THE UNIVERSITY OF CALGARY

On Wavelet Projection Methods for the Numerical Solution of Differential Equations

by

Blaine Edward Campbell

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF MATHEMATICS AND STATISTICS

CALGARY, ALBERTA

October, 1999

Abstract

This work presents an investigation of some initial attempts at the numerical solution of ordinary and partial differential equations using wavelet projection methods. A brief discussion of the history of wavelet analysis and an overview of relevant theory is presented. The representation of operators is given, most notably that of spatial derivatives and functions of spatial derivatives; emphasis is placed on the derivation of the standard and non-standard forms. Latto's method for the evaluation of connection coefficients is discussed. These results are applied via Bertoluzza and Naldi's collocation method for the solution of ordinary differential equations, and is followed by Amaratunga, Williams, Qian, and Weiss' Galerkin approach to the same problem. A class of non-linear partial differential equations is solved numerically by the method of Beylkin and Keiser, using a semi-group approach and quadrature formulation. Examples are given for each method.

Acknowledgments

Thanks to the many friends who provided encouragement and support over the (somewhat protracted) period of completing this thesis. Particular thanks to: Bimal Patel; Blair Nicolle; Suzanne and the entire Deterville family; Michael Placonouris; Marissa Kochanski; Lawrence Leong; Werner Karsten; Gwynneth Kirk; Gerald Simon; Shawn O'Gallagher; Jessica Main; and Rochus Schmid.

Greatly helpful suggestions were provided along the way by Dr. Indy Lagu, who towards the end probably read this work more times than I did. This thesis would be in a sorry state of disrepair without the doctor's input.

A considerable debt of gratitude is owed to Dr. Len Bos—I count myself very lucky to have had him as my supervisor. Without Dr. Bos' insights and support, my degree would likely remain unfinished.

Above all else, thanks to my family, Don, Val, and Peter, for all their continued love and encouragement.

Table of Contents

List of Figures

Chapter 1

Introduction

In recent years, wavelet analysis has gained considerable popularity as a powerful new mathematical tool, and has found favour in a wide variety of fields. This work considers one particular and highly promising area of application, the wavelet-based numerical solution of differential equations.

The initial spark for wavelet theory was provided by J. Morlet, a French research scientist working in seismic analysis. Morlet collaborated with A. Grossmann, a theoretical physicist, to develop a "geometrical formalism of the continuous wavelet transform, based on invariance under affine groups—namely translation and dilatation" [21, p. 398]. In 1985, Y. Meyer recognised in Morlet and Grossmann's 1984 paper the Calderón identity as implicit in the wavelet admissibility condition. Meyer went on to collaborate with Grossmann and I. Daubechies to create frames for $\mathbf{L}^2(\mathbf{R})$ using wavelets.

Perhaps the key success behind the effectiveness of wavelets was found by Meyer and S. Mallat in 1988 with the introduction of multi-resolution analysis (M.R.A.). M.R.A. provides a dyadic, localized framework for the analysis of $\mathbf{L}^2(\mathbf{R})$, and leads to a general method for building orthogonal wavelet bases. Such bases were presented in Daubechies' seminal 1988 paper, "Orthonormal Bases of Compactly Supported Wavelets" [18]. Since then there has been considerable, growing interest in wavelet theory throughout the mathematics, scientific, and engineering communities.

In the last ten years, wavelet analysis' arsenal of tools has continued to expand. Of importance are S. Mallat's decomposition and recomposition pyramid schemes for

discrete wavelet projections (see [14], [18]), and the fast wavelet transform introduced by Beylkin, Coifman, and Rokhlin in [8].

As the theory has evolved so have the variety of applications expanded. Today wavelets are finding uses in many areas of signal analysis, *e.g.*, seismic data analysis and acoustic sampling, image analysis (most notably edge detection), functional analysis (such as differentiability and singularity detection), and numerical analysis. This thesis focuses on a selection of techniques used in the numerical solution of differential equations.

Our interest in this topic developed from investigations into numerical methods for honest plotting, which led to the paper "Singularity Detection and Processing with Wavelets" by S. Mallat and W.L. Hwang [32]. This resulted in a search for methods marrying wavelet theory with other fields of application (partial differential equations in particular).

Interest in using wavelets to numerically solve differential equations is spurred by the properties of wavelets and the difficulties inherent in existing methods. Many differential equations give rise to "interesting" features, such as boundary shocks and cusps (generally, high frequency behaviour). Simple numerical methods have difficulty resolving such features. This has resulted in a number of specialized methods, often developed for specific classes of problems, or even individual differential equations. The localisation and adaptive properties of wavelets have led many to believe more general and robust techniques can be developed using these functions.

This thesis is by no means meant to act as an exhaustive study of the approaches currently investigated by the research community. Rather, we focus our interest on a small selection of the initial attempts at creating numerical wavelet-based methods. This

thesis is meant to provide an investigation into the validity of these techniques, and not to perform a large-scale, rigorous comparison between methods.

Many techniques have been developed using wavelet theory which fall outside our scope of interest. For instance, Dahmen and Dahlke's groups have focused on the use of biorthogonal wavelet based methods; others are considering the construction of multidimensional wavelets for the solution of problems arising from turbulence (see, for example, [17] and [33]).

