python is an object-oriented interpreted language that is ideal for prototyping complex mathematical systems. Python is open source, has a large community of users and developers in scientific computing, and has many resources for multi-threaded and distributed computing.

Our computing stack includes Python 2.7, its standard library, and other third party libraries that include NumPy, SciPy, Matplotlib, and MPI4PY. NumPy and SciPy are popular software packages used in science, engineering, and mathematics. These libraries consist of a Python object-oriented interface to precompiled C and Fortran code such as BLAS and LAPACK, and support fast, vectorized arithmetic operations and a wide assortment of functions. Since NumPy and SciPy rely internally on BLAS and LAPACK, scientific computing in Python is about as fast as MATLAB, which is an interpreted language that also relies on BLAS and LAPACK internally.

Matplotlib is a package that can be used to create quality 2D plots; see [22]. It is a standard in Python's scientific computing community. Matplotlib can be easily adopted by MATLAB users due to the similarity of its syntax.

Several ODE solvers are available in Python. The standard adaptive fifth order Dormand-Prince Runge-Kutta procedure for IVPs is implemented as an option for the `complex_ode` function in `scipy.integrate`. One BVP solver is `bvp_solver`, which is implemented as a SciKit. The non-native MATLAB solver `bvp6c`, built by Nick Hale and implementing sixth order collocation, has been also been repurposed for use in Python.

STABLAB is available in a GitHub repository in both Python and MATLAB; see [4]. The Python implementation contains scripts that compute the Evans function for traveling waves for Burgers equation, the gKdV equation, the Boussinesq equation, and combustion with a high Lewis number.

5.3 Solitons in the gKdV system

In this section we construct the Evans function for soliton solutions of the gKdV equation, and use numerical continuation to track the unstable eigenvalue-eigenfunction pair. We also provide Python code that computes the Evans function.

The gKdV equation is given by

$$u_t + u_{xxx} + \frac{1}{p}(u^p)_x = 0, \quad p \geq 2, \tag{5.1}$$

with solitary wave solutions of the form

$$\hat{u}(x-st) = \frac{p(p+1)}{2}\operatorname{sech}^2\left(\frac{1-p}{2}(x-st)\right)^{1/(p-1)};$$

see Section 3.2. Transforming (5.1) into its moving frame $(x,t) \to (x-st,t)$ and linearizing about the steady state solution $\hat{u}$ results in the eigenvalue problem

$$\lambda v - sv' + v''' + (\hat{u}^{p-1}v)' = 0.$$

By a possible rescaling, we may assume that $s = 1$. Substituting $w = \int_{-\infty}^{x} v$ and integrating from $-\infty$ to x yields the eigenvalue problem in integrated coordinates,

$$\lambda w - w' + w''' + \hat{u}^{p-1} w' = 0.$$

The eigenvalue problem can be written as a first-order system $W' = A(x; \lambda, p)W$, with

$$A(x; \lambda, p) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\lambda & 1 - \hat{u}^{p-1} & 0 \end{pmatrix} \quad \text{and} \quad W = \begin{pmatrix} w \\ w' \\ w'' \end{pmatrix}. \tag{5.2}$$

For an unstable eigenvalue λ, our root following approach requires an initial estimate of the corresponding eigenfunction $W(x)$; see Section 3.2.2. Since the scaling invariance of $W(x)$ necessitates a phase condition at $x = 0$, we reformulate (5.2) on the domain $x \in [0, \infty)$. Letting $Y(x) = W(-x)$, we obtain

$$\begin{pmatrix} W \\ Y \\ \lambda \end{pmatrix}' = \begin{pmatrix} A(x; \lambda, \mu) & 0 & 0 \\ 0 & -A(-x; \lambda, \mu) & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} W \\ Y \\ \lambda \end{pmatrix}, \quad x \in [0, \infty). \tag{5.3}$$

The phase condition can be imposed at $x = 0$ by fixing a coordinate of W. The boundary conditions consist of 3 matching conditions, 3 projective conditions at $x = \infty$, and the phase condition. Letting $P_+(\lambda, \mu)$ be the projection onto the unstable manifold of $A_+(\lambda, \mu)$ and $P_-(\lambda, \mu)$ the projection onto the stable manifold of $A_-(\lambda, \mu)$, the boundary conditions are implemented on a truncated interval $[0, L]$ by

$$\begin{pmatrix} W(0) - Y(0) \\ P_+(\lambda, \mu) W(L) \\ P_-(\lambda, \mu) Y(L) \\ W_1(0) - W_1^* \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{5.4}$$

where $L > 0$ is large enough to capture the asymptotic behavior of the eigenfunction. Numerical continuation is then used to track the eigenvalue-eigenfunction pair as p varies. Python functions implementing (5.2), (5.4), and the right hand side of (5.3) are contained in the file `gKdV.py`, shown in Listing 5.1.

5.4 Evans computation and roottracking in the gKdV system

5.4.1 gKdV driver file. Our Python code for the gKdV system is contained in two files, `gKdV.py` and `gKdVdriver.py`. The driver file contains most of the control code for the gKdV system, and imports the gKdV Evans matrix, the eigenfunction ODE, and the associated boundary conditions from `gKdV.py`. In this section we will provide a short description of the code contained in `gKdVdriver.py`.

`gKdVdriver.py` begins with a series of Python import statements. Basic mathematical and plotting functions are imported from NumPy, SciPy, and Matplotlib. The core STABLAB algorithms are implemented in four Python modules, contained in the files `bin.py`, `contour.py`, `manifold.py`, and `evans.py`. Several of the functions in these modules are also imported into `gKdVdriver.py`, along with the BVP solver `bvp6c` and `roottracking.py`.

```
from __future__ import division, print_function
import numpy as np
from numpy import array, ones, zeros, linspace, flipud, conj, dot
import matplotlib.pyplot as plt


from core.contour import winding_number
from core.evans import emcset, compute_Evans
from core.roottracking import eigf_init_guess, mod_drury
from core.pybvp6c.bvp6c import bvp6c, bvpinit, deval, struct
from gKdV import A, G, G_jacobian, bcs_G, bcs_G_jacobian
```

Next the gKdV system is initialized for Evans function computation with several STABLAB variables, implemented as Python dictionaries. Variable `p` contains gKdV system parameters, and `s` is used to evaluate wave profiles for the system. Variables `e`, `m`, and `c` are used by functions defined in `evans.py`, `manifold.py`, and `contour.py`, respectively, and are initialized with the `emcset` function.

