SYMBOLIC VERIFICATION OF OPERATOR AND MATRIX LAX PAIRS FOR SOME COMPLETELY INTEGRABLE NONLINEAR PARTIAL DIFFERENTIAL EQUATIONS

by

Jennifer Larue

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematical and Computer Sciences).

Golden, Colorado
Date _____

Signed: _____
Jennifer Larue

Signed: _____
Dr. Willy Hereman
Thesis Advisor

Golden, Colorado
Date _____

Signed: _____
Dr. Tracy Camp
Professor and Head
Department of Mathematical and Computer Sciences

ABSTRACT

A completely integrable nonlinear partial differential equation (PDE), such as the Korteweg-de Vries equation, can be replaced by a system of linear PDEs in an auxiliary function whose compatibility requires that the original nonlinear PDE holds. That system of compatible linear PDEs is called a Lax pair. Two equivalent formulations are being investigated: one uses a Lax pair of differential operators leading to linear scalar PDEs of high order; the other uses a Lax pair of matrices leading to first-order PDEs.

The purpose of this research project is to design and implement software in *Mathematica* to symbolically test previously found Lax pairs in both operator and matrix forms and to help correct minor errors in the operator form. Furthermore, if a candidate Lax pair with undetermined coefficients is given, the software assists with computing these coefficients. For the class of evolution equations of fifth-order, it is shown how candidate Lax pairs (in operator form) can be generated using the scaling symmetry of the nonlinear PDE. Using this method, we find an extra Lax pair for the Sawada–Kotera equation, which we did not expect. We also showed that the specific instances of this class of equations that were known in the literature are the only instances that admit Lax pairs of certain forms. The outcomes of this project will facilitate future investigations of methods to compute and verify Lax pairs as well as conservation laws of novel completely integrable PDEs.

TABLE OF CONTENTS

LIST OF SYMBOLS

General nonlinear PDE . . . . . $\Delta = 0$

First operator in a Lax pair . . . . . $\mathcal{L}$

Second operator in a Lax pair . . . . . $\mathcal{M}$

Eigenfunction for an operator Lax pair . . . . . ψ

Total derivative operator for the space variable x . . . . . $\mathcal{D}_x$

Repeated applications of $\mathcal{D}_x$ n times . . . . . $\mathcal{D}_x^n$

Identity Operator . . . . . $\mathcal{I}$

Zero Operator . . . . . $\mathcal{O}$

First matrix in a Lax pair . . . . . $\mathbf{X}$

Second matrix in a Lax pair . . . . . $\mathbf{T}$

Vector containing eigenfunctions for a matrix Lax pair . . . . . Ψ

Arbitrary square matrix for gauge transformation . . . . . $\mathbf{G}$

Constant spectral parameter . . . . . λ

Equal when evaluated on solutions of the nonlinear PDE ($\Delta = 0$) . . . . . $\doteq$

Commutator of a and b, i.e., $ab - ba$ . . . . . $[a, b]$

Imaginary number ($\sqrt{-1}$) i

LIST OF ABBREVIATIONS

Inverse scattering transform method	IST method
Partial differential equation	PDE
Ordinary differential equation	ODE
Camassa-Holm equation	CH equation
Harry Dym equation	HD equation
Hirota–Satsuma equation	HS equation
Kadomtsev-Petviashvili equation	KP equation
Kaup-Kuperschmidt equation	KK equation
Korteweg–de Vries equation	KdV equation
Liouville equation	LV equation
Nonlinear Schrödinger equation	NLS equation
Sawada-Kotera equation	SK equation
Short pulse equation	SP equation
Sine–Gordon equation	SG equation
Sinh–Gordon equation	ShG equation
Ziber-Shabat-Mikhailov system	ZSM system

ACKNOWLEDGEMENTS

Thanks to my advisor, Dr. Willy Hereman, and to the undergraduate students who helped search for Lax pairs: Jacob Rezac, William (Tony) McCollom, and Janeen Neri.

I also wish to thank the Department of Mathematical and Computer Science at CSM for their support. I especially wish to thank Jody Lowther for her help in all paperwork and non-technical areas.

Thanks also goes to the National Science Foundation for their support. My research was supported in part under award no. CCF-0830783.

Thanks to my brother, Justin de Vesine, for checking readability of my code and supporting me in general.

Many thanks to my parents, David M. Larue and Suzanne Wolfram, for supporting me throughout this endeavor and to Trisha, my best friend, for believing in me and keeping me going.

Jennifer Larue

Colorado School of Mines

jwlarue@pobox.com

May 2011

Dedicated to my father, David M. Larue.

This project would not have been possible without you.

CHAPTER 1

INTRODUCTION

In this chapter, we will first outline a history of wave equations and Lax pairs. Then we will discuss the flow of the rest of the thesis. Nonlinear partial differential equations (PDEs) are notoriously hard to solve and it seems that no general algorithm to solve nonlinear PDEs exists. However, there are certain types of nonlinear PDEs whose initial value problems can be solved using the inverse scattering transform (IST) method: see [1] for more information. Lax pair, as defined and studied in this work, form the cornerstone of the IST method and are instrumental discovering conservation laws. If the nonlinear PDE is of first order in time, it is usually referred to as an evolution equation. Exact solutions in terms of elementary functions are available for some of these equations. This can be useful for understanding the nonlinear model better and can also be used to test the accuracy of numerical solving methods.

Integrable nonlinear PDEs are found in numerous physical problems. For example, the Korteweg-de Vries (KdV) equation and the modified KdV (mKdV) describe shallow water waves, ion acoustic waves in plasmas, and many more. The nonlinear Schrödinger (NLS) equation models surface waves in deep water and nonlinear optics as well as many others. The sine-Gordon (SG) equation governs the swing in a line of identical pendulums hanging from a horizontal torsion wire that can move in vertical planes perpendicular to the wire due to gravity [2] as well as other nonlinear phenomena.

A completely integrable nonlinear PDE, such as the Korteweg-de Vries equation, can be replaced by a system of linear PDEs in an auxiliary function whose compatibility requires that the original nonlinear PDE holds. That system of compatible linear PDEs is called a Lax pair. Two equivalent formulations are being investigated: one uses a Lax pair of differential operators leading to linear scalar PDEs of high

order; the other uses a Lax pair of matrices leading to first-order PDEs.

The purpose of this research project is to design and implement software in *Mathematica* to symbolically test previously found Lax pairs in both operator and matrix forms and to help correct minor errors in the operator form. Furthermore, if a Lax pair with undetermined coefficients is given, the software computes these coefficients. For the class of evolution equations of fifth-order, it is shown how candidate Lax pairs (in operator form) can be generated using the scaling symmetry of the nonlinear PDE. Using this method, we find an extra Lax pair for the Sawada–Kotera (SK) equation, which we did not expect. We also showed that the specific instances of this class of equations that were known in the literature are the only instances that admit Lax pairs of certain forms. The outcomes of this project will facilitate future investigations of methods to compute and verify Lax pairs as well as conservation laws of novel completely integrable PDEs.

To test if a Lax pair is valid for a PDE is fairly straightforward when done by hand; however, the computations quickly get very long and complicated. Later in this thesis, we will take the simplest example known (the KdV equation) to illustrate the computations. Even for this simple case, the computations are quite complicated, so we want to let the computer do the symbolic computations. The goal of this master's project is to take given various Lax pairs from the literature (or ones that were computed by some other means) and, after putting in a standard form if necessary, test the validity of each Lax pair using the software we developed. Letting the computer do the routine but lengthy computations prevents mathematical errors. As with many mathematical computations, a small error at the beginning would cascade throughout the rest of the test, to the frustration of the researcher carrying out the calculation!

It is not uncommon to find in the literature examples of Lax pairs where the base form of the operators are correct, but there are minor errors in the signs or the coefficients. Assuming the base form of the operator Lax pair is correct, the user can

enter the Lax pair with unspecified constants, and then the software developed for this thesis can be used to help determine the value of those constants (or relations between those constants) that would give a correct Lax pair.

1.1 Background

Solitons were first observed by J. Scott Russell, a Scottish navel engineer, in 1834. He recounted his observations in a report [3] to the British Association for the Advancement of Science in 1844 as follows:

> I believe I shall introduce this phenomenon by describing the circumstances of my own first acquaintance with it. I was observing the motion of a boat which was rapidly drawn along a narrow channel by a pair of horses, when the boat suddenly stopped – not so the mass of water in the channel which it had put in motion; it accumulated round the prow of the vessel in a state of violent agitation, then suddenly leaving it behind, rolled forward with great velocity, assuming the form of a large solitary elevation, a rounded, smooth and well-defined heap of water, which continued its course along the channel apparently without change of form or diminution of speed. I followed it on horseback, and overtook it still rolling on at a rate of some eight or nine miles an hour, preserving its original figure some thirty feet long and a foot to a foot and a half in height. Its height gradually diminished, and after a chase of one or two miles I lost it in the windings of the channel. Such, in the month of August 1834, was my first chance interview with that singular and beautiful phenomenon which I have called the Wave of Translation, a name which it now very generally bears.

However, the scientific community of his time did not seem impressed by Russell's discovery. His contemporary George Airy strongly disagreed with Russell's findings

and, in fact, did not believe that solitary water waves existed [4].

About 50 years later, the controversy over solitary water waves was finally resolved by Korteweg and de Vries (1895). They derived the unidirectional equation (now called the KdV equation) which governs moderately small, shallow water waves and has solutions which include solitary waves. It is important to note that, independently, Boussinesq (1872) and Rayleigh (1876) derived the bi-directional version of the KdV equation and found the hyperbolic secant-squared solution for the free surface [5]. In 1963, Kruskal and Zabusky [6] carried out experiments to understand the redistribution of the energy of masses connected by springs in a lattice. They assumed that the interaction force exerted by the springs is exponential (instead of linear as in Hooke's law). In the continuous limit, i.e., when the distance between the neighboring masses goes to zero, the vibration of the masses is governed by the KdV equation. A remarkable phenomenon was noticed. When these solitary waves collide with each other, they appear to scramble; yet they emerge from the interaction retaining their former shapes and speeds except for a phase shift. Indeed, the larger wave is further ahead than it would have been with without interaction; similarly, the smaller wave is further behind [1]. The unusual collision phenomenon of the solitary waves inspired Kruskal to name these special solutions *solitons* [7]. That name is chosen well because it captures the *solitary* as well as the *partial-like* behavior of the waves.

Solitary waves and solitons are solutions of a nonlinear PDE or system of PDEs. These PDEs are completely integrable, which means that they have infinitely many conservation laws and symmetries. They can be solved with the IST method which is a nonlinear analog of the Fourier transform (for more information, see [8]).

Peter Lax became interested in the mathematical properties of the nonlinear PDEs and their solutions. In his seminal 1968 paper [9], Lax introduced the concept of a Lax pair as a way to "linearize" these complicated PDEs. A Lax pair takes a higher order,

completely integrable, non-linear PDE and expresses it as a system of *linear* equations involving a pair of differential operators or as a system of first-order linear differential equations written in matrix form. The operators, denoted by $\mathcal{L}$ and $\mathcal{M}$, allow us to replace the original PDE by a system of high order linear equations in an auxiliary function, whose compatibility requires that the original nonlinear PDE holds. This circumvents the difficulty of working with a *nonlinear* PDE but introduces the issue of working with differential operators. The matrix, or zero-curvature, formulation, denoted by $\mathbf{X}$ and $\mathbf{T}$, reduces the original PDE to a first-order, linear problem and only requires basic matrix operations; however, for a n^{th} order $\mathcal{L}$ there will be n^2 entries in each $\mathbf{X}$ and $\mathbf{T}$. For instance, in many of our examples $\mathcal{L}$ is second order, so $\mathbf{X}$ and $\mathbf{T}$ are 2×2 matrices. Each of the entries in these matrices can get long and complicated. This is the drawback of using the matrix formulation.

If a non-trivial Lax pair can be found for a given PDE, then that PDE is guaranteed to be completely integrable, i.e., it has an infinite number of conservation laws and soliton solutions at any order. Having a Lax pair is essential for the application of the IST. Furthermore, other researchers have developed algorithms to compute conservation laws from Lax pairs [10, 11].

1.2 Outline of thesis

The thesis is organized as follows. In Chapter 2, we will show multiple compatibility tests for both operator and matrix Lax pairs, first in general and then with specific examples. We show that matrix Lax pairs are not unique; they can be converted using a gauge transformation. We will also outline the methods to convert Lax pairs in operator form to matrix form and vice versa.

Chapter 3 shows examples of PDEs and their known Lax pairs (either of operator form, matrix form, or both) that have been tested successfully using our *Mathematica* package, **laxpairtester.m** [12], and related routines and data files that were devel-

oped for this project.

Chapter 4 describes the algorithms used in **laxpairtester.m**. Each section of the code was written as a separate `Module` in *Mathematica*. This approach not only prevents conflicts when using the same variables but also makes the software more adaptable to different applications.

Chapter 5 shows the flexibility of **laxpairtester.m** by studying the class of evolution equations of fifth-order PDEs, which includes three different fully integrable equations for specific ratios of the parameters. Using the scaling symmetry of the nonlinear PDE, we generate candidate Lax pairs for each of the three fully integrable equations. We then use **laxpairtester.m** to help determine the unspecified coefficients of each Lax pair. With the help of our software, we discovered what we believe to be a previously unknown Lax pair for the SK equation. Finally, we take an unknown Lax pair in operator form which includes the base form of each of the three equations' Lax pairs, with unspecified coefficients and obtain all three parameter constraints and each Lax pair.

Reflecting on the research we performed and the software we developed, we will draw some conclusions in Chapter 6 and indicated areas of future research.

CHAPTER 2

DERIVATION OF LAX EQUATIONS

The purpose of this chapter is to introduce the concept of Lax pairs and show a number of derivations and tests for operator and matrix Lax pairs. First, we will demonstrate how to derive the Lax equation for the operator formulation and how the Lax equation can be used to check an operator Lax pair for the KdV equation. Next, we will do a similar derivation of the Lax equation for the matrix formulation and show the test used on a matrix Lax pair for the KdV equation. We will show that a matrix Lax pair is not unique, and it can be converted and possibly simplified using a gauge transformation. We will outline a method to convert an operator Lax pair into a matrix Lax pair using the relationships between the formulations, and we will show a similar method to convert back, from matrix to operator. We will show a second method for testing an operator Lax pair that is direct but would be complicated to code or use. Finally, we will show an alternate form of an operator Lax pair that is used in the literature regularly.

In this chapter, we will consider nonlinear PDEs, $\Delta = 0$, in one space variable, x, and one time variable, t.

For example, $\Delta = u_t + \alpha u u_x - \beta u_{xx} = 0$, which is called the Burgers equation for $u(x,t)$. Subscripts denote partial derivatives and $u_{nx} = \frac{\partial^n u}{\partial x^n}$, n integer.

2.1 Compatibility test for an operator Lax pair

To have a valid Lax pair for a given nonlinear PDE, $\Delta = 0$, the following linear equations must hold:

$$\mathcal{L}\psi = \lambda\psi \tag{2.1}$$

and

$$\psi_t = \mathcal{M}\psi, \tag{2.2}$$

where $\mathcal{L}$ and $\mathcal{M}$ are linear differential operators, $\mathcal{L}$ and $\mathcal{M}$ preferably do not, but may, contain λ, λ is an eigenvalue of $\mathcal{L}$ ($\lambda_x = 0$), and ψ is an eigenfunction of $\mathcal{L}$.

The pair $(\mathcal{L}, \mathcal{M})$ is called the Lax pair of the nonlinear PDE of equations. (2.1) and (2.2) are compatible on the solutions of the nonlinear PDE. Equation (2.1) is a Schrödinger equation for ψ; equation (2.2) governs the time evolution of ψ. We assume that the problem is *isospectral*, which means that the eigenvalues do not change in time, i.e., $\lambda_t = 0$.

Now we derive the Lax equation to be satisfied by $\mathcal{L}$ and $\mathcal{M}$. If we differentiate (2.1) with respect to t, we get

$$\mathcal{L}_t\psi + \mathcal{L}\psi_t = \lambda\psi_t. \tag{2.3}$$

Substituting (2.2) into (2.3) yields

$$\mathcal{L}_t\psi + \mathcal{L}\mathcal{M}\psi = \lambda\mathcal{M}\psi = \mathcal{M}\lambda\psi. \tag{2.4}$$

Substituting (2.1) into (2.4) gives

$$\begin{aligned} \mathcal{L}_t\psi + \mathcal{L}\mathcal{M}\psi &= \mathcal{M}\mathcal{L}\psi, \\ \mathcal{L}_t\psi + \mathcal{L}\mathcal{M}\psi - \mathcal{M}\mathcal{L}\psi &= 0, \\ (\mathcal{L}_t + [\mathcal{L}, \mathcal{M}])\,\psi &\doteq 0, \end{aligned} \tag{2.5}$$

where $\doteq$ means "evaluated on the given nonlinear PDE" or we have substituted any solution of the original PDE, $\Delta = 0$, into (2.5), that must hold for all ψ. Hence,

$$\mathcal{L}_t + [\mathcal{L}, \mathcal{M}] \doteq \mathcal{O}, \tag{2.6}$$

where $\mathcal{O}$ is the zero operator. Equation (2.6) is called the Lax equation, and only evaluates to $\mathcal{O}$ when $\Delta = 0$. If the solution of $\Delta = 0$ is not substituted, then in most

cases, $\mathcal{L}_t + [\mathcal{L}, \mathcal{M}] = c\Delta\mathcal{I}$ where c is a constant and $\mathcal{I}$ is the identity operator. $\mathcal{L}$ and $\mathcal{M}$ are non-commutating differential operators, so we have to be careful about the evaluation of the Lax equation. $\mathcal{L}_t$ denotes the partial derivative of $\mathcal{L}$ with respect to time. It can be computed as follows: since $(\mathcal{L}\psi)_t = \mathcal{L}_t\psi + \mathcal{L}\psi_t$ it follows that

$$\mathcal{L}_t\psi = (\mathcal{L}\psi)_t - \mathcal{L}\psi_t.$$

The commutator, as defined by

$$[\mathcal{L}, \mathcal{M}] = \mathcal{L}\mathcal{M} - \mathcal{M}\mathcal{L},$$

can be computed with operator calculus, or alternatively, with differential calculus since

$$[\mathcal{L}, \mathcal{M}]\psi = \mathcal{L}\mathcal{M}\psi - \mathcal{M}\mathcal{L}\psi.$$

Now we turn to the most famous example. For the Korteweg-de Vries (KdV) equation [2],

$$\Delta = u_t + \alpha u u_x + u_{3x} = 0, \tag{2.7}$$

where $u = u(x, t)$ and α is a real parameter, it is well-known [9] that

$$\mathcal{L} = \mathcal{D}_x^2 + \frac{1}{6}\alpha u\mathcal{I} \tag{2.8}$$

and

$$\mathcal{M} = -4\mathcal{D}_x^3 - \alpha u\mathcal{D}_x + \left(a(t) - \frac{1}{2}\alpha u_x\right)\mathcal{I} \tag{2.9}$$

where $\mathcal{D}_x$ is the total derivative operator for the space variable x. $\mathcal{D}_x^n$ denotes repeated applications of $\mathcal{D}_x$ (n times). $\mathcal{I}$ is the identity operator, and $a(t)$ is an arbitrary

function of time t. It is straightforward to compute

$$\mathcal{L}_t = \frac{1}{6}\alpha u_t \mathcal{I}, \tag{2.10}$$

$$\begin{aligned} \mathcal{L}\mathcal{M} &= -4\mathcal{D}_x^5 - \tfrac{5}{3}\alpha u \mathcal{D}_x^3 + \left(a(t) - \tfrac{5}{2}\alpha u_x\right)\mathcal{D}_x^2 \\ &\quad - \left(\tfrac{1}{6}\alpha^2 u^2 + 2\alpha u_{xx}\right)\mathcal{D}_x \\ &\quad + \left(\tfrac{1}{6}\alpha a(t) u - \tfrac{1}{12}\alpha^2 u u_x - \tfrac{1}{2}\alpha u_{3x}\right)\mathcal{I}, \end{aligned} \tag{2.11}$$

and

$$\begin{aligned} \mathcal{M}\mathcal{L} &= -4\mathcal{D}_x^5 - \tfrac{5}{3}\alpha u \mathcal{D}_x^3 + \left(a(t) - \tfrac{5}{2}\alpha u_x\right)\mathcal{D}_x^2 \\ &\quad - \left(\tfrac{1}{6}\alpha^2 u^2 + 2\alpha u_{xx}\right)\mathcal{D}_x \\ &\quad + \left(\tfrac{1}{6}\alpha a(t) u - \tfrac{1}{4}\alpha^2 u u_x - \tfrac{2}{3}\alpha u_{3x}\right)\mathcal{I}. \end{aligned} \tag{2.12}$$

Hence,

$$\mathcal{L}_t + [\mathcal{L}, \mathcal{M}] = \frac{1}{6}\alpha\left(u_t + \alpha u u_x + u_{3x}\right)\mathcal{I}. \tag{2.13}$$

Notice that (2.13) is indeed a constant times (2.7), so when we substitute a solution of (2.7) we get $\mathcal{O}$. Therefore, (2.8) and (2.9) form a valid Lax pair for (2.7), which means that

$$\mathcal{L}\psi = \psi_{xx} + \frac{1}{6}\alpha u \psi = \lambda\psi \tag{2.14}$$

and

$$\psi_t = \mathcal{M}\psi = -4\psi_{3x} - \alpha u \psi_x + \left(a(t) - \frac{1}{2}\alpha u_x\right)\psi \tag{2.15}$$

are compatible modulo (2.7).

2.2 Compatibility test for a matrix Lax pair

For $\mathbf{X}$ and $\mathbf{T}$ to be a valid matrix Lax pair, the following vector equations must hold:

$$\Psi_x = \mathbf{X}\Psi \tag{2.16}$$

and

$$\Psi_t = \mathbf{T}\Psi. \tag{2.17}$$

where Ψ is a vector. Both $\mathbf{X}$ and $\mathbf{T}$ will be dependent on λ. The number of components of Ψ is determined by the order of $\mathcal{L}$. When $\mathcal{L}$ is of order 2, the vector contains two components,

$$\Psi = \begin{bmatrix} \psi \\ \varphi \end{bmatrix}, \tag{2.18}$$

and the matrices $\mathbf{X}$ and $\mathbf{T}$ will be square of size two. When $\mathcal{L}$ is of order n, the vector contains n components. If we cross-differentiate (2.16) and (2.17), we get

$$\begin{aligned} \Psi_{xt} &= \mathbf{X}_t\Psi + \mathbf{X}\Psi_t, \\ \Psi_{tx} &= \mathbf{T}_x\Psi + \mathbf{T}\Psi_x. \end{aligned} \tag{2.19}$$

Hence,

$$\mathbf{X}_t\Psi + \mathbf{X}\Psi_t = \mathbf{T}_x\Psi + \mathbf{T}\Psi_x \tag{2.20}$$

or

$$\mathbf{X}_t\Psi - \mathbf{T}_x\Psi + \mathbf{X}\Psi_t - \mathbf{T}\Psi_x = \mathbf{0}. \tag{2.21}$$

Substituting (2.16) and (2.17) into (2.21) gives

$$\left(\mathbf{X}_t - \mathbf{T}_x + [\mathbf{X}, \mathbf{T}]\right)\Psi = \mathbf{0}, \tag{2.22}$$

where $[\mathbf{X}, \mathbf{T}] = \mathbf{XT} - \mathbf{TX}$ is the commutator of the matrices $\mathbf{X}$ and $\mathbf{T}$. If we eliminate Ψ from (2.22), we get the matrix Lax equation, or zero-curvature equation,

$$\mathbf{X}_t - \mathbf{T}_x + [\mathbf{X}, \mathbf{T}] \doteq \mathbf{0}. \tag{2.23}$$

Using (2.23), we can test if some matrices $\mathbf{X}$ and $\mathbf{T}$ are a valid Lax pair for $\Delta = 0$; i.e., the zero-curvature equation, (2.23), should be satisfied if and only if $\Delta = 0$.

$\mathbf{X}$ and $\mathbf{T}$ are square matrices, so (2.23) is straightforward to evaluate.

For example, using the KdV equation (2.7), it is well-known [13] that

$$\mathbf{X} = \begin{pmatrix} 0 & 1 \\ \lambda - \frac{1}{6}\alpha u & 0 \end{pmatrix} \tag{2.24}$$

and

$$\mathbf{T} = \begin{pmatrix} a(t) + \frac{1}{6}\alpha u_x & -\left(4\lambda + \frac{1}{3}\alpha u\right) \\ -4\lambda^2 + \frac{1}{3}\alpha\lambda u + \frac{1}{18}\alpha^2 u^2 + \frac{1}{6}\alpha u_{xx} & a(t) - \frac{1}{6}\alpha u_x \end{pmatrix}. \tag{2.25}$$

Let us now verify that (2.24) and (2.25) indeed are a valid matrix Lax pair for (2.7). Thus, we compute

$$\mathbf{X}_t = \frac{1}{6}\alpha \begin{pmatrix} 0 & 0 \\ -u_t & 0 \end{pmatrix}, \tag{2.26}$$

$$\mathbf{T}_x = \frac{1}{6}\alpha \begin{pmatrix} u_{xx} & -2u_x \\ \frac{2}{3}\alpha u u_x + 2\lambda u_x + u_{3x} & -u_{xx} \end{pmatrix}, \tag{2.27}$$

and

$$\mathbf{XT} - \mathbf{TX} = -\frac{1}{6}\alpha \begin{pmatrix} -u_{xx} & 2u_x \\ \left(-2\lambda + \frac{1}{3}\alpha u\right) u_x & u_{xx} \end{pmatrix}. \tag{2.28}$$

Hence,

$$\mathbf{X}_t - \mathbf{T}_x + \mathbf{XT} - \mathbf{TX} = -\frac{1}{6}\alpha \begin{pmatrix} 0 & 0 \\ u_t + u_{3x} + \alpha u u_x & 0 \end{pmatrix}. \tag{2.29}$$

Notice that (2.29) is nearly the zero matrix; it only has the original PDE in the bottom left entry. Once we substitute a solution of (2.7), equation (2.29) yields the zero matrix. Since the Lax equation evaluates to $\mathbf{0}$, the given $\mathbf{X}$ and $\mathbf{T}$ are indeed a

valid Lax pair for the KdV equation.

2.3 Gauge equivalent Lax pairs in matrix form

To show that $\mathbf{X}$ and $\mathbf{T}$ are not unique, we will first derive and then use the following gauge transformation:

$$\tilde{\mathbf{X}} = \mathbf{G}\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}_x\mathbf{G}^{-1}, \tag{2.30}$$

$$\tilde{\mathbf{T}} = \mathbf{G}\mathbf{T}\mathbf{G}^{-1} + \mathbf{G}_t\mathbf{G}^{-1}, \tag{2.31}$$

where $\mathbf{G}(x,t)$ is an arbitrary invertible matrix.

To derive $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{T}}$, let

$$\tilde{\Psi} = \mathbf{G}\Psi \tag{2.32}$$

be our transformation for *any* matrix $\mathbf{G}$. For $(\tilde{\mathbf{X}}, \tilde{\mathbf{T}})$ to be a valid Lax pair

$$\tilde{\Psi}_x = \tilde{\mathbf{X}}\tilde{\Psi}, \tag{2.33}$$

$$\tilde{\Psi}_t = \tilde{\mathbf{T}}\tilde{\Psi} \tag{2.34}$$

must be true. Substituting (2.32) into the right hand side of (2.33) yields

$$\tilde{\Psi}_x = \tilde{\mathbf{X}}\mathbf{G}\Psi. \tag{2.35}$$

Taking the derivative of (2.32) with respect to x yields

$$\tilde{\Psi}_x = (\mathbf{G}\Psi)_x \tag{2.36}$$

Equating (2.35) and (2.36) gives

$$\tilde{\mathbf{X}}\mathbf{G}\Psi \;=\; (\mathbf{G}\Psi)_x \;=\; \mathbf{G}_x\Psi + \mathbf{G}\Psi_x.$$

Replacing Ψ_x from (2.16), we get

$$\tilde{\mathbf{X}}\mathbf{G}\Psi \;=\; \mathbf{G}_x\Psi + \mathbf{G}\mathbf{X}\Psi \;=\; (\mathbf{G}_x + \mathbf{G}\mathbf{X})\Psi.$$

In order to cancel $\mathbf{G}\Psi$ from both sides of this equation, we will insert the identity matrix $\mathbf{I} = \mathbf{G}^{-1}\mathbf{G}$ on the right hand side. Thus

$$\tilde{\mathbf{X}}\mathbf{G}\Psi \;=\; (\mathbf{G}_x + \mathbf{G}\mathbf{X})\mathbf{G}^{-1}\mathbf{G}\Psi \;=\; (\mathbf{G}_x\mathbf{G}^{-1} + \mathbf{G}\mathbf{X}\mathbf{G}^{-1})\mathbf{G}\Psi.$$

Since $\mathbf{G}$ is an arbitrary matrix,

$$\tilde{\mathbf{X}} = \mathbf{G}\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}_x\mathbf{G}^{-1}$$

which is the same as (2.30).