The next chapter provides a brief overview of the wavelet theory relevant to our discussion. This presentation is kept brief as a number of introductory works on wavelets are widely available. Chapter 3 extends wavelet representation to that of operators. This is of key importance, as a successful wavelet-based technique will (preferably) perform all of its calculations within a wavelet context, rather than moving between physical and frequency space at each iteration. A brief study of general integral operators is given, which introduces structural properties that are of great importance in the use of derivative operators. (Moreover, the inverse of a derivative operator is typically an integral operator.) Chapters 4 and 5 use this framework for the numerical solution of differential equations. Chapter 4 considers second order boundary value ordinary differential equations, while Chapter 5 looks at partial differential evolution equations. Of special note, we develop an implementation of Beylkin, Keiser, and Vozovoi's exact linear part scheme [11] for non-linear evolution equations, and apply it to Burger's equation. We conclude with some numerical examples of each of the presented techniques and a brief discussion of the observed behaviour.

The following definitions are used throughout the subsequent discussion.

Definition 1.1 *A sequence of reals* $\{c_n\}$ *is said to be in* $\ell^\infty(\mathbf{R})$ *if and only if*

$$\|\{c_n\}\|_{\ell^\infty} := \sup_{n\in\mathbf{Z}} \{|c_n|\} < \infty.$$

Definition 1.2 *A sequence of reals* $\{c_n\}$ *is said to be in* $\ell^1(\mathbf{R})$ *if and only if*

$$\|\{c_n\}\|_{\ell^1} := \sum_{n\in\mathbf{Z}} |c_n| < \infty.$$

Definition 1.3 *A sequence of reals* $\{c_n\}$ *is said to be in* $\ell^2(\mathbf{R})$ *if and only if*

$$\|\{c_n\}\|_{\ell^2} := \left(\sum_{n\in\mathbf{Z}} c_n^2\right)^{1/2} < \infty.$$

Definition 1.4 *A function* f *is said to be in* $\mathbf{L}^\infty(\mathbf{R})$ *if and only if*

$$\|f\|_\infty := \sup_{x\in\mathbf{R}} |f(x)| < \infty.$$

Definition 1.5 *A function* f *is said to be in* $\mathbf{L}^1(\mathbf{R})$ *if and only if*

$$\|f\|_1 := \int_{\mathbf{R}} |f(x)|\, dx < \infty.$$

Definition 1.6 *A function* f *is said to be in* $\mathbf{L}^2(\mathbf{R})$ *if and only if*

$$\|f\|_2 := \left(\int_{\mathbf{R}} f(x)^2\, dx\right)^{1/2} < \infty.$$

Definition 1.7 *The inner product of* $f \in \mathbf{L}^2(\mathbf{R})$ *with* $g \in \mathbf{L}^2(\mathbf{R})$ *is given by*

$$\langle f, g\rangle = \int_{\mathbf{R}} f(x)\overline{g(x)}\, dx.$$

Definition 1.8 *A vector* $\mathbf{u}$*, indexed on* k*, is indicated by* $\{u\}_k$.

Chapter 2

An Overview of Wavelet Theory

We must first provide an overview of wavelet theory relevant to the forthcoming chapters. For a more thorough presentation of the theory, the reader is directed to some of the following sources: Daubechies' seminal paper [18] that initiated the wide-spread interest in wavelets, and her book [20]; Chui's first book on wavelets [14]; Z. Hasham's Master's thesis [23]; the paper by Jawerth and Sweldens on multiresolution analysis [26]; and the book edited by Koornwinder *et al.* [29].

Why should one consider using wavelets? The Fourier transform is the tool most often used in signal analysis and decomposition and pseudo-spectral methods for the solution of differential equations. There are, however, a number of reasons to avoid the use of Fourier transforms.

Consider for any function $f \in \mathbf{L}^2(\mathbf{R})$ the continuous forward and inverse Fourier transforms defined by

$$\begin{aligned} (\mathcal{F}f)(z) &= \widehat{f}(z) := \int_{\mathbf{R}} f(x) e^{-2\pi i x z}\, dx, \\ (\mathcal{F}^{-1}\widehat{f})(x) &= f(x) := \int_{\mathbf{R}} \widehat{f}(z) e^{2\pi i x z}\, dz. \end{aligned} \tag{2.1}$$

There are two key problems with (2.1). First, the Fourier transform projects the $\mathbf{L}^2(\mathbf{R})$ function f into a basis of complex exponentials that is not itself in $\mathbf{L}^2(\mathbf{R})$. Second, we require f at every point of the real line in order to determine its frequency (Fourier value) at a particular point z. It is also worth noting that the expansion given by (2.1) is continuous, whereas a discrete expansion is often preferable in applications. These problems are overcome in wavelet theory.

2.1 Basic Wavelet Theory

A so-called *mother wavelet* (or simply "wavelet") is any function $\psi \in \mathbf{L}^2(\mathbf{R})$ which satisfies the admissibility condition

$$C_\psi := \int_{-\infty}^{\infty} |\xi|^{-1} \left|\widehat{\psi}(\xi)\right|^2 d\xi < \infty, \tag{2.2}$$

i.e., $\widehat{\psi} \in \mathbf{L}^2(\mathbf{R}, d\xi/|\xi|)$. If additionally we have $\psi \in \mathbf{L}^1(\mathbf{R})$, then $\widehat{\psi} \in \mathbf{C}(\mathbf{R})$ ([14, p. 25]), so (2.2) implies $\widehat{\psi}(0) = 0$ and thus $\int_{\mathbf{R}} \psi(x)\, dx = 0$. These add up to good localization (and hence decay) properties (see [14], [18], [23]) in both the physical and frequency domains, a property which the Fourier basis functions, $e^{2\pi i n x}$, do not have.