The call to `emcset` specifies the type of the traveling wave as a front or a pulse. It also specifies the number of growth modes and decay modes in the Evans system at $x = \pm\infty$ as 2 and 1, respectively. Our computations will use the adjoint formulation for the Evans system near $x = -\infty$; this is a small simplification that reduces the dimension of the manifold that must be evolved from $x = -\infty$ (from $k = 2$ to $k = 1$).

```
p = {'p':10}
s = {'I':1,'R':5,'L':-5,'A':A}
s,e,m,c = emcset(s,'pulse',2,1,'adj_reg_polar')
```

Next we construct a contour in the right half-plane, over which we compute the Evans function. To compute the Evans function, growth and decay manifolds must be evolved from $x \pm \infty$ to $x = 0$. Analytic bases for these manifolds are initialized at infinity with a numerical method based on Kato's reduced ODE; see Section 2.3 for details and references. `c['ksteps']` specifies the number of Kato steps between points in the complex plane where the Evans function is computed. This value can be increased as needed to ensure the initializing bases vary smoothly.

The `compute_Evans` function requires the contour and several STABLAB variables as input. Variable `s` is a Python dictionary that usually contains the numerical solution of the traveling wave; however, since solitons in the gKdV system have an explicit formula, this formula is provided in the implementation of the Evans matrix A. The `compute_Evans` function evaluates the Evans function over the contour and the winding number is computed; see Figure 5.1.

```
c['ksteps'] = 1; points = 50
preimage = (  5.5+5*np.exp(
                 2*np.pi*1j*np.linspace(0,1,points+(points-1)*c['ksteps']))   )

preimage2, w = compute_Evans(preimage,c,s,p,m,e)
```

```
print('Evans Computation Successful\nThe winding number is',winding_number(w))
plt.plot(np.real(w),np.imag(w),'*-k',linewidth=2)
plt.show()
```

To implement roottracking for the gKdV system, we begin by resetting the STABLAB variables. Note in particular the new call to the `emcset` function: our roottracking approach requires a `'reg_reg_polar'` formulation for the Evans function.

We set the system parameter and the length of the numerical domain of the eigenfunction to $p = 5.2$ and $L = 20$, respectively. A new STABLAB variable `r` is initialized for use in `roottracking.py`. The function `eigf_init_guess` constructs an initial estimate for the eigenfunction W. Analytic bases for the growth and decay manifolds at $x = \pm\infty$ are evolved to $x = 0$ using a small modification of Drury's method; see equation 3.9. These analytic bases are evolved for $\lambda = 0.098035$, an unstable eigenvalue of the gKdV system when $p = 5.2$. W is then constructed using (3.6) and (3.7).

```
# Roottracking: find initial estimate of eigenfunction W
L = 20.
p, s = {'p':5.2,'p_final':10.}, {'I':L,'R':L,'L':-L,'A':A}
s, e, m, c = emcset(s,'pulse',2,1,'reg_reg_polar')
r={'N':200,'root':0.098035,'method':mod_drury,'options':{'RelTol':1e-8,'AbsTol':1e
    -9}}
r, x1, x2 = eigf_init_guess(s,e,m,c,p,r)
for j in range(0,m['n']):
    plt.plot(x1,np.flipud(r['W'][j+m['n']-1,:]) ,'k')
    plt.plot(np.flipud(x2), r['W'][j,:],'k')
plt.title('Eigenfunction for gKdV equation')
plt.xlabel('x'); plt.ylabel('W')
plt.show()
```

Naive numerical continuation is used to follow the eigenvalue-eigenfunction pair from $p = 5.2$ to $p = 10$. Options to `bvp6c` are set by defining attributes on the Python class `opt_G`, and helper functions `bvpinit` and `deval` are used to create initial guesses for `bvp6c` and to evaluate its solutions. The variable `r['ph']` will be used by the BVP solver to impose a phase condition for the eigenfunction.

```
# Roottracking: numerically continue in (lambda, W) as p varies through p_array
N = int(round((p['p_final']-p['p'])/.05) + 1)
p_array = linspace(p['p'],p['p'] + .05*(N-1),N)
root_array = zeros(N); root_array[0] = r['root']

opt_G = struct()
opt_G.abstol, opt_G.reltol = 1e-9, 1e-8
opt_G.nmax, opt_G.stats = 2000, 'on'
s.update({ 'guess':r['W'], 'ph':[1,r['W'][1,0]] })
for j in range(0,N):
    p['p'], s['I'] = p_array[j], 10-(10/4.8)*(p['p']-10)
    s['R'], s['L'] = s['I'], -s['I']
    opt_G.fjacobian = lambda x,y: G_jacobian(x,y,s,p)
    opt_G.bcjacobian = lambda x,y: bcs_G_jacobian(x,y,s,p)
    if j==0:
        solinit = bvpinit(linspace(0,s['I'],len(s['guess'][1])),s['guess'])
    else:
        solinit  = bvpinit(linspace(0,s['I'],250),yint)
    s['sol_G'] = bvp6c(lambda x,y: G(x,y,s,p),
                        lambda ya,yb: bcs_G(ya,yb,s,p), solinit, opt_G)
    yint,_ = deval(s['sol_G'],linspace(0,s['I'],250))
    root_array[j] = yint[-1,0]
    print('\n\nj = '+str(j)+':', '\nParameter p = ', p['p'])
    print('lambda = ', root_array[j], '\nNumerical Infinity = ', s['I'])
plt.figure()
plt.plot(p_array,root_array,'-*k')
plt.show()
```

5.4.2 gKdV system functions.