Similarly, to derive $\tilde{\mathbf{T}}$ we substitute (2.32) into (2.34),

$$\tilde{\Psi}_t(\mathbf{G}\Psi)_t = \mathbf{G}_t\Psi + \mathbf{G}\Psi_t = \tilde{\mathbf{T}}\mathbf{G}\Psi, \tag{2.37}$$

and replace Ψ_t using (2.17),

$$\mathbf{G}_t\Psi + \mathbf{G}\mathbf{T}\Psi \;=\; (\mathbf{G}_t + \mathbf{G}\mathbf{T})\Psi \;=\; \tilde{\mathbf{T}}\mathbf{G}\Psi.$$

As before, we insert the identity matrix (this time into the left hand side),

$$(\mathbf{G}_t + \mathbf{G}\mathbf{T})\mathbf{G}^{-1}\mathbf{G}\Psi \;=\; (\mathbf{G}_t\mathbf{G}^{-1} + \mathbf{G}\mathbf{T}\mathbf{G}^{-1})\mathbf{G}\Psi \;=\; \tilde{\mathbf{T}}\mathbf{G}\Psi.$$

Hence,

$$\tilde{\mathbf{T}} = \mathbf{G}\mathbf{T}\mathbf{G}^{-1} + \mathbf{G}_t\mathbf{G}^{-1}$$

which matches (2.31).

To verify that $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{T}}$ are indeed a valid Lax pair for the same PDE, using (2.30) and (2.31) we compute:

$$\begin{aligned}
\tilde{\mathbf{X}}_t &= \mathbf{G}_t\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}\mathbf{X}_t\mathbf{G}^{-1} + \mathbf{G}\mathbf{X}(\mathbf{G}^{-1})_t + \mathbf{G}_{xt}\mathbf{G}^{-1} + \mathbf{G}_x(\mathbf{G}^{-1})_t, \\
\tilde{\mathbf{T}}_x &= \mathbf{G}_x\mathbf{T}\mathbf{G}^{-1} + \mathbf{G}\mathbf{T}_x\mathbf{G}^{-1} + \mathbf{G}\mathbf{T}(\mathbf{G}^{-1})_x + \mathbf{G}_{tx}\mathbf{G}^{-1} + \mathbf{G}_t(\mathbf{G}^{-1})_x,
\end{aligned}$$

$$\begin{aligned}
\tilde{\mathbf{X}}\tilde{\mathbf{T}} &= \left(\mathbf{G}\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}_x\mathbf{G}^{-1}\right)\left(\mathbf{G}\mathbf{T}\mathbf{G}^{-1} + \mathbf{G}_t\mathbf{G}^{-1}\right) \\
&= \mathbf{G}\mathbf{X}\mathbf{T}\mathbf{G}^{-1} + \mathbf{G}\mathbf{X}\mathbf{G}^{-1}\mathbf{G}_t\mathbf{G}^{-1} + \mathbf{G}_x\mathbf{T}\mathbf{G}^{-1} + \mathbf{G}_x\mathbf{G}^{-1}\mathbf{G}_t\mathbf{G}^{-1},
\end{aligned}$$

and

$$\begin{aligned}
\tilde{\mathbf{T}}\tilde{\mathbf{X}} &= \left(\mathbf{G}\mathbf{T}\mathbf{G}^{-1} + \mathbf{G}_t\mathbf{G}^{-1}\right)\left(\mathbf{G}\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}_x\mathbf{G}^{-1}\right) \\
&= \mathbf{G}\mathbf{T}\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}\mathbf{T}\mathbf{G}^{-1}\mathbf{G}_x\mathbf{G}^{-1} + \mathbf{G}_t\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}_t\mathbf{G}^{-1}\mathbf{G}_x\mathbf{G}^{-1}.
\end{aligned}$$

Hence,

$$\begin{aligned}
\tilde{\mathbf{X}}_t - \tilde{\mathbf{T}}_x &= \mathbf{G}_t\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}\mathbf{X}_t\mathbf{G}^{-1} \\
&\quad + \mathbf{G}\mathbf{X}(\mathbf{G}^{-1})_t + \mathbf{G}_{xt}\mathbf{G}^{-1} + \mathbf{G}_x(\mathbf{G}^{-1})_t - \mathbf{G}_x\mathbf{T}\mathbf{G}^{-1} \\
&\quad - \mathbf{G}\mathbf{T}_x\mathbf{G}^{-1} - \mathbf{G}\mathbf{T}(\mathbf{G}^{-1})_x - \mathbf{G}_{tx}\mathbf{G}^{-1} - \mathbf{G}_t(\mathbf{G}^{-1})_x
\end{aligned}$$

and

$$\begin{aligned}
\tilde{\mathbf{X}}\tilde{\mathbf{T}} - \tilde{\mathbf{T}}\tilde{\mathbf{X}} &= \mathbf{G}\mathbf{X}\mathbf{T}\mathbf{G}^{-1} + \mathbf{G}\mathbf{X}\mathbf{G}^{-1}\mathbf{G}_t\mathbf{G}^{-1} + \mathbf{G}_x\mathbf{T}\mathbf{G}^{-1} + \mathbf{G}_x\mathbf{G}^{-1}\mathbf{G}_t\mathbf{G}^{-1} \\
&\quad -\mathbf{G}\mathbf{T}\mathbf{X}\mathbf{G}^{-1} - \mathbf{G}\mathbf{T}\mathbf{G}^{-1}\mathbf{G}_x\mathbf{G}^{-1} - \mathbf{G}_t\mathbf{X}\mathbf{G}^{-1} - \mathbf{G}_t\mathbf{G}^{-1}\mathbf{G}_x\mathbf{G}^{-1}.
\end{aligned}$$

To replace $(\mathbf{G}^{-1})_x$ in terms of $\mathbf{G}_x$, we compute

$$(\mathbf{G}\mathbf{G}^{-1})_x = \mathbf{G}_x\mathbf{G}^{-1} + \mathbf{G}(\mathbf{G}^{-1})_x = \mathbf{I}_x = \mathbf{0},$$

from which it follows that

$$(\mathbf{G}^{-1})_x = -\mathbf{G}^{-1}\mathbf{G}_x\mathbf{G}^{-1}.$$

Thus,

$$\begin{aligned}
&\tilde{\mathbf{X}}_t - \tilde{\mathbf{T}}_x + [\tilde{\mathbf{X}}, \tilde{\mathbf{T}}] = \\
&\quad \mathbf{G}_t\mathbf{X}\mathbf{G}^{-1} - \mathbf{G}_t\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}_x\mathbf{T}\mathbf{G}^{-1} - \mathbf{G}_x\mathbf{T}\mathbf{G}^{-1} \\
&\qquad + \mathbf{G}_{xt}\mathbf{G}^{-1} - \mathbf{G}_{tx}\mathbf{G}^{-1} + \mathbf{G}\mathbf{T}\mathbf{G}^{-1}\mathbf{G}_x\mathbf{G}^{-1} - \mathbf{G}\mathbf{T}\mathbf{G}^{-1}\mathbf{G}_x\mathbf{G}^{-1} \\
&\qquad + \mathbf{G}_x\mathbf{G}^{-1}\mathbf{G}_t\mathbf{G}^{-1} - \mathbf{G}_x\mathbf{G}^{-1}\mathbf{G}_t\mathbf{G}^{-1} + \mathbf{G}\mathbf{X}\mathbf{G}^{-1}\mathbf{G}_t\mathbf{G}^{-1} \\
&\qquad - \mathbf{G}\mathbf{X}\mathbf{G}^{-1}\mathbf{G}_t\mathbf{G}^{-1} + \mathbf{G}_t\mathbf{G}^{-1}\mathbf{G}_x\mathbf{G}^{-1} - \mathbf{G}_t\mathbf{G}^{-1}\mathbf{G}_x\mathbf{G}^{-1} \\
&\qquad + \mathbf{G}\mathbf{X}\mathbf{T}\mathbf{G}^{-1} - \mathbf{G}\mathbf{T}\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}\mathbf{X}_t\mathbf{G}^{-1} - \mathbf{G}\mathbf{T}_x\mathbf{G}^{-1} \\
&\quad = \mathbf{G}\mathbf{X}\mathbf{T}\mathbf{G}^{-1} - \mathbf{G}\mathbf{T}\mathbf{X}\mathbf{G}^{-1} + \mathbf{G}\mathbf{X}_t\mathbf{G}^{-1} - \mathbf{G}\mathbf{T}_x\mathbf{G}^{-1} \\
&\quad = \mathbf{G}\left(\mathbf{X}_t - \mathbf{T}_x + \mathbf{X}\mathbf{T} - \mathbf{T}\mathbf{X}\right)\mathbf{G}^{-1}.
\end{aligned} \tag{2.38}$$

Using $\mathbf{X}_t - \mathbf{T}_x + [\mathbf{X}, \mathbf{T}] \doteq \mathbf{0}$, it follows that

$$\tilde{\mathbf{X}}_t - \tilde{\mathbf{T}}_x + \tilde{\mathbf{X}}\tilde{\mathbf{T}} - \tilde{\mathbf{T}}\tilde{\mathbf{X}} \doteq \mathbf{0}. \tag{2.39}$$

Therefore, if $(\mathbf{X}, \mathbf{T})$ is a valid Lax pair for some PDE, then $(\tilde{\mathbf{X}}, \tilde{\mathbf{T}})$, defined by (2.30) and (2.31), is also a Lax pair for the same PDE. A topic for future investigation is looking for a $\mathbf{G}$ that will reduce a complicated Lax pair, $\mathbf{X}$ and $\mathbf{T}$, to a simpler Lax pair, $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{T}}$.

2.4 Converting an operator Lax pair into a matrix Lax pair

Given an operator Lax pair, $\mathcal{L}$ and $\mathcal{M}$, the goal is to find a corresponding matrix Lax pair, $\mathbf{X}$ and $\mathbf{T}$.

Starting from (2.1) and (2.2) involving $\mathcal{L}$ and $\mathcal{M}$, we want to convert these scalar equations into a linear system using matrices $\mathbf{X}$ and $\mathbf{T}$. This is analogous to the matrix

version of systems of ordinary differential equations (ODEs), where one replaces one scalar equation of n^{th} order by a system of n equations of first order. In the cases we have studied so far, we can set

$$\varphi = \psi_x. \tag{2.40}$$

If $\mathcal{L}$ is of order n, we must introduce $(n-1)$ new dependent variables. The vector Ψ is then defined as in (2.16) and (2.17).

Supposing $\mathcal{L}$ is of order 2, we can write (2.16) and (2.17) in explicit form,

$$\begin{pmatrix} \psi_x \\ \varphi_x \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} \psi \\ \varphi \end{pmatrix} \tag{2.41}$$

and

$$\begin{pmatrix} \psi_t \\ \varphi_t \end{pmatrix} = \begin{pmatrix} E & F \\ G & H \end{pmatrix} \begin{pmatrix} \psi \\ \varphi \end{pmatrix}, \tag{2.42}$$

with unknown elements A through H.

For example, using the KdV equation, (2.7), we have from (2.40) $\varphi = \psi_x$, and by solving (2.14) for ψ_{xx} we get

$$\varphi_x = \psi_{xx} = \left(\lambda - \frac{1}{6}\alpha u\right)\psi \tag{2.43}$$

that leads to

$$\begin{pmatrix} \psi_x \\ \varphi_x \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \lambda - \frac{1}{6}\alpha u & 0 \end{pmatrix} \begin{pmatrix} \psi \\ \varphi \end{pmatrix}. \tag{2.44}$$

So, A through D are computed yielding $\mathbf{X}$ given in (2.24).

The remaining task is to compute E through H. To do so, we substitute (2.43)

and its differential consequences,

$$\begin{aligned} \varphi_x &= \lambda\psi - \tfrac{1}{6}\alpha u\psi, \\ \varphi_{xx} &= \lambda\varphi - \tfrac{1}{6}\alpha u\varphi - \tfrac{1}{6}\alpha u_x\psi, \end{aligned} \tag{2.45}$$

into (2.15). Hence,

$$\begin{aligned} \psi_t &= -4\varphi_{xx} - \alpha u\varphi + \left(a(t) - \tfrac{1}{2}\alpha u_x\right)\psi \\ &= -4\left(\lambda\varphi - \tfrac{1}{6}\alpha u\varphi - \tfrac{1}{6}\alpha u_x\psi\right) - \alpha u\varphi + \left(a(t) - \tfrac{1}{2}\alpha u_x\right)\psi \\ &= \left(-4\lambda + \tfrac{2}{3}\alpha u - \alpha u\right)\varphi + \left(\tfrac{2}{3}\alpha u_x + a(t) - \tfrac{1}{2}\alpha u_x\right)\psi \\ &= \left(a(t) + \tfrac{1}{6}\alpha u_x\right)\psi - \left(4\lambda + \tfrac{1}{3}\alpha u\right)\varphi \end{aligned} \tag{2.46}$$

and, likewise

$$\begin{aligned} \varphi_t &= \psi_{xt} \\ &= \Big(\left(a(t) + \tfrac{1}{6}\alpha u_x\right)\psi - \left(4\lambda + \tfrac{1}{3}\alpha u\right)\varphi\Big)_x \\ &= \left(a(t) + \tfrac{1}{6}\alpha u_x\right)\psi_x + \tfrac{1}{6}\alpha u_{xx}\psi - \left(4\lambda + \tfrac{1}{3}\alpha u\right)\varphi_x - \tfrac{1}{3}\alpha u_x\varphi \\ &= \left(a(t) + \tfrac{1}{6}\alpha u_x\right)\varphi + \tfrac{1}{6}\alpha u_{xx}\psi - \left(4\lambda + \tfrac{1}{3}\alpha u\right)\left(\lambda - \tfrac{1}{6}\alpha u\right)\psi - \tfrac{1}{3}\alpha u_x\varphi \\ &= \left(a(t) - \tfrac{1}{6}\alpha u_x\right)\varphi + \left(\tfrac{1}{6}\alpha u_{xx} + \tfrac{2}{3}\alpha\lambda u + \tfrac{1}{18}\alpha^2u^2 - 4\lambda^2 - \tfrac{1}{3}\alpha\lambda u\right)\psi \\ &= \left(-4\lambda^2 + \tfrac{1}{3}\alpha\lambda u + \tfrac{1}{18}\alpha^2u^2 + \tfrac{1}{6}\alpha u_{xx}\right)\psi + \left(a(t) - \tfrac{1}{6}\alpha u_x\right)\varphi \end{aligned} \tag{2.47}$$

leading to

$$\begin{pmatrix} \psi_t \\ \varphi_t \end{pmatrix} = \begin{pmatrix} a(t) + \dfrac{1}{6}\alpha u_x & -\left(4\lambda + \dfrac{1}{3}\alpha u\right) \\ -4\lambda^2 + \dfrac{1}{3}\alpha\lambda u + \dfrac{1}{18}\alpha^2u^2 + \dfrac{1}{6}\alpha u_{xx} & a(t) - \dfrac{1}{6}\alpha u_x \end{pmatrix} \begin{pmatrix} \psi \\ \varphi \end{pmatrix} \tag{2.48}$$

that yields the $\mathbf{T}$ found in (2.25).

2.5 Converting a matrix Lax pair into an operator Lax pair

Given a matrix Lax pair, $\mathbf{X}$ and $\mathbf{T}$, the goal now is to find a corresponding operator Lax pair, $\mathcal{L}$ and $\mathcal{M}$.

For the cases we have studied, we can use (2.41) and (2.42) with $A = 0$ and $B = 1$.

First, to find $\mathcal{L}$, we expand the right hand side of (2.41),

$$\begin{aligned} \psi_x &= A\psi + B\varphi = \varphi, \\ \varphi_x &= C\psi + D\varphi, \end{aligned} \tag{2.49}$$

and eliminate φ and all its differential consequences, i.e., $\varphi_x, \varphi_{xx}, \ldots$. After all eliminations were done, we would have an operator $\mathcal{L}$ that is λ dependent.

We would then expand the right hand side of (2.42),

$$\begin{aligned} \psi_t &= E\psi + F\varphi, \\ \varphi_t &= G\psi + H\varphi, \end{aligned} \tag{2.50}$$

and eliminate φ along with all its differential consequences. If we remove φ and its derivatives from (2.50), we will find a low order operator $\mathcal{M}$ that contains λ. We can then eliminate λ using (2.1) if we find an operator $\mathcal{L}$ that is independent of λ.

For example, using the KdV equation,(2.7), and matrix Lax pair, (2.24) and (2.25), we get

$$\begin{aligned} \psi_x &= \varphi, \\ \varphi_x &= \psi_{xx}, \\ &= \left(\lambda - \tfrac{1}{6}\alpha u\right)\psi. \end{aligned} \tag{2.51}$$

Defining

$$\tilde{\mathcal{L}} = \mathcal{D}_x^2 - \left(\lambda - \frac{1}{6}\alpha u\right)\mathcal{I}, \tag{2.52}$$

we have $\tilde{\mathcal{L}}\psi = 0$. In this case, we can easily eliminate λ by defining $\mathcal{L}$ as follows:

$$\mathcal{L} = \tilde{\mathcal{L}} + \lambda\mathcal{I} = \mathcal{D}_x^2 + \frac{1}{6}\alpha u\mathcal{I}, \tag{2.53}$$

that equals $\mathcal{L}$ in (2.8) and obviously $\mathcal{L}\psi = \lambda\psi$.

Now, to find $\mathcal{M}$, we use (2.25) and (2.50). Thus, we get

$$\psi_t = \left(a(t) + \frac{1}{6}\alpha u_x\right)\psi - \left(4\lambda + \frac{1}{3}\alpha u\right)\varphi \tag{2.54}$$

and

$$\varphi_t = \left(-4\lambda^2 + \frac{1}{3}\alpha u\lambda + \frac{1}{18}\alpha^2 u^2 + \frac{1}{6}\alpha u_{xx}\right)\psi + \left(a(t) - \frac{1}{6}\alpha u_x\right)\varphi. \tag{2.55}$$

Replacing φ by ψ_x in (2.54) yields

$$\psi_t = -\left(4\lambda + \frac{1}{3}\alpha u\right)\psi_x + \left(a(t) + \frac{1}{6}\alpha u_x\right)\psi. \tag{2.56}$$

If we define

$$\tilde{\mathcal{M}} = -\left(4\lambda + \frac{1}{3}\alpha u\right)\mathcal{D}_x + \left(a(t) + \frac{1}{6}\alpha u_x\right)\mathcal{I}, \tag{2.57}$$

then $\tilde{\mathcal{M}}$ is a first order operator that depends on λ. To remove λ from (2.57), we replace $\lambda\psi_x$ in (2.56) by differentiating (2.14) with respect to x, yielding

$$\lambda\psi_x = (\mathcal{L}\psi)_x = \psi_{3x} + \frac{1}{6}\alpha u_x\psi + \frac{1}{6}\alpha u\psi_x. \tag{2.58}$$

Doing so, we find

$$\psi_t = -4\psi_{3x} - \alpha u\psi_x + \left(a(t) - \frac{1}{2}\alpha u_x\right)\psi. \tag{2.59}$$

Since (2.2) defines $\mathcal{M}$, removing the ψ gives

$$\mathcal{M} = -4\mathcal{D}_x^3 - \alpha u \mathcal{D}_x + \left(a(t) - \frac{1}{2}\alpha u_x \right) \mathcal{I}.$$

So, we obtain $\mathcal{M}$ given in (2.15).

2.6 Direct compatibility test of an operator Lax pair

There is also a direct, but harder to program, way to test the compatibility of an operator Lax pair with a given PDE. To do so, we cross-differentiate (2.1) and (2.2) and then equate the results.

Let n be the order of $\mathcal{L}$, and let $\mathcal{L}$ be linear in $\mathcal{D}_x^n$. Using (2.1) we first compute $(\psi_{nx})_t$. Next, from (2.2) we compute $(\psi_t)_{nx} = \mathcal{D}_x^n(\mathcal{M}\psi)$. Finally, we set them equal. This will again require replacement of the differential consequences of ψ from (2.1) and (2.2).

For example, using the KdV equation, (2.7), we can substitute (2.8) and (2.9) into (2.1) and (2.2) yielding (2.14) and (2.15).

Differentiating (2.14) with respect to t and (2.15) twice with respect to x gives

$$\psi_{xxt} = \left((\lambda - \frac{1}{6}\alpha u)\psi \right)_t \tag{2.60}$$

and

$$\psi_{txx} = \Big(-4\psi_{3x} - \alpha u \psi_x + \left(a(t) - \frac{1}{2}\alpha u_x \right) \psi \Big)_{xx}. \tag{2.61}$$

Setting (2.60) and (2.61) equal gives

$$\begin{aligned} -\tfrac{1}{6}\alpha u_t \psi + \left(\lambda - \tfrac{1}{6}\alpha u\right) \psi_t \;\; &= \;\; -4\psi_{5x} - \alpha u \psi_{3x} + \left(a(t) - \tfrac{5}{2}\alpha u_x \right) \psi_{xx} \\ &\quad -2\alpha u_{xx}\psi_x - \tfrac{1}{2}\alpha u_{3x}\psi. \end{aligned} \tag{2.62}$$

We can now replace ψ_t in (2.62) from (2.15) and all x derivatives of ψ of order 2 and

higher from (2.14). After all possible ψ_{kx} (with $k \geq 2$) are eliminated, (2.62) reduces to

$$-\frac{1}{6}\alpha\left(u_t + \alpha u u_x + u_{3x}\right)\psi = 0. \tag{2.63}$$

Note that the left hand side of (2.63) is a multiple of (2.7), so when we substitute from the PDE, (2.7), we get zero on both sides.

We used the Lax equation because the direct compatibility would be difficult to code; however, in more general cases, where the Lax equations are not of the standard form given in (2.1) and (2.2), all cross differentiations would need to be considered. To accomplish that, we would have to implement the differential Gröbner basis method (for more information, see [14]). This is beyond the scope of this thesis.

2.7 Alternate form of an operator Lax pair

The defining equations (2.1) and (2.2) as well as the Lax equation (2.6) can be written in an equivalent form. Indeed, defining

$$\tilde{\mathcal{L}} = \mathcal{L} - \lambda\mathcal{I} \tag{2.64}$$

and

$$\tilde{\mathcal{M}} = \mathcal{M} - \mathcal{D}_t, \tag{2.65}$$

(2.1) and (2.2) become

$$\tilde{\mathcal{L}}\psi = \mathcal{O} \tag{2.66}$$

and

$$\tilde{\mathcal{M}}\psi = \mathcal{O}. \tag{2.67}$$

Furthermore, if we replace $\mathcal{L}$ and $\mathcal{M}$ in (2.6), then

$$(\tilde{\mathcal{L}} + \lambda\mathcal{I})_t + [(\tilde{\mathcal{L}} + \lambda\mathcal{I}), (\tilde{\mathcal{M}} + \mathcal{D}_t)] = \mathcal{O}, \tag{2.68}$$

and by expanding

$$\begin{aligned}
& \tilde{\mathcal{L}}_t + \mathcal{O} + (\tilde{\mathcal{L}} + \lambda\mathcal{I})(\tilde{\mathcal{M}} + \mathcal{D}_t) - (\tilde{\mathcal{M}} + \mathcal{D}_t)(\tilde{\mathcal{L}} + \lambda\mathcal{I}) \\
= \; & \tilde{\mathcal{L}}_t + (\tilde{\mathcal{L}}\tilde{\mathcal{M}} + \lambda\mathcal{I}\tilde{\mathcal{M}}) + (\tilde{\mathcal{L}}\mathcal{D}_t + \lambda\mathcal{I}\mathcal{D}_t) - (\tilde{\mathcal{M}}\tilde{\mathcal{L}} + \tilde{\mathcal{M}}\lambda\mathcal{I}) - (\mathcal{D}_t\tilde{\mathcal{L}} + \mathcal{D}_t\lambda\mathcal{I}) \\
= \; & \tilde{\mathcal{L}}_t + \tilde{\mathcal{L}}\tilde{\mathcal{M}} - \tilde{\mathcal{M}}\tilde{\mathcal{L}} + \lambda\tilde{\mathcal{M}} + \tilde{\mathcal{L}}\mathcal{D}_t + \lambda\mathcal{D}_t - \lambda\tilde{\mathcal{M}}\mathcal{I} - \mathcal{D}_t\tilde{\mathcal{L}} - \lambda\mathcal{D}_t\mathcal{I} \\
= \; & \tilde{\mathcal{L}}_t + [\tilde{\mathcal{L}}, \tilde{\mathcal{M}}] + \lambda\tilde{\mathcal{M}} + \tilde{\mathcal{L}}\mathcal{D}_t + \lambda\mathcal{D}_t - \lambda\tilde{\mathcal{M}} - \mathcal{D}_t\tilde{\mathcal{L}} - \lambda\mathcal{D}_t \\
= \; & [\tilde{\mathcal{L}}, \tilde{\mathcal{M}}] + \tilde{\mathcal{L}}_t + \tilde{\mathcal{L}}\mathcal{D}_t - \mathcal{D}_t\tilde{\mathcal{L}} \\
= \; & [\tilde{\mathcal{L}}, \tilde{\mathcal{M}}] \\
= \; & \mathcal{O}.
\end{aligned}$$

So the Lax equation, (2.6), can be replaced by

$$\left[\tilde{\mathcal{L}}, \tilde{\mathcal{M}}\right] \doteq \mathcal{O}. \tag{2.69}$$

For example, using the KdV equation, (2.7), we get

$$\tilde{\mathcal{L}} = \mathcal{D}_x^2 + \left(\frac{1}{6}\alpha u - \lambda\right)\mathcal{I} \tag{2.70}$$

and

$$\tilde{\mathcal{M}} = -4\mathcal{D}_x^3 - \alpha u\mathcal{D}_x + \left(a(t) - \frac{1}{2}\alpha u_x\right)\mathcal{I} - \mathcal{D}_t. \tag{2.71}$$

CHAPTER 3

EXAMPLES OF LAX PAIRS FOR NONLINEAR PDES

The purpose of this chapter is to introduce several examples of PDEs, together with their corresponding operator and matrix Lax pairs. All the presented examples have been verified using our new *Mathematica* package **laxpairtester.m** [12]. If we do not report the operator or matrix forms of the Lax pair for an example, then that means the conversion algorithms of Sections 2.2 and 2.4 did not straightforwardly apply.

In this chapter, λ is the spectral parameter, and $a(t)$ is an arbitrary function of time t.

3.1 The Korteweg–de Vries equation

The Korteweg–de Vries equation was shown in equation (2.7). The operator Lax pair for the KdV equation was shown in (2.8) and (2.9). A real matrix Lax pair for the KdV equation was shown in (2.24) and (2.25).

A second Lax pair [1] for the KdV equation is a complex matrix pair,

$$\mathbf{X}_1 = \begin{pmatrix} -ik & \frac{1}{6}\alpha u \\ -1 & ik \end{pmatrix} \tag{3.1}$$

and

$$\mathbf{T}_1 = \begin{pmatrix} -4ik^3 + \frac{1}{3}\alpha iku - \frac{1}{6}\alpha u_x & \frac{1}{3}\alpha\left(2k^2u - \frac{1}{6}\alpha u^2 + iku_x - \frac{1}{2}u_{xx}\right) \\ -4\lambda^2 + \frac{1}{3}\alpha u & 4i\lambda^3 - \frac{1}{3}\alpha i\lambda u + \frac{1}{6}\alpha u_x \end{pmatrix}. \tag{3.2}$$

The Lax pairs (2.24)–(2.25) and (3.1)–(3.2) are related by a simple gauge transfor-

mation,

$$\mathbf{G}_1 = \begin{pmatrix} 1 & 0 \\ ik & 1 \end{pmatrix},$$

where $\lambda = -k^2$. A third Lax pair [15] for the KdV equation is a complex matrix pair,

$$\mathbf{X}_2 = \begin{pmatrix} -ik & 1 \\ -\frac{1}{6}\alpha u & ik \end{pmatrix} \tag{3.3}$$

and

$$\mathbf{T}_2 = \begin{pmatrix} -4ik^3 + \frac{1}{3}\alpha iku + \frac{1}{6}\alpha u_x & 4\lambda^2 - \frac{1}{3}\alpha u \\ \frac{1}{3}\alpha\left(-2k^2u + \frac{1}{6}\alpha u^2 + iku_x + \frac{1}{2}u_{xx}\right) & 4i\lambda^3 - \frac{1}{3}\alpha i\lambda u - \frac{1}{6}\alpha u_x \end{pmatrix}. \tag{3.4}$$

The Lax pairs (3.1)–(3.2) and (3.3)–(3.4) are related by another simple gauge transformation,

$$\mathbf{G}_2 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},$$

we also require $k \to -k$.