For a wavelet ψ, define $\psi_{a,b}$ as

$$\psi_{a,b} := |a|^{-1/2} \psi\left(\frac{x-b}{a}\right).$$

Then the continuous forward and inverse wavelet transforms ([14, p. 7]) for $f \in \mathbf{L}^2(\mathbf{R})$ are defined by

$$\begin{aligned} (\mathcal{W}f)(a,b) &= \int_{\mathbf{R}} f(x) \overline{|a|^{-1/2} \psi\left(\frac{x-b}{a}\right)}\, dx, \\ f(x) &= C_\psi^{-1} \int_{\mathbf{R}} \int_{\mathbf{R}} (\mathcal{W}f)(a,b) |a|^{-1/2} \psi\left(\frac{x-b}{a}\right) db\, \frac{da}{a^2}. \end{aligned} \tag{2.3}$$

In the majority of applications one chooses $a = 2^{-m}, m \in \mathbf{Z}$, and $b = k2^{-m}, k \in \mathbf{Z}$, giving wavelets $\psi_{m,k}$ (noting a slight change in notation), where

$$\psi_{m,k}(x) = 2^{m/2} \psi(2^m x - k).$$

This results in a discrete partitioning of the frequency parameter, a, and of the translation parameter, b, into disjoint bands. The resulting discrete transform is then

$$(Q_{m,k} f) := 2^{m/2} \int_{\mathbf{R}} f(x) \overline{\psi(2^m x - k)}\, dx. \tag{2.4}$$

Definition 2.1 *A function $\psi \in \mathbf{L}^2(\mathbf{R})$ generates a* **frame** $\{\psi_{m,k}\}$ *of $\mathbf{L}^2(\mathbf{R})$ if for all $f \in \mathbf{L}^2(\mathbf{R})$ and for some constants $A, B \in \mathbf{R}$,*

$$A \|f\|_2^2 \leq \sum_{j,k \in \mathbf{Z}} |\langle f, \psi_{m,k} \rangle|^2 \leq B \|f\|_2^2.$$

If ψ generates a frame then in certain cases we have a convenient series representation

$$f(x) = \sum_{m \in \mathbf{Z}} \sum_{k \in \mathbf{Z}} (Q_{m,k} f) \, \tilde{\psi}_{m,k}(x) \tag{2.5}$$

where $\tilde{\psi}$ is called the *dual* of ψ (provided $\tilde{\psi}$ exists).

It should be noted that the existence of a dual $\tilde{\psi}$ is not in general guaranteed for all wavelets ψ that generate a frame. To ensure the existence of a dual, we must restrict our consideration to that of Riesz bases.

Definition 2.2 *A set of functions of the form $\{f(\cdot - k) : k \in \mathbf{Z}\}$, $f \in \mathbf{L}^2(\mathbf{R})$, satisfies the* **Riesz condition** *with* Riesz bounds *A and B if for any sequence $\{c_k\} \in \ell^2(\mathbf{R})$,*

$$A \|\{c_k\}\|_{\ell^2}^2 \leq \left\| \sum_{k \in \mathbf{Z}} c_k f(\cdot - k) \right\|_{\ell^2}^2 \leq B \|\{c_k\}\|_{\ell^2}^2.$$

Definition 2.3 *A set of functions $\{f_j\}$ is called a* **Riesz basis** *if:*

(1) $\{f_j\}$ satisfies the Riesz condition;

(2) span$\{f_j\}$ *is dense in $\mathbf{L}^2(\mathbf{R})$.*

Definition 2.4 ([23, p. 65]) *Suppose $\{\psi_{j,k}\}$ is a Riesz basis of $\mathbf{L}^2(\mathbf{R})$. Then $\tilde{\psi} \in \mathbf{L}^2(\mathbf{R})$ is called the* **dual** *of ψ if, for all $j, k, l, m \in \mathbf{Z}$,*

$$\left\langle \psi_{j,k}, \tilde{\psi}_{l,m} \right\rangle = \delta_{jl} \delta_{km},$$

(the biorthogonality condition*) where δ_{jk} is the Kronecker delta symbol.*

Theorems 5.23 and 5.24 of [14, pp. 154-155] provide sufficient conditions for the existence of a dual $\tilde{\psi}$ if ψ generates a Riesz basis.

Although we still have infinite integrations and summations in (2.5) (a problem dealt with below), we have our first advantage over the Fourier transform—the representation of $\mathbf{L}^2(\mathbf{R})$ functions in a basis of $\mathbf{L}^2(\mathbf{R})$ functions.