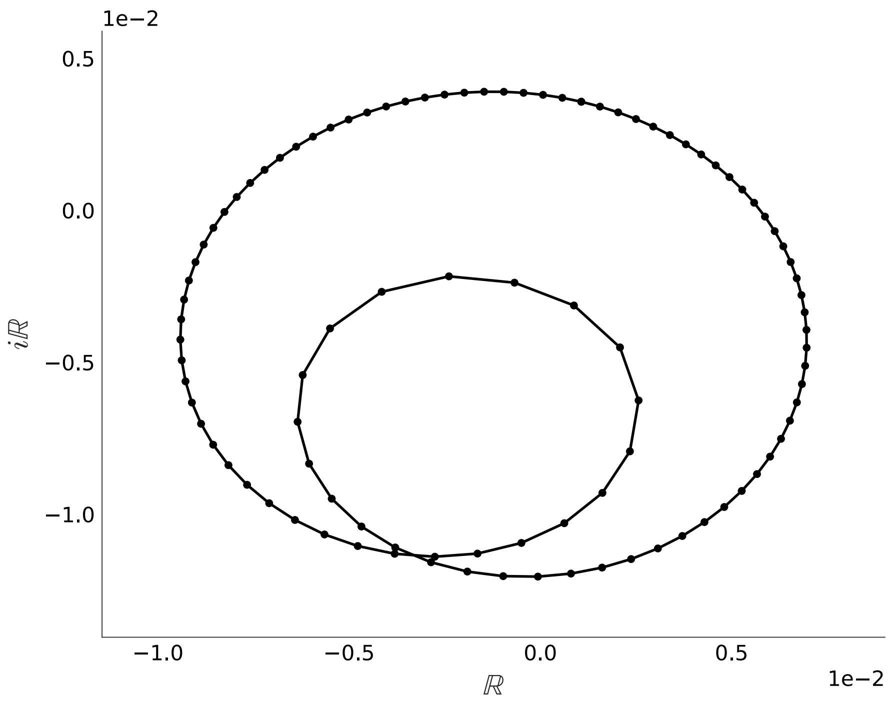


Figure 5.1: Evans function output for the gKdV system with $p = 10$ on a contour in the right half plane. The winding number of 1 indicates an instability.

Lines 1-4

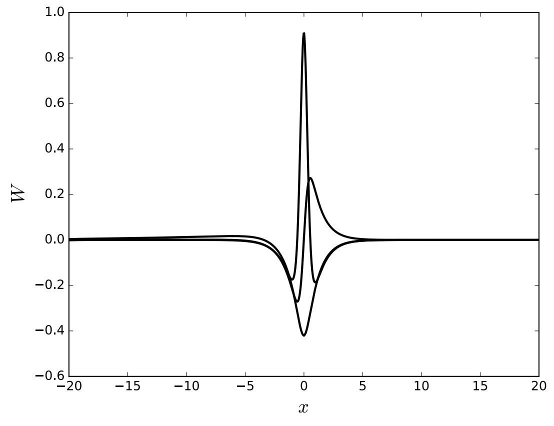
We import mathematical functions from NumPy and SciPy. The BVP solver `bcp6c` requires Jacobian functions for the ODE and the boundary conditions as arguments. This can be done using the `Jacobian` function from Numdifftools.

Lines 6-11

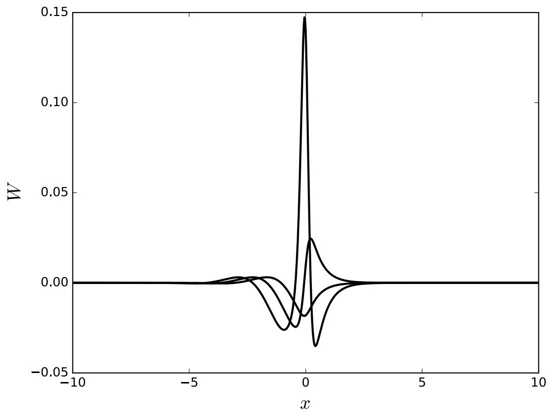
Here the Evans matrix A is defined. Note that STABLAB variables `s` and `p` are supplied as arguments. System parameters are contained in `p`, and `s` usually stores the numerical solution of the wave profile. However, since the gKdV system has an explicit formula for its wave profile, that formula is hard-coded in lines 7-8. Also notice that λ is coded as `lmbda`; in the Python language `lambda` is a keyword used to create or modify functions in-line. This can be seen in the Jacobian functions defined later in this file.

Lines 13-56

The ODE function (5.3) and its derivative are implemented as Python functions `G`



(a)



(b)

Figure 5.2: Subplots **(a)** and **(b)** graph the eigenfunctions W corresponding to the unstable eigenvalue λ for $p = 5.2$ and $p = 10$, respectively. The eigenvalue-eigenfunction pair satisfies (3.4) and (3.5).

and `G_jacobian`. The boundary conditions (5.4) and their derivative functions are implemented as `bcs_G` and `bcs_G_jacobian`. We also define the function `Flinear`, which is used to evaluate the Evans matrix at $x = \pm\infty$. The results of `Flinear` are used to provide projective conditions for the eigenfunction.

```
from scipy import linalg; orth = linalg.orth
from numdifftools import Jacobian
from numpy import array, ones, zeros, linspace, flipud, conj, dot, cosh, copy
from core.bin import projection2, projection1

def A(x,lmbda,s,p):
    u=(.5*p['p']*(p['p']+1))**(1./(p['p']-1))*(
                    cosh(.5*(1-p['p'])*x)    )**(-2./(p['p']-1))
    return array([  [0,      1.,              0 ],
                    [0,      0,               1.],
                    [-lmbda, 1-u**(p['p']-1), 0 ]     ])

def G(x,y,s,p):
    n = 3
    yr = copy(y[:n+1]); yr[-1] = y[-1]
```