3.2 The Camassa–Holm equation

The Camassa–Holm (CH) equation [16, 17] is a nonlinear PDE,

$$u_{xxt} - u_t - 3uu_x - 2\kappa u_x + 2u_x u_{xx} + uu_{3x} = 0, \tag{3.5}$$

where κ is a real parameter (which can be scaled to any value).

The operator Lax pair [15] for the CH equation is

$$\mathcal{L} = \mathcal{D}_x^2 - \left(\frac{1}{4} + \lambda(\kappa - 1 + u - u_{xx})\right)\mathcal{I}, \tag{3.6}$$

$$\mathcal{M} = \left(\frac{1}{2\lambda} - u\right)\mathcal{D}_x + \frac{1}{2}\left[u_x + a(t)\right]\mathcal{I}. \tag{3.7}$$

The matrix Lax pair [15] for the CH equation is

$$\mathbf{X} = \begin{pmatrix} 0 & 1 \\ \frac{1}{4} + \lambda(\kappa + u - u_{xx}) & 0 \end{pmatrix} \tag{3.8}$$

and

$$\mathbf{T} = \begin{pmatrix} \frac{1}{2}(a(t) + u_x) & \frac{1}{2\lambda} - u \\ \frac{1}{2}\left(\kappa + \frac{1}{4\lambda} + \frac{1}{2}u\right) - \lambda(\kappa u + u^2 - u u_{xx}) & \frac{1}{2}(a(t) - u_x) \end{pmatrix}, \tag{3.9}$$

3.3 The short pulse equation

The short pulse (SP) equation [18] is a nonlinear PDE,

$$u_{xt} - \alpha u - \beta\left(u^3\right)_{xx} = u_{xt} - \alpha u - 3\beta u\left(2u_x^2 + u u_{xx}\right) = 0, \tag{3.10}$$

where α and β are positive, real parameters.

The matrix Lax pair [19] for the SP equation is

$$\mathbf{X} = \begin{pmatrix} \sqrt{\alpha}\lambda & \sqrt{6\beta}\lambda u_x \\ \sqrt{6\beta}\lambda u_x & -\sqrt{\alpha}\lambda \end{pmatrix} \tag{3.11}$$

and

$$\mathbf{T} = \begin{pmatrix} \sqrt{\alpha}\left(\frac{1}{4\lambda} + 3\beta\lambda u^2\right) & -\sqrt{\frac{3}{2}\alpha\beta u} + 3\sqrt{6\beta^3}\lambda u^2 u_x \\ \sqrt{\frac{3}{2}\alpha\beta u} + 3\sqrt{6\beta^3}\lambda u^2 u_x & -\sqrt{\alpha}\left(\frac{1}{4\lambda} + 3\beta\lambda u^2\right) \end{pmatrix}. \tag{3.12}$$

One choice [19] is $\alpha = 1$ and $\beta = \frac{1}{6}$.

3.3.1 The Lax's fifth-order KdV equation

The Lax's fifth-order KdV equation due to Lax [9] is a nonlinear PDE,

$$u_t + \alpha u^2 u_x + \beta u_x u_{xx} + \gamma u u_{3x} + u_{5x} = 0, \tag{3.13}$$

where α, β, γ satisfy the following relationships:

$$\alpha = \frac{3}{10}\gamma^2, \quad \beta = 2\gamma. \tag{3.14}$$

The operator Lax pair for the Lax's fifth-order KdV equation will be shown in Section 5.2.1. The matrix Lax pair [15] for the Lax's fifth-order KdV equation is

$$\mathbf{X} = \begin{pmatrix} 0 & \frac{1}{\lambda^2} \\ -\frac{1}{10}\gamma\lambda^2 u & 0 \end{pmatrix} \tag{3.15}$$

and

$$\mathbf{T} = \frac{1}{500}\gamma \begin{pmatrix} 10\left(3\gamma u u_x + 5u_{3x}\right) & -\frac{10}{\lambda^2}\left(3\gamma u^2 + 10u_{xx}\right) \\ \lambda^2\left(3\gamma^2 u^3 + 30\gamma (u_x)^2 + 40\gamma u u_{xx} + 50u_{4x}\right) & -10\left(3\gamma u u_x + 5u_{3x}\right) \end{pmatrix}. \tag{3.16}$$

A common choice [20] for the parameters is $\gamma = 10$, $\alpha = 30$, and $\beta = 20$.

3.3.2 The Sawada–Kotera equation

The SK equation [21] is shown in equation (3.13) where α, β, γ satisfy the following relationships:

$$\alpha = \frac{1}{5}\gamma^2, \quad \beta = \gamma. \tag{3.17}$$

An operator Lax pair for the SK equation will be shown in Section 5.2.2. The matrix Lax pair [15] for the SK equation is

$$\mathbf{X} = \begin{pmatrix} 0 & \frac{1}{\lambda^2} \\ -\frac{1}{5}\gamma\lambda^2 u & 0 \end{pmatrix} \tag{3.18}$$

and

$$\mathbf{T} = \frac{1}{125}\gamma \begin{pmatrix} 5\left(\gamma u u_x + 5u_{3x}\right) & -\frac{5}{\lambda^2}\left(\gamma u^2 + 10u_{xx}\right) \\ \lambda^2\left(\gamma^2 u^3 + 5\gamma u_x^2 + 15\gamma u u_{xx} + 25u_{4x}\right) & -5\left(\gamma u u_x + 5u_{3x}\right) \end{pmatrix}. \tag{3.19}$$

Two common choices [20, 22] for the parameters are $\gamma = -15$, $\alpha = 45$, and $\beta = -15$, and $\gamma = 5$, $\alpha = 5$, and $\beta = 5$.

3.3.3 The Kaup–Kupershmidt equation

The Kaup–Kupershmidt (KK) equation [23] is shown in equation (3.13) where α, β, γ satisfy the following relationships:

$$\alpha = \frac{1}{5}\gamma^2, \quad \beta = \frac{5}{2}\gamma. \tag{3.20}$$

The operator Lax pair for the KK equation will be shown in Section 5.2.3. The matrix Lax pair [15, 24] for the KK equation is

$$\mathbf{X} = \begin{pmatrix} 0 & \frac{1}{\lambda^2} \\ -\frac{1}{20}\gamma\lambda^2 u & 0 \end{pmatrix} \tag{3.21}$$

and

$$\mathbf{T} = \frac{1}{1000}\gamma \begin{pmatrix} 10\left(4\gamma u u_x + 5u_{3x}\right) & -\frac{20}{\lambda^2}\left(2\gamma u^2 + 5u_{2x}\right) \\ \lambda^2\left(2\gamma^2 u^3 + 40\gamma u_x^2 + 45\gamma u u_{2x} + 50u_{4x}\right) & -10\left(4\gamma u u_x + 5u_{3x}\right) \end{pmatrix}. \tag{3.22}$$

A common choice [20] for the parameters is $\gamma = 10$, $\alpha = 20$ and $\beta = 25$.

3.4 The sine–Gordon equation

The sine–Gordon (SG) equation [25] is a nonlinear PDE,

$$u_{xt} - \sin(u) = 0. \tag{3.23}$$

The matrix Lax pair [26] for the SG equation (3.23) is

$$\mathbf{X} = \begin{pmatrix} -i\lambda & -\frac{1}{2}u_x \\ \frac{1}{2}u_x & i\lambda \end{pmatrix} \tag{3.24}$$

and

$$\mathbf{T} = \frac{1}{4\lambda} i \begin{pmatrix} \cos(u) & \sin(u) \\ \sin(u) & -\cos(u) \end{pmatrix}. \tag{3.25}$$

3.5 The sinh–Gordon equation

The sinh–Gordon (ShG) [25] equation is a nonlinear PDE,

$$u_{xt} - \sinh(u) = 0. \tag{3.26}$$

The matrix Lax pair [13, p. 39] for the ShG equation is

$$\mathbf{X} = \begin{pmatrix} -i\lambda & -\frac{1}{2}iu_x \\ \frac{1}{2}iu_x & i\lambda \end{pmatrix} \tag{3.27}$$

and

$$\mathbf{T} = \frac{1}{4\lambda} i \begin{pmatrix} \cosh(u) & \sinh(u) \\ \sinh(u) & -\cosh(u) \end{pmatrix}. \tag{3.28}$$

3.6 The Boussinesq system

The general Boussinesq equation [27] is a nonlinear PDE,

$$u_t + \beta u u_{xx} + \beta (u_x)^2 + \epsilon u_{xx} + \alpha u_{4x} = 0, \tag{3.29}$$

where α, β, ϵ are real parameters and $\epsilon = 0$ is valid, which can be transformed into a nonlinear system [28],

$$\begin{aligned} u_t &= -3v_x, \\ v_t &= -u_{3x} + 8uu_x. \end{aligned} \tag{3.30}$$

The operator Lax pair for the Boussinesq system is

$$\mathcal{L} = \mathcal{D}_x^3 - 2u\mathcal{D}_x - (u_x + iv)\mathcal{I}, \tag{3.31}$$

$$\mathcal{M} = i(3\mathcal{D}_x^2 - 4u\mathcal{I}). \tag{3.32}$$

3.7 The Harry Dym equation

The Harry Dym (HD) equation [29] is a nonlinear PDE,

$$u_t + 2u^3 u_{3x} = 0. \tag{3.33}$$

The operator Lax pair [30] for the HD equation is

$$\mathcal{L} = -u^2\mathcal{D}_x^2 - 2uu_x\mathcal{D}_x, \tag{3.34}$$

$$\mathcal{M} = -8u^2\mathcal{D}_x^3 - 36u^2u_x\mathcal{D}_x^2 - 12u(2(u_x)^2 + uu_{xx})\mathcal{D}_x. \tag{3.35}$$

3.8 The Hirota–Satsuma equation

The Hirota–Satsuma (HS) equation [31] is a nonlinear PDE,

$$u_{xxt} - 2u_t u_x = 0 \tag{3.36}$$

(not to be confused with the well-known coupled system of KdV type equations [32, 33]). The matrix Lax pair [34] for the HS equation is

$$\mathbf{X} = \begin{pmatrix} 0 & 1 & 3u_x \\ 0 & 0 & -\lambda \\ 1 & 0 & 0 \end{pmatrix} \tag{3.37}$$

and

$$\mathbf{T} = \begin{pmatrix} 0 & -\dfrac{1}{\lambda}u_{xt} & u_t \\ u_t & 0 & -u_{xt} \\ 0 & -\dfrac{1}{\lambda}u_t & 0 \end{pmatrix}. \tag{3.38}$$

3.9 The Liouville equation

The Liouville (LV) equation [35] is a nonlinear PDE,

$$u_{xt} - e^{2u} = 0. \tag{3.39}$$

The matrix Lax pair [36] for the LV equation is

$$\mathbf{X} = \begin{pmatrix} u_x & \lambda \\ \lambda & -u_x \end{pmatrix} \tag{3.40}$$

and

$$\mathbf{T} = \begin{pmatrix} 0 & -\dfrac{e^{2u}}{\lambda} \\ 0 & 0 \end{pmatrix}. \tag{3.41}$$

3.10 The Ziber–Shabat–Mikhailov system

The Ziber–Shabat–Mikhailov (ZSM) system [37] is a nonlinear PDE system,

$$\begin{aligned} u_{xt} - e^{u+v} + e^{-u} &= 0, \\ u_{xt} - e^{u+v} + e^{-v} &= 0. \end{aligned} \tag{3.42}$$

The matrix Lax pair [38] for the ZSM equation is

$$\mathbf{X} = \begin{pmatrix} u_x & \lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda & 0 & 0 & 0 \\ 0 & 0 & -u_x & \lambda & 0 & 0 \\ 0 & 0 & 0 & v_x & \lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda \\ \lambda & 0 & 0 & 0 & 0 & -v_x \end{pmatrix} \tag{3.43}$$

and

$$\mathbf{T} = \frac{1}{\lambda} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & e^{u+v} \\ e^{-u} & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{-u} & 0 & 0 & 0 & 0 \\ 0 & 0 & e^{u+v} & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{-v} & 0 & 0 \\ 0 & 0 & 0 & 0 & e^{-v} & 0 \end{pmatrix}. \tag{3.44}$$

3.11 The Kadomtsev–Petviashvili equation

The Kadomtsev–Petviashvili (KP) equation [39] is a nonlinear PDE,

$$(u_t + \alpha u u_x + u_{3x})_x + 3u_{yy} = u_{xt} + \alpha \left(u u_{xx} + (u_x)^2 \right) + 3\sigma^2 u_{yy} + u_{4x} = 0. \tag{3.45}$$

The operator Lax pair [13, p. 212] for the KP equation is

$$\mathcal{L} = \mathcal{D}_x^2 + \sigma \mathcal{D}_y + \frac{1}{6}\alpha u \mathcal{I}, \tag{3.46}$$

$$\mathcal{M} = -4\mathcal{D}_x^3 - \alpha u \mathcal{D}_x - \frac{1}{2}\alpha \left(u_x - \sigma \int u_y \, dx - a(t) \right) \mathcal{I}. \tag{3.47}$$

3.12 The Burgers equation

The Burgers equation [40] is a nonlinear PDE,

$$u_{tt} + \alpha u u_x + \beta u_{xx} = 0. \tag{3.48}$$

The operator Lax pair [15] for the Burgers equation is

$$\mathcal{L} = \mathcal{D}_x + \frac{\alpha}{2\beta} u \mathcal{I}, \tag{3.49}$$

$$\mathcal{M} = -\beta \mathcal{D}_x^2 - \alpha u \mathcal{D}_x. \tag{3.50}$$

CHAPTER 4

ALGORITHMS

In this chapter, we will discuss the algorithms used in various parts of the *Mathematica* package, **laxpairtester.m**.

While writing the functions `testLM` and `testXT`, we started with the simplest case, i.e., the KdV equation. Once the code worked for that case, we moved onto the next case, the CH equation, and added complexity until the software could handle that case. We continued this process until we had a dozen examples that worked. Doing so, we had added enough complexity to the code that new examples most likely fall into the class of equations the software can deal with.

Without going into all details, the software can test Lax pairs of polynomial PDEs of any number of dependent variables, any number of independent spatial variables, and one independent time variable. Integral terms are allowed in both the PDE and the Lax pair, but such examples have not been thoroughly tested.

4.1 Invoking the code and debugging

To invoke **laxpairtester.m**, the user will first set the directory of the notebook to wherever the package and data files are saved. Next, the user will `Get laxPackage.m` to initialize all needed files. The user will then run `laxWork` (see Section 4.8) with optional debugging flags, e.g., `laxWork[printTestLM->True]` would run the program with all debug print statements in `testLM`, Section 4.2, turned on. Finally, the user will be prompted to choose which tester to run and what example to use.

4.2 $\mathcal{L}$ and $\mathcal{M}$ tester

The function `testLM` takes a possible operator Lax pair and returns whether it is valid or not. The input is the possible $\mathcal{L}$ and $\mathcal{M}$, the original PDE, the variable

the user wishes to apply $\mathcal{L}$ and $\mathcal{M}$ to (usually ψ), the list of dependent variables, the list of independent spatial variables, the list of independent time variables (can only handle one currently), and an optional flag for if the $\mathcal{L}$ and $\mathcal{M}$ are user input or computed.

First we run the given $\mathcal{L}$ through the function `highLDer` described in Section 4.5 which finds the highest derivative in $\mathcal{L}$ with respect to the first independent spatial variable given in the data file (usually x). We then run $\mathcal{L}$ and $\mathcal{M}$ through `convertIn` to make all variables consistent for the rest of the program using the flags

```
lmExist = True
xtExist = False
```

because when using `testLM` we can only assume that $\mathcal{L}$ and $\mathcal{M}$ exist and do not want `convertIn` to attempt to convert $\mathbf{X}$ and $\mathbf{T}$. We then print the PDE using `printPDE` which returns the string "given" or "computed" depending on the internal flag. We can use that string in printing $\mathcal{L}$ and $\mathcal{M}$. Next, several sets of rules are computed. First, we compute a set of general substitution rules for the independent variables using `genRuleFinder`. Next, the rule for ψ_t is derived using (2.2). Finally, the general rule for $(\psi_{mx})_{nx}$ where m is the highest derivative of ψ in $\mathcal{L}$ and n is the derivative which needs to be replaced minus m.

Now to compute the different parts of (2.69). $\mathcal{L}_t$ is computed using the derivative operator, `Dop`, from Section 4.4, $\mathcal{LM}$ is computed by applying $\mathcal{M}$ to ψ and $\mathcal{L}$ to the result. $\mathcal{ML}$ is computed by applying $\mathcal{L}$ to ψ and $\mathcal{M}$ to the result.

4.3 X and T tester

The function `testXT` takes a possible matrix Lax pair and returns whether it is valid or not. The input is the possible $\mathbf{X}$ and $\mathbf{T}$, the PDE the Lax pair is to represent, the list of dependent variables, the list of independent spatial variables, and the list of independent temporal variables.

First we convert, using `convertIn` Section 4.7, all variables to our internal variables. Next, we generate the set of replacement rules, using `genRuleFinder` Section 4.6, for the dependent variables, u, v are common choices. We compute each piece of the matrix Lax equation (2.23) separately, $\mathbf{X}_t, \mathbf{T}_x, \mathbf{XT}$, and $\mathbf{TX}$. Then we combine them into the Lax equation, `laxMat`. Finally, we substitute the dependent variables into `laxMat` and simplify; the resulting matrix, `zeroMat`, is checked, using `jZeroQ` (4.13), to see if it equals the zero matrix. If `zeroMat` does equal zero, we print that the matrix pair is a valid Lax pair otherwise, we print that the matrix pair is not a valid Lax pair and print the final resulting matrix.

4.4 Derivative operator

It was important to define a new function to compute the partial derivative with respect to time of the operator $\mathcal{L}$, i.e. $\mathcal{L}_t$ Based on the chain rule

$$\mathcal{L}_t\psi = (\mathcal{L}\psi)_t - \mathcal{L}\psi_t,$$

we define a new function

```
Dop[op_, var_] := (D[op[#], var] - op[D[#, var]]) &.
```

`Dop` can now be used throughout the code for the derivative of any operator with respect to any variable.

4.5 Highest derivative finder

The function `highLDer` takes as input a single internal independent variable (the one we wish to find the highest derivative of) and a pure function. We create a pattern to allow searching within our function for derivatives which include our internal variable. We create a function `findxD` which returns the number of derivatives when we input our pattern and we set `findxD` as `Listable`. Next, we create a func-

tion `findxDD` which, when fed the `DownValues`[1] of a pure function, walks through each term and runs it through `findxD` and returns a list of values. This list, named `derList`, of values is the list of all derivatives taken with respect to the independent variable in the function. We then return the `Max` of `derList` as the highest derivative with respect to the independent variable in this function.

The function `highDer` is identical to `highLDer` except that it takes any expression other then a pure function as input and works directly with the expression instead of with the `DownValues`.

4.6 General PDE rule generator

The function `genRuleFinder` solves the original PDE for each dependent variable which have derivatives with respect to time t (there must be the same number of equations in the PDE system as dependent variables) and creates rules for each, i.e., $u_t \to \ldots$. It takes as input a list of the dependent variables, a list of the independent spatial variables, a list of the independent time variables (only t at this time), and the list of PDE equations (one or more equations).

We first define a couple of functions:

- `prefixSymbols` takes as input a variable and a prefix for that variable and returns a combination of those symbols, e.g., it takes as input n and x then returns nx.

- `makeList` takes a variable and a number and creates a list from them, e.g., it takes as input x and 3 then returns $x, 3$.

- `patternMaker` takes a variable and a differential list and returns a differential pattern that can be matched, e.g., it takes as input x and $x, 3$ then returns

[1] From *Mathematica*'s help: `DownValues`$[f]$ gives a list of transformation rules corresponding to all downvalues defined for the symbol f. This function allows us to directly manipulate the values of the definition of a pure function.

`D[uu[i_][x],{x,3}]`.

Next, we create a list of symbols using the user given independent variables and `prefixSymbols` using n as our prefix input, and then we create a list of pattern matching symbols using the list we just created. Next, we `Thread makeList` using our list of all independent variables (including time) and our list of pattern matching symbols to end with a list such as `{{x,nx},{t,nt}}`, and then we add to that list our `uu[i_]` with our list of all independent variables applied to it. Finally, we `Apply D` to the list we just created and name it `pattern`.

Once we have created all of our pattern matching variables, we create a list of all of the derivatives (for each of the independent spatial variables) and then we take that and find the highest derivative for each of independent spatial variable. Next, we solve our list of equations (our expressions set equal to zero) for the highest derivatives of each independent spatial variable (the number of equations must equal the number of independent spatial variables). This becomes our basic set of replacement rules for the PDE. Finally, we make a pattern matching set of rules which replaces any derivative of u (down to the highest derivative in each variable) and replaces it using the correct rule differentiated as many times as necessary.

We also include an integration rule in the first independent spatial variable listed in the data file. This allows some examples that include integration to be checked. We return the list of general rules for replacement of the PDE.

4.7 Input converter

The function `convertIn` takes the following as input: the given PDE, a list of the dependent variables, a list of the independent spatial variables, a list of the independent time variables (only t currently), the operator Lax pair along with a flag to tell if they exist, the matrix Lax pair along with a flag to tell if they exist, and finally the variable we apply the pure functions to (usually the eigenfunction ψ).

We first convert the PDE by running it through `finduRules`, described in Section 4.10, and then through `findxRules`, described in Section 4.9. We then convert the dependent variables and then the independent variables using `findxRules` in both cases. Finally, we convert the variables in ψ.

4.8 Main driver

The main driver includes several `Modules`:

- `laxWork` takes as optional input any debugging options the user wishes to use and runs the rest of the code (see Section 4.1). It sets all debugging print statements and runs `newMenu`.

- `newMenu` prints out the choices of if you wish to test a matrix Lax pair or an operator Lax pair. Either choice will route to `testMenu`. After a data file has been loaded, `newMenu` will call `testXT` from Section 4.3 or `testLM` from Section 4.2 depending.

- `testMenu` allows the user to choose if they wish to use a prewritten example, upload a data file they provide, return to the previous menu, or quit the program. If they choose to use a previous example, they are routed to `examplesMenu`. Otherwise, if they choose to upload their own file, then they will be asked to make sure their file is prepared, and, if so, to input the name of the file.

- `examplesMenu` prints out all included tested examples for the user to choose from, as well as options to go back to the previous menu or quit the program. Each example will include if it has a matrix Lax pair (denoted by XT), an operator Lax pair (denoted by LM), or both. Once the user selects the number of the example they wish to see, the data file is loaded and `newMenu` is finished.

- `Options[laxPairs]` defaults debugging to off.

4.9 Converting independent variables

The function `findxRules` takes as input an equation or a list of equations, the number of independent spatial variables, the list of the independent spatial variables (given as input by the user), and a flag which tells if we are converting to or from internal variables. The function creates two lists of rules: `x2xxRule` creates a list of rules to convert x and y to $xx[1]$ and $xx[2]$, etc., `xx2xRule` creates a list of rules to convert $xx[1]$ back to x, etc. Then it applies the correct rule to the equation using an `If` statement

```
If[
    forwardRule,
    eqListOut = eqList/.x2xxRule,
    eqListOut = eqList/.xx2xRule
  ]
```

and outputs the resulting equation.

4.10 Converting dependent variables

The function `finduRules` takes as input an equation or a list of equations, the number of dependent variables, the list of dependent variables (given as input by the user), and a flag which tells if we are converting to or from internal variables. The function creates two lists of rules: `u2uuRule` creates a list of rules to convert u to $uu[1]$, v to $uu[2]$, etc., `uu2uRule` creates a list of rules to convert $uu[1]$ back to u, etc. Then it applies the correct rule to the equation using an `If` statement

```
If[
    forwardRule,
    eqListOut = eqList/.u2uuRule,
    eqListOut = eqList/.uu2uRule
  ]
```

and outputs the resulting equation.

4.11 Pretty print

The function `prettyPrint` is used to convert the user given variables, u, x, etc., to internal variables, $uu[1]$, $xx[1]$, etc., and vice versa. This allows users to use any variables they choose in their data file. Input is the equation (can be single equation or a list of equations), a flag to tell if the input is a pure function or not, and a flag to tell which direction we are converting the variables. First we check that our conversion flag has a value of True or False, defaulting to False if no value is given. We computed the number of spatial variables and the number of dependent variables.

If the equation was a pure function, we run `finduRules` on the `DownValues` of the given equation along with the input of our number of different variables and our conversion flag; however, we do not convert x because the pure function will get applied to $xx[1]$ later as needed.

If the equation was of regular form or a list of regular equations, we run `finduRules` first and then we run `findxRules` on the returned result. The function defaults to this usage.

In both cases, `prettyPrint` returns the converted equation or pure function.

4.12 PDE printer

The function `printPDE` takes as input the list of PDEs, the number of equations in the PDE list, and whether the PDE was computed in the program or given by the user. It then decides if it should use "given" or "computed"

```
If[
   compFlag,
   givenComp = ``computed'',
   givenComp = ``given'',
   givenComp = ``computed''
  ]
```

and “equation” or “system” in the later print statements

```
If[
  numPDE == 1,
  systemVar = ``equation'',
  systemVar = ``system''
 ].
```

Then it prints out a pretty version of the PDE using `prettyPrint` from Section 4.11, with each part equaling zero. It returns `givenComp` for future printing (so we do not have to decide if the PDE was given or computed for the rest of the current run).

4.13 Checking for zero

The function `jZeroQ` takes the expression (or list of expressions) that might be zero and a flag to let tell if the expression has any chance of being zero (if the given Lax pair has unspecified constants, the expression will not be zero and attempting to `Simplify` will take a long time and many calculations). If the equation should be zero, we use the *Mathematica* function `PossibleZeroQ` on all pieces of the list of equations. It then returns the value True, if all pieces of the list of equations are zero, or False, if they are not.

4.14 Abnormal end

The function `Abend` takes a list of notes to print (the input is optional). It then prints out the notes or a default message

```
The program has encountered an error.
Please check inputs and try again.
The program will now end.
```

CHAPTER 5

APPLICATIONS

In this chapter, we will investigate the operator Lax pairs of a family of nonlinear PDEs. More precisely, we will use our *Mathematica* software, **laxpairtester.m**, to help test and determine operator Lax pairs for different PDEs related to the family of PDEs. Doing so, we discovered a, to our knowledge, previously unknown Lax pair for the SK equation (5.47) .

In this chapter, λ is the spectral parameter, $a(t)$ is an arbitrary function of time t, and α, β, γ are non-zero parameters.

5.1 Fifth-order Korteweg–de Vries equations

We will consider the family of fifth-order KdV [20, 41] equations,

$$u_t + \alpha u^2 u_x + \beta u_x u_{xx} + \gamma u u_{3x} + u_{5x} = 0, \tag{5.1}$$

in a dimensionless version, leading to several possible relationships among the parameters α, β, and γ. Finally, we will show and then derive several operator Lax pairs for the completely integrable equations in this family.