Now we must deal with the lack of localisation (*i.e.*, the requirement that the entire signal be used to determine the frequency value at one point). One common approach in classical Fourier analysis is to use the Gabor transform ([23])

$$(G_{a,b}f)(z) := \int_{\mathbf{R}} f(t)e^{-\imath zt}g_a(t-b)\,dt,$$

where

$$g_a(t) = \frac{1}{2\sqrt{\pi a}}e^{-t^2/4a}.$$

The kernel $g_a(t-b)$ is said to be a *window function* with centre b and radius $\sqrt{a}$. Introducing $g_a(t-b)$ into the Fourier transform allows one to look at a local neighbourhood of the function (signal) f, eliminating the need to consider the entire function (the resulting physical-frequency window has dimensions $[b-\sqrt{a}, b+\sqrt{a}] \times [z-1/2\sqrt{a}, z+1/2\sqrt{a}]$). The choice of a and b is not however directly related to the choice of frequency z, as the size of the window remains fixed for all frequencies. Because high frequency behaviour requires greater accuracy, this fixed window size is a hindrance. Without going into details (the reader is directed to [14] and [23]), the use of wavelets gives a physical-frequency window which narrows and grows taller for high frequencies (hence giving greater physical accuracy) but broadens and shrinks for low frequencies. Window behaviour is controlled by the choice of m and k in (2.5). In a sense we have "automatic" localisation. Figure 2.1 illustrates how windows appear at different octaves (values of m).

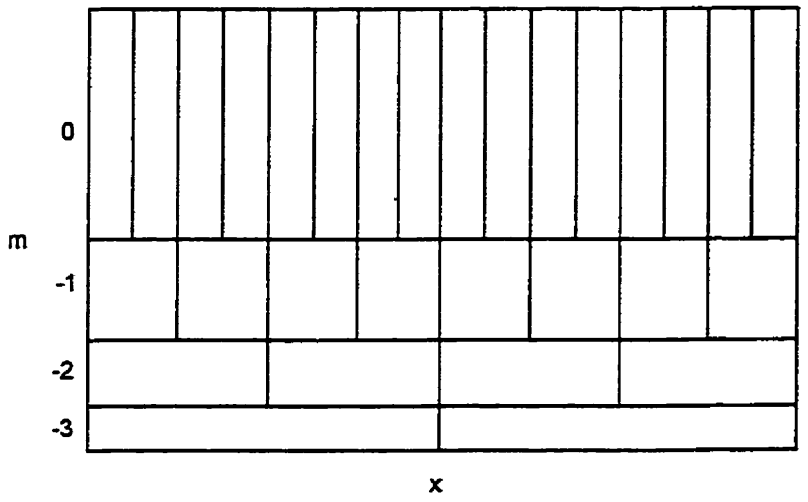


Figure 2.1: Space vs. frequency plot of windows for scales −3 to 0.

The usefulness of this result is summarized in the following theorem, found in the paper by Liandrat *et al.* ([31, p. 230])—comparable theorems can be found in [26, p. 6], [32, p. 620], and [42, p. 130]. A definition must first be provided.

Definition 2.5 *Let $n \in \mathbf{Z}^+$ and $n \leq \alpha < n+1$. A function $f \in \mathbf{C}^n(\mathbf{R})$ is said to be* **uniformly Lipschitz-α** *if and only if there exists a constant $A > 0$ such that for all $x, h \in \mathbf{R}$,*

$$\left| f^{(n)}(x+h) - f^{(n)}(x) \right| \leq A|h|^{\alpha - n}.$$

Theorem 2.6 *Take $\psi \in \mathbf{C}^n(\mathbf{R})$ with $n+1$ vanishing moments, i.e.,*

$$\int_{\mathbf{R}} x^p \psi(x)\, dx = 0, \quad \text{for} \quad 0 \leq p \leq n. \tag{2.6}$$

Let $0 < \alpha < n$, $\alpha \in \mathbf{R} \setminus \mathbf{Z}$, $f \in \mathbf{L}^2(\mathbf{R})$, and $[a, b]$ an interval. The function $f(x)$ is uniformly Lipschitz-α over the interval $[a, b]$ if and only if for any $m, k \in \mathbf{Z}$ such that

$2^{-m}k \in (a,b)$,

$$|\langle f, \psi_{m,k} \rangle| = O\left(2^{-(\alpha+\frac{1}{2})m}\right).$$

The proof of this theorem is beyond the scope of this work, and is therefore omitted.

This result shows that the decay of the wavelet coefficients depends on the local smoothness of the function in a window determined by the choice of resolution m. By examining the coefficients of a function's wavelet transform, we then have a powerful method to investigate a function's local behaviour. Figure 2.4 demonstrates such localized predominance of wavelet coefficients near non-smooth features. (The derivation of the plotted set of coefficients is explained in Section 2.4.)

As seen in the following chapters, this automatic localisation is extremely useful in solving differential equations. Due to Theorem 2.6, a wavelet basis (in a sense) automatically focuses on high frequency components, enabling us to better discern high frequency behaviour such as cusps, steep gradients, and boundary shocks. Standard differential equation methods have difficulty dealing with such features.

2.2 Multiresolution Analysis

The idea of multiresolution analysis (M.R.A.) is to use the wavelet theory described in Section 2.1 to generate a series of nested subspaces which act as successive approximations to $\mathbf{L}^2(\mathbf{R})$. In order to construct such a set of subspaces we require additional properties of the wavelet functions.