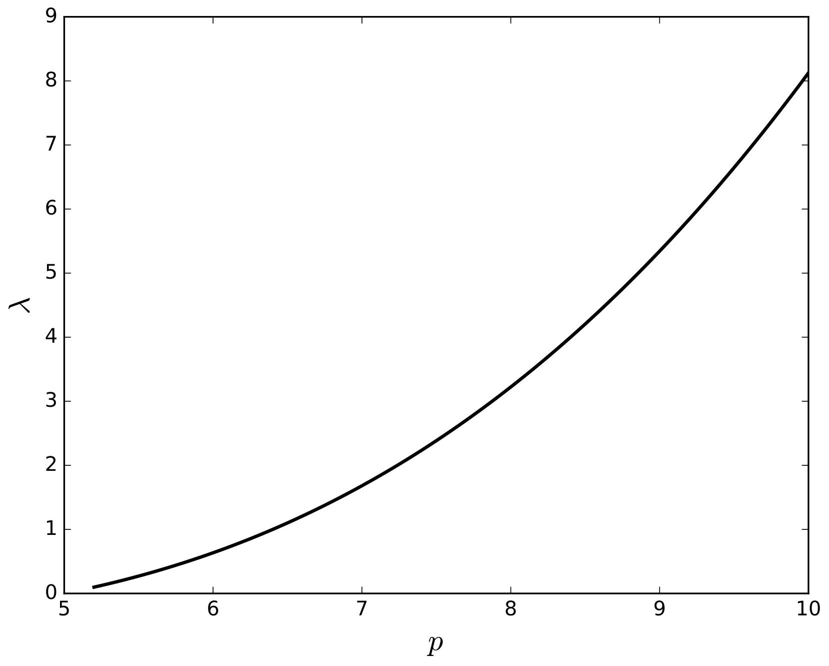


Figure 5.3: Unstable eigenvalues for the gKdV system for $5.2 \leq p \leq 10$. These eigenvalues are tracked using the root following method discussed in Chapter 3.2; see [19].

```
    yl = copy(y[n:])

    out = zeros(2*n+1)
    out[:n] =    (s['R']/s['I'])*dot(A(s['R']/s['I']*x,yr[-1],s,p), yr[:-1])
    out[n:2*n] = (s['L']/s['I'])*dot(A(s['L']/s['I']*x,yl[-1],s,p), yl[:-1])
    return out

def G_jacobian(x,y,s,p):
    try:
        g = Jacobian(lambda z:G(x,z,s,p),step = 1e-7)
    except:
        g = Jacobian(lambda z:G(x,z,s,p),step = 1e-8)
    return g(y)

def bcs_G(Ya,Yb,s,p):
    n = 3
    aa = array([0, 0, 0, (Ya[-1] + Yb[-1])/2. ])
    AM = Flinear(aa); AP = AM
    P, _ = projection2(AM,-1,1e-8); LM = orth(P)
    P, _ = projection2(AP,1,1e-8);  LP = orth(P)

    out = zeros((2*n+1,),dtype='complex')
    out[:3] = Ya[0:n]-Ya[n:2*n]             # matching conditions = 3
    out[3] = Ya[s['ph'][0]] - s['ph'][1]    # phase condition = 1
    out[4] = LM.T.dot(Yb[n:2*n])[0]         # at -infty; = 1
    out[5:] = LP.T.dot(Yb[0:n])             # at +infty; = 2
    return out

def bcs_G_jacobian(ya,yb,s,p):
    try:
        ga = Jacobian(lambda z:bcs_G(z,yb,s,p),step=1e-6)
        gb = Jacobian(lambda z:bcs_G(ya,z,s,p),step=1e-6)
    except:
        ga = Jacobian(lambda z:bcs_G(z,yb,s,p),step=1e-8)
        gb = Jacobian(lambda z:bcs_G(ya,z,s,p),step=1e-8)
    return ga(ya), gb(yb)

def Flinear(y):
    return array([ [0,      1,  0],
                   [0,      0,  1],
                   [-y[-1], 1,  0]  ])
```

Listing 5.1: gKdV.py

5.5 Detonation profiles and Evans computation in the RNS system

5.5.1 RNS driver file. We include here our Python code for numerically resolving detonation profiles and computing the associated Evans function. The code is contained in two files, `RNSdriver.py` and `RNS.py`, and can be obtained at [4]. Most of the control code is contained in `RNSdriver.py`, which sets up the RNS system, calls BVP solving routines from `RNS.py`, and computes the Evans function over a semicircular contour in the right half-plane.

The driver file begins with a series of Python statements importing STABLAB routines and other third-party libraries. The core STABLAB algorithms are implemented in four Python modules, contained in the files `bin.py`, `contour.py`, `manifold.py`, and `evans.py`. We import `pybvp6c`, a Python implementation of a BVP solver using a 6th order collocation method, `bvp6c`. We also import `MPI` from MPI4Py to support parallel Evans function computation. Basic mathematical and plotting functions are imported from NumPy, SciPy, and Matplotlib.

```
from __future__ import division, print_function
import sys
sys.path.append('/fslhome/joshualy/research2016/pystablabDec2016/')
import numpy as np
from numpy import linspace, array, zeros, ones, conj, real, imag, sqrt
from scipy.io import loadmat
import matplotlib.pyplot as plt
import pickle
from mpi4py import MPI

from core.bin import projection2, projection1
from core.contour import winding_number, semicirc2, analytic_basis
from core.evans import compute_Evans, emcset
from core.pybvp6c.bvp6c import bvp6c, bvpinit, deval
from core.pybvp6c.structure_variable import *
from RNS import detonation_profile_solver, A

def plot_profile(s,p):
    xint = linspace(0,s['I'],200)**2.
    Sxint,_ = deval(s['sol'],xint)
    l, c = ['$u$','$e$','$y$','$z$'], ['b','g','c','r']
```

```
        for j in range(4):
            plt.plot(s['R']*xint,Sxint[j,:],c[j],linewidth=2,label=l[j])
            plt.plot(s['L']*xint,Sxint[4+j,:],c[j],linewidth=2)
        plt.legend(loc='best',fontsize=18)
        plt.axis([s['L'],s['R'],-.05,1.05])
```