5.2 Nondimensionalization

We wish to nondimensionalize (5.1) as much as possible to see what ratios of the parameters are important. Let us use the following scaling transformation:

$$x = \frac{\tilde{x}}{A}, \quad t = \frac{\tilde{t}}{B}, \quad u = C\tilde{u}, \tag{5.2}$$

and we are transforming $u(x,t) \to \tilde{u}(\tilde{x},\tilde{t})$. We can substitute (5.2) into (5.1) term by term:

$$u_t = C\tilde{u}_{\tilde{t}}\frac{\partial \tilde{t}}{\partial t} = BC\tilde{u}_{\tilde{t}}, \tag{5.3}$$

$$\alpha u^2 u_x = \alpha AC^3\tilde{u}^2\tilde{u}_{\tilde{x}}, \tag{5.4}$$

$$\beta u_x u_{xx} = \beta A^3C^2\tilde{u}_{\tilde{x}}\tilde{u}_{\tilde{x}\tilde{x}}, \tag{5.5}$$

$$\gamma u u_{3x} = \gamma A^3C^2\tilde{u}\tilde{u}_{3\tilde{x}}, \tag{5.6}$$

$$u_{5x} = A^5C\tilde{u}_{5\tilde{x}}. \tag{5.7}$$

Hence, (5.1) becomes

$$BC\tilde{u}_{\tilde{t}} + \alpha AC^3\tilde{u}^2\tilde{u}_{\tilde{x}} + \beta A^3C^2\tilde{u}_{\tilde{x}}\tilde{u}_{\tilde{x}\tilde{x}} + \gamma A^3C^2\tilde{u}\tilde{u}_{3\tilde{x}} + A^5C\tilde{u}_{5\tilde{x}} = 0. \tag{5.8}$$

We wish to make the coefficients of as many terms as possible equal one. To accomplish this, we will first divide the entire equation by BC,

$$\tilde{u}_{\tilde{t}} + \alpha\frac{A}{B}C^2\tilde{u}^2\tilde{u}_{\tilde{x}} + \beta\frac{A^3}{B}C\tilde{u}_{\tilde{x}}\tilde{u}_{\tilde{x}\tilde{x}} + \gamma\frac{A^3}{B}C\tilde{u}\tilde{u}_{3\tilde{x}} + \frac{A^5}{B}\tilde{u}_{5\tilde{x}} = 0. \tag{5.9}$$

Next, we choose to make the coefficient of the $\tilde{u}_{5\tilde{x}}$ term equal one by setting $B = A^5$, and replace B in all the terms,

$$\tilde{u}_{\tilde{t}} + \alpha\left(\frac{C}{A^2}\right)^2\tilde{u}^2\tilde{u}_{\tilde{x}} + \beta\frac{C}{A^2}\tilde{u}_{\tilde{x}}\tilde{u}_{\tilde{x}\tilde{x}} + \gamma\frac{C}{A^2}\tilde{u}\tilde{u}_{3\tilde{x}} + \tilde{u}_{5\tilde{x}} = 0. \tag{5.10}$$

Now, we choose to clear out the coefficient of the $\tilde{u}\tilde{u}_{3\tilde{x}}$ term by setting $\gamma\frac{C}{A^2} = 1$. This allows us to use the following replacement rule:

$$\frac{C}{A^2} = \frac{1}{\gamma}, \tag{5.11}$$

yielding,

$$\tilde{u}_{\tilde{t}} + \frac{\alpha}{\gamma^2}\tilde{u}^2\tilde{u}_{\tilde{x}} + \frac{\beta}{\gamma}\tilde{u}_{\tilde{x}}\tilde{u}_{\tilde{x}\tilde{x}} + \tilde{u}\tilde{u}_{3\tilde{x}} + \tilde{u}_{5\tilde{x}} = 0. \tag{5.12}$$

Thus far, $B = A^5$, $C = \frac{A^2}{\gamma}$, and A is still free. Since the coefficients in (5.12) do not depend on A,, it is clear that two ratios define the equation:

$$\frac{\alpha}{\gamma^2}, \qquad \frac{\beta}{\gamma}, \tag{5.13}$$

and our transforming variables are

$$\tilde{x} = Ax, \quad \tilde{t} = A^5 t, \quad u = \frac{A^2}{\gamma}\tilde{u}. \tag{5.14}$$

We can let $A = 1$, so

$$\tilde{x} = x, \quad \tilde{t} = t, \quad u = \frac{1}{\gamma}\tilde{u}. \tag{5.15}$$

5.2.1 Lax's fifth-order KdV equation

The fifth-order KdV equation due to Lax [9] is a nonlinear PDE, found in (5.1) where α, β, γ satisfy the following relationships:

$$\alpha = \frac{3}{10}\gamma^2, \quad \beta = 2\gamma. \tag{5.16}$$

The operator Lax pair [9, 15] for Lax's fifth-order KdV equation,

$$u_t + \frac{3}{10}\gamma^2 u^2 u_x + 2\gamma u_x u_{xx} + \gamma u u_{3x} + u_{5x} = 0, \tag{5.17}$$

is

$$\mathcal{L} = \mathcal{D}_x^2 + \frac{1}{10}\gamma u\mathcal{I}, \tag{5.18}$$

$$\begin{aligned}\mathcal{M} &= -16\mathcal{D}_x^5 - 4\gamma u\mathcal{D}_x^3 - 6\gamma u_x\mathcal{D}_x^2 - \left(\frac{3}{10}\gamma^2u^2 + 5\gamma u_{xx}\right)\mathcal{D}_x \\ &\quad - \left(a(t) + \frac{3}{10}\gamma^2uu_x + \frac{3}{2}\gamma u_{3x}\right)\mathcal{I}.\end{aligned} \tag{5.19}$$

The matrix Lax pair for Lax's fifth-order KdV equation was shown in Section 3.3.1.

5.2.2 The Sawada–Kotera equation

The Sawada–Kotera (SK) equation [21] is a nonlinear PDE, found in (5.1), where α, β, γ satisfy the following relationships:

$$\alpha = \frac{1}{5}\gamma^2, \quad \beta = \gamma. \tag{5.20}$$

An operator Lax pair [22] for the SK equation,

$$u_t + \frac{1}{5}\gamma^2u^2u_x + \gamma u_xu_{xx} + \gamma uu_{3x} + u_{5x} = 0, \tag{5.21}$$

is

$$\mathcal{L} = \mathcal{D}_x^3 + \frac{1}{5}\gamma u\mathcal{D}_x, \tag{5.22}$$

$$\mathcal{M} = 9\mathcal{D}_x^5 + 3\gamma u\mathcal{D}_x^3 + 3\gamma u_x\mathcal{D}_x^2 + \left(\frac{1}{5}\gamma^2u^2 + 2\gamma u_{xx}\right)\mathcal{D}_x + a(t)\mathcal{I}. \tag{5.23}$$

The matrix Lax pair for the SK equation was shown in Section 3.3.2.

5.2.3 The Kaup–Kupershmidt equation

The Kaup–Kupershmidt (KK) equation [23] is a nonlinear PDE, found in (5.1), where α, β, γ satisfy the following relationships:

$$\alpha = \frac{1}{5}\gamma^2, \quad \beta = \frac{5}{2}\gamma. \tag{5.24}$$

The operator Lax pair [15] for the KK equation,

$$u_t + \frac{1}{5}\gamma^2 u^2 u_x + \frac{5}{2}\gamma u_x u_{xx} + \gamma u u_{3x} + u_{5x} = 0, \tag{5.25}$$

is

$$\mathcal{L} = \mathcal{D}_x^3 + \frac{1}{5}\gamma u \mathcal{D}_x + \frac{1}{10}\gamma u_x \mathcal{I}, \tag{5.26}$$

$$\begin{aligned} \mathcal{M} &= 9\mathcal{D}_x^5 + 3\gamma u \mathcal{D}_x^3 + \frac{9}{2}\gamma u_x \mathcal{D}_x^2 + \left(\frac{1}{5}\gamma^2 u^2 + \frac{7}{2}\gamma u_{xx}\right)\mathcal{D}_x \\ &\quad + \left(a(t) + \frac{1}{5}\gamma^2 u u_x + \gamma u_{3x}\right)\mathcal{I}. \end{aligned} \tag{5.27}$$

The matrix Lax pair for the KK equation was shown in Section 3.3.3.

5.2.4 Scaling invariance

We can verify that (5.1) is invariant under the scaling symmetry,

$$(x, t, u) \to \left(\kappa^{-1}x, \kappa^{-5}t, \kappa^2 u\right), \tag{5.28}$$

where κ is a scaling parameter, by assuming that (5.1) scales uniformly via

$$(x, t, u) \to \left(\kappa^a x, \kappa^b t, \kappa^c u\right) = (\tilde{x}, \tilde{t}, \tilde{u}),$$

with undetermined, integer, exponents a, b, and c. Transforming our original equation term by term, using the chain rule as needed, yields

$$u_t = \kappa^{-c}\tilde{u}_{\tilde{t}}\frac{\partial \tilde{t}}{\partial t} = \kappa^{b-c}\tilde{u}_{\tilde{t}}, \tag{5.29}$$

$$\alpha u^2 u_x = \alpha\left(\kappa^{-c}\tilde{u}\right)^2 \kappa^{-c}\tilde{u}_{\tilde{x}}\frac{\partial \tilde{x}}{\partial x} = \alpha\kappa^{-3c}\tilde{u}^2\tilde{u}_{\tilde{x}}\kappa^a = \alpha\kappa^{a-3c}\tilde{u}^2\tilde{u}_{\tilde{x}}, \tag{5.30}$$

and, similarly,

$$\beta u_x u_{2x} = \beta\kappa^{3a-2c}\tilde{u}_{\tilde{x}}\tilde{u}_{2\tilde{x}}, \tag{5.31}$$

$$\gamma u u_{3x} = \gamma \kappa^{3a-2c} \tilde{u} \tilde{u}_{3\tilde{x}}, \tag{5.32}$$

and

$$u_{5x} = \kappa^{5a-c} \tilde{u}_{5\tilde{x}}. \tag{5.33}$$

So, the transformed equation is

$$\kappa^{b-c} \tilde{u}_{\tilde{t}} + \kappa^{a-3c} \alpha \tilde{u}^2 \tilde{u}_{\tilde{x}} + \kappa^{3a-2c} \beta \tilde{u}_{\tilde{x}} \tilde{u}_{2\tilde{x}} + \kappa^{3a-2c} \gamma \tilde{u} \tilde{u}_{3\tilde{x}} + \kappa^{5a-c} \tilde{u}_{5\tilde{x}}. \tag{5.34}$$

One obtains (5.1) in the new variables $(\tilde{x}, \tilde{t}, \tilde{u})$ if $b - c = a - 3c = 3a - 2c = 5a - c$. Without loss of generality, we can set $a = -1$, yielding $b = -5$ and $c = 2$, which determines (5.28).

Defining the *weight*, W, of a variable (or operator) as the exponent of κ in (5.28), we see that $W(x) = -1$, or $W\left(\frac{\partial}{\partial x}\right) = 1$; $W(t) = -5$, or $W\left(\frac{\partial}{\partial t}\right) = 5$; and $W(u) = 2$. The weight of a term is defined as the sum of the weights of each variable (or operator) contained in that term, i.e.,

$$W(u_{5x}) = W(u) + 5W\left(\frac{\partial}{\partial x}\right) = 7$$

In all the examples we have looked at so far, it appears that the scaling symmetry carries over to the operator Lax pair. We will assume that the terms of $\mathcal{L}$ and the terms of $\mathcal{M}$ will each, respectively, have the same scaling weight. This will permit us, in the remainder of this chapter, to focus our attention on a manageable set of possible forms for a Lax pair associated with a PDE. Doing so, we will determine and verify Lax pairs for the three completely integrable PDEs in the family (5.1).

5.3 Computing Lax pairs

Now we are going to generate the Lax pair for each of our three completely integrable equations, (5.17), (5.21), and (5.25).

For Lax's fifth-order KdV equation, (5.17), one can see that each term in $\mathcal{L}$ has weight 2 whereas each term in $\mathcal{M}$ has weight 5, for the SK equation, (5.21), the weights of the terms in $\mathcal{L}$ and $\mathcal{M}$ are 3 and 5, respectively, and for the KK equation, (5.25), they are also 3 and 5. There is a very limited number of terms that can be of weight 2, 3, or 5, so we will start with all possible terms of the necessary weight with unspecified, real coefficients and solve for the constants to find the Lax pair. Since we fix the order of $\mathcal{L}$ in each case, we can assume that the coefficients of the first term, $\mathcal{D}_x^2$ and $\mathcal{D}_x^3$ respectively, is one.

In each case, we will use the following procedure to compute the unknown $\mathcal{L}$ and $\mathcal{M}$.

- Take the candidate $\mathcal{L}$ and $\mathcal{M}$ pair with unspecified coefficients.
- Determine each term in the Lax equation, (2.6), i.e., $\mathcal{L}_t$, $\mathcal{LM}$, and $\mathcal{ML}$.
- Combine the terms into the Lax equation.
- Substitute high derivatives of ψ (from $\mathcal{L}\psi = \lambda\psi$) and any derivative which involve u_t (from the PDE) into the Lax equation.
- Set the resulting expression identically equal to zero, thereby satisfying the Lax equation.
- Create a nonlinear system of equations for the undetermined coefficients by grouping like terms and setting their coefficients to zero.
- Use *Mathematica*'s `Reduce` function to obtain the solutions for each unspecified coefficient.
- Substitute the solution for the coefficients into the candidate Lax pair to obtain a valid Lax pair for each equation.

5.3.1 Derivation of an operator Lax pair for Lax's fifth-order KdV equation

To find a candidate $\mathcal{L}$ and $\mathcal{M}$ for Lax's fifth-order KdV equation, (5.17), we will assume that all the terms in $\mathcal{L}$ have weight 2, whereas all the terms in $\mathcal{M}$ have weight 5. The candidates for $\mathcal{L}$ and $\mathcal{M}$ are then linear combinations of such terms with undetermined coefficients. We can always scale one term to have a leading coefficient of 1. Since $\mathcal{L}$ must have the $\mathcal{D}_x^2$ term, we set its coefficient equal to one. Hence, the candidate $\mathcal{L}$ and $\mathcal{M}$ are

$$\begin{aligned} \mathcal{L} &= \mathcal{D}_x^2 + c_1 u\mathcal{I}, \\ \mathcal{M} &= c_2\mathcal{D}_x^5 + c_3 u\mathcal{D}_x^3 + c_4 u_x\mathcal{D}_x^2 + (c_5 u^2 + c_6 u_{xx})\,\mathcal{D}_x \\ &\quad + (a(t) + c_7 uu_x + c_8 u_{3x})\,\mathcal{I}, \end{aligned} \tag{5.35}$$

where $c_1, \ldots, c_8$ are unknown constants. We will substitute (5.35) into the Lax equation (2.6) and require that the result is zero. So, we compute

$$\mathcal{L}_t = c_1 u_t \mathcal{I},$$

$$\begin{aligned} \mathcal{L}\mathcal{M} &= \mathcal{D}_x^2\left[c_2\mathcal{D}_x^5 + c_3 u\mathcal{D}_x^3 + c_4 u_x\mathcal{D}_x^2 + (c_5 u^2 + c_6 u_{xx})\,\mathcal{D}_x\right. \\ &\quad \left. + (a(t) + c_7 uu_x + c_8 u_{3x})\,\mathcal{I}\right] + c_1 u\left[c_2\mathcal{D}_x^5 + c_3 u\mathcal{D}_x^3 + c_4 u_x\mathcal{D}_x^2\right. \\ &\quad \left. + (c_5 u^2 + c_6 u_{xx})\,\mathcal{D}_x + (a(t) + c_7 uu_x + c_8 u_{3x})\,\mathcal{I}\right] \\ &= c_2\mathcal{D}_x^7 + c_3 u_{xx}\mathcal{D}_x^3 + 2c_3 u_x\mathcal{D}_x^4 + c_3 u\mathcal{D}_x^5 + c_4 u_{3x}\mathcal{D}_x^2 + 2c_4 u_{xx}\mathcal{D}_x^3 \\ &\quad + c_4 u_x\mathcal{D}_x^4 + 2c_5 uu_{xx}\mathcal{D}_x + 2c_5(u_x)^2\mathcal{D}_x + 4c_5 uu_x\mathcal{D}_x^2 + c_5 u^2\mathcal{D}_x^3 \\ &\quad + c_6 u_{4x}\mathcal{D}_x + 2c_6 u_{3x}\mathcal{D}_x^2 + c_6 u_{xx}\mathcal{D}_x^3 + c_7 2(u_x)^2\mathcal{D}_x + c_7 3u_x u_{xx}\mathcal{I} \\ &\quad + c_7 2uu_{xx}\mathcal{D}_x + a(t)\mathcal{D}_x^2 + c_7 uu_x\mathcal{D}_x^2 + c_7 uu_{3x}\mathcal{I} + c_8 u_{3x}\mathcal{D}_x^2 \\ &\quad + 2c_8 u_{4x}\mathcal{D}_x + c_8 u_{5x}\mathcal{I} + c_1 c_2 u\mathcal{D}_x^5 + c_1 c_3 u^2\mathcal{D}_x^3 + c_1 c_4 uu_x\mathcal{D}_x^2 \\ &\quad + (c_1 c_5 u^3 + c_1 c_6 uu_{xx})\,\mathcal{D}_x + (c_1 a(t)u + c_1 c_7 u^2 u_x + c_1 c_8 uu_{3x})\,\mathcal{I} \end{aligned}$$

$$
\begin{aligned}
&= c_2\mathcal{D}_x^7 + (c_1c_2+c_3)u\mathcal{D}_x^5 + (2c_3+c_4)u_x\mathcal{D}_x^4 \\
&\quad +(c_3+2c_4+c_6)u_{xx}\mathcal{D}_x^3 + (c_1c_3+c_5)u^2\mathcal{D}_x^3 \\
&\quad +(c_4+2c_6+c_8)u_{3x}\mathcal{D}_x^2 + (c_1c_4+4c_5+c_7)uu_x\mathcal{D}_x^2 \\
&\quad +c_1c_5u^3\mathcal{D}_x + (c_1c_6+2c_5+2c_7)uu_{xx}\mathcal{D}_x \\
&\quad +2(c_5+c_7)(u_x)^2\mathcal{D}_x + (c_6+2c_8)u_{4x}\mathcal{D}_x \\
&\quad +c_1c_7u^2u_x\mathcal{I} + (c_1c_8+c_7)uu_{3x}\mathcal{I} + 3c_7u_xu_{xx}\mathcal{I} + c_8u_{5x}\mathcal{I}
\end{aligned}
$$

and similarly,

$$
\begin{aligned}
\mathcal{ML} &= c_2\mathcal{D}_x^5\left(\mathcal{D}_x^2+c_1u\mathcal{I}\right) + c_3u\mathcal{D}_x^3\left(\mathcal{D}_x^2+c_1u\mathcal{I}\right) + c_4u_x\mathcal{D}_x^2\left(\mathcal{D}_x^2+c_1u\mathcal{I}\right) \\
&\quad + \left(c_5u^2+c_6u_{xx}\right)\mathcal{D}_x\left(\mathcal{D}_x^2+c_1u\mathcal{I}\right) + \left(c_7uu_x+c_8u_{3x}\right)\mathcal{I}\left(\mathcal{D}_x^2+c_1u\mathcal{I}\right) \\
&= c_2\mathcal{D}_x^7 + (c_1c_2+c_3)u\mathcal{D}_x^5 + (5c_1c_2+c_4)u_x\mathcal{D}_x^4 \\
&\quad +(c_1c_3+c_5)u^2\mathcal{D}_x^3 + (10c_1c_2+c_6)u_{xx}\mathcal{D}_x^3 \\
&\quad +(10c_1c_2+c_8)u_{3x}\mathcal{D}_x^2 + (3c_1c_3+c_1c_4+c_7)uu_x\mathcal{D}_x^2 \\
&\quad +c_1c_5u^3\mathcal{D}_x + 2c_1c_4u_x^2\mathcal{D}_x + (3c_1c_3+c_1c_6)uu_{xx}\mathcal{D}_x + 5c_1c_2u_{4x}\mathcal{D}_x \\
&\quad +(c_1c_5+c_1c_7)u^2u_x\mathcal{I} + (c_1c_4+c_1c_6)u_xu_{xx}\mathcal{I} \\
&\quad +(c_1c_3+c_1c_8)uu_{3x}\mathcal{I} + c_1c_2u_{5x}\mathcal{I}.
\end{aligned}
$$

Hence,

$$
\begin{aligned}
&\mathcal{L}_t + [\mathcal{L},\mathcal{M}] \\
&\qquad = c_1u_t\mathcal{I} + (2c_3-5c_1c_2)u_x\mathcal{D}_x^4 + (c_3+2c_4-10c_1c_2)u_{xx}\mathcal{D}_x^3 \\
&\qquad\quad +(c_4+2c_6-10c_1c_2)u_{3x}\mathcal{D}_x^2 - (3c_1c_3+c_1c_4+c_7)uu_x\mathcal{D}_x^2 \\
&\qquad\quad +(c_1c_4+4c_5+c_7)uu_x\mathcal{D}_x^2 + (2c_5+2c_7-3c_1c_3)uu_{xx}\mathcal{D}_x \\
&\qquad\quad +2(c_5+c_7-2c_1c_4)u_x^2\mathcal{D}_x + (c_6+2c_8-5c_1c_2)u_{4x}\mathcal{D}_x \\
&\qquad\quad -c_1c_5u^2u_x\mathcal{I} + (c_7-c_1c_3)uu_{3x}\mathcal{I} \\
&\qquad\quad +(3c_7-c_1c_4+c_1c_6)u_xu_{xx}\mathcal{I} + (c_8-c_1c_2)u_{5x}\mathcal{I} \\
&\qquad \doteq \mathcal{O}.
\end{aligned}
\tag{5.36}
$$

Using (2.8) and (5.35),

$$\mathcal{L}\psi = \left(\mathcal{D}_x^2 + c_1 u\mathcal{I}\right)\psi = \lambda\psi,$$

and therefore

$$D_x^2 = (\lambda - c_1 u)\mathcal{I}.$$

Consequently, by differentiation with respect to x,

$$D_x^3 = -c_1 u_x \mathcal{I} + (\lambda - c_1 u)\mathcal{D}_x$$

etc. After replacing $\mathcal{D}_x^2, \mathcal{D}_x^3$, and $\mathcal{D}_x^4$ in (5.36) and u_t from (5.1), we require

$$\begin{aligned}
&\mathcal{L} + [\mathcal{L}, \mathcal{M}] = \\
&\quad 2\big(c_1\left(5c_1c_2 - 2c_3 - c_4\right) + c_5 + c_7\big)u_x^2\mathcal{D}_x - \lambda(10c_1c_2 - c_3 - 2c_4)u_{xx}\mathcal{D}_x \\
&\quad +2\big(c_1\left(5c_1c_2 - 2c_3 - c_4\right) + c_5 + c_7\big)uu_{xx}\mathcal{D}_x - (5c_1c_2 - c_6 - 2c_8)u_{4x}\mathcal{D}_x \\
&\quad +\lambda(c_1(10c_1c_2 - 5\lambda c_2 - 7c_3) + 2\lambda c_3 + 4c_5)uu_x\mathcal{I} \\
&\quad -c_1\left(\tfrac{3}{10}\gamma^2 + 5c_1^2c_2 - 5c_1c_3 + 5c_5\right)u^2u_x\mathcal{I} \\
&\quad -\big(c_1\left(2\gamma - 15c_1c_2 + 3c_3 + 3c_4 + c_6\right) - 3c_7\big)u_xu_{xx}\mathcal{I} \\
&\quad -\big(c_1\left(\gamma - 10c_1c_2 + c_3 + c_4 + 2c_6\right) - c_7\big)uu_{3x}\mathcal{I} \\
&\quad -\lambda(10c_1c_2 - c_4 - 2c_6)u_{3x}\mathcal{I} - \big(c_1(1 + c_2) - c_8\big)u_{5x}\mathcal{I} \\
&\quad \equiv \mathcal{O}.
\end{aligned} \tag{5.37}$$

Setting the coefficients of (5.37) equal to zero, we must solve the following nonlinear system:

$$\begin{aligned}
c_1\left(5c_1c_2 - 2c_3 - c_4\right) + c_5 + c_7 &= 0, \\
10c_1c_2 - c_3 - 2c_4 &= 0, \\
5c_1c_2 - c_6 - 2c_8 &= 0, \\
c_1(10c_1c_2 - 5\lambda c_2 - 7c_3) + 2\lambda c_3 + 4c_5 &= 0 \\
c_1\left(\tfrac{3}{10}\gamma^2 + 5c_1^2c_2 - 5c_1c_3 + 5c_5\right) &= 0,
\end{aligned}$$

$$
\begin{aligned}
c_1\left(2\gamma - 15c_1c_2 + 3c_3 + 3c_4 + c_6\right) - 3c_7 &= 0,\\
c_1\left(\gamma - 10c_1c_2 + c_3 + c_4 + 2c_6\right) - c_7 &= 0,\\
10c_1c_2 - c_4 - 2c_6 &= 0,\\
c_1\left(1 + c_2\right) - c_8 &= 0.
\end{aligned}
$$

Using *Mathematica*'s `Reduce` function, we obtain the following solution:

$$
\begin{aligned}
c_1 &= \frac{1}{10}\gamma, & c_2 &= -16, & c_3 &= -4\gamma, & c_4 &= -6\gamma,\\
c_5 &= -\frac{3}{10}\gamma^2, & c_6 &= -5\gamma, & c_7 &= -\frac{3}{10}\gamma^2, & c_8 &= -\frac{3}{2}\gamma.
\end{aligned}
\tag{5.38}
$$

Substituting (5.38) into (5.35) yields the Lax pair, with the assumed weights, found in (5.18) and (5.19) as expected. Notice that λ has dropped out of all c_i in (5.38), which is necessary for $\mathcal{L}$ and $\mathcal{M}$ to be independent of λ.

5.3.2 Derivation of two operator Lax pairs for the SK equation

As before, to find a candidate $\mathcal{L}$ and $\mathcal{M}$ for the SK equation, (5.21), we will assume that all the terms in $\mathcal{L}$ have weight 3 and all the terms in $\mathcal{M}$ have weight 5. We can always scale one term to have a leading coefficient of 1, so since $\mathcal{L}$ must be a third order operator, we set the $\mathcal{D}_x^3$ coefficient equal to one. Thus, the candidate $\mathcal{L}$ and $\mathcal{M}$ are

$$
\begin{aligned}
\mathcal{L} &= \mathcal{D}_x^3 + c_1 u\mathcal{D}_x + c_2 u_x\mathcal{I},\\
\mathcal{M} &= c_3\mathcal{D}_x^5 + c_4 u\mathcal{D}_x^3 + c_5 u_x\mathcal{D}_x^2 + \left(c_6u^2 + c_7u_{xx}\right)\mathcal{D}_x\\
&\quad + \left(a(t) + c_8uu_x + c_9u_{3x}\right)\mathcal{I},
\end{aligned}
\tag{5.39}
$$

where $c_1, \ldots, c_9$ are undetermined constants. As in Section 5.3.1, $\mathcal{L}$ and $\mathcal{M}$ must satisfy the Lax equation (2.6). Therefore, we compute

$$
\mathcal{L}_t = c_1u_t\mathcal{D}_x + c_2u_{xt}\mathcal{I}, \tag{5.40}
$$

$$
\begin{aligned}
\mathcal{LM} \;=\;& c_3\mathcal{D}_x^8 + (c_1c_3+c_4)u\mathcal{D}_x^6 + (c_2c_3+3c_4+c_5)u_x\mathcal{D}_x^5 \\
&+(c_1c_4+c_6)u^2\mathcal{D}_x^4 + (3c_4+3c_5+c_7)u_{xx}\mathcal{D}_x^4 \\
&+(c_1c_4+c_2c_4+c_1c_5+6c_6+c_8)uu_x\mathcal{D}_x^3 \\
&+(c_4+3c_5+3c_7+c_9)u_{3x}\mathcal{D}_x^3 + c_1c_6u^3\mathcal{D}_x^2 \\
&+(c_2c_5+6c_6+3c_8)u_x^2\mathcal{D}_x^2 + (c_1(c_5+c_7)+3(2c_6+c_8))uu_{xx}\mathcal{D}_x^2 \\
&+(c_5+3c_7+3c_9)u_{4x}\mathcal{D}_x^2 + (c_2c_6+c_1(2c_6+c_8))u^2u_x\mathcal{D}_x \\
&+(6c_6+c_2c_7+9c_8)u_xu_{xx}\mathcal{D}_x \\
&+(2c_6+c_1c_7+3c_8+c_1c_9)uu_{3x}\mathcal{D}_x + (c_7+3c_9)u_{5x}\mathcal{D}_x \\
&+(c_1+c_2)c_8uu_x^2\mathcal{I} + c_1c_8u^2u_{xx}\mathcal{I} + 3c_8u_{xx}^2\mathcal{I} \\
&+(4c_8+c_2c_9)u_xu_{3x}\mathcal{I} + (c_8+c_1c_9)uu_{4x}\mathcal{I} + c_9u_{6x}\mathcal{I}
\end{aligned}
\tag{5.41}
$$

and similarly,

$$
\begin{aligned}
\mathcal{ML} \;=\;& c_3\mathcal{D}_x^8 + (c_1c_3+c_4)u\mathcal{D}_x^6 + (5c_1c_3+c_2c_3+c_5)u_x\mathcal{D}_x^5 \\
&+(10c_1c_3+5c_2c_3+c_7)u_{xx}\mathcal{D}_x^4 + (c_1c_4+c_6)u^2\mathcal{D}_x^4 \\
&+(10c_1c_3+10c_2c_3+c_9)u_{3x}\mathcal{D}_x^3 + (3c_1c_4+c_2c_4+c_1c_5+c_8)uu_x\mathcal{D}_x^3 \\
&+c_1c_6u^3\mathcal{D}_x^2 + c_5(2c_1+c_2)u_x^2\mathcal{D}_x^2 + (3c_1c_4+3c_2c_4+c_1c_7)uu_{xx}\mathcal{D}_x^2 \\
&+5c_3(c_1+2c_2)u_{4x}\mathcal{D}_x^2 + (c_1c_6+c_2c_6+c_1c_8)u^2u_x\mathcal{D}_x \\
&+(c_1c_5+2c_2c_5+c_1c_7+c_2c_7)u_xu_{xx}\mathcal{D}_x + c_3(c_1+5c_2)u_{5x}\mathcal{D}_x \\
&+(c_1c_4+3c_2c_4+c_1c_9)uu_{3x}\mathcal{D}_x + c_2c_8uu_x^2\mathcal{I} + c_2c_6u^2u_{xx}\mathcal{I} \\
&+c_2c_7u_{xx}^2\mathcal{I} + c_2(c_5+c_9)u_xu_{3x}\mathcal{I} + c_2c_4uu_{4x}\mathcal{I} + c_2c_3u_{6x}\mathcal{I}.
\end{aligned}
\tag{5.42}
$$