The essential requirements to build an M.R.A. (see, for instance, [14] and [20]) are given as follows:

(1) there exist (nested) subspaces $V_m \subset \mathbf{L}^2(\mathbf{R})$, $m \in \mathbf{Z}$, such that

$$\cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots;$$

(2) $\bigcap_{m\in\mathbf{Z}} V_m = \{0\}$ and $\overline{\bigcup_{m\in\mathbf{Z}} V_m} = \mathbf{L}^2(\mathbf{R})$ (the *non-redundancy* and *representation conditions*);

(3) $f \in V_m$ if and only if $f(2\,\cdot) \in V_{m+1}$ (the *octave condition*);

(4) there exists a function $\phi \in V_0$ such that for all $k \in \mathbf{Z}$, $\{\phi_{m,k}\}_k$ constitutes a Riesz basis for V_m (*i.e.*, $V_m = \text{span}\{\phi_{m,k} : k \in \mathbf{Z}\}$, and $\{\phi_{m,k}\}_k$ satisfies the Riesz condition).

Define P_m to be the projection operator of a function $f \in \mathbf{L}^2(\mathbf{R})$ onto V_m by

$$\begin{aligned} (P_m f)(x) &:= \sum_{k\in\mathbf{Z}} f_{m,k}\phi_{m,k}(x) \quad \text{where} \\ f_{m,k} &:= \int_{\mathbf{R}} f(x)\tilde{\phi}_{m,k}(x)\,dx, \end{aligned} \tag{2.7}$$

for $\tilde{\phi}$ the dual of ϕ. As a result of (1) and (2) we have that $\lim_{m\to\infty} P_m f = f$ in $\mathbf{L}^2(\mathbf{R})$. Requirement (3) implies that the V_m correspond to different scales (frequency bands). Finally, as a result of (4) we additionally have that $f \in V_m$ implies $f(\cdot - 2^m k) \in V_m$ for all $k \in \mathbf{Z}$. The function ϕ is referred to as a *father wavelet* or *scaling function*.

By (1) we may decompose the subspaces V_m by

$$V_{m+1} = V_m \bigoplus W_m \quad \text{where} \quad W_m \perp V_m; \tag{2.8}$$

so that W_m is the orthogonal compliment of V_m in V_{m+1}. Then we define an operator $Q_m : f \to W_m$ by $Q_m := P_{m+1} - P_m$. As a result of the conditions (1) and (2) and the definition of W_m :

(5) $\mathbf{L}^2(\mathbf{R}) = \bigoplus_{m\in\mathbf{Z}} W_m$.

Finally by Theorem 5.1.1 of [20, p. 135], we have a very important result:

(6) there exists a function ψ such that $W_0 = \overline{\text{span}\{\psi_{0,k} : k \in \mathbf{Z}\}}$, and consequently $W_m = \overline{\text{span}\{\psi_{m,k} : k \in \mathbf{Z}\}}$.

Now as $\phi \in V_0 \subset V_1$ and $\psi \in W_0 \subset V_1$, and by condition (4), there exist sequences $\{h_\ell\} \in \ell^2(\mathbf{Z})$ and $\{g_\ell\} \in \ell^2(\mathbf{Z})$ such that

$$\phi(x) = \sqrt{2}\sum_k h_k \phi(2x-k) \quad \text{and} \tag{2.9}$$

$$\psi(x) = \sqrt{2}\sum_k g_k \phi(2x-k). \tag{2.10}$$

Equations (2.9) and (2.10) are referred to as the *two-scale relations*.

The sequences $\{h_\ell\}$ and $\{g_\ell\}$ are respectively referred to as low-pass and high-pass filters. The pair constitute a *quadrature mirror filter* (Q.M.F.), a term which arises from the theory of sub-band coding schemes (see the discussion on p. 162 of [20]). As we are only considering orthonormal $\phi_{0,k}$ in the upcoming sections, it can be shown that the $\psi_{0,k}$ are also orthonormal and we may take $g_n = (-1)^{n-1}h_{-n}$ (see [20, p. 135] for the details).

2.3 Daubechies Wavelets and Coiflets

Now that we have the M.R.A. framework in which to work, we give a very brief overview of the wavelets used in the following chapters.

The first and perhaps most widely used set of wavelets are those derived by Daubechies in [18] and [20] (see also [16] and almost any other introductory paper on wavelets). One

notation for these wavelets is db$M$ or DM, where M indicates the number of vanishing moments (*cf.* (2.6)). (The other convention, as used in [1] and [4], is to refer to Daubechies wavelets by the number of coefficients in either filter band of the Q.M.F.) The most important feature of Daubechies wavelets is that they have compact support of $[0, 2M-1]$, with $L := 2M$ (finite) low-pass Q.M.F. coefficients (the $\{h_\ell\}$). Additional properties worth noting are

$$\sum_{k=0}^{L-1} h_k = \sqrt{2}, \quad \sum_{k=0}^{L-1} h_k h_{k+2n} = \delta_{0n}, \quad \sum_{k=0}^{L-1} h_k^2 = 1, \tag{2.11}$$

and

$$\phi_M \dot{\in} C^\alpha(\mathbf{R}), \text{ where } \alpha := (L-1)\left(1 - \frac{\log 3}{2\log 2}\right)$$

(note that this is an asymptotic result), where ϕ_M is the father wavelet corresponding to DM. These are results of the construction of Daubechies wavelets; their derivation falls outside of the focus of this work. (For further details, see [14], [16], and especially Sections 2A and 4 of [18] and Chapter 6 of [20].) Figure 2.2 shows the Daubechies 10^{th} order mother wavelet.