We import a file of paired values (E_A, k). The reaction rate k is computed to make the length of the reaction interval approximately equal for inviscid detonations in the ZND system as E_A increases; see [3]. The Rankine-Hugoniot conditions (4.19) can be computed using the `produce_endstates` function.

```
    k_EA=np.genfromtxt('data',skip_header=0,skip_footer=0,delimiter=',',dtype='float32'
        )
    # k_EA contains values of EA,k for EA = 1.6:0.01:7.1
    k_EA = k_EA[60:350:10,:] # values of EA,k for EA = 2.2:0.1:5.0

    def produce_endstates(q,gamma,e_minus):
        B=(gamma+1.)*(gamma*e_minus+1.)
        u_plus=(B-sqrt(B**2.-gamma*(gamma+2)*(1+2.*e_minus*(gamma+1.)+2.*q) )
                 )/(gamma + 2.)
        e_plus=u_plus*e_minus+u_plus*(1.-u_plus)/gamma
        return e_plus,u_plus
```

The `driver_continuation` function accepts a list of paired E_A and k values, and computes detonation profiles for each pair using naive numerical continuation. The parameters and numerical solutions are permanently stored using Python's `pickle` library. This code can easily be generalized to compute detonation profiles across other parameter regimes.

Next the RNS system is initialized for Evans function computation with several STABLAB variables, implemented as Python dictionaries. Variable `p` contains RNS system parameters, and `s` contains parameters needed to numerically compute wave profiles. All RNS system parameters are initialized as described in Section 4.7.2. Variables `e`, `m`, and `c` are used by functions defined in `evans.py`, `manifold.py`, and `contour.py`, respectively, and are initialized with the `emcset` function.

```
def driver_continuation(list_k_EA):
    # Define STABLAB dictionaries s and p
    epsilon = 0.1
    p={'g':0.2,    'mu':epsilon/3., 'eta':epsilon/3.,   'nu':epsilon, 'q':0.623,
       'beta':0.1, 'Ti':0.06641216,  #'EA':k_EA[0,0],       'k':k_EA[0,1],
       'c':1.,     'z_minus':1.,    'z_plus':0.,        'y_minus':0.,
       'y_plus':0.,'u_minus':1.,    'e_minus':0.0623,   'xi':0.          }
    p['e_plus'], p['u_plus'] = produce_endstates(p['q'], p['g'], p['e_minus'])
    s={'I':1.,'side':1,'n':4,'rarray':array([0,1,2,3]),'larray':array([4,5,6,7]),
        'R':8.,'L':-4.,'A':A}
    # right and left end states
    s['UR'] = array([p['u_plus'], p['e_plus'], 0., p['z_plus']])
    s['UL'] = array([p['u_minus'], p['e_minus'], 0., p['z_minus']])
    p['UR'], p['UL'] = s['UR'], s['UL']

    # Computes and saves profile and STABLAB variables s,p for EA, k in list_k_EA
    for EA, k in list_k_EA:
        p['EA'], p['k'] = EA, k
        sol, p = detonation_profile_solver(s,p)
        s['sol'] = sol

        print("EA = ", EA)
        str_=('profile_nu_1/euler_plot_EA_%1.2f'%EA).replace('.','_')
        File = open(str_,'wb')
        pickle.dump([s,p],File); File.close()
```

The `driver_Evans_xi` function accepts the name of a pickled Python object, and extracts the numerical solution in two STABLAB variables `s` and `p`. We construct a semicircular contour in the right half-plane centered at the origin. A small notch is cut in the contour to avoid the zero in the Evans function at the origin. The call to `emcset` specifies the type of the traveling wave as a front or a pulse. It also specifies the number of growth modes and decay

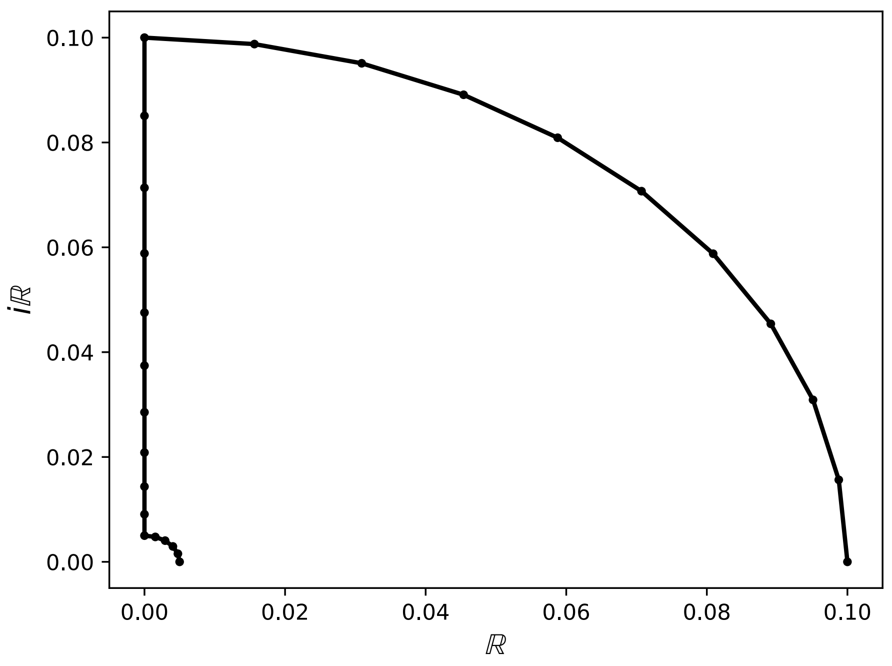


Figure 5.4: The upper half of a semicircular contour in the right half-plane. The lower half can be ignored due to the symmetry $D(\overline{\lambda}) = \overline{D(\lambda)}$.

modes in the Evans system at $x = \pm\infty$ to be 4 and 5, and specifies the adjoint formulation for the Evans system near $x = +\infty$; see (1.20).

Because the Evans function satisfies $D(\overline{\lambda}) = \overline{D(\lambda)}$, the Evans function need only be computed over contour points in the upper right quadrant of the complex plane; see Figure 5.4. The Evans function is then evaluated over the contour for a range of values ξ with the `compute_Evans` function.