Substitution of (5.40)–(5.42) into (2.6) then yields

$$
\mathcal{L}_t + [\mathcal{L},\mathcal{M}] = \mathfrak{A} + \mathfrak{B} \doteq \mathcal{O} \tag{5.43}
$$

where

$$
\mathfrak{A} = (3c_4-5c_1c_3)u_x\mathcal{D}_x^5 + (3c_4+3c_5-10c_1c_3-5c_2c_3)u_{xx}\mathcal{D}_x^4
$$

$$
\begin{aligned}
\mathfrak{B} = {} & 2(3c_6 - c_1c_4)uu_x\mathcal{D}_x^3 + (c_4 + 3c_5 + 3c_7 - 10c_1c_3 - 10c_2c_3)u_{3x}\mathcal{D}_x^3 \\
& +(6c_6 + 3c_8 - 2c_1c_5)u_x^2\mathcal{D}_x^2 + (c_1c_5 + 6c_6 + 3c_8 - 3c_1c_4 - 3c_2c_4)uu_{xx}\mathcal{D}_x^2 \\
& +(c_5 + 3c_7 + 3c_9 - 5c_1c_3 - 10c_2c_3)u_{4x}\mathcal{D}_x^2 \\
& +c_1u_t\mathcal{D}_x + c_1c_6u^2u_x\mathcal{D}_x \\
& +(6c_6 + 9c_8 - c_1c_7 - c_1c_5 - 2c_2c_5)u_xu_{xx}\mathcal{D}_x \\
& +(2c_6 + c_1c_7 + 3c_8 - c_1c_4 - 3c_2c_4)uu_{3x}\mathcal{D}_x \\
& +(c_7 + 3c_9 - c_1c_3 - 5c_2c_3)u_{5x}\mathcal{D}_x \\
& +c_2u_{xt}\mathcal{I} + c_1c_8uu_x^2\mathcal{I} + (c_1c_8 - c_2c_6)u^2\mathcal{I}u_{xx} + (3c_8 - c_2c_7)u_{xx}^2\mathcal{I} \\
& +(4c_8 - c_2c_5)u_xu_{3x}\mathcal{I} + (c_8u + c_1c_9 - c_2c_4)uu_{4x}\mathcal{I} + (c_9 - c_2c_3)\mathcal{I}u_{6x}.
\end{aligned}
$$

After replacing $\mathcal{D}_x^3, \ldots, \mathcal{D}_x^5$ in (5.43) and u_t from (5.1), we require

$$
\mathcal{L}_t + [\mathcal{L}, \mathcal{M}] = \mathfrak{C} + \mathfrak{D} \tag{5.44}
$$

where

$$
\begin{aligned}
\mathfrak{C} = {} & \Big(\lambda(5c_1c_3 - 3c_4) + c_1(10c_1c_3 + 5c_2c_3 - 6c_4 - 2c_5) - 3c_2c_4 + 6c_6 + 3c_8\Big)u_x^2\mathcal{D}_x^2 \\
& +\big(c_1(10c_1c_3 + 5c_2c_3 - 6c_4 - 2c_5) - 3c_2c_4 + 6c_6 + 3c_8\big)uu_{xx}\mathcal{D}_x^2 \\
& -\big(5c_3(c_1 + 2c_2) - c_5 - 3c_7 - 3c_9\big)u_{4x}\mathcal{D}_x^2 \\
& -\big(c_1\left(\tfrac{1}{5}\gamma^2 + 5c_1^2c_3 - 5c_1c_4 + 5c_6\right)\big)u^2u_x\mathcal{D}_x \\
& +\Big(c_1(15c_1c_3 - \gamma + 25c_2c_3 - 6c_4 - 4c_5 - c_7) \\
& \quad +c_2(5c_2c_3 - 9c_4 - 5c_5) + 6c_6 + 9c_8\Big)u_xu_{xx}\mathcal{D}_x \\
& -\big(5c_3(2c_1 + c_2) - 3c_4 - 3c_5\big)\lambda u_{xx}\mathcal{D}_x \\
& +\big(c_1(10c_1c_3 - \gamma + 10c_2c_3 - 2c_4 - 3c_5 - 2c_7) - 3c_2c_4 + 2c_6 + 3c_8\big)uu_{3x}\mathcal{D}_x \\
& +\big(c_1(1 + c_3) + 5c_2c_3 - c_7 - 3c_9\big)u_{5x}\mathcal{D}_x \\
& +\big(c_1(5c_1c_3 - 5c_4) + 6c_6\big)\lambda uu_x\mathcal{I} \\
& -\big(c_1(5c_1c_2c_3 - 5c_2c_4 - c_8) + c_2\left(\tfrac{2}{5}\gamma^2 + 6c_6\right)\big)uu_x^2\mathcal{I} \\
& +\big(c_1c_8 - c_2\left(\tfrac{1}{5}\gamma^2 + c_6\right)\big)u^2u_{xx}\mathcal{I}
\end{aligned}
$$

and

$$\begin{aligned}\mathfrak{D} = &\big(c_2(10c_1c_3 + 5c_2c_3 - \gamma - 3c_4 - 3c_5 - c_7) + 3c_8\big)u_{xx}^2\mathcal{I}\\ &+\big(c_2(15c_1c_3 + 10c_2c_3 - 2\gamma - 4c_4 - 4c_5 - 3c_7) + 4c_8\big)u_xu_{3x}\mathcal{I}\\ &-\big(10c_3(c_1 + c_2) - c_4 - 3c_5 - 3c_7\big)\lambda u_{3x}\mathcal{I}\\ &+\big(c_1c_9 - c_2(\gamma + c_4) + c_8\big)uu_{4x}\mathcal{I} - \big(c_2(1 + c_3) - c_9\big)u_{6x}\mathcal{I}.\end{aligned}$$

Again, we set each coefficient in (5.44) equal to zero and solve the resulting nonlinear system:

$$\begin{aligned}\lambda(5c_1c_3 - 3c_4) + c_1(10c_1c_3 + 5c_2c_3 - 6c_4 - 2c_5) - 3c_2c_4 + 6c_6 + 3c_8 &= 0,\\ c_1(10c_1c_3 + 5c_2c_3 - 6c_4 - 2c_5) - 3c_2c_4 + 6c_6 + 3c_8 &= 0,\\ 5c_3(c_1 + 2c_2) - c_5 - 3c_7 - 3c_9 &= 0,\\ c_1\left(\tfrac{1}{5}\gamma^2 + 5c_1^2c_3 - 5c_1c_4 + 5c_6\right) &= 0,\\ c_1(15c_1c_3 - \gamma + 25c_2c_3 - 6c_4 - 4c_5 - c_7) + c_2(5c_2c_3 - 9c_4 - 5c_5) + 6c_6 + 9c_8 &= 0,\\ 5c_3(2c_1 + c_2) - 3c_4 - 3c_5 &= 0,\\ c_1(10c_1c_3 - \gamma + 10c_2c_3 - 2c_4 - 3c_5 - 2c_7) - 3c_2c_4 + 2c_6 + 3c_8 &= 0,\\ c_1(1 + c_3) + 5c_2c_3 - c_7 - 3c_9 &= 0,\\ c_1(5c_1c_3 - 5c_4) + 6c_6 &= 0,\\ c_1(5c_1c_2c_3 - 5c_2c_4 - c_8) + c_2\left(\tfrac{2}{5}\gamma^2 + 6c_6\right) &= 0,\\ c_1c_8 - c_2\left(\tfrac{1}{5}\gamma^2 + c_6\right) &= 0,\\ c_2(10c_1c_3 + 5c_2c_3 - \gamma - 3c_4 - 3c_5 - c_7) + 3c_8 &= 0,\\ c_2(15c_1c_3 + 10c_2c_3 - 2\gamma - 4c_4 - 4c_5 - 3c_7) + 4c_8 &= 0,\\ 10c_3(c_1 + c_2) - c_4 - 3c_5 - 3c_7 &= 0,\\ c_1c_9 - c_2(\gamma + c_4) + c_8 &= 0,\\ c_2(1 - c_3) - c_9 &= 0.\end{aligned}$$

Using `Reduce`, we find the following two solutions:

$$\begin{aligned} c_1 &= \tfrac{1}{5}\gamma, & c_2 &= \tfrac{1}{5}\gamma, & c_3 &= 9, \\ c_4 &= 3\gamma & c_5 &= 6\gamma, & c_6 &= \tfrac{1}{5}\gamma^2, \\ c_7 &= 5\gamma, & c_8 &= \tfrac{2}{5}\gamma^2, & c_9 &= 2\gamma \end{aligned} \tag{5.45}$$

and

$$\begin{aligned} c_1 &= \tfrac{1}{5}\gamma, & c_2 &= 0, & c_3 &= 9, \\ c_4 &= 3\gamma & c_5 &= 3\gamma, & c_6 &= \tfrac{1}{5}\gamma^2, \\ c_7 &= 2\gamma, & c_8 &= 0, & c_9 &= 0. \end{aligned} \tag{5.46}$$

Substituting (5.45) into (5.39) gives us a Lax pair [12] for the SK equation:

$$\begin{aligned} \mathcal{L} &= \mathcal{D}_x^3 + \tfrac{1}{5}\gamma u\mathcal{D}_x + \tfrac{1}{5}\gamma u_x\mathcal{I}, \\ \mathcal{M} &= 9\mathcal{D}_x^5 + 3\gamma u\mathcal{D}_x^3 + 6\gamma u_x\mathcal{D}_x^2 + \left(\tfrac{1}{5}\gamma^2u^2 + 5\gamma u_{xx}\right)\mathcal{D}_x \\ &\quad + \left(a(t) + \tfrac{2}{5}\gamma^2uu_x + 2\gamma u_{3x}\right)\mathcal{I}, \end{aligned} \tag{5.47}$$

which, to the best of our knowledge, was previously undiscovered, and substituting (5.46) into (5.39) gives the Lax pair shown in (5.22) and (5.23).

5.3.3 Derivation of an operator Lax pair for the KK equation

For the KK equation, (5.25), we will assume all terms in $\mathcal{L}$ to have weight 3 and all terms in $\mathcal{M}$ to have weight 5. Therefore, we will use the same candidates, (5.39), for $\mathcal{L}$ and $\mathcal{M}$ as in Section 5.3.2.

All the work for the KK equation is identical to the SK case, until we replace $\mathcal{D}_x^3, \ldots, \mathcal{D}_x^5$ and u_t in (5.43) which yields

$$\mathcal{L} + [\mathcal{L}, \mathcal{M}] = \mathfrak{A} \equiv \mathcal{O} \tag{5.48}$$

where

$$
\begin{aligned}
\mathfrak{A} = &-\lambda(5c_1c_3 - 3c_4)u_x\mathcal{D}_x^2 - \big(5c_3(c_1 + 2c_2) - c_5 - 3c_7 - 3c_9\big)u_{4x}\mathcal{D}_x^2 \\
&+\big(c_1(10c_1c_3 + 5c_2c_3 - 6c_4 - 2c_5) - 3c_2c_4 + 6c_6 + 3c_8\big)u_x^2\mathcal{D}_x^2 \\
&+\big(c_1(10c_1c_3 + 5c_2c_3 - 6c_4 - 2c_5) - 3c_2c_4 + 6c_6 + 3c_8\big)uu_{xx}\mathcal{D}_x^2 \\
&+c_1\left(5c_1^2c_3 - 5c_1c_4 + \tfrac{1}{5}\gamma^2 + 5c_6\right)u^2u_x\mathcal{D}_x \\
&-\lambda(10c_1c_3 + 5c_2c_3 - 3c_4 - 3c_5)u_{xx}\mathcal{D}_x \\
&\Big(c_1\left(15c_1c_3 - \tfrac{5}{2}\gamma + 25c_2c_3 - 6c_4 - 4c_5 - c_7\right) \\
&\quad +c_2\big(5c_2c_3 - 9c_4 - 5c_5\big) + 6c_6 + 9c_8\Big)u_xu_{xx}\mathcal{D}_x \\
&+\big(c_1(10c_1c_3 - \gamma + 10c_2c_3 - 2c_4 - 3c_5 - 2c_7) - 3c_2c_4 + 2c_6 + 3c_8\big)uu_{3x}\mathcal{D}_x \\
&-\big(c_1(1 + c_3) + 5c_2c_3 - c_7 - 3c_9\big)u_{5x}\mathcal{D}_x \\
&+\lambda(c_1(5c_1c_3 - 5c_4) + 6c_6)uu_x\mathcal{I} + \left(c_1c_8 - c_2\left(\tfrac{1}{5}\gamma^2 - c_6\right)\right)u^2u_{xx}\mathcal{I} \\
&-\Big(c_1(5c_1c_2c_3 - 5c_2c_4 - c_8) + c_2\left(\tfrac{2}{5}\gamma^2 + 6c_6\right)\Big)uu_x^2\mathcal{I} \\
&+\Big(c_2\left(10c_1c_3 + 5c_2c_3 - \tfrac{5}{2}\gamma - 3c_4 - 3c_5 - c_7\right) + 3c_8\Big)u_{xx}^2\mathcal{I} \\
&-\lambda\big(10c_3(c_1 + c_2) - c_4 - 3c_5 - 3c_7\big)u_{3x}\mathcal{I} \\
&+\Big(c_2\left(15c_1c_3 + 10c_2c_3 - \tfrac{7}{2}\gamma - 4c_4 - 4c_5 - 3c_7\right) + 4c_8\Big)u_xu_{3x}\mathcal{I} \\
&+\big(c_1c_9 - c_2(\gamma - c_4) + c_8\big)uu_{4x}\mathcal{I} - \big(c_2(1 - c_3) + c_9\big)u_{6x}\mathcal{I}
\end{aligned}
$$

Setting each coefficient equal to zero yields a nonlinear system which must be solved,

$$
\begin{aligned}
5c_1c_3 - 3c_4 &= 0, \\
5c_3(c_1 + 2c_2) - c_5 - 3c_7 - 3c_9 &= 0, \\
c_1(10c_1c_3 + 5c_2c_3 - 6c_4 - 2c_5) - 3c_2c_4 + 6c_6 + 3c_8 &= 0, \\
c_1\left(5c_1^2c_3 - 5c_1c_4 + \tfrac{1}{5}\gamma^2 + 5c_6\right) &= 0, \\
10c_1c_3 + 5c_2c_3 - 3c_4 - 3c_5 &= 0,
\end{aligned}
$$

$$
\begin{aligned}
c_1\left(15c_1c_3-\tfrac{5}{2}\gamma+25c_2c_3-6c_4-4c_5-c_7\right)& \\
+c_2(5c_2c_3-9c_4-5c_5)+6c_6+9c_8&=0,\\
c_1(10c_1c_3-\gamma+10c_2c_3-2c_4-3c_5-2c_7)-3c_2c_4+2c_6+3c_8&=0,\\
c_1(1+c_3)+5c_2c_3-c_7-3c_9&=0,\\
c_1(5c_1c_3-5c_4)+6c_6&=0,\\
c_1c_8-c_2\left(\tfrac{1}{5}\gamma^2-c_6\right)&=0,\\
c_1(5c_1c_2c_3-5c_2c_4-c_8)+2c_2\left(\tfrac{1}{5}\gamma^2+3c_6\right)&=0,\\
c_2\left(10c_1c_3+5c_2c_3-\tfrac{5}{2}\gamma-3c_4-3c_5-c_7\right)+3c_8&=0,\\
10c_3(c_1+c_2)-c_4-3c_5-3c_7&=0,\\
c_2\left(15c_1c_3+10c_2c_3-\tfrac{7}{2}\gamma-4c_4-4c_5-3c_7\right)+4c_8&=0,\\
c_1c_9-c_2(\gamma-c_4)+c_8&=0,\\
c_2(1-c_3)+c_9&=0.
\end{aligned}
$$

Using `Reduce`, we obtain the following solution:

$$
\begin{array}{lllllllll}
c_1 & = & \frac{1}{5}\gamma, & c_2 & = & \frac{1}{10}\gamma, & c_3 & = & 9,\\
c_4 & = & 3\gamma & c_5 & = & \frac{9}{2}\gamma, & c_6 & = & \frac{1}{5}\gamma^2,\\
c_7 & = & \frac{7}{2}\gamma, & c_8 & = & \frac{1}{5}\gamma^2, & c_9 & = & \gamma
\end{array}
\tag{5.49}
$$

Substituting (5.49) into (5.39) yields the Lax pair shown in (5.26) and (5.27).

5.4 Derivation of some operator Lax pairs for a family of fifth-order KdV equations

For the entire family (5.1) of fifth-order KdV equations, we assume that

$$
\begin{aligned}
\mathcal{L} &= c_1\mathcal{D}_x^3+c_2u\mathcal{D}_x+c_3\mathcal{I}u_x+c_4\mathcal{D}_x^2+c_5u\mathcal{I},\\
\mathcal{M} &= c_6\mathcal{D}_x^5+c_7u\mathcal{D}_x^3+c_8u_x\mathcal{D}_x^2+c_9u^2\mathcal{D}_x\\
&\quad +c_{10}u_{xx}\mathcal{D}_x+c_{11}uu_x\mathcal{I}+c_{12}u_{3x}\mathcal{I},
\end{aligned}
\tag{5.50}
$$

where $c_1, \ldots, c_{12}$ are arbitrary constants. As before, we compute the Lax equation (2.6) and solve for the undetermined coefficients including, this time, α and β in terms of the parameter γ.

We go through the computations twice. First assuming that $c_1 \neq 0$ and then assuming that $c_1 = 0$. We use `Solve` early in **laxpairtester.m**, so there is a built-in assumption that the coefficient of the highest-order $\mathcal{D}_x$ in $\mathcal{L}$ is not zero.

Assuming $c_1 \neq 0$, we derive the Lax equation using (5.50) and find, after all possible replacements are finished,

$$\mathcal{L}_t + [\mathcal{L}, \mathcal{M}] = \tfrac{1}{c_1^3}\big(\mathfrak{A}\mathcal{D}_x^2 + \mathfrak{B}\mathcal{D}_x + \mathfrak{C}\mathcal{I}\big) \equiv \mathcal{O}, \tag{5.51}$$

where

$$\begin{aligned}
\mathfrak{A} \;=\;& c_1^3\big(c_1(c_8 + 3c_{10} + 3c_{12}) - 5c_2c_6 - 10c_3c_6\big)u_{4x} \\
&+\Big(c_1\big(\lambda c_1(3c_1c_7 - 5c_2c_6) - c_4^2(c_4c_7 - 5c_5c_6)\big) + 5c_2c_4^3c_6\Big)u_x \\
&-c_1\big(c_1^2(2c_4c_9 + 6c_5c_7) - 2c_1c_2(3c_4c_7 + 5c_5c_6) + 10c_2^2c_4c_6\big)uu_x \\
&+c_1^2\big(3c_1^2(2c_9 + c_{11}) - 2c_1c_2(3c_7 + c_8) - 3c_1c_3c_7 + 5c_2c_6(2c_2 + c_3)\big)u_x^2 \\
&-c_1^2\big(c_1(10c_5c_6 + c_4c_7 + 2c_4c_8 + c_4c_{10}) - 10c_4c_6(c_2 + c_3)\big)u_{3x} \\
&+c_1c_4\big(c_1(2c_4c_7 + c_4c_8 + 10c_5c_6) - 5c_4c_6(2c_2 + c_3)\big)u_{xx} \\
&+c_1^2\big(3c_1^2(2c_9 + c_{11}) - 2c_1c_2(3c_7 + c_8) - 3c_1c_3c_7 + 5c_2c_6(2c_2 + c_3)\big)uu_{xx},
\end{aligned}$$

$$\begin{aligned}
\mathfrak{B} \;=\;& -\lambda c_1\big(c_1(c_4c_7 + 5c_5c_6) - 5c_2c_4c_6\big)u_x \\
&+\Big(c_1\big(c_1c_5(5c_5c_6 + c_4c_7) - c_2c_4(10c_5c_6 + c_4c_7)\big) + 5c_2^2c_4^2c_6\Big)uu_x \\
&-c_1c_2\big(c_1^2(\alpha + 5c_9) - 5c_2(c_1c_7 - c_2c_6)\big)u^2u_x \\
&+c_1\Big(2c_1^2(c_4c_9 + c_4c_{11} - 3c_5c_7 - c_5c_8) + 5c_2c_4c_6(c_2 - c_3) \\
&\qquad +c_1(c_2c_4c_7 + c_3c_4c_7 + 15c_2c_5c_6 + 5c_3c_5c_6)\Big)u_x^2 \\
&+c_1^2\lambda\left(3c_1(c_7 + c_8) - 5c_6(2c_2 + c_3)\right)u_{xx}
\end{aligned}$$

$$
\begin{aligned}
&+c_1\Big(c_1^2(2c_4c_9+2c_4c_{11}-6c_5c_7-3c_5c_8)+c_1c_2c_4(2c_7+c_8)\\
&\qquad +5c_1c_5c_6(4c_2+c_3)-2c_2c_4c_6(5c_2+c_3)\Big)uu_{xx}\\
&+c_1^2\Big(3c_1^2(2c_9+3c_{11})-c_1c_2(\beta+6c_7+4c_8+c_{10})\\
&\qquad -c_1c_3(9c_7+5c_8)+5c_6(3c_2^2+5c_2c_3+c_3^2)\Big)u_xu_{xx}\\
&+c_1^2\Big(c_1^2(2c_9+3c_{11})-c_1c_2(\gamma u+2c_7+3c_8+2c_{10})\\
&\qquad -3c_1c_3c_7+10c_2c_6(c_2+c_3)\Big)uu_{3x}\\
&+c_1^3\big(c_4(c_{10}+2c_{12})-5c_5c_6\big)u_{4x}\\
&+c_1^3\big(c_1(c_{10}+3c_{12})-c_2(1+c_6)-5c_3c_6\big)u_{5x},
\end{aligned}
$$

and

$$
\begin{aligned}
\mathfrak{C} \;=\;& \Big(c_1\left(5c_1c_5^2c_6+c_1c_4c_5c_7-5c_2c_4c_5c_6-5c_3c_4c_5c_6-c_3c_4^2c_7\right)+5c_2c_3c_4^2c_6\Big)u_x^2\\
&+c_1\big(c_1^2(c_2c_{11}-2\alpha c_3-6c_3c_9)+5c_2c_3(c_1c_7-5c_2c_6)\big)uu_x^2\\
&-\lambda c_1\big(c_1(2c_4c_7+c_4c_8+10c_5c_6)-5c_4c_6(2c_2+c_3)\big)u_{xx}\\
&+c_1\big(c_2c_5(2c_4c_7+c_4c_8+10c_5c_6)-5c_4c_5c_6(2c_2+c_3)\big)uu_{xx}\\
&+c_1^3\left(c_2c_{11}-c_3(\alpha+c_9)\right)u^2u_{xx}\\
&+c_1\Big(c_1^2(3c_4c_{11}-\beta c_5-4c_5c_8-c_5c_{10}-6c_5c_7)-5c_3c_4c_6(3c_2+c_3)\\
&\qquad +c_1\big(15c_2c_5c_6+3c_3c_4c_7+c_3c_4c_8+20c_3c_5c_6\big)\Big)u_xu_{xx}\\
&+c_1^2\big(3c_1^2c_{11}-c_1c_3(\beta+3c_7+3c_8+c_{10})+5c_3c_6(2c_2+c_3)\big)(u_{xx})^2\\
&+\lambda c_1^2\big(c_1(c_7+3c_8+3c_{10})-10c_6(c_2+c_3)\big)u_{3x}\\
&+c_1^2\big(c_1c_4c_{11}-c_1c_5(\gamma+2c_7+3c_8+c_{10})+10c_5c_6(c_2+c_3)\big)uu_{3x}\\
&+c_1^2\big(4c_1^2c_{11}-c_1c_3(\beta+\gamma+4c_7+4c_8+3c_{10})+5c_3c_6(3c_2+2c_3)\big)u_xu_{3x}\\
&+c_1^3\big(c_1c_{11}+c_2c_{12}-c_3(\gamma+c_7)\big)uu_{4x}+\lambda c_4\big(c_1c_4c_7+5c_1c_5c_6-5c_2c_4c_6\big)u_x\\
&+\big(c_1(6\lambda c_1^2c_9-5\lambda c_1c_2c_7+5\lambda c_2^2c_6-5c_4c_5^2c_6-c_4^2c_5c_7)+5c_2c_4^2c_5c_6\big)uu_x\\
&-c_1c_5\big(c_1^2(\alpha+7c_9)-5c_2(c_1c_7-c_2c_6)\big)u^2u_x\\
&+c_1^3\big(c_4c_{12}-c_5(1+c_6)\big)u_{5x}+c_1^3\big(c_1c_{12}-c_3(1+c_6)\big)u_{6x}.
\end{aligned}
$$

It is of interest to some that there are very few occurrences of α, β, and γ in $\mathfrak{A}, \mathfrak{B}$

and $\mathfrak{C}$.

Equation (5.51) leads to the following nonlinear system (recall that $c_1 \neq 0$):

$$
\begin{aligned}
c_1(c_8 + 3c_{10} + 3c_{12}) - 5c_6(c_2 + 2c_3) &= 0,\\
c_1(3\lambda c_1^2 c_7 - 5\lambda c_1 c_2 c_6 - 5c_4^2 c_5 c_6 - c_4^3 c_7) + 5c_2 c_4^3 c_6 &= 0,\\
c_4(c_1^2 c_9 - 3c_1 c_2 c_7 + 5c_2^2 c_6) + c_1 c_5(3c_1 c_7 - 5c_2 c_6) &= 0,\\
3c_1^2(2c_9 + c_{11}) - c_1(6c_2 c_7 + 2c_2 c_8 + 3c_3 c_7) + 5c_2 c_6(2c_2 + c_3) &= 0,\\
c_1(c_4 c_7 + 2c_4 c_8 + c_4 c_{10} + 10c_5 c_6) - 10c_4 c_6(c_2 + c_3) &= 0,\\
c_4\big(c_2(2c_4 c_7 + c_4 c_8 + 10c_5 c_6) - 5c_4 c_6(2c_2 + c_3)\big) &= 0,\\
c_1(c_4 c_7 + 5c_5 c_6) - 5c_2 c_4 c_6 &= 0,\\
c_1\big(c_1 c_5(5c_5 c_6 + c_4 c_7) - c_2 c_4(10c_5 c_6 + c_4 c_7)\big) + 5c_2^2 c_4^2 c_6 &= 0,\\
c_2\big(c_1^2(\alpha + 5c_9) - 5c_2(c_1 c_7 - c_2 c_6)\big) &= 0,\\
c_4\big(2c_1^2(c_9 + c_{11}) + c_1 c_7(c_2 + c_3) - 5c_2 c_6(c_2 + c_3)\big)&\\
-c_1 c_5\big(2c_1(3c_7 + c_8) - 5c_6(3c_2 c_3)\big) &= 0,\\
3c_1(c_7 + c_8) - 5c_6(2c_2 + c_3) &= 0,\\
c_1^2(2c_4 c_9 + 2c_4 c_{11} - 6c_5 c_7 - 3c_5 c_8) + c_1 c_2 c_4(2c_7 + c_8)&\\
+5c_1 c_5 c_6(4c_2 + c_3) - 2c_2 c_4 c_6(5c_2 + c_3) &= 0,\\
3c_1^2(2c_9 + 3c_{11}) - c_1 c_2(\beta + 6c_7 + 4c_8 + c_{10})&\\
-c_1 c_3(9c_7 + 5c_8) + 5c_6(3c_2^2 + 5c_2 c_3 + c_3^2) &= 0,\\
c_1^2(2c_9 + 3c_{11}) - c_1 c_2(\gamma u + 2c_7 + 3c_8 + 2c_{10})&\\
-3c_1 c_3 c_7 + 10c_2 c_6(c_2 + c_3) &= 0,\\
c_4(c_{10} + 2c_{12}) - 5c_5 c_6 &= 0,\\
c_1(c_{10} + 3c_{12}) - c_2(1 + c_6) - 5c_3 c_6 &= 0,\\
c_1\left(5c_1 c_5^2 c_6 + c_4 c_5(c_1 c_7 - 5c_2 c_6 - 5c_3 c_6) - c_3 c_4^2 c_7\right) + 5c_2 c_3 c_4^2 c_6 &= 0,\\
c_1^2(c_2 c_{11} - 2\alpha c_3 - 6c_3 c_9) + 5c_2 c_3(c_1 c_7 - 5c_2 c_6) &= 0,\\
c_1(2c_4 c_7 + c_4 c_8 + 10c_5 c_6) - 5c_4 c_6(2c_2 + c_3) &= 0,\\
c_2 c_5(2c_4 c_7 + c_4 c_8 + 10c_5 c_6) - 5c_4 c_5 c_6(2c_2 + c_3) &= 0,
\end{aligned}
$$

$$
\begin{aligned}
c_2c_{11} - c_3(\alpha + c_9) &= 0,\\
c_1^2(3c_4c_{11} - \beta c_5 - 4c_5c_8 - c_5c_{10} - 6c_5c_7) - 5c_3c_4c_6(3c_2 + c_3)&\\
+c_1(15c_2c_5c_6 + 3c_3c_4c_7 + c_3c_4c_8 + 20c_3c_5c_6) &= 0,\\
3c_1^2c_{11} - c_1c_3(\beta + 3c_7 + 3c_8 + c_{10}) + 5c_3c_6(2c_2 + c_3) &= 0,\\
c_1(c_7 + 3c_8 + 3c_{10}) - 10c_6(c_2 + c_3) &= 0,\\
c_1c_4c_{11} - c_1c_5(\gamma + 2c_7 + 3c_8 + c_{10}) + 10c_5c_6(c_2 + c_3) &= 0,\\
4c_1^2c_{11} - c_1c_3(\beta + \gamma + 4c_7 + 4c_8 + 3c_{10}) + 5c_3c_6(3c_2 + 2c_3) &= 0,\\
c_1c_{11} + c_2c_{12} - c_3(\gamma + c_7) &= 0,\\
c_4(c_1c_4c_7 + 5c_1c_5c_6 - 5c_2c_4c_6) &= 0,\\
c_1(6\lambda c_1^2c_9 - 5\lambda c_1c_2c_7 + 5\lambda c_2^2c_6 - 5c_4c_5^2c_6 - c_4^2c_5c_7) + 5c_2c_4^2c_5c_6 &= 0,\\
c_5\big(c_1^2(\alpha + 7c_9) - 5c_2(c_1c_7 - c_2c_6)\big) &= 0,\\
c_4c_{12} - c_5(1 + c_6) &= 0,\\
c_1c_{12} - c_3(1 + c_6) &= 0,
\end{aligned}
$$

which, after using `Reduce`, yields three solutions to α, β, and c_i for $i = 1, \ldots, 12$ in terms of γ. First,

$$
\begin{array}{lllllllll}
\multicolumn{3}{c}{\alpha = \frac{1}{5}\gamma^2} & \multicolumn{3}{c}{\beta = \gamma} & \multicolumn{3}{c}{,}\\
c_1 &=& 1, & c_2 &=& \frac{1}{5}\gamma, & c_3 &=& 0,\\
c_4 &=& 0, & c_5 &=& 0, & c_6 &=& 9,\\
c_7 &=& 3\gamma & c_8 &=& 3\gamma, & c_9 &=& \frac{1}{5}\gamma^2,\\
c_{10} &=& 2\gamma, & c_{11} &=& 0, & c_{12} &=& 0,
\end{array}
\tag{5.52}
$$

which, after replacement in (5.50), leads to one operator Lax pair, (5.47), for the SK equation, (5.21).