Coiflets constitute the other family of wavelets often used in subsequent chapters. Coiflets are also compactly supported, and allow for both the wavelet and scaling functions to have a large number of zero moments. This wavelet family was suggested by R. Coifman in 1989 and presented by Daubechies in [19] and [20] (see also [35]). The Coiflet family developed in [8] is slightly different from that found in [20]; we follow the latter's approach. It must be noted that in either case, Coiflets are *nearly interpolating* for a fine enough scale j, *i.e.*, $\langle f, \phi_{j,k} \rangle \approx 2^{-j/2} f(2^j k)$ (see, for example, [8, p. 150]).

Of primary interest in Chapter 5, the scaling functions for Coiflets constructed in [8]

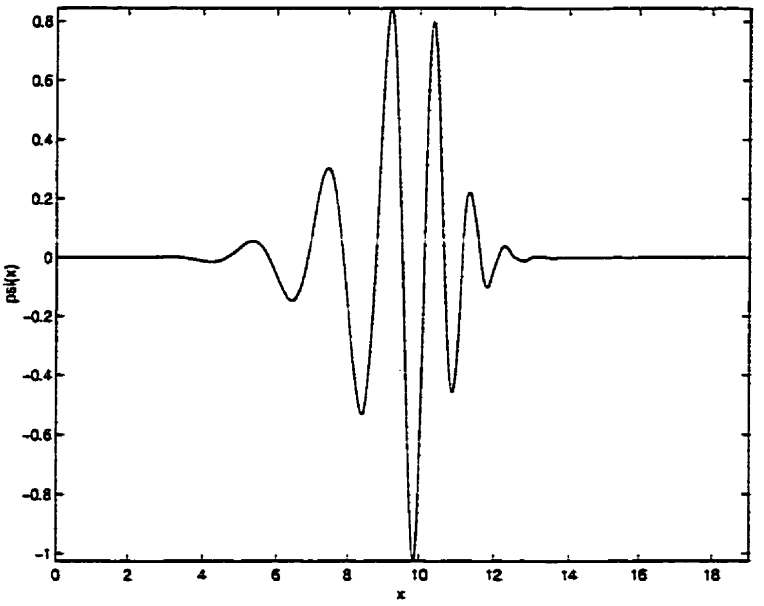


Figure 2.2: A plot of the D10 mother wavelet.

have *shifted vanishing moments, i.e.,*

$$\int_{\mathbf{R}} (x-\chi)^m \phi(x)\,dx = 0, \qquad m = 0, \ldots, M$$

where

$$\chi := \int_{\mathbf{R}} x\phi(x)\,dx.$$

As pointed out in [35, p. 10], for L the length of $\{h_\ell\}$ (the low-pass filter), χ falls within $[0, L-1]$. Figure 2.3 shows the scaling function of a Coiflet that is frequently used in Chapter 6.

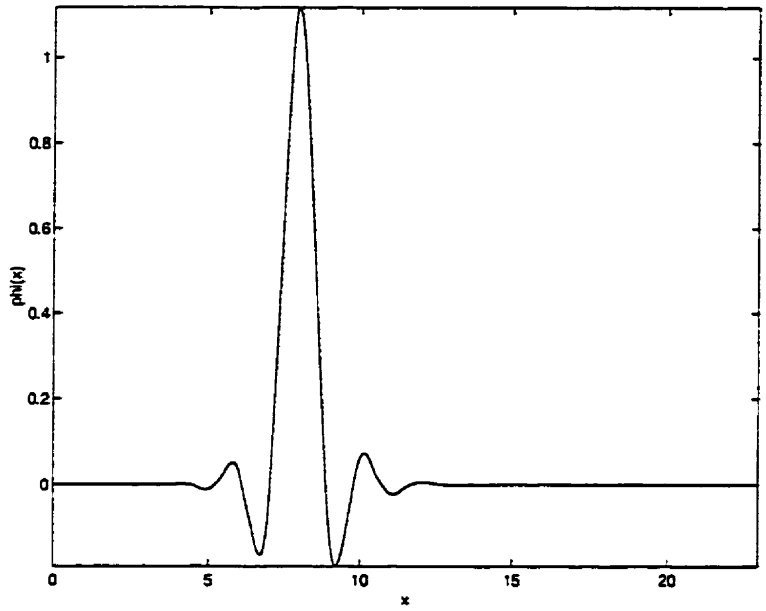


Figure 2.3: A plot of a Coiflet scaling function (father wavelet), with $L = 24$, $M = 7$.

2.4 Mallat's Decomposition Algorithm

Mallat's decomposition and recomposition algorithms for the representation of vectors in a so-called *wavelet basis* is the last matter we must consider. This is often referred to as the *pyramid scheme*, and is based on the Laplacian pyramid scheme of Burt and Adelson ([18, p. 922]). Again we omit most of the details—a number of references deal with the Mallat algorithm, including [9], [10], [14], [18], [20], [26], and [28].

One starts with a vector $\mathbf{u} \in \ell^2(\mathbf{R})$. We must first choose a multi-resolution analysis (a fixed set of nested subspaces $\{V_m\}$ with corresponding scaling function ϕ and wavelet ψ). For the finite expansion, we take V_{-J} to be our coarsest subspace, and V_0 to be the finest.