To compute the Evans function, growth and decay manifolds must be evolved from $x\pm\infty$ to $x = 0$. Analytic bases for these manifolds are initialized at infinity with a numerical method based on Kato's reduced ODE; see [29] for details and references. `c['ksteps']` specifies the number of Kato steps between points in the complex plane where the Evans function is computed. This value can be increased as needed to ensure that the initializing bases vary smoothly in λ.

```
def driver_Evans_xi(filename,list_xi):

    File_ = open(filename,'rb')
```

```
        s, p = pickle.load(File_)
        File_.close()

        preimage=semicirc2(circpnts=10,imagpnts=10,innerpnts=5,ksteps=c['ksteps'],
                r=0.1,spread=2,inner_radius=5*10**(-3.),lambda_steps=c['lambda_steps'])

        s['output'] = []
        for xi in list_xi:
            p['xi'] = xi
            print("EA, xi = ", p['EA'],xi)
            [s,e,m,c] = emcset(s,'front',4,5,'default')
            preimage2, halfw = compute_Evans(preimage,c,s,p,m,e)

            w = np.empty((2*len(halfw)),dtype='complex')
            halfw = halfw/halfw[0]
            w[:len(halfw)] = halfw
            w[len(halfw):] = conj(np.flipud(halfw))
            try: s['output'].append((xi,int(winding_number(w)),w))
            except: pass
        print("\n")
        for xi,wnd,_ in s['output']:
            print("EA, xi, wnd = ", p['EA'],xi,wnd)
        File = open(filename,'wb')
        pickle.dump([s,p],File); File.close()
```

The MPI4Py library allows us to easily parallelize Evans function computation. For example, running `mpirun -n 5 python RNSdriver.py` on BYU's supercomputer executes the code below 5 times on separate processes, each having a value of `RANK` between 1 and 5.

```
# Parallelizes Evans function computation; since Evans function
# computation is embarrassingly parallel we can assign one call to
# driver_Evans_xi to its own process.
COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()
```

```
# SIZE = COMM.Get_size()
# print("The number of processes is ",SIZE)
L = np.arange(SIZE)
try:
    # Computes the Evans function over a semicircular contour for several values of
         xi
    # Uses pickle to save the output.
    EA = k_EA[RANK,0]
    filename=('profile_nu_1/euler_plot_EA_%1.2f'%EA).replace('.','_')
    driver_Evans_xi(filename,list_xi=array([0.,0.2,0.4,0.6,0.8]))
    print("EA = ", EA)
except:
    print("Error in Evans function computation")
```

5.5.2 RNS profile solver.

Lines 1-10

Mathematical functions are imported from NumPy and SciPy. The BVP solver `bcp6c` requires Jacobian functions for the ODE and the boundary conditions as arguments. This can be done using the `Jacobian` function from Numdifftools.

Lines 11-86

Here the Evans matrix `A` is defined; see (4.29). Note that STABLAB variables `s` and `p` are supplied as arguments. These variables contain system parameters and the numerical solution of the wave profile, respectively.

Lines 88-134

The boundary condition function and its derivative are implemented as `bc` and `bc_jacobian`; see (4.24) and (4.20). We also define the function `Flinear`, which is used to evaluate the Evans matrix at infinity. The results of `Flinear` are used to provide projective conditions for the detonation profiles.

Lines 136-167

The ODE function and its derivative are implemented as Python functions `double_F`, `ode_euler`, `ode_jacobian`, and `phi`; see (4.17) and (4.20).

Lines 169-186

The function `guess_function` provides an initial estimate of the detonation profile when a solution at a nearby parameter value is not available.