Second,

$$\alpha = \frac{1}{5}\gamma^2 \qquad \beta = \gamma \qquad ,$$

$$\begin{array}{lllllllll} c_1 &=& 1, & c_2 &=& \frac{1}{5}\gamma, & c_3 &=& \frac{1}{5}\gamma, \\ c_4 &=& 0, & c_5 &=& 0, & c_6 &=& 9, \\ c_7 &=& 3\gamma, & c_8 &=& 6\gamma, & c_9 &=& \frac{1}{5}\gamma^2, \\ c_{10} &=& 5\gamma, & c_{11} &=& \frac{2}{5}\gamma^2, & c_{12} &=& 2\gamma, \end{array} \tag{5.53}$$

which, after replacement in (5.50), leads to a second operator Lax pair, (5.22) and (5.23), for the SK equation, (5.21).

Finally, third,

$$\alpha = \frac{1}{5}\gamma^2 \qquad \beta = \frac{5}{2}\gamma \qquad ,$$

$$\begin{array}{lllllllll} c_1 &=& 1, & c_2 &=& \frac{1}{5}\gamma, & c_3 &=& \frac{1}{10}\gamma, \\ c_4 &=& 0, & c_5 &=& 0, & c_6 &=& 9, \\ c_7 &=& 3\gamma & c_8 &=& \frac{9}{2}\gamma, & c_9 &=& \frac{1}{5}\gamma^2, \\ c_{10} &=& \frac{7}{2}\gamma, & c_{11} &=& \frac{1}{5}\gamma^2, & c_{12} &=& \gamma, \end{array} \tag{5.54}$$

which, after replacement in (5.50), leads to the operator Lax pair, (5.26) and (5.27), for the KK equation, (5.25).

Similarly, if we assume $c_1 = 0$, we get the solution where $c_4 \neq 0$

$$\alpha = \frac{3}{10}\gamma^2 \qquad \beta = 2\gamma \qquad ,$$

$$\begin{array}{lllllllll} c_1 &=& 0 & c_2 &=& 0 & c_3 &=& 0 \\ c_4 &=& 1 & c_5 &=& \frac{1}{10}\gamma, & c_6 &=& -16, \\ c_7 &=& -4\gamma, & c_8 &=& -6\gamma, & c_9 &=& -\frac{3}{10}\gamma^2, \\ c_{10} &=& -5\gamma, & c_{11} &=& -\frac{3}{10}\gamma^2, & c_{12} &=& -\frac{3}{2}\gamma. \end{array} \tag{5.55}$$

which, after replacement in (5.50), leads to the operator Lax pair, (5.18) and (5.19),

for the Lax's fifth-order KdV equation, (5.17).

As expected, substituting (5.52) into (5.50) yields one Lax pair, (5.22) and (5.23), for the SK equation, (5.21). Substituting (5.53) into (5.50) yields a second Lax pair, (5.47), for the SK equation, (5.21). Substituting (5.54) into (5.50) yields a Lax pair, (5.26) and (5.27), for the KK equation, (5.25). Finally, substituting (5.55) into (5.50) yields a Lax pair, (5.18) and (5.19) for Lax's fifth-order KdV equation, (5.17).

It is surprising that such a large nonlinear system, under a single application of `Reduce`, resolves quickly to all variables being given in terms of γ, including α and β. We have shown that these three sets of ratios for α and β in terms of γ are the *only* ratios which provide nontrivial solutions for this family of PDEs, (5.1), and these orders of Lax pairs. This discovery may lead to some possible deeper connections during future study.

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

In this thesis, we gave a brief overview of the history of Lax pairs and the tests required to check Lax pair candidates. We laid out examples of PDEs along with their Lax pairs that we verified were valid. We showed the two forms of Lax pairs, i.e., operator and matrix forms, and illustrated how to convert from one form to another.

We have introduced a new *Mathematica* package, **laxpairtester.m**, which automatically tests if a Lax pair of a PDE is valid. As far as we know, this is the first *Mathematica* package offered which carries out the tedious, computational testing.

This new package works for all polynomial PDEs which contain one or more independent variables and one or more dependent variables. It also can take a possible Lax pair with some unspecified constants and, as long as the Lax pair is in the correct form, assist in determining these constants so that the Lax pair works for the given PDE. This will allow us to correct small errors in Lax pairs which are reported in the literature, find a Lax pair for a PDE using candidate forms of the Lax pair along with undetermined coefficients, or, in the best case, find all the constants and Lax pairs for a given family of PDEs and candidate forms.

For simple cases, the package can transform the operator Lax pair into a matrix Lax pair and vice versa. For more complicated examples, the complexity of the Lax pairs increases so much that it becomes infeasible for the current code to do the transformation. In the future, one could try to find better algorithms for these transformations which will allow the code to work with any Lax pair.

We showed the key algorithms used in **laxpairtester.m**. Finally, we showed the versatility of **laxpairtester.m** by taking a general Lax pair candidate and solving for the unspecified coefficients to find new Lax pairs. We also found a possibly new Lax pair for the SK equation with the help of our software, and we showed that these

three known instances of this family of PDEs are the only ones that admit a Lax pair of the given forms.

In the future, we hope to extend these algorithms to overcome the difficulties encountered: allow more flexibility in the form of Lax pair given, improve the ability to handle integrals in a Lax pair, and fully automate the construction of candidate Lax pairs with undetermined coefficients by exploiting the scaling properties of the nonlinear PDE. This could be a big step towards the development of *Mathematica* code that can automatically find new Lax pairs. We would examine whether numerical methods could give insights into the existence and form of Lax pairs in various cases. We also hope to grant the ability to transform Lax pairs in both directions. ❋

REFERENCES CITED

[1] M. J. Ablowitz and H. Segur. The inverse scattering transform on the infinite integral. In *Solitons and the Inverse Scattering Transform*, volume 4 of *Studies in Applied Mathematics*, chapter 1, pages 1–91. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1981.

[2] Th. W. Ruijgrok. Solitons. *Acta Phys. Pol. B*, B14(3):139–156, 1983.

[3] J. Scott Russell. Report on waves. *Report of the fourteenth Meeting of the British Association for the Advancement of Science*, pages 311–390 + 57 plates, 1844.

[4] T. Aktosun. Inverse scattering transform and the theory of solitons. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 4960–4971. Springer New York, 2009.

[5] A. C. Newell. The history of the soliton. *J. Appl. Mech*, 50(4b):1127–1138, 1983.

[6] M. D. Kruskal and N. J. Zabusky. Progress on the Fermi-Pasta-Ulam nonlinear string problem. *Princeton Plasma Phys. Lab. Annu. Rep.*, pages 301–308, 1963.

[7] M. D. Kruskal and N. J. Zabusky. Interaction of "solitons" in a collisionless plasma and the recurrence of initial states. *Phys. Rev. Lett.*, 15(6):240–243, 1965.

[8] M.J. Ablowitz and H. Segur. *Solitons and the Inverse Scattering Transform*. SIAM studies in applied mathematics. Society for Industrial and Applied Mathematics, 1981.

[9] P. D. Lax. Integrals of nonlinear equations of evolution and solitary waves. *Comm. Pure Appl. Math.*, 21(5):467–490, 1968.

[10] V. E. Zakharov and A. B. Shabat. Exact theory of two-dimensional selffocusing and one-dimensional selfmodulation of waves in nonlinear media. *Sov. Phys. JETP*, 34:62–69, 1972.

[11] V. G. Drinfel'd and V. V. Sokolov. Lie algebras and equations of korteweg-de vries type. *J. Math. Sci.*, 30:1975–2036, 1985.

[12] J. Larue and W. Hereman. **laxpairtester.m**: A *Mathematica* package for the symbolic verification of operator and matrix Lax pairs for completely integrable nonlinear PDEs. Software is available at `http://inside.mines.edu/~whereman` under scientific software, 2011.

[13] M.J. Ablowitz and P.A. Clarkson. *Solitons, nonlinear evolution equations and inverse scattering*. London Mathematical Society lecture note series. Cambridge University Press, 1991.

[14] W.W. Adams and P. Loustaunau. *An introduction to Gröbner bases*. Graduate studies in mathematics. American Mathematical Society, 1994.

[15] W. Hereman. A list of nonlinear PDEs and DDEs with their Lax pairs in operator and matrix form. Private Communication, 2011.

[16] Camassa, R. and Holm, D. An integrable shallow water equation with peaked solitons. *Phys. Rev. Lett.*, 71(11):1661–1664, 1993.

[17] R. Camassa and D.D. Holm. A new integrable shallow water equation. *Adv. Appl. Mech.*, 31:1–33, 1994.

[18] T. Schäfer and C.E. Wayne. Propagation of ultra-short optical pulses in cubic nonlinear media. *Phys. D: Nonlinear Phenom.*, 196(1-2):90–105, 2004. SPE equation.

[19] A. Sakovich and S. Sakovich. The short pulse equation is integrable. *J. Phys. Soc. Jpn.*, 74(1):239, 2005.

[20] Ü. Göktaş and W. Hereman. Symbolic computation of conserved densities for systems of nonlinear evolution equations. *J. Symb. Comput.*, 24(5):591–621, 1997.

[21] K. Sawada and T. Kotera. A method for finding n-soliton solutions of the KdV equation and KdV-like equation. *Prog. Theor. Phys.*, 51(5):1355–1367, 1974.

[22] J. M. Dye and A. Parker. On bidirectional fifth-order nonlinear evolution equations, Lax pairs, and directionally dependent solitary waves. *J. Math. Phys.*, 42(6):2567–2589, 2001.

[23] D. J. Kaup. On the inverse scattering problem for cubic eigenvalue problems of the class $\psi_{xxx} + 6Q\psi_x + 6R\psi = \lambda\psi$. *Stud. Appl. Math.*, 62:189–216, 1980.

[24] M. C. Nucci. Pseudopotentials, Lax equations and Bäcklund transformations for nonlinear evolution equations. *J. Phys. A: Math. Gen*, 21:73–79, 1988.

[25] L. Bianchi. Lezioni di geometria differenziale. *Spoerri, Pisa*, 1:360, 1902.

[26] M. J. Ablowitz, D. J. Kaup, A. C. Newell, and H. Segur. Method for solving the sine-Gordon equation. *Phys. Rev. Lett.*, 30(25):1262–1264, Jun 1973.

[27] J. Boussinesq. Théorie de lintumescence liquide appelée onde solitaire ou de translation se propageant dans un canal rectangulaire. *C. R. Acad. Sci. (Paris)*, 72:755–759, 1871.

[28] P. Deift, C. Tomei, and E. Trubowitz. Inverse scattering and the Boussinesq equation. *Commun. Pure Appl. Math.*, 35(5):567–628, 1982.

[29] M. Kruskal. Nonlinear wave equations. In J. Moser, editor, *Dynamical Systems, Theory and Applications*, volume 38 of *Lecture Notes in Physics*, pages 310–354. Springer Berlin / Heidelberg, 1975.

[30] F. Gesztesy and K. Unterkofler. Isospectral deformations for Strum-Liouville and Dirac-type operators and associated nonlinear evolution equations. *Rep. Math. Phys.*, 31(2):113–137, 1992.

[31] R. Hirota and J. Satsuma. N-Soliton Solutions of Model Equations for Shallow Water Waves. *J. Phy. Soc. Jpn.*, 40:611, 1976.

[32] R. Hirota and J. Satsuma. Soliton solutions of a coupled Korteweg–de Vries equation. *Phys. Lett. A*, 85(8–9):407–408, 1981.

[33] J. Satsuma and R. Hirota. A coupled KdV equation is one case of the four-reduction of the KP hierarchy. *J. Phy. Soc. Jpn.*, 51(10):3390–3397, 1982.

[34] M. Musette and R. Conte. Algorithmic method for deriving lax pairs from the invariant Painlevé analysis of nonlinear partial differential equations. *J. Math. Phys.*, 32(6):1450, 1991.

[35] J. Liouville. Sur l' équation aux différence partielles $\frac{d^2 \log \lambda}{dudv} \pm \frac{\lambda}{2a^2} = 0$. *C.R. Acad. Sci. Paris*, 36:71–72, 1853.

[36] W. Pingfeng, G. Xiwen, X. Zhuang, and Z. Huanqiang. A note on infinite conservation laws in liouville equation. *Chin. Phys. Lett.*, 9(7):337, 1992.

[37] A.V. Zhiber and A.B. Shabat. Klein–Gordon equations with a nontrivial group. *Sov. Phys. Dokl.*, 24(8):607–609, 1979.

[38] H.-Q. Zhou, L.-J. Jiang, and Q. Jiang. Connection between infinite conservation laws in a coupled Ziber-Shabat-Mikhailov equation and a coupled Kaup-Kupershmidt equation. *Phys. Lett. A*, 143(6-7):288–292, 1990.

[39] B.B. Kadomtsev and V.I. Petviashvili. On the stability of solitary waves in weakly dispersing media. In *Sov. Phys. Dokl.*, volume 15, pages 539–541, 1970.

[40] J.M. Burgers. *Mathematical examples illustrating relations occurring in the theory of turbulent fluid motion*. Noord-Hollandsche Uitg. Mij., 1939.

[41] W. Hereman and A. Nuseir. Symbolic methods to construct exact solutions of nonlinear partial differential equations. *Mathematics and Computers in Simulation*, 43(1):13–27, 1997.

APPENDIX A - DATA FILES

Samples of existing data files are included in the following pages. Any new data file must contain all lines which are not 'commented out' by `(* *)`. It is recommended that an existing data file be copied and modified.

Three example data files are included here: the KdV equation (basic example with both operator and matrix forms), the Boussinesq system (includes two equations in the system input), and the fully general Lax's fifth-order KdV equation (includes constants and parameters).

```
(* ::Package:: *)

(* jwl_kdv.m *)
(* Menu item 1-1 *)
(* Last Updated:  24 August, 2010, 21:30 by jwl at home *)


(***  KdV Equation  ***)

titleInput = "KdV Equation";

(*
The lhs of all equations in the pde system (of form lhs = 0)
in list form (separated by commas.)
*)
eqInput={D[u[x,t],t]+\[Alpha]*u[x,t]*D[u[x,t],x]+D[u[x,t],{x,3}]};

(* The X Matrix *)
xMatInput={{0,1},{(\[Lambda]-(\[Alpha]/6)*u[x,t]),0}};
(* The T Matrix *)
tMatInput={{a[t]+(\[Alpha]/6)*D[u[x,t],x],
            -(4*\[Lambda]+(\[Alpha]/3)*u[x,t])},
```

```
        {(-4*\[Lambda]^2+(\[Alpha]/3)\[Lambda]*u[x,t])
            +(\[Alpha]^2/18)*u[x,t]^2+(\[Alpha]/6)*D[u[x,t],{x,2}],
         a[t]-(\[Alpha]/6)*D[u[x,t],x]}};

(* The L Operator *)
laxLOpInput[x_,t_]:=(D[#,{x,2}]+\[Alpha]/6*u[x,t]#)&;
(* The M Operator *)
laxMOpInput[x_,t_]:=(-4*D[#,{x,3}]-\[Alpha]*u[x,t]*D[#,{x,1}]
                    -((\[Alpha]/2)*D[u[x,t],{x,1}]-a[t])#)&;

(* List of dependent variables *)
depVarListInput = {u};
(* List of independent space-like variables *)
indepSpaceVarListInput = {x};
(*
List of independent time-like variables
(currently only works with one time-like variable)
*)
indepTimeVarListInput = {t};

(* The function L and M will be applied to *)
laxOpFunInput=\[Psi];

(*
If you only have the X and T possible LAX Pair
or only have the L and M possible LAX Pair,
set those values you do not have to Null.
*)


(* jwl_kdv.m *)
(* end of file *)
```

```
(* ::Package:: *)

(* d_jwlBousDeift0306.m *)
(* Menu item 1-14 *)
(* Last Updated: 6 March, 2011, 15:05 by jwl at home *)

(*** Boussinesq from Deift ***)
titleInput = "Boussinesq System from Deift";
eqInput={D[u[x,t],t]+3*D[v[x,t],x],
      D[v[x,t],t]+D[u[x,t],{x,3}]-8*u[x,t]*D[u[x,t],x]};

(* The X Matrix *)
xMatInput= Null;
(* The T Matrix *)
tMatInput= Null;

(* The L Operator *)
laxLOpInput[x_,t_]:= (D[#,{x,3}]-u[x,t]*D[#,x]-D[u[x,t]*#,x]
                      -I*v[x,t]*#)&;
(* The M Operator *)
laxMOpInput[x_,t_]:= I*(3*D[#,{x,2}]-4*u[x,t]*#)&
(* List of dependent variables *)
depVarListInput = {u,v};
(* List of independent space-like variables *)
indepSpaceVarListInput = {x};
(* List of independent time-like variables
(currently only works with one time-like variable) *)
indepTimeVarListInput = {t};
(* The function L and M will be applied to *)
laxOpFunInput=\[Psi];
(* d_jwlBousDeift0306.m *)
(* end of file *)
```

```
(* ::Package:: *)

(* jwl_le.m *)
(* Menu item 1-8 *)

(* Last Updated:  26 January, 2011, 17:00 by jwl at csm *)


(***  Fully General Fifth-order KdV Equation  ***)

titleInput = "Fully General Fifth-order KdV Equation";

(*
The lhs of all equations in the pde system (of form lhs = 0)
in list form (separated by commas.)
*)
eqInput={D[u[x,t],t]+\[Alpha]*u[x,t]^2*D[u[x,t],x]
    +\[Beta]*D[u[x,t],x]*D[u[x,t],{x,2}]
    +\[Gamma]*u[x,t]*D[u[x,t],{x,3}]+D[u[x,t],{x,5}]};

(* The X Matrix *)
xMatInput= Null;
(* The T Matrix *)
tMatInput=Null;

(* The L Operator *)
laxLOpInput[x_,t_]:=(c[1]*D[#,{x,3}]+c[2]*u[x,t]*D[#,x]
    + c[3]*D[u[x,t],x]*# + c[4]*D[#,{x,2}]+c[5]*u[x,t]*#)&;
(* The M Operator *)
laxMOpInput[x_,t_]:=(c[6]*D[#,{x,5}]+c[7]*u[x,t]*D[#,{x,3}]
    + c[8]*D[u[x,t],x]*D[#,{x,2}]
    + (c[9]*u[x,t]^2+c[10]*D[u[x,t],{x,2}])D[#,x]
    + (c[11]*u[x,t]*D[u[x,t],x]+c[12]*D[u[x,t],{x,3}])*#)&;
```

```
(* List of dependent variables *)
depVarListInput = {u};
(* List of independent space-like variables *)
indepSpaceVarListInput = {x};
(*
List of independent time-like variables
(currently only works with one time-like variable)
*)
indepTimeVarListInput = {t};

(* The function L and M will be applied to *)
laxOpFunInput=\[Psi];

(* parameters in the equation (assumed to not equal zero) *)
parametersInput = {\[Alpha],\[Beta],\[Gamma]};

(* unknown constants in Lax Pair to be solved for *)
constantsInput = Table[c[myi],{myi,12}];



(* jwl_le.m *)
(* end of file *)
```