Define f by

$$f := \sum_n u_n \phi_{0,n} \in V_0.$$

As we are using an M.R.A., we may decompose $V_0 = W_{-1} \oplus V_{-1}$, *i.e.*, a "blurred" version V_{-1} (often referred to as the average), and W_{-1}, the difference between V_0 and V_{-1}. Then for projection operators

$$\begin{aligned} P_j &: \mathbf{L}^2(\mathbf{R}) \to V_j \quad \text{and} \\ Q_j &: \mathbf{L}^2(\mathbf{R}) \to W_j \end{aligned}$$

we can write

$$f = P_{-1}f + Q_{-1}f = \sum_k c_k^{-1} \phi_{-1,k} + \sum_k d_k^{-1} \psi_{-1,k}. \tag{2.12}$$

The coefficients c_k^{-1} and d_k^{-1} are given by

$$\begin{aligned} c_k^{-1} &= \langle \phi_{-1,k}, f \rangle \\ &= \sum_n u_n \langle \phi_{-1,k}, \phi_{0,n} \rangle \\ &= \sum_n h_{n-2k} u_n \quad \text{and} \\ d_k^{-1} &= \sum_n g_{n-2k} u_n, \end{aligned}$$

where h and g are the quadrature mirror filters for our choice of multiresolution analysis.

Now, by the two-scale relations and re-indexing, it can be shown that

$$\phi_{n,k}(x) = \sum_m h_{m-2k} \phi_{n+1,m}(x) \tag{2.13}$$

and

$$\psi_{n,k}(x) = \sum_m g_{m-2k} \phi_{n+1,m}(x). \tag{2.14}$$

Then (2.12) can be extended to sub-spaces V_j, $j < -1$, via the nesting property of M.R.A. Define operators $H, G : \ell^2(\mathbf{Z}) \to \ell^2(\mathbf{Z})$ by

$$\{H\mathbf{a}\}_k = \sum_n h_{n-2k} a_n$$
$$\{G\mathbf{a}\}_k = \sum_n g_{n-2k} a_n,$$

then the coefficients for V_j are related to those of V_{j+1} by

$$\mathbf{c}^j = H\mathbf{c}^{j+1}, \quad \mathbf{d}^j = G\mathbf{c}^{j+1}. \tag{2.15}$$

This can be viewed as an averaging and differencing process. That is to say, $\mathbf{c}^j$ is considered to be an "average" of $\mathbf{c}^{j+1}$, as it is a coarser approximation of the original function f (this follows as the support of ϕ_j is twice that of ϕ_{j+1} by the definition of $\phi_{j,k}$, and $c_k^j := \langle \phi_{j,k}, f \rangle$). The $\mathbf{d}^j$ then represent the difference in "information" between $\mathbf{c}^j$ and $\mathbf{c}^{j+1}$.

After J applications of (2.15), we retain $\{\mathbf{d}^{-1}, \mathbf{d}^{-2}, \ldots, \mathbf{d}^{-J}, \mathbf{c}^{-J}\}$. This vector constitutes the Mallat wavelet decomposition. Because each down-sampling step produces half as many coefficients as present in the previous step, if the original vector $\{u_n\}$ has N_s non-zero entries, then there are N_s non-zero entries in the wavelet decomposition. Thanks to Theorem 2.6, this representation more easily allows for compression (by the application of *ϵ-thresholding, i.e.*, removing all values below a given threshold value ϵ), with little loss of detail. An example of Mallat decomposition is seen in Figure 2.4.

The infinite Mallat decomposition is almost identical to the above. Here one uses property (2.2), giving a bi-infinite vector $\{\ldots, \mathbf{d}^{-2}, \mathbf{d}^{-1}, \mathbf{d}^0, \mathbf{d}^1, \mathbf{d}^2, \ldots\}$ as the expansion.

To reconstruct (recompose) the original signal, given a Mallat wavelet decomposition, one proceeds as follows. Using (2.13) and (2.14) and noting orthogonality of nested sub-