Lines 188-213

The function `detonation_profile_solver` pulls everything in `RNS.py` together to compute detonation profiles numerically for given values of STABLAB variables `s` and `p`. Functions `detonation_profile_solver` and `A` are the only ones that need to be explicitly imported from `RNS.py` into `RNSdriver.py`.

```
 1  from __future__ import division
 2  from math import exp
 3  from numpy import linspace, array, zeros, ones, tanh, diff, newaxis
 4  from scipy import linalg
 5  from numdifftools import Jacobian
 6
 7  from core.bin import projection2, soln
 8  from core.pybvp6c.bvp6c import bvp6c, bvpinit, deval
 9  from core.pybvp6c.structure_variable import *
10
11  def A(x,lmbda,s,p):
12      # multiD-rNS Evans matrix in Eulerian coordinates
13      xi = p['xi']
14
15      # profile variables and their derivatives
16      temp = soln(x,s)
17      u, e, y, z = temp[:,0]
18      u_x, e_x, y_x, z_x = ode_euler(0,temp,s,p)
19
```

```
    eta, beta = p['eta'], p['beta']
    mu, nu = p['mu'], p['nu']
    g, q = p['g'], p['q']
    phi_e, Dphi_e = phi(p,e)

    mu2 = 2.*mu + eta
    eta2 = mu + eta
    rho = 1./u
    pressure = g*rho*e
    f1 = pressure + (mu - eta)*u_x
    f2 = q*rho*phi_e
    f3 = lmbda + phi_e
    f4 = lmbda*rho + nu*xi**2.-q*rho*z*Dphi_e

    a11 = -lmbda*rho
    a16 = -lmbda*rho**2.
    a17 = 1j*xi*rho

    a21 = -u_x
    a26 = lmbda*rho + mu*xi**2.

    a31 = -1j*xi*pressure
    a36 = -1j*xi*pressure*rho
    a37 = lmbda*rho + mu2*xi**2.
    a38 = 1j*xi*g*rho

    a41 = -e_x + z*f2
    a42 = -u_x
    a46 = rho*z*f2
    a47 = 1j*xi*f1
    a48 = f4
    a49 = -f2

    a51 = -rho*z*f3
    a56 = -rho**2.*z*f3
    a57 = 1j*xi*rho*z
    a58 = rho*z*Dphi_e
    a59 = rho*(beta*xi**2. + f3)

    a61 = -pressure/mu2
    a62 = 1./mu2
    a66 = (1.-pressure*rho)/mu2
    a67 = -1j*xi*eta2/mu2
    a68 = g*rho/mu2

    a73 = 1./mu
    a76 = -1j*xi*eta2/mu
    a77 = 1./mu

    a84 = 1./nu
    a86 = (pressure - mu2*u_x)/nu
    a88 = 1./nu

    a91 = (beta*z_x - u*z)/beta
    a95 = u/beta
    a96 = rho*z_x
    a99 = u/beta

    return array(   [   [a11, 0,     0,   0,   0,   a16, a17, 0,   0],
                        [a21, 0,     0,   0,   0,   a26, 0,   0,   0],
                        [a31, 0,     0,   0,   0,   a36, a37, a38, 0],
                        [a41, a42,   0,   0,   0,   a46, a47, a48, a49],
                        [a51, 0,     0,   0,   0,   a56, a57, a58, a59],
                        [a61, a62,   0,   0,   0,   a66, a67, a68, 0],
                        [0,   0,     a73, 0,   0,   a76, a77, 0,   0],
                        [0,   0,     0,   a84, 0,   a86, 0,   a88, 0],
                        [a91, 0,     0,   0,   a95, a96, 0,   0,   a99]     ] )

def bc(ya,yb,s,p):
    n = s['n']
    out = zeros(8)
    out[0:n] = ya[s['rarray']]-ya[s['larray']]  #   4 matching conditions
    out[n] = yb[s['larray'][3]]-1.              #   right side condition on z
    out[n+1:n+3] = s['LP'].T.dot(yb[s['rarray']] - s['UR'])     #   projection at -
        infinity, 2 dim
    out[n+3] = ya[0]-0.5*(p['u_minus']+p['u_plus'])         # 1 phase condition
    return out

def bc_jacobian(ya,yb,s,p):
    try:
        ga = Jacobian(lambda z:bc(z,yb,s,p),step=1e-7)
        gb = Jacobian(lambda z:bc(ya,z,s,p),step=1e-7)
    except:
        ga = Jacobian(lambda z:bc(z,yb,s,p),step=1e-9)
        gb = Jacobian(lambda z:bc(ya,z,s,p),step=1e-9)
    return ga(ya), gb(yb)

def Flinear(U,p):
    # Jacobian of the rNS profile ODEs at end states UL or UR.
    u,e,y,z = U
    phi_val, D_phi_val = phi(p,e)

    dudu = (1. - p['g']*e/u**(2.))/(2.*p['mu'] + p['eta'])
    dude = p['g']/(u*(2.*p['mu'] + p['eta']))
    dudy = 0.
    dudz = 0.

    dedu = (1.-u + p['g']*p['e_minus'])/p['nu']
    dede = 1./p['nu']
    dedy = p['q']/p['nu']
    dedz = p['q']/p['nu']

    dydu = y/p['beta'] + z*phi_val/u**(2.)
    dyde = -z*D_phi_val/u
    dydy = u/p['beta']
    dydz = -phi_val/u

    dzdu = -y/p['beta']
    dzde = 0.
    dzdy = -u/p['beta']
    dzdz = 0.

    return array([ [dudu,   dude,   dudy,     dudz],
                   [dedu,   dede,   dedy,     dedz],
                   [dydu,   dyde,   dydy,     dydz],
                   [dzdu,   dzde,   dzdy,     dzdz]   ])

def ode_jacobian(x,y,s,p):
    out = zeros((8,8))
    out[:4,:4] = (1.*s['R'])*Flinear(y[s['rarray']],p)
    out[4:,4:] = (1.*s['L'])*Flinear(y[s['larray']],p)
    return out

def double_F(x,y,s,p):
    n = s['n']; out = zeros(2*n)
    out[0:n] = (s['R']/s['I'])*ode_euler(x,y[s['rarray']],s,p)
    out[n:] = (s['L']/s['I'])*ode_euler(x,y[s['larray']],s,p)
    return out

def ode_euler(x,w,s,p):
    u,e,y,z = w
    phi_val, _ = phi(p,e)

    # Note that k does not show up in ODE equations because it is defined
    # in the phi function.
    return array( [ ((u-1.) + p['g']*(e/u - p['e_minus']))/(2.*p['mu'] + p['eta']),
            ((e-p['e_minus'])-.5*(u-1.)**2.+(u-1.)*p['g']*p['e_minus']+p['q']*(y+z
                -1.) )/p['nu'],
            u*y/p['beta'] - z*phi_val/u,
            -u*y/p['beta'] ] )

def phi(p,e):
    # Ignition function
    T = e/p['c']        # Here p.c := 1.
    if T > p['Ti']:
        phi = p['k']*exp(-p['EA']/(T-p['Ti']))
        Dphi = (p['EA']/(T-p['Ti'])**2)*phi
        return phi, Dphi
    else:
        return 0., 0.

def guess_function(x,s,p):
    slope = array([8,5,1.3,0.8]); slope = slope[:,newaxis]
    intercept = array([0,1.25,0.4,0.9]); intercept = intercept[:,newaxis]
    a = (0.5*(s['UR']+s['UL'])*ones((len(x),1))).T
    c = (0.5*(s['UR']-s['UL'])*ones((len(x),1))).T

    out = zeros((2*s['n'],len(x)))
    out[0:4] = a+c*tanh(slope*x*(s['R']/s['I'])-intercept*ones(x.shape))
    out[4:8] = a+c*tanh(slope*x*(s['L']/s['I'])-intercept*ones(x.shape))

    dzr = diff(out[3,:])/diff(x)    # y =-beta*z'/u
    temp = zeros(len(x)); temp[:-1] = dzr; temp[-1] = dzr[-1];
    out[2,:] = (s['I']/s['R'])*(-p['beta'])*temp/out[0,:]

    dzl = diff(out[7,:])/diff(x)
    temp[:-1] = dzl; temp[-1] = dzl[-1];
    out[6,:] = (s['I']/s['L'])*(-p['beta'])*temp/out[4,:]
    return out

def detonation_profile_solver(s,p):
    # Solves detonations in Eulerian coordinates
    AM = Flinear(s['UL'],p)
    s['LM'] = linalg.orth(projection1(AM,-1,0)[0].T)
    AP = Flinear(s['UR'],p)
    s['LP'] = linalg.orth(projection1(AP,1,-1e-8)[0].T)

    pre_double_F= lambda x,y: double_F(x,y,s,p)
    pre_bc=lambda ya,yb: bc(ya,yb,s,p)
    pre_ode_jacobian=lambda x,y: ode_jacobian(x,y,s,p)
    pre_bc_jacobian=lambda x,y: bc_jacobian(x,y,s,p)

    options = struct()
    options.abstol, options.reltol = 1e-9, 1e-8
    options.nmax, options.stats = 20000, 'on'
    options.fjacobian  = pre_ode_jacobian
    options.bcjacobian = pre_bc_jacobian

    xarray = linspace(0,1,150)**(2.)
    if 'sol' not in s.keys():
        guess_array = guess_function(xarray,s,p)
    else:
        guess_array, _ = deval(s['sol'],xarray)
    solinit = bvpinit(xarray,guess_array)
    sol = bvp6c(pre_double_F,pre_bc,solinit,options)
    return sol, p
```