APPENDIX B - CODE

The entirety of the *Mathematica* package **laxpairtester.m** [12] is given in the following pages. Softcopy forms of **laxpairtester.m** and data files are included on the CD and on the website found in [12]. The choices we made for formatting this code were made with an eye toward human readability. This was validated by a review of this section by a professional programmer.

```
(* ::Package:: *)

BeginPackage["laxPackage`"]
laxPackage::usage =
        "laxPackage uploads all needed files for laxWork
                to run properly."

    Get["m_jwlFileNameAssociations.m"];
    Get[fna["jwl_lax.m"]];
    Get[fna["jwl_xttst.m"]];
    Get[fna["jwl_lmtst.m"]];
    Get[fna["jwl_CheckForZero.m"]];
    Get[fna["jwl_Abend.m"]];
    Get[fna["jwl_genrulefinder.m"]];
    Get[fna["jwl_uuconvert.m"]];
    Get[fna["jwl_xxconvert.m"]];
    Get[fna["jwl_prettyprint.m"]];
    Get[fna["jwl_highLDerFinder.m"]];
    Get[fna["jwl_highDerFinder.m"]];
    Get[fna["jwl_printpde.m"]];
    Get[fna["jwl_ConvertInput.m"]];

    Dop[op_,var_] := (D[op[#],var]-op[D[#,var]])&

    Unprotect[Integrate];
```

```
    Plus[Integrate[f_,x__],Integrate[g_,x__]]^:=Integrate[f+g,x]/;magic
    Protect[Integrate];

EndPackage[ ]
(* Set printLS to True if you wish to see the load statements for all
   loaded files *)
printLS["laxPackage.m of Oct 7, 2010 Sucessfully Loaded."];
(* ### ### ###       End Package: laxPackage       ### ### ### *)

(* ::Package:: *)

(* ### ### ### ###                Function: testLM                ### ### ### ### *)
(*****************************************************************************)
(* textLM[lOp_, mOp_, expr_, opts___?OptionQ]                                    *)
(* Purpose: To test given L and M operators                                      *)
(* Input:   A L operator                                                         *)
(*          A M operator                                                         *)
(*          The original list of differential funtions (the PDE)                 *)
(*          (Optional) Print Flags                                               *)
(* Output:  The L and M Lax pair and the orignal equation                        *)
(*          The answer if the Lax pair is valid                                  *)
(* Created: 26 January 2010, 22:45 by jwl at home                                *)
(* Code is in File:  jwllmtst.m                                                  *)
(* Last Modified:  01 Feb 2011, 23:00 by jwl at home                             *)
(*****************************************************************************)
Off[General::spell1];
testLM[lIn_,mIn_,pdeIn_,givenvar1_,depVarList_,
        indepSpaceVarList_,indepTimeVarList_,
        inputCompFlag___] :=
   Module[{myeqlist = {}},

If[debugLMT,printLMT=Print,Clear[printLMT],Clear[printLMT]];
printLMT["------------------------------------------------------"];
```

```
printLMT["Start of testLM"];

(* Checks if the given L and M are Null *)
If[DownValues[lIn][[1,2]]===Null||DownValues[mIn][[1,2]]===Null,
                    abend[{"The given L and/or M are Null",
                    "Please retry with valid L and M"}]];

(* Tells what the highest derivative of x is in L *)
highDerL = highLDer[indepSpaceVarList[[1]],lIn];

lmExist = True;
xtExist = False;
convertIn[depVarList,pdeIn,indepSpaceVarList,indepTimeVarList,
                     lIn,mIn,Null,Null,givenvar1,lmExist,xtExist];

pde[i_]:=pdeList[[i]];

givenComp = printPDE[pdeList,numPDE,inputCompFlag];
Print[" "];

(* Print L in a readable form *)
Print["The ", givenComp, " L operator is"];
Print[prettyPrint[Apply[lOp,joinedVarList],False,False]];
Print[" "];

(* Print M in a readable form *)
Print["The ", givenComp, " M operator is"];
Print[prettyPrint[Apply[mOp,joinedVarList],False,False]];
Print[" "];

(* Finds set of general substitution rules for u,v,etc. *)
ruleuMultixt = genRuleFinder[Most[convertedJoinedVarList],
{Last[convertedJoinedVarList]},pdeList,convertedDepVarList];
```

```
printLMT["LMT : ruleuMultixt = "];
printLMT[ruleuMultixt];

Print["L operator applied to "<>
 ToString[prettyPrint[givenvar, False, False]]<> " is"];
Print[prettyPrint[Apply[lOp, joinedVarList][givenvar],
      False, False]];
Print[" "];

Print["M operator applied to "<>
 ToString[prettyPrint[givenvar, False, False]]<> " is"];
Print[prettyPrint[Apply[mOp, joinedVarList][givenvar],
      False, False]];
Print[" "];

(* Deriving \[Psi]_t and setting a rule for it *)
rulepsit={D[givenvar,{t,1}]->Apply[mOp,
   convertedJoinedVarList][givenvar]};
printLMT["LMT : Dt"<>ToString[givenvar]<> " rule is"];
printLMT[rulepsit];

(* Deriving \[Psi]_ (mx) (where m is the max derivative of x
 in L) and setting a rule for it *)
rulepsimx=
      Solve[Apply[lOp, convertedJoinedVarList][givenvar]
           ==\[Lambda]*givenvar,
           D[givenvar,{xx[1], highDerL}]][[1]];
printLMT[ToString[givenvar]<> "_("
         <>ToString[highDerL]<>"x) rule is"];
printLMT[rulepsimx];

(* Rule that takes \[Psi]_ (mx) and replaces each instance of
 derivatives mx or higher with (\[Psi]_ (mx))_ (nx) where n is
```

```
 number of derivatives - m. *)

rulepsimultix={Derivative[(q_/;q>=highDerL),0][\[Psi]][xx[1],t]->
 (D[D[\[Psi]][xx[1],t],{xx[1],highDerL}]/.rulepsimx,
  {xx[1],q-highDerL}])};
printLMT["LMT : rulepsimultix = "];
printLMT[rulepsimultix];

(* Deriving L_t using the Operator Derivative Rule *)
dtLOp[patJoinedVarList] := (Dop[Apply[lOp,joinedVarList],
  joinedVarList[[-1]]][#]&);
printLMT["LMT : DtL is"];
printLMT[dtLOp[convertedJoinedVarList]];
printLMT["LMT : DtL applied to "<> ToString[givenvar]<> " is"];
printLMT[dtLOp[convertedJoinedVarList][givenvar]];

(* Deriving LM *)
lm[patJoinedVarList]:=(Factor[Apply[lOp,joinedVarList][
   Apply[mOp,joinedVarList][#]]]&);
printLMT["LMT : L applied to M is"];
printLMT[lm[convertedJoinedVarList]];
printLMT["LMT : L applied to M applied to "
   <> ToString[givenvar]<> " is"];
printLMT[lm[convertedJoinedVarList][givenvar]];

(* Deriving ML *)
ml[patJoinedVarList]:=(Factor[Apply[mOp,joinedVarList][
   Apply[lOp,joinedVarList][#]]]&);
printLMT["LMT : M applied to L is"];
printLMT[ml[convertedJoinedVarList]];
printLMT["LMT : M applied to L applied to "
   <> ToString[givenvar]<> " is"];
printLMT[ml[convertedJoinedVarList][givenvar]];
```

```
(* Lax Equation before any replacement *)
laxOp[patJoinedVarList]:=((dtLOp[joinedVarList][#]
   + lm[joinedVarList][#]-ml[joinedVarList][#])&);
printLMT["LMT : The lax operator test before being applied to "];
printLMT[ToString[givenvar]<> " is:"];
printLMT[laxOp[convertedJoinedVarList]];
eq1=Factor[laxOp[convertedJoinedVarList][givenvar]];
Print["Lax operator test applied to "
   <> ToString[prettyPrint[givenvar,False,False]]<>
   " before any assumptions is"];
Print[prettyPrint[eq1,False,False]];
(* Checking that Lax Equation does NOT equal zero
    before subsitution *)
If[jZeroQ[eq1,False],abend[{"In testLM, eq1=0",
           "The operater pair has no dependence on the pde",
           "The program will now abort"}]];

(* Then replace \[Psi]_t *)
(* Lax Equation after replacement of \[Psi]_t
   from M\[Psi]=Subscript[\[Psi], t] *)
eq2=Factor[eq1/.rulepsit];
printLMT["LMT : Lax operator test applied to "
   <> ToString[givenvar]
   <> " after replacing \!\(\*SubscriptBox[\"\[Psi]\", \"t\"]\) is"];
printLMT[eq2];
printLMT["LMT : The above (eq2) should not have any D_t in it."];
If[jZeroQ[eq2,False],abend[{"In testLM, eq2=0",
           "The operater pair has no dependence on the pde",
           "The program will now abort"}]];

(* Then replace all higher order \[Psi]_ (xx) *)
(* Lax Equation after replacement of \[Psi]_ {a (xx)}
```

```
    from L\[Psi]=\[Lambda]\[Psi] *)
eq3=Factor[eq2//.rulepsimultix];
printLMT["LMT : Lax operator test applied to "<> ToString[givenvar]<>
    " after replacing \!\(\*SubscriptBox[\"\[Psi]\",\"xx\"]\) is"];
printLMT[eq3];
printLMT["LMT : The above (eq3) should have no D_t ,",
      " high orders of D_x in it ."];
If[jZeroQ[eq3,False],abend[{"In testLM, eq3=0",
            "The operater pair has no dependence on the pde",
            "The program will now abort"}]];

(* Lax Equation after final replacement with equation *)
eq4=Factor[eq3//.ruleuMultixt];
printLMT["LMT : Lax operator test applied to "
    <> ToString[givenvar]
    <> " after all possible subsitutions ."];
printLMT[eq4];
printLMT["LMT : The above (eq4) should have no D_t ,
high orders of D_x, and no u_t in it ."];

(* Checking for integrals, will take the derivative of the
 eq4 if there are any integrals found. Will then test that
 for zero and print a warning to the user to check the
 answer if it computes to zero. *)
If[jZeroQ[eq4,False],eq5=eq4,
intTotalList[fun_]:=Join[
    Cases[fun,___*Integrate[a_,b__]->{b},Infinity],
    Cases[fun,Integrate[a_,b__]->{b},Infinity]];
listCount1[ll_,v_]:=Max[Map[Count[#,v]&,ll]];
listCount2[ll_,vv_]:=MapThread[{#,listCount1[ll,#]}&,{vv}];
conIndepSpVarList = findxRules[indepSpaceVarList,
    numSpVar,indepSpaceVarList,True];
intDerRules = listCount2[intTotalList[eq4],conIndepSpVarList];
```

```
printLMT["LMT : Converted spatial variable list."];
printLMT[conIndepSpVarList];
printLMT["LMT : List of types of integrals in eq4."];
printLMT[intTotalList[eq4]];
printLMT["LMT : Max Integral rules."];
printLMT[intDerRules];

If[intTotalList[eq4]!= List[],
    eq5=Apply[D[eq4/givenvar,#]&,intDerRules];intFlag=True,
    eq5=eq4;intFlag=False,
  Print["Program Failed: Please print debug statements to find problem"]
  ];
];
printLMT["LMT : Test after everything. Should be zero."];
printLMT[eq5];

(* Testing if the final answer is zero *)
(* If there are integrals in the final answer, we first differentiate
   the final answer, then we check if it is zero.
   We also include a warning that the answer may not be true
   because there were integrals *)
If[jZeroQ[eq5,False],
    Print["The ", givenComp, " L and M operators are a valid Lax pair"];
    If[intFlag,Print["-----------------------------------"];
          Print[Style["Disclaimer",Red],
           ": The result may need hand evaluation. ",
           "Integrals were found in the result and were ",
           "differentiated out. ",
           "The result before differentiation was:"];
          Print[eq4]],
    Print["The ", givenComp, " L and M operators are not ",
     "a valid Lax pair"];
```

```
    Print["and the following is the result of the Lax equation"];
    Print[prettyPrint[eq4, False, False]],
    Print["Program Failed:",
      " Please print debug statements to find problem"]];

printLMT["End of testLM"];
printLMT["------------------------------------------------"];
]
printLS["jwllmtst.m of Feb 01, 2011 Sucessfully Loaded."];
(* ## ## ##       End Function: testLM       ## ## ## *)

(* ::Package:: *)

(* ## ## ## ##                Function: testXT                ## ## ## ## *)
(*********************************************************************)
(* TextXT[xMat1_, tMat1_, pdeIn_, givenvar1_, depVarList_,           *)
(*        indepSpaceVarList_,                                        *)
(*        indepTimeVarList_, inputCompFlag___]                       *)
(* Purpose: To test given X and T matrices                           *)
(* Input:   A X matrix                                               *)
(*          A T matrix                                               *)
(*          The original list of differential funtions (the PDE)     *)
(*          The given variable list {\[Psi]}                         *)
(*          The list of dependent variables {u, v, etc}              *)
(*          The list of independent space variables {x,y, etc}       *)
(*          The list of independent time variable                    *)
(*               can only have one input, normally {t}               *)
(*          (Optional) Computed Flag: True if the pair was           *)
(*             computed within the program, False if it was given    *)
(* Output:  The X and T lax pair and the orignal equation            *)
(*          The answer Yes or No for if the lax pair is valid        *)
(* Created: 23 January 2010, 20:45 by jwl at home                    *)
(* Code is in File:  m_jwlxttst.m                                    *)
```

```
(* Last Modified:  7 March 2011, 12:10 by jwl at home                    *)
(*************************************************************************)
Off[General::spell1];
testXT[xMat1_, tMat1_, pdeIn_, givenvar1_, depVarList_, indepSpaceVarList_,
        indepTimeVarList_, inputCompFlag___] :=
Module[{dtXmat, dxTmat, xtMat, txMat, laxMat, zeroMat},

If[debugXTT, printXTT=Print, Clear[printXTT], Clear[printXTT]];
printXTT["-------------------------------------------------------"];
printXTT["Start of testXT"];

(* Checks if the given X and T are Null or not *)
If[xMat1==Null||tMat1==Null, abend[{"The given X and/or T are Null",
                    "Please retry with valid X and T"}]];

lmExist = False;
xtExist = True;
convertIn[depVarList, pdeIn, indepSpaceVarList, indepTimeVarList,
                    Null, Null, xMat1, tMat1, givenvar1, lmExist, xtExist];
givenComp = printPDE[pdeList, numPDE, inputCompFlag];

Print["The ", givenComp, " X matrix is"];
Print[prettyPrint[xMat, False, False]//MatrixForm];
Print[" "];

Print["The ", givenComp, " T matrix is"];
Print[prettyPrint[tMat, False, False]//MatrixForm];
Print[" "];

(* Finds set of general subsitution rules for u,v,etc. *)
ruleuMultixt = genRuleFinder[Most[convertedJoinedVarList],
{Last[convertedJoinedVarList]}, pdeList, convertedDepVarList];
printXTT["XTT : ruleuMultixt = "];
```

```
printXTT[ruleuMultixt];

dtXmat=D[xMat,t];
printXTT["XTT : dtXmat="];
printXTT[dtXmat//MatrixForm];

dxTmat=D[tMat,prettyPrint[x,False,True]];
printXTT["XTT : dxTmat="];
printXTT[dxTmat//MatrixForm];

xtMat = xMat.tMat;
printXTT["XTT : xtMat="];
printXTT[xtMat//MatrixForm];

txMat = tMat.xMat;
printXTT["XTT : txMat="];
printXTT[txMat//MatrixForm];

laxMat=dtXmat-dxTmat+xtMat-txMat;
printXTT["XTT : Before expansion , the Lax Equation matrix is"];
printXTT[laxMat//MatrixForm];

printXTT["XTT : After expansion , the Lax Equation matrix is"];
printXTT[Expand[laxMat]//MatrixForm];
printXTT["XTT : There should only be a multiple (or some form of) "<>
        "the original pde left in one corner of the above matrix."];

zeroMat=Simplify[laxMat/. ruleuMultixt];
printXTT["XTT : After subsitution of the original pde, "<>
        "the below matrix should be the zero matrix"];
printXTT[zeroMat//MatrixForm];

If[jZeroQ[Expand[zeroMat],True],
```

```
    Print["The ", givenComp, " X and T matrices are a valid Lax pair"],
    Print["The ", givenComp, " X and T matrices are not a valid
            Lax pair"];
    Print["and the following is the resulting matrix"];
    Print[zeroMat//MatrixForm],
    Print["Program Failed:
          Please print debug statements to find problem"]];

printXTT["End of testXT"];
printXTT["------------------------------------------------------------"];
];
printLS["jwlxttst.m of Mar 07, 2011 Sucessfully Loaded."];
(* ## ## ##       End Function: testXT       ## ## ## *)

(* ::Package:: *)

(* ## ## ## ##                Function: convertIn            ## ## ## ## *)
(*********************************************************************)
(* convertIn[depVarList_, pdeIn_, indepSpaceVarList_,                  *)
(*             indepTimeVarList_, lIn_, mIn_, xIn_, tIn_,              *)
(*             givenvar1_, lmExist_, xtExist_]                         *)
(* Purpose: Converts user input to internal variables                  *)
(* Input:   The list of dependent variables                            *)
(*          The original PDE (or system)                               *)
(*          The list of independent space variables                    *)
(*          The list of independent time variables (currently only t)  *)
(*          The operator L (or Null if it does not exist)              *)
(*          The operator M (or Null if it does not exist)              *)
(*          The matrix X (or Null if it does not exist)                *)
(*          The matrix T (or Null if it does not exist)                *)
(*          The given variable (normally \[Psi])                       *)
(*          A flag to tell if L and M exist (True if they do)          *)
(*          A flag to tell if X and T exist (True if they do)          *)
```

```
(* Output:  All inputs converted to internal variables                *)
(* Created: 22 June 2010, 12:10 by jwl at home                        *)
(* Code is in File:  jwlConvertInput.m                                *)
(* Last Modified:  26 Sept 2010, 00:25 by jwl at home                 *)
(*********************************************************************)
Off[General::spell1];
convertIn[depVarList_,pdeIn_,indepSpaceVarList_,indepTimeVarList_,
            lIn_,mIn_,xIn_,tIn_,givenvar1_,lmExist_,xtExist_] :=
   Module[{},

If[debugCI,printCI=Print,Clear[printCI],Clear[printCI]];

printCI["-------------------------------------------------"];
printCI["Start of convertIn"];

numVar = Length[depVarList];
numPDE = Length[pdeIn];
numSpVar = Length[indepSpaceVarList];

printCI["CI : number of dependent variables : " <>ToString[numVar]];
printCI["CI : number of equations in system : " <>ToString[numPDE]];
printCI["CI : number of independent spacial variables : "
            <>ToString[numSpVar]];

pdeList1 = finduRules[pdeIn,numVar,depVarList,True];
convertedDepVarList = finduRules[depVarList,numVar,depVarList,True];
If[lmExist,
lOp = prettyPrint[lIn,True,True];
mOp = prettyPrint[mIn,True,True];
];
If[xtExist,
xMat = prettyPrint[xIn,False,True];
tMat = prettyPrint[tIn,False,True];
```

```
];

pdeList = findxRules[pdeList1,numSpVar,indepSpaceVarList,True];
givenvar = findxRules[givenvar1,numSpVar,indepSpaceVarList,True];
joinedVarList = Join[indepSpaceVarList,indepTimeVarList];
convertedJoinedVarList=findxRules[joinedVarList,numSpVar,
                                  indepSpaceVarList,True];
patJoinedVarList = Map[\!\(\*
TagBox[
StyleBox[
RowBox[{"Pattern", "[",
RowBox[{"#", ",",
RowBox[{"Blank", "[", "]"}]}], "]"}],
ShowSpecialCharacters->False,
ShowStringCharacters->True,
NumberMarks->True],
FullForm]\)&,joinedVarList];

givenvar = Apply[givenvar1,convertedJoinedVarList];

printCI["CI : pdeList1 : " <>ToString[pdeList1,StandardForm]];
printCI["CI : L Operator after subsitution : "];
printCI[ToString[Apply[lOp,convertedJoinedVarList],StandardForm]];
printCI["CI : M Operator after subsitution : "];
printCI[ToString[Apply[mOp,convertedJoinedVarList],StandardForm]];
printCI["CI : pdeList : " <>ToString[pdeList,StandardForm]];
printCI["CI : pre-converted given var : " <>ToString[givenvar1]];
(*
printCI["CI : given variable : " <>ToString[givenvar,StandardForm]];
*)
printCI["CI : Converted Independent Variable List : "];
printCI[ToString[convertedJoinedVarList,StandardForm]];
printCI["CI : Converted Dependent Variable List : "];
```

```
printCI[ToString[convertedDepVarList,StandardForm]];

printCI["End of convertIn"];
printCI["-------------------------------------------------"];

Return[{lOp,mOp,pdeList,givenvar,convertedJoinedVarList,
        convertedDepVarList,xMat,tMat}];
]
printLS["jwlConvertInput.m of Sept 26, 2010 Sucessfully Loaded."];
(* ### ### ###       End Function: convertIn       ### ### ### *)

(* ::Package:: *)

(* ### ### ### ###              Function: genRuleFinder          ### ### ### ### *)
(*********************************************************************)
(* genRuleFinder[indepSpaceVarList_,indepTimeVarList_,                *)
(*                 eqListIn_,depVarList_]                             *)
(* Purpose: To compute the general u_t, v_t, etc rule                 *)
(*               from the pde (or sysmte)                             *)
(* Input:   The list of independent space variables                   *)
(*          The list of independent time variables                    *)
(*          The original list of differential funtions (the PDE)      *)
(*          The list of dependent variables                           *)
(* Output:  A list of generalized rules for u_t, v_t, etc             *)
(* Created: 08 August 2010, 11:45 by jwl at home                      *)
(* Code is in File:  jwlgenrulefinder.m                               *)
(* Last Modified:  18 March 2011, 02:00 by jwl at home                *)
(*********************************************************************)
Off[General::spell1];
genRuleFinder[indepSpaceVarList_,indepTimeVarList_,eqListIn_,
              depVarList_,inputCompFlag___] :=
   Module[{myeqlist = {}},
```

```
If[debugGRF, printGRF=Print, Clear[printGRF], Clear[printGRF]];
printGRF["------------------------------------------------------------"];
printGRF["Start of genRuleFinder"];

prefixSymbol[n_, x_]:=Symbol[n<>SymbolName[x]];
makeList[var_, dervar_]:={var, dervar};
patternMaker[var_, yes_]:= D[uu[i_][var], yes];

(* determining the number of variables and the number of equations
  in the system *)
numVar=Length[depVarList];
printGRF["numVar : " <>ToString[numVar]];
numPDE=Length[eqListIn];
printGRF["numPDE : "<>ToString[numPDE, StandardForm]];
numSpVar = Length[indepSpaceVarList];
printGRF["numSpVar : "<>ToString[numSpVar, StandardForm]];
joinedVarList = Join[indepSpaceVarList, indepTimeVarList];
printGRF["joinedVarList : "<>ToString[joinedVarList, StandardForm]];
nonConvertedVarList = prettyPrint[joinedVarList, False, False];
printGRF["nonConvertedVarList : "<>
            ToString[nonConvertedVarList, StandardForm]];
symbolVarList=Map[prefixSymbol["n",#]&, nonConvertedVarList];
printGRF["symbolVarList : "<>ToString[symbolVarList, StandardForm]];
symbolVarBlkList=Map[\!\(\*
TagBox[
StyleBox[
RowBox[{"Pattern", "[",
RowBox[{"#", ",",
RowBox[{"Blank", "[", "]"}]}]}], "]"}],
ShowSpecialCharacters->False,
ShowStringCharacters->True,
NumberMarks->True],
FullForm]\)&, symbolVarList];
```

```
printGRF["symbolVarBlkList : "<>ToString[symbolVarBlkList,StandardForm]];

eqList = eqListIn;

printGRF["At GRF In: List of dependent variables: "
          <> ToString[depVarList,StandardForm]];
printGRF["At GRF In: List of independent space variables: "
          <> ToString[indepSpaceVarList,StandardForm]];
printGRF["At GRF In: List of independent time variables: "
          <> ToString[indepTimeVarList,StandardForm]];
printGRF["At GRF In: List of equations: "
          <> ToString[eqListIn,StandardForm]];

pattern1=Thread[makeList[joinedVarList,symbolVarBlkList]];
printGRF["pattern1 : "<>ToString[pattern1,StandardForm]];
newPattern = Prepend[pattern1,Apply[uu[ni_],joinedVarList]];
printGRF["newPattern : "<>ToString[newPattern,StandardForm]];
pattern = Apply[D,newPattern];
printGRF["The computed pattern for derivatives is: "
<> ToString[pattern,StandardForm]];
Clear[i];
foundList = Prepend[symbolVarList,ni];
printGRF["foundList : "<>ToString[foundList,StandardForm]];
symbolFoundList=Map[\!\(\*
TagBox[
StyleBox[
RowBox[{"Pattern", "[",
RowBox[{"#", ",",
RowBox[{"Blank", "[", "]"}]}], "]"}],
ShowSpecialCharacters->False,
ShowStringCharacters->True,
NumberMarks->True],
FullForm]\)&,foundList];
```

```
printGRF["symbolFoundList : "<>ToString[symbolFoundList,StandardForm]];
pattern2[symbolFoundList]=Thread[makeList[joinedVarList,symbolVarList]];
printGRF["pattern2 : "<>ToString[pattern2[symbolVarList],StandardForm]];

findD[pattern] := Evaluate[foundList];
SetAttributes[findD,Listable];
findDD[b_]:=Block[{a=b},
                  printGRF["Block input is:"];
                  printGRF[ToString[b,StandardForm]];
                  findD[Extract[{a},Position[{a},pattern]]]
                 ];
tableOfAllDers=findDD[eqList];
printGRF["The list of{i,xder,tder} for D[u[i],{x,xder},{t,tder}]
for all u's in the system : "<>ToString[tableOfAllDers,StandardForm]];

(* Making table[i] *)
For[j=1,j<=numVar,j++,table[j]={};
    For[i=1,i<=Length[tableOfAllDers],i++,
        If[
            tableOfAllDers[[i,1]]==j,
            AppendTo[table[j],tableOfAllDers[[i]]]
           ]
       ];
    printGRF["List of uu[" <> ToString[j] <> "] derivatives: "
<> ToString[table[j]]];
];

(*
Takes {i,x,y,t} and turns it into a sortable list of form
{t,x+y,x,y}. This allows it to be sorted by highest t der,
highest sum of x,y,... der, then highest ders from x down.
*)
newList[list_]:=Join[{list[[-1]]},
```

```
    {Apply[Plus,Drop[Rest[list],-1]]},Drop[Rest[list],-1]];
(*
Takes {t,x+y,x,y} and i and turns it back into a list of form
{i,x,y,t}.
*)
revertingNewList[list_,i_]:=Join[{i},Drop[list,2],{list[[1]]}];

tableHighDer={};
For[kk=1,kk<=numVar,kk++,
      myTempList = Map[newList,table[kk]];
      AppendTo[tableHighDer, revertingNewList[Sort[myTempList][[-1]],kk]]
    ];

printGRF["List of highest derivatives: " <> ToString[tableHighDer]];

rhsEq=ConstantArray[0,numPDE];
printGRF["Computed right hand side of system (list of zeros should be
as many as equations in the system) : " <> ToString[rhsEq]];

(*
   highDer contains {ii,xx} where ii denotes the uu[ii] and xx
   denotes the highest derivative of u[ii] in x
*)
findSolveVar[highDer_,list1_]:=
       Apply[D,Prepend[pattern2[list1],
         Apply[uu[list1[[1]]],highDer]
         ]
       ];
solveVar=Map[findSolveVar[joinedVarList,#]&,tableHighDer];
printGRF["Variables (and derivatives) which need to be solved for :"];
printGRF[ToString[solveVar,StandardForm]];

solvedRules=Flatten[Solve[eqList==rhsEq,solveVar]];
```

```
printGRF["Specific rules for solved variables : "
          <> ToString[solvedRules, StandardForm]];

Print["Non-generalized list of rules :"];
Print[ToString[
      prettyPrint[solvedRules, False, False]//TableForm, StandardForm]];

derPatFun[varbk_, var_, num_]:=HoldPattern[(varbk/;var>=num)];
solvedRulesGeneral[derPat_, symList_, nonSymList_, listInput_,
                        varList_, solvedRuleList_]:=Apply[
   Apply[Derivative, ReleaseHold[
   MapThread[derPat, {Rest[symList], Rest[nonSymList],
              Rest[listInput]}]]][uu[listInput[[1]]], varList]->
   Apply[D, Prepend[MapThread[{#1,#2-#3}&,{varList, Rest[nonSymList],
            Rest[listInput]}],
   (Apply[D, Prepend[MapThread[{#1,#2}&,{joinedVarList, Rest[listInput]}],
   Apply[uu[listInput[[1]]], varList]]]/.solvedRules)]];
rulesGen = MapThread[solvedRulesGeneral[derPatFun, symbolFoundList,
            foundList, #1, joinedVarList, #2]&, {tableHighDer, solvedRules}];
rulesGenOut = prettyPrint[rulesGen, False, False];
printGRF["List of general rules (before subsitution of variables) : "
          <> ToString[rulesGen, StandardForm]];
printGRF["List of general rules (after subsitution of variables) : "
          <> ToString[rulesGenOut, StandardForm]];

rulesGen=Join[rulesGen, Map[Integrate[#, xx[1]]&, solvedRules, {2}]];

printGRF["new rulesGen"];
printGRF[rulesGen];

printGRF["End of genRuleFinder"];
printGRF["-------------------------------------------------------"];
```

```
Return[rulesGen];
]
printLS["jwlgenrulefinder.m of Aug 22, 2010 Sucessfully Loaded."];
(* ## ## ##       End Function: genRuleFinder       ## ## ## *)

(* ::Package:: *)

(* ## ## ## ##          Function: highDerFinder          ## ## ## ## *)
(*********************************************************************)
(* highDerFinder[indepVarIn_, depVarIn_, joinedVarList_, funIn_]      *)
(* Purpose: To find the highest derivative taken w.r.t. to            *)
(*             a certain independent variable                         *)
(* Input:   An indepdent variable                                     *)
(*          A list of all dependent variables                         *)
(*          A list of all independent variables (spacial and time)    *)
(*          The expression we wish to search                          *)
(* Output:  The value of the highest derivative searched for          *)
(* Created: 08 August 2010, 11:45 by jwl at home                      *)
(* Code is in File:  m_jwlhighDerFinder.m                             *)
(* Last Modified:  18 March 2011, 02:00 by jwl at home                *)
(*********************************************************************)
Off[General::spell1];
highDerFinder[indepVarIn_, depVarIn_, joinedVarList_, funIn_] :=
   Module[{printHDF, depVar, xxvar, patx, findxD, findxDD, derList, highDer},

If[debugHDF, printHDF=Print, Clear[printHDF], Clear[printHDF]];
printHDF["-------------------------------------------------------"];
printHDF["Start of highDerFinder"];

printHDF["HDF: varIn = " <> ToString[indepVarIn]];
depVar = Apply[depVarIn, joinedVarList];
printHDF["HDF: Variable to be found ="];
printHDF[ToString[depVar, StandardForm]];
```

```
printHDF["HDF : Function to be tested ="];
printHDF[ToString[funIn,StandardForm]];

xxvar=indepVarIn;
patx=HoldPattern[D[Hold[depVar],{Hold[xxvar],myx_}]];
printHDF["HDF : patx Held ="];
printHDF[ToString[patx,StandardForm]];

patx=(patx/.{Hold[xxvar]->xxvar,Hold[depVar]->depVar});
printHDF["HDF : patx ="];
printHDF[ToString[patx,StandardForm]];

findxD[ReleaseHold[patx]]:={myx};
Attributes[findxD]={Listable};
findxDD[b_]:=findxD[Extract[{b},Position[{b},ReleaseHold[patx]],
                    Unevaluated]];
derList = findxDD[Evaluate[funIn]];
printHDF["HDF : derList ="];
printHDF[ToString[derList,StandardForm]];

highDer = Max[derList];
printHDF["HDF : highDer ="];
printHDF[ToString[highDer,StandardForm]];

printHDF["highDer ="<>ToString[highDer]];

printHDF["End of highDerFinder"];
printHDF["-------------------------------------------------------"];

Return[highDer];
]
printLS["jwlhighDerFinder.m of Mar 18, 2011 Sucessfully Loaded."];
(* ### ### ###      End Function: highDerFinder      ### ### ### *)
```

```
(* ::Package:: *)

(* ## ## ## ##              Function: highLDer                ## ## ## ## *)
(*********************************************************************)
(* highLDer[indepVarIn_,pureFunIn_]                                    *)
(* Purpose: To find the highest derivative in a pure function          *)
(*              with respect to a given variable                       *)
(* Input:   An independent variable, such as x                         *)
(*          A pure function, such as L                                 *)
(* Output:  A value which is the highest derivative with respect       *)
(*              to x found in L                                        *)
(* Created: 08 August 2010, 11:45 by jwl at home                       *)
(* Code is in File:  jwllmtst.m                                        *)
(* Last Modified:  01 Feb 2011, 21:45 by jwl at home                   *)
(*********************************************************************)
Off[General::spell1];
highLDer[indepVarIn_,pureFunIn_] :=
   Module[{},

If[debugHDL,printHDL=Print,Clear[printHDL],Clear[printHDL]];
printHDL["————————————————————————————————————————————————————"];