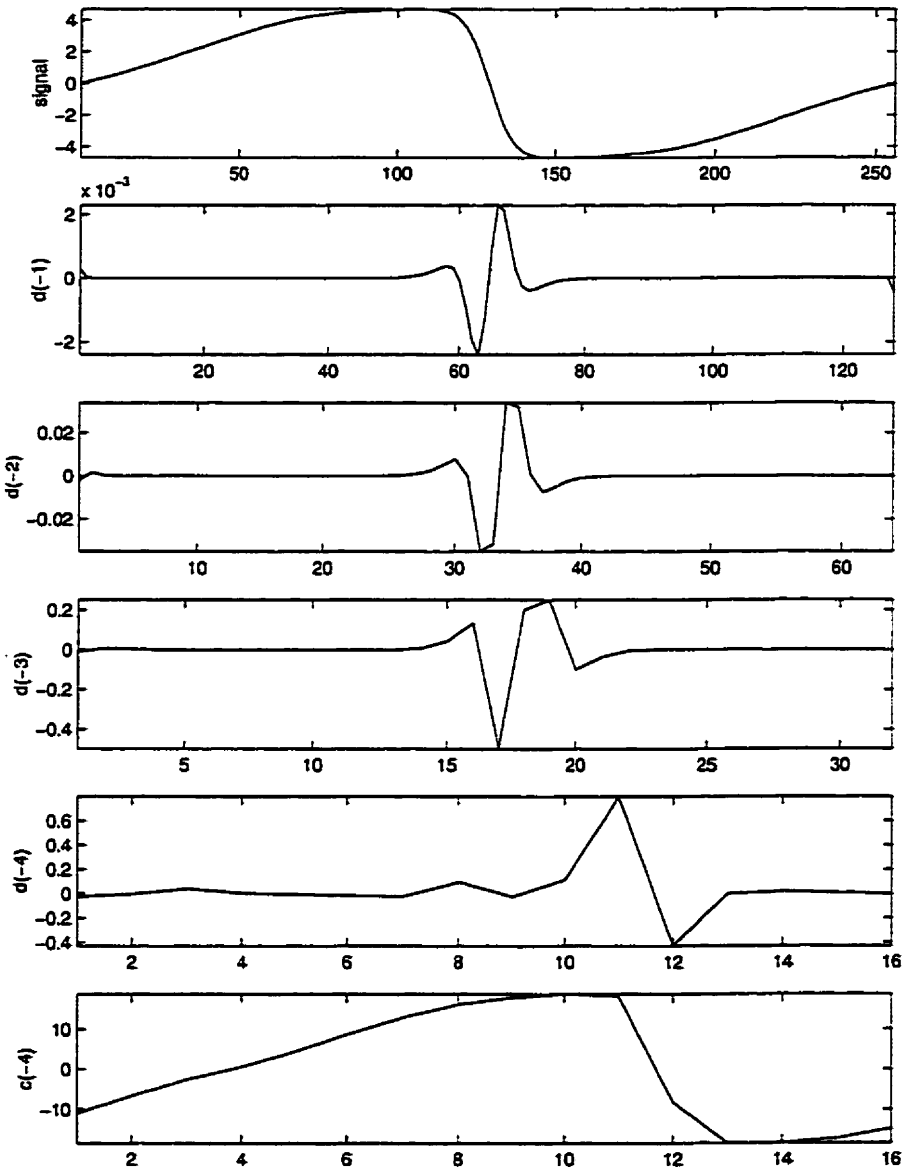


Figure 2.4: An example Mallat decomposition using $J = -4$ and Daubechies-4 wavelets. Note the presence of large wavelet coefficients in the region of the steep gradient. The horizontal axis corresponds to the index of the coefficients, while the vertical axis corresponds to the coefficient values.

spaces, we have

$$\langle \phi_{n,l}, \phi_{n+1,k} \rangle = h_{k-2l}$$

and

$$\langle \psi_{n,l}, \psi_{n+1,k} \rangle = g_{k-2l}.$$

Then the coefficients c_n^{j+1} are given in ([24, p. 70]) as

$$\begin{aligned} c_n^{j+1} &= \langle f, \phi_{j+1,n} \rangle \\ &= \langle P_{j+1} f, \phi_{j+1,n} \rangle \\ &= \langle P_j f + Q_j f, \phi_{j+1,n} \rangle \\ &= \left\langle \sum_l c_l^j \phi_{j,l}, \phi_{j+1,n} \right\rangle + \left\langle \sum_l d_l^j \psi_{j,l}, \phi_{j+1,n} \right\rangle \\ &= \sum_l h_{n-2l} c_l^j + \sum_l g_{n-2l} d_l^j. \end{aligned} \tag{2.16}$$

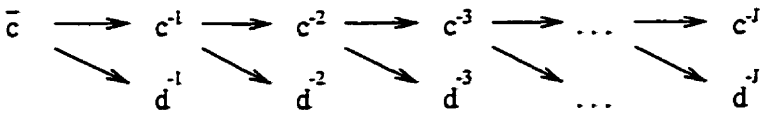
Defining the adjoint operators H^* and G^* by $\{H^*\mathbf{a}\}_n := \sum_k h_{n-2k} a_k$ and $\{G^*\mathbf{a}\}_n := \sum_k g_{n-2k} a_k$, (2.16) may be rewritten as

$$\mathbf{c}^{j+1} = H^* \mathbf{c}^j + G^* \mathbf{d}^j.$$

Hence we have a finite reconstruction algorithm. The pyramid nature of these schemes can be seen in Figure 2.5.

The next chapter extends the wavelet representation of vectors to that of operators.

Decomposition



Recomposition

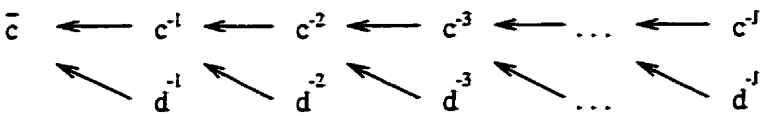


Figure 2.5: Visualisation of the Mallat algorithms.

Chapter 3

Representation of Operators

In this chapter we consider the representation of operators in bases of compactly supported wavelets, as discussed in [5], [7], [8], [9], [10], and [28]. We begin with the *standard form* (s-form) in Section 3.1, and proceed to the *non-standard form* (ns-form) in Section 3.2. Section 3.3 discusses the representation of differential operators $\partial_x^p \equiv d^p/dx^p$. Finally, in Section 3.4 we consider the ns-form of analytic functions of the operator ∂