Listing 5.2: RNS.py

Bibliography

[1] J. C. Alexander and R. Sachs, *Linear instability of solitary waves of a Boussinesq-type equation: a computer assisted computation*, Nonlinear World, 2 (1995), pp. 471–507.

[2] B. Barker, *Evans function computation.*, Master's thesis, Brigham Young University, Provo, 2009.

[3] B. Barker, J. Humpherys, G. Lyng, and K. Zumbrun, *Viscous hyperstabilization of detonation waves in one space dimension*, SIAM J. Appl. Math., 75 (2015), pp. 885–906.

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

[5] B. Barker, J. Humpherys, K. Rudd, and K. Zumbrun, *Stability of viscous shocks in isentropic gas dynamics*, Comm. Math. Phys., 281 (2008), pp. 231–249.

[6] B. Barker, J. Humpherys, and K. Zumbrun, *STABLAB: A MATLAB-Based Numerical Library for Evans Function Computation*, July 2009. http://www.impact.byu.edu/stablab/.

[7] J. C. Bronski, *Semiclassical eigenvalue distribution of the Zakharov-Shabat eigenvalue problem*, Phys. D, 97 (1996), pp. 376–397.

[8] J. R. Dormand and P. J. Prince, *A family of embedded Runge-Kutta formulae*, J. Comput. Appl. Math., 6 (1980), pp. 19–26.

[9] J. J. Erpenbeck, *Stability of steady-state equilibrium detonations*, Phys. Fluids, 5 (1962), pp. 604–614.

[10] ——, *Stability of idealized one-reaction detonations*, Phys. Fluids, 7 (1964), pp. 684–696.

[11] A. Ghazaryan, J. Humpherys, and J. Lytle, *Spectral behavior of combustion fronts with high exothermicity*, SIAM J. Appl. Math., 73 (2013), pp. 422–437.

[12] N. Hale and D. R. Moore, *A sixth-order extension to the matlab package bvp4c of j. kierzenka and l. shampine*, Tech. Rep. NA-08/04, Oxford University Computing Laboratory, May 2008.

[13] D. Henry, *Geometric theory of semilinear parabolic equations*, vol. 840 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1981.

[14] P. Howard and K. Zumbrun, *Pointwise estimates and stability for dispersive-diffusive shock waves*, Arch. Ration. Mech. Anal., 155 (2000), pp. 85–169.

[15] J. Humpherys, *On the shock wave spectrum for isentropic gas dynamics with capillarity*, J. Differential Equations, 246 (2009), pp. 2938–2957.

[16] J. Humpherys, O. Lafitte, and K. Zumbrun, *Stability of isentropic Navier-Stokes shocks in the high-Mach number limit*, Comm. Math. Phys., 293 (2010), pp. 1–36.

[17] J. Humpherys, G. Lyng, and K. Zumbrun, *Multidimensional stability of large-amplitude navier-stokes shocks*.

[18] J. Humpherys, G. Lyng, and K. Zumbrun, *Spectral stability of ideal-gas shock layers*, Arch. Ration. Mech. Anal., 194 (2009), pp. 1029–1079.

[19] J. Humpherys and J. Lytle, *Root following in Evans function computation*, SIAM J. Numer. Anal., 53 (2015), pp. 2329–2346.

[20] J. Humpherys and K. Zumbrun, *An efficient shooting algorithm for Evans function calculations in large systems*, Phys. D, 220 (2006), pp. 116–126.

[21] J. C. Humpherys, *Spectral energy methods and the stability of shock waves*, ProQuest LLC, Ann Arbor, MI, 2002. Thesis (Ph.D.)–Indiana University.

[22] J. D. Hunter, *Matplotlib: A 2d graphics environment*, Computing In Science & Engineering, 9 (2007), pp. 90–95.

[23] T. Kato, *Perturbation theory for linear operators*, Classics in Mathematics, Springer-Verlag, Berlin, 1995. Reprint of the 1980 edition.

[24] C. Mascia and K. Zumbrun, *Pointwise Green's function bounds and stability of relaxation shocks*, Indiana Univ. Math. J., 51 (2002), pp. 773–904.

[25] ——, *Pointwise Green function bounds for shock profiles of systems with real viscosity*, Arch. Ration. Mech. Anal., 169 (2003), pp. 177–263.

[26] R. L. Pego and M. I. Weinstein, *Eigenvalues, and instabilities of solitary waves*, Philos. Trans. Roy. Soc. London Ser. A, 340 (1992), pp. 47–94.

[27] A. Pogan, J. Yao, and K. Zumbrun, *$O(2)$ Hopf bifurcation of viscous shock waves in a channel*, Phys. D, 308 (2015), pp. 59–79.

[28] D. H. Sattinger, *On the stability of waves of nonlinear parabolic systems*, Advances in Math., 22 (1976), pp. 312–355.

[29] K. Zumbrun, *A local greedy algorithm and higher-order extensions for global numerical continuation of analytically varying subspaces*, Quart. Appl. Math., 68 (2010), pp. 557–561.

[30] K. Zumbrun and P. Howard, *Pointwise semigroup methods and stability of viscous shock waves*, Indiana Univ. Math. J., 47 (1998), pp. 741–871.