printHDL["Start of highLDer"];

printHDL["HDL: varIn =" <> ToString[indepVarIn]];
printHDL["HDL: Function to be tested ="];
printHDL[Definition[pureFunIn]];
(*printHDL[Information[pureFunIn]];*)
(*printHDL[ToString[Evaluate[pureFunIn,StandardForm]]];
printHDL[ToString[pureFunIn[x,t][\[Psi][x,t]],TraditionalForm]];*)

xxvar=indepVarIn;
patx=HoldPattern[D[#1,{Hold[xxvar],myx_}]];
patx=patx/.Hold[xxvar]->xxvar;
```

```
Clear[findxD];
findxD[patx]:={myx};
Attributes[findxD]={Listable};
findxDD[b_]:=findxD[Extract[{b},Position[{b},patx],Unevaluated]];

derList = findxDD[DownValues[Evaluate[pureFunIn]]];
printHDL["HDL : List of derivatives : derList ="];
printHDL[derList];

highDer = Max[derList];
printHDL["HDL : highest derivative : highDer ="];
printHDL[highDer];

printHDL["End of highLDer"];
printHDL["-------------------------------------------------"];

Return[highDer];
]
printLS["jwlhighLDerFinder.m of Feb 1, 2011 Sucessfully Loaded."];
(* ### ### ###       End Function: highLDer       ### ### ### *)

(* ::Package:: *)

(* ### ### ### ### ###            Begin jwllax.m            ### ### ### ### ### *)
(* ### ### ### ### ### ### ### ### ### ### ###  ### ### ### ### ### ### ### ### ### ### ### *)
(* ### ### ### ### ###            Function: newMenu         ### ### ### ### ### *)
(*************************************************************************)
(* newMenu (no arguments)                                                *)
(* Purpose: A menu to choose between what type of lax pair work done     *)
(* Author: Jennifer Larue                                                *)
(* Input:   When the program is first run, this menu is printed for      *)
(*          the user to choose what type of test or conversion they      *)
(*          wish to do.                                                  *)
```

```
(* Output:  The menu is loaded for whatever choice they made          *)
(* Adapted from:  data/newzeala/reu2004/reu2004-0811/software/        *)
(*                whnemenu.m, Created 1 June 2004 at CSM              *)
(* Code is in File:  jwllax.m                                         *)
(* Last Modified:  22 March, 2010, 10:45 by jwl at csm                *)
(*********************************************************************)

newMenu :=
Module[{choice, makeChoice, makeChoice1, choice2, choice3},

  (* Make sure dependent and independent variables are clear.         *)
  Clear[u, x, y, z, t];

  Print[" "];
  Print["  *** Menu Interface *** "];
  Print["-------------------------------------------------"];
  Print["    1) Test X and T Matrix Lax Pair"];
  Print["    2) Test L and M Operator Lax Pair"];
  Print["  qq) Exit the Program "];
  Print["-------------------------------------------------"];

  choice = Input["ENTER YOUR CHOICE: "];

  makeChoice := Switch[choice,
      1, testMenu,
      2, testMenu,
      3, testMenu,
      qq, Print["All computations are being discontinued."]; Abort[],
      _,  choice = Input["Please enter a choice from the menu."];
          makeChoice
    ]; (* end Switch *)

  Check[makeChoice, Abort[]];
```

```
  (* Clear all local variables not being returned.                    *)
  Clear[makeChoice, choice2, choice3];

(*********************************************************************)
(* ## ## ## ## ##              Start the program.       #### ## ## ## *)
(*********************************************************************)
 (* makeChoice := Switch[choice,
       1, Get[fna["jwlxttst.m"]],
       2, Get[fna["jwllmtst.m"]],
       3, Get[fna["jwllm2xt.m"]]
    ]; (* end Switch *)
  *)

Print["The "<>ToString[titleInput]];
Print["————————————————————"];

  makeChoice := Switch[choice,
       1, testXT[xMatInput,tMatInput,eqInput,
                 laxOpFunInput,depVarListInput,
                 indepSpaceVarListInput,
                 indepTimeVarListInput,compFalse],
       2, testLM[laxLOpInput,laxMOpInput,eqInput,
                 laxOpFunInput,depVarListInput,
                 indepSpaceVarListInput,indepTimeVarListInput,
                 compFalse],
       3, transformLMtoXT[laxLOpInput[x,t],laxMOpInput[x,t],pdeRule,
            laxOpFunInput,\[CurlyPhi][x,t]]
    ]; (* end Switch *)

  Check[makeChoice, Abort[]];
  Clear[choice];
]; (* end Module newMenu *)
```

```
(* ## ## ## ## ##          Function: testMenu          ## ## ## ## ## *)

(*********************************************************************)
(* xtTestMenu (no arguments)                                         *)
(* Purpose: A menu to choose between the 1-D and the multi-D cases   *)
(* Authors: Maxi von Eye, Lindsay Auble, Scott Danford               *)
(* Input:   After the user has chosen to test a given X and T        *)
(*          the user can choose to pick a case example or give       *)
(*          their own.                                               *)
(* Output:  The menu for case examples is loaded.                    *)
(* Adapted from:  data/newzeala/reu2004/reu2004-0811/software/       *)
(*                whnemenu.m, Created 1 June 2004 at CSM             *)
(* Code is in File:  jwllax.m                                        *)
(* Last Modified:  25 February, 2010, 13:31 by jwl at home           *)
(*********************************************************************)

testMenu :=
Module[{choice, makeChoice, choice2, choice3},

  (* Make sure dependent and independent variables are clear. *)
  Clear[u, x, y, z, t];

  Print[" "];
  Print["   *** Menu Interface *** "];
  Print["-------------------------------------------------"];
  Print["    1) Case Examples"];
  Print["   tt) Take Equation or System from a File"];
  Print["   uu) Go to original menu"];
  Print["   qq) Exit the Program "];
  Print["-------------------------------------------------"];
```

```
  choice = Input["ENTER YOUR CHOICE: "];

  makeChoice := Switch[choice,
      1, examplesMenu,
      uu, newMenu,
      tt, Print["Make sure that you have prepared the data file for the"
            <>" system you want to test (similar to the data files we"
            <>" supplied)."];
          choice2 = Input["If your file is ready, press 1, else 2: "];
          If[choice2 === 1,
           choice3 = InputString["Enter the name of your data file: "];
            Get[choice3],
            Print["All computations are being discontinued."];
            Print["Prepare your data file first, then restart the
                  program."];
            Abort[];
          ],
      qq, Print["All computations are being discontinued."]; Abort[],
      _,  choice = Input["Please enter a choice from the menu."];
          makeChoice
    ]; (* end Switch *)

  Check[makeChoice, Abort[]];

  (* Clear all local variables not being returned. *)
  Clear[choice, makeChoice, choice2, choice3];

]; (* end Module testMenu *)


(* ## ## ## ## ##          Function: examplesMenu          ## ## ## ## ## *)

(*****************************************************************)
```

```
(* examplesMenu (no arguments)                                          *)
(* Purpose: This is the menu for known Lax Pair Cases.                  *)
(* Input:   When program is first run, the menu is printed for user     *)
(*          to choose which case they want to run. (by choosing a       *)
(*          number from the list)                                       *)
(* Output:  Loads the chosen case.                                      *)
(* Adapted From:  conservationlaws/dplemenu.m                           *)
(* Code is in File:  m_jwllax.m                                         *)
(* Last Modified:  4 March, 2011, 11:50 by jwl at home                  *)
(************************************************************************)

(************************************************************************)
(* printpage[n, page]: a subroutine for menu                            *)
(************************************************************************)

printpage[n_, page_] := Module[{lenpage, choice},
  lenpage = Length[page[n]];
  Print[" "];
  Print["  *** Menu Interface ***  (page: ", n, ")"];
  Print["-------------------------------------------------"];
  For[i=1, i <= lenpage, i++,
    Print[Part[Part[page[n], i], 1]]];
  Print[" nn) Next Page"];
  Print[" qq) Exit the Program"];
  Print["-------------------------------------------------"];
  choice = Input["Enter Your Choice: "];
  Return[choice]
]; (* end Module printpage *)
(************************************************************************)
(* examplesMenu: creates the menu                                       *)
(************************************************************************)

examplesMenu :=
```

```
Module[{counterpage = 1, menulist, numpages, page, choice1, control,
    lenmenulist, i},

  menulist = {
   {"  1) KdV Equation (XT or LM) (d_jwlkdv.m)"},
   {"  2) Camassa-Holm Equation (XT or LM) (d_jwlch.m)"},
   {"  3) Lax fifth-order KdV Equation (XT or LM) (d_jwlle.m)"},
   {"  4) Sawada-Kotera Equation (XT or LM) (d_jwlsk.m)"},
   {"  5) Kaup-Kupershmidt Equation (XT or LM) (d_jwlkk.m)"},
   {"  6) KdV Equation (XT complex) (d_jwlkdv_complex.m)"},
   {"  7) Modified KdV Equation (XT) (d_jwlmkdv.m)"},
   {"  8) Sine-Gordon Equation (XT) (d_jwlsineG.m)"},
   {"  9) Sinh-Gordon Equation (XT) (d_jwlsinhG.m)"},
   {"  10) Ziber-Shabat-Mikhailov System (XT) (d_jwlzsm.m)"},
   {"  11) Short Pulse Equation (XT) (d_jwlspe.m)"},
   {"  12) Liouville Equation (XT) (d_jwlliou.m)"},
   {"  13) Sawada-Kotera Equation Alternate (LM)
          (d_jwlsk_dye_parker.m)"},
   {"  14) Hirota-Satsuma System (LM) (d_jwlhs.m)"},
   {"  15) Boussinesq System from Deift (LM) (d_jwlbousdeift.m)"},
   {"  16) Boussinesq System from Ivey-Hierarchy (LM) (d_jwlbousv2.m)"},
   {"  17) Burgers Equation (LM) (d_jwlbur.m)"},
   {"  18) Harry Dym Equation (LM) (d_jwlhdym.m)"},
   {"  19) Harry Dym Equation alternate Lax Pair (LM) (d_jwlhdymv2.m)"},
   {"  20) Kadomtsev-Petviashvili Equation (LM) (d_jwlkp.m)"},
   {"  uu) Go up one menu"},
   {"  mm) Go back to first menu"}
  }; (* closes menulist *)

  lenmenulist = Length[menulist];
  numpages = Ceiling[lenmenulist/10];
  For[i = 1, i <= numpages, i++,
    page[i] = If[lenmenulist >= (i*10),
```

```
    menulist[[Table[k, {k, (i-1)*10+1, i*10}]]],
    menulist[[Table[k, {k, (i-1)*10+1, lenmenulist}]]]
    ] (* end If *)
]; (* end For *)

choice1 = printpage[counterpage, page];

control := (
  Switch[choice1,
    nn, If[counterpage < numpages, counterpage++;
        choice1 = printpage[counterpage, page]; control,
        counterpage = 1; choice1 = printpage[1, page]; control],
     uu, testMenu,
     mm, newMenu,
    qq, Print["All computations are being discontinued."]; Abort[],
     1, Get[fna["d_jwlkdv.m"]],
     2, Get[fna["d_jwlch.m"]],
     3, Get[fna["d_jwlle.m"]],
     4, Get[fna["d_jwlsk.m"]],
     5, Get[fna["d_jwlkk.m"]],
     6, Get[fna["d_jwlkdv_complex.m"]],
     7, Get[fna["d_jwlmkdv.m"]],
     8, Get[fna["d_jwlsineG.m"]],
     9, Get[fna["d_jwlsinhG.m"]],
     10, Get[fna["d_jwlzsm.m"]],
     11, Get[fna["d_jwlspe.m"]],
     12, Get[fna["d_jwlliou.m"]],
     13, Get[fna["d_jwlsk_dye_parker.m"]],
     14, Get[fna["d_jwlhs.m"]],
     15, Get[fna["d_jwlbousdeift.m"]],
     16, Get[fna["d_jwlbousv2.m"]],
     17, Get[fna["d_jwlbureq.m"]],
     18, Get[fna["d_jwlhdym.m"]],
```

```
        19, Get[fna["d_jwlhdymv2.m"]],
        20, Get[fna["d_jwlkp.m"]],
       _,   choice1 = Input["Please enter a choice from the menu."];
            control
    ]; (* closes Switch *)
  ); (* end control *)

  control;

  Check[control, Abort[]];

(* Clear all local variables not being returned. *)
  Clear[counterpage, menulist, numpages, page, choice1, control,
      lenmenulist, i]
]; (* end Module examplesMenu *)


(*     The list of Options controls print  and debug statements.      *)
Options[laxPairs] = {
    printLoadStatements -> False, printTestXT -> False,
    printTestLM -> False, printAbend -> False,
    printJZeroQ -> False, printhighDerFinder -> False
    }; (* End of Options[laxPairs] *)


(* ## ## ## ## ##          Function: laxWork          ## ## ## ## ## *)
(*********************************************************************)
(* laxWork[Options]                                                  *)
(* Purpose: To adjust program options and to open the menu.          *)
(* Input:   Any changes to options found under Options[laxPairs]     *)
(* Output:  None                                                     *)
(* Code is in File:  jwllax.m                                        *)
(* Created: 2 May, 2008, by DP                                       *)
```

```
(* Modified from ConservationLawsMD by jwl                              *)
(* Last Modified:  23 June, 2010, 00:40 by jwl at home                  *)
(*********************************************************************)

laxWork[opts___?OptionQ] := Module[{},
  verboseTest=opts;

  (* Setting default case (for print statements) to false.
     This tells the program that the item is Not computed
     but is instead, given from a data file. *)
  compFalse = False;
  (* Set all debug print statements.  Any print option set to True *)
  (* turns on all debug statements for that function. *)

  debugXTT = printTestXT /. {opts} /.
      Options[laxPairs];
  debugLMT = printTestLM /. {opts} /.
      Options[laxPairs];
  debugABED = printAbend /. {opts} /.
      Options[laxPairs];
  debugJZQ = printJZeroQ /. {opts} /.
      Options[laxPairs];
  debugGRF = printGenRuleFinder /. {opts} /.
      Options[laxPairs];
  debugHDL = printhighLDer /. {opts} /.
      Options[laxPairs];
  debugHDF = printhighDerFinder /. {opts} /.
      Options[laxPairs];

  Clear[printLOW, printHIGH];

  loadingTag = printLoadStatements /. {opts} /. Options[laxPairs];
  If[loadingTag, printLS = Print];
```

```
  (* Start the program by calling the menu function. *)
  newMenu
] (* end Module laxWork *)

printLS["jwllax.m of Mar 18, 2011 Sucessfully Loaded."
];
(* ## ## ## ## ##          End jwllax.m          ## ## ## ## ## *)

(* ::Package:: *)

(* ## ## ## ##          Function: jZeroQ          ## ## ## ## *)

(*********************************************************************)
(* jZeroQ[givenEq_, opts___?OptionQ]                                 *)
(* Purpose: To test if a given equation or matrix is zero            *)
(* Input:   An equation or a matrix                                  *)
(*          (Optional) Print Flags                                   *)
(* Output:  True if the equation equals zero or if all entries       *)
(*          in the matrix are zero. False otherwise.                 *)
(* Created: 20 June 2010, 21:45 by jwl at home                       *)
(* Code is in File:  jwlCheckForZero.m                               *)
(* Last Modified:  18 March 2011, 01:45 by jwl at home               *)
(*********************************************************************)
Off[General::spell1];
jZeroQ[givenEq_,knownZero_] :=
   Module[{printJZQ, zeroTestResult},
If[debugJZQ, printJZQ=Print, Clear[printJZQ], Clear[printJZQ]];
printJZQ["-------------------------------------------------------"];
printJZQ["Start of jZeroQ"];

printJZQ["At JZQ In: the equation that is input is "];
printJZQ[givenEq];
```

```
(* Takes equation or matrix input and tests if it is likely zero *)

(*
Curly Braces are to convert a scaler input
into a list (so Flatten will work)
zeroQ will work with scalers and matrices (lists of lists...)
*)
zeroTestResult=Apply[And, Flatten[PossibleZeroQ[{givenEq}]]];
printJZQ["Before testing with integration addition,
          The given equation/matrix is zero is:"];
printJZQ[ToString[zeroTestResult]];
If[knownZero,
If[
!zeroTestResult,
magic = True;
zeroTestResult=Apply[And, Flatten[PossibleZeroQ[{Simplify[givenEq]}]]];
magic = False;
]
];

printJZQ["After testing with integration addition,
          The given equation/matrix is zero is:"];
printJZQ[ToString[zeroTestResult]];

printJZQ["End of jZeroQ"];
printJZQ["-------------------------------------------------------"];

Return[zeroTestResult];
]
printLS["jwl_CheckForZero.m of Mar 18, 2011 Sucessfully Loaded."];
(* ### ### ###       End Function: jZeroQ       ### ### ### *)
```

```
(* ::Package:: *)

(* ## ## ## ##          Function: finduRules          ## ## ## ## *)
(*********************************************************************)
(* finduRules[eqList_,numVarIn_,depVarList_,forwardRule_]            *)
(* Purpose: To convert user dependent variables to internal          *)
(*             and vice versa                                        *)
(* Input:   A list of expressions                                    *)
(*          The number of dependent variables                        *)
(*          The list of user dependent variables                     *)
(*          A direction flag where                                   *)
(*             True takes user variable to internal                  *)
(*             False takes internal variable to user                 *)
(* Output:  The list of expressions with dependent                   *)
(*              variables converted                                  *)
(* Created: 08 August 2010, 11:45 by jwl at home                     *)
(* Code is in File:  jwllmtst.m                                      *)
(* Last Modified:  08 August 2010, 11:45 by jwl at home              *)
(*********************************************************************)
Off[General::spell1];
finduRules[eqList_,numVarIn_,depVarList_,forwardRule_] :=
   Module[{},

If[debugFUR,printFUR=Print,Clear[printFUR],Clear[printFUR]];
printFUR["-------------------------------------------------"];
printFUR["Start of finduRules"];

printFUR["FUR In : eqList : "];
printFUR[ToString[eqList,StandardForm]];
printFUR["FUR In : forwardRule : "];
printFUR[ToString[forwardRule,StandardForm]];

u2uuRule = Map[depVarList[[#]]->uu[#]&,Range[numVarIn]];
```

```
uu2uRule = Map[uu[#]->depVarList[[#]]&,Range[numVarIn]];
printFUR["FUR : u2uuRule : " <> ToString[u2uuRule,StandardForm]];
printFUR["FUR : uu2uRule : " <> ToString[uu2uRule,StandardForm]];

If[forwardRule,
    eqListOut = eqList/.u2uuRule,
    eqListOut = eqList/.uu2uRule
  ];

printFUR["End of finduRules"];
printFUR["-------------------------------------------------------"];

Return[eqListOut];
]
printLS["jwluuConvert.m of Aug 23, 2010 Sucessfully Loaded."];
(* ### ### ###       End Function: finduRules       ### ### ### *)

(* ::Package:: *)

(* ### ### ### ###              Function: findxRules              ### ### ### ### *)
(*********************************************************************)
(* findxRules[eqList_,numspVarIn_,inSpVarList_,forwardRule_]          *)
(* Purpose: To convert user independent variables to internal         *)
(*             and vice versa                                         *)
(* Input:   A list of expressions                                     *)
(*          The number of spacial variables                           *)
(*          The list of user spacial variables                        *)
(*          A direction flag where                                    *)
(*             True takes user variable to internal                   *)
(*             False takes internal variable to user                  *)
(* Output:  The list of expressions with independent                  *)
(*              variables converted                                   *)
(* Created: 08 August 2010, 11:45 by jwl at home                      *)
```

```
(* Code is in File:  jwllmtst.m                                          *)
(* Last Modified:  08 August 2010, 11:45 by jwl at home                  *)
(*************************************************************************)
Off[General::spell1];
findxRules[eqList_,numspVarIn_,inSpVarList_,forwardRule_] :=
   Module[{},

If[debugFXR,printFXR=Print,Clear[printFXR],Clear[printFXR]];
printFXR["-------------------------------------------------"];
printFXR["Start of findxRules"];

printFXR["FXR In : eqList : "];
printFXR[ToString[eqList,StandardForm]];
printFXR["FXR In : Number of Spatial Variables : "];
printFXR[ToString[numspVarIn,StandardForm]];
printFXR["FXR In : Spatial Variable List : "];
printFXR[ToString[inSpVarList,StandardForm]];
printFXR["FXR In : forwardRule : "];
printFXR[ToString[forwardRule,StandardForm]];

x2xxRule = Map[inSpVarList[[#]]->xx[#]&,Range[numspVarIn]];
xx2xRule = Map[xx[#]->inSpVarList[[#]]&,Range[numspVarIn]];
printFXR["x2xxRule : " <> ToString[x2xxRule,StandardForm]];
printFXR["xx2xRule : " <> ToString[xx2xRule,StandardForm]];

If[forwardRule,
    eqListOut = eqList/.x2xxRule,
    eqListOut = eqList/.xx2xRule
  ];

printFXR["End of findxRules"];
printFXR["-------------------------------------------------"];
```

```
Return[eqListOut];
]
printLS["jwlxxConvert.m of Aug 23, 2010 Sucessfully Loaded."];
(* ### ### ###       End Function: findxRules       ### ### ### *)

(* ::Package:: *)

(* ### ### ### ###                Function: printPDE              ### ### ### ### *)
(*********************************************************************)
(* printPDE[pde_,numPDE_,compFlag_]                                  *)
(* Purpose: To print the original PDE (or system) in a pretty form   *)
(* Input:   The list of equations in the PDE/system                  *)
(*          The number of equations in the list                      *)
(*          A flag telling where the PDE came from                   *)
(*              True for computed and False for user input           *)
(* Output:  Returns the variable containing "given" or "computed"    *)
(*              (for printing purposes only, comes from the flag)    *)
(* Created: 22 June 2010, 12:10 by jwl at home                       *)
(* Code is in File:  jwlPrintPDE.m                                   *)
(* Last Modified:  23 June 2010, 00:25 by jwl at home                *)
(*********************************************************************)
Off[General::spell1];
printPDE[pde_,numPDE_,compFlag_] :=
   Module[{},

If[debugPPDE,printPPDE=Print,Clear[printPPDE],Clear[printPPDE]];
printPPDE["-------------------------------------------------"];
printPPDE["Start of printPDE"];
printPPDE["At PPDE In: The given PDE is: "];
printPPDE[ToString[pde,StandardForm]];
printPPDE["At PPDE In: The number of PDEs is: "];
printPPDE[ToString[numPDE,StandardForm]];
printPPDE["At PPDE In: The computed flag is set to: "];
```

```
printPPDE[ToString[compFlag,StandardForm]];
printPPDE["-------------------------------------------------------"];

mypde[i_]:=pde[[i]];
(*
Test if input is coming from data file or computed work
True if from computed work, False if from data file
*)
If[
    compFlag,
    givenComp = "computed",
    givenComp = "given",
    givenComp = "computed" (* Leaving default as computed *)
];
If[
    numPDE==1,
    systemVar = "equation",
    systemVar = "system"
];

Print["The original ", systemVar, " is"];
For[j=1,j<=numPDE, j++,
Print[ToString[prettyPrint[pde[[j]],False,False],StandardForm]<>" = 0"];
];
Print[" "];

printPPDE["End of printPDE"];
printPPDE["-------------------------------------------------------"];

Return[givenComp];
](* End of printPDE Module *)
printLS["jwlPrintPDE.m of Sept 26, 2010 Sucessfully Loaded."];
(* ## ## ##       End Function: printPDE       ## ## ## *)
```

```
(* ::Package:: *)

(* ## ## ## ##          Function: prettyPrint          ## ## ## ## *)
(*********************************************************************)
(* prettyPrint[myEq_, pureFunTag_, convertBack_, inputCompFlag___]   *)
(* Purpose: Convert to and from internal variables                   *)
(* Input:   An equation or list of equations                         *)
(*            (Can be a pure function)                               *)
(*          Conversion direction flag                                *)
(*            (True for u to uu, False for uu to u)                  *)
(* Output:  The converted equation or list of equations              *)
(* Created: 08 August 2010, 11:45 by jwl at home                     *)
(* Code is in File:  jwlprettyprint.m                                *)
(* Last Modified:  31 December 2010, 01:45 by jwl at home            *)
(*********************************************************************)
Off[General::spell1];
prettyPrint[myEq_, pureFunTag_, convertBack_, inputCompFlag___] :=
  Module[{myEqOut, convertTest},

(* pureFunTag = True if myEq is a pure function *)
(* convertBack = True if you want to convert u to uu and x to xx *)
If[debugPP, printPP=Print, Clear[printPP], Clear[printPP]];

printPP["-----------------------------------------------"];
printPP["Start of prettyPrint"];

If[convertBack==True,
  convertTest = True,
  convertTest = False,
  convertTest = False
 ];
printPP["convertTest : "<>ToString[convertTest]];
```

```
myNumSpVar = Length[indepSpaceVarListInput];
mySpVarList = indepSpaceVarListInput;

myNumVar = Length[depVarListInput];
myVarList = depVarListInput;

If[pureFunTag,
  DownValues[myEq] =
      finduRules[DownValues[myEq],myNumVar,myVarList,convertTest];
      printPP["Returned Function (applied to x,t) : "];
      printPP[ToString[Apply[myEq,{x,t}],StandardForm]];
      Return[myEq],
  myEqOut =
      findxRules[finduRules[myEq,myNumVar,myVarList,convertTest],
                 myNumSpVar,mySpVarList,convertTest];
      Return[myEqOut],
  myEqOut =
      findxRules[finduRules[myEq,myNumVar,myVarList,convertTest],
                 myNumSpVar,mySpVarList,convertTest];
      Return[myEqOut]
];

printPP["End of prettyPrint"];
printPP["-----------------------------------------------"];
]
printLS["jwlprettyprint.m of Dec 31, 2010 Sucessfully Loaded."];
(* ## ## ##      End Function: prettyPrint      ## ## ## *)
```

```
(* ::Package:: *)

(* ## ## ## ##                Function: abend                ## ## ## ## *)
(*********************************************************************)
(* abend[notesToPrint___]                                              *)
(* Purpose: To test given L and M operators                            *)
(* Input:   A list of print statements for before the Abort            *)
(* Example Input: abend[{"Line 1", "Line 2", "Line 3"}]                *)
(*                abend[{}] (leads to default notes)                   *)
(* Output:  None. Aborts the program.                                  *)
(* Created: 22 June 2010, 12:10 by jwl at home                         *)
(* Code is in File:  jwlAbend.m                                        *)
(* Last Modified:  18 March 2011, 01:45 by jwl at home                 *)
(*********************************************************************)
Off[General::spell1];
abend[notesToPrint___] :=
   Module[{printDefault, printNotes, printABED},

If[debugABED, printABED=Print, Clear[printABED], Clear[printABED]];
printABED["------------------------------------------------------------"];
printABED["Start of abend"];

printABED["At ABED In: The list of notes is: "];
printABED[notesToPrint];
printABED["------------------------------------------------------------"];

Clear[printNotes, printDefault];

(* Checking if there are notes given, if not, will use default notes *)
If[Length[notesToPrint]==0,
   printDefault=Print,
   printNotes=Print,
   printNotes=Print
```

```
  ]; (* End of If *)

For[
    i=1,i<=Length[notesToPrint],i++,
    printNotes[notesToPrint[[i]]];
]; (* End of For *)

printDefault["The program has encountered an error."];
printDefault["Please check inputs and try again."];
printDefault["The program will now end."];

printABED["-------------------------------------------------------"];
printABED["At ABED 1: The program will now abort"];

printABED["End of abend"];
printABED["-------------------------------------------------------"];
Abort[];

]
printLS["jwlAbend.m of Mar 18, 2011 Sucessfully Loaded."];
(* ## ## ##       End Function: abend       ## ## ## *)

(* ::Package:: *)

(* ## ## ## ##              fileNameAssociations              ## ## ## ## *)
(* this Mathematica file lists symbols and corresponding filenames    *)
(* Filename = "fileNameAssociations.m"                                *)
(* Creates a global variable "fileNameAssociationsList" containing a  *)
(* list of pairs of symbolic strings and the associated real          *)
(* file name.                                                         *)
(* From that list, produces the function "fileNameAssociations" that  *)
(* takes the symbol and returns the filename.                         *)
```

```
(* Add entries here.  No comma after last pair. *)
(* Creates the list "fileNameAssociationsList"  *)
fileNameAssociationsList =
{
    {"extractCoefficientSystem.m","jwlExtractCoefficientSystem0526.m"},
    {"jwl_ConvertRules.m","m_jwlConvertRules0604.m"},
    {"jwl_lax.m","m_jwllax0318.m"},
    {"jwl_lmtst.m","m_jwllmtst0201.m"},
    {"jwl_xttst.m","m_jwlxttst0307.m"},
    {"jwl_uuconvert.m","m_jwluuconvert0823.m"},
    {"jwl_xxconvert.m","m_jwlxxconvert0317.m"},
    {"jwl_genrulefinder.m","m_jwlgenrulefinder0101.m"},
    {"jwl_highLDerFinder.m","m_jwlhighLDerFinder0201.m"},
    {"jwl_highDerFinder.m","m_jwlhighDerFinder0318.m"},
    {"jwl_prettyprint.m","m_jwlprettyprint1231.m"},
    {"jwl_CheckForZero.m","m_jwlCheckForZero0318.m"},
    {"jwl_Abend.m","m_jwlAbend0318.m"},
    {"jwl_printpde.m","m_jwlPrintPDE0926.m"},
    {"jwl_ConvertInput.m","m_jwlConvertInput0926.m"},
    (* Data Files *)
    {"d_jwlbousdeift.m","d_jwlBousDeift0306.m"},
    {"d_jwlbousv2.m","d_jwlBousv20306.m"},
    {"d_jwlbureq.m","d_jwlbureq0201.m"},
    {"d_jwlch.m","d_jwlch0926.m"},
    {"d_jwlhdym.m","d_jwlhdym0306.m"},
    {"d_jwlhdymv2.m","d_jwlhdymalternate0306.m"},
    {"d_jwlhs.m","d_jwlhs0304.m"},
    {"d_jwlkdv.m","d_jwlkdv0824.m"},
    {"d_jwlkdv_coeff.m","d_jwlkdvCoeff0526.m"},
    {"d_jwlkdv_complex.m","d_jwlkdvComplex0824.m"},
    {"d_jwlkk.m","d_jwlkk0201.m"},
    {"d_jwlkp.m","d_jwlkp0830.m"},
    {"d_jwlle.m","d_jwlle0201.m"},
```

```
    {"d_jwlliou.m","d_jwlliou0306.m"},
    {"d_jwlmkdv.m","d_jwlmkdv0304.m"},
    {"d_jwlpdeconst.m","d_jwlpdeconst0113.m"},
    {"d_jwlsineG.m","d_jwlsineG0824.m"},
    {"d_jwlsinhG.m","d_jwlsinhG0824.m"},
    {"d_jwlsk.m","d_jwlsk0201.m"},
    {"d_jwlsk_dye_parker.m","d_jwlskDyeParker0824.m"},
    {"d_jwlspe.m","d_jwlspe0824.m"},
    {"d_jwlzsm.m","d_jwlzsm0306.m"}
};
(* End fileNameAssociationsList *)
(* Create the function "fileNameAssociations" *)
(* from the list "fileNameAssociationsList"   *)
Scan[(fna[#[[1]]] = #[[2]]) &, fileNameAssociationsList];
(* Announce completion *)
printLS["jwlFileNameAssociations.m of Feb 1, 2011 Succefully Loaded. ",
      "File Associations: ",ToString[Length[fileNameAssociationsList]]];
(* ## ## ##       End: fileNameAssociations       ## ## ## *)
```

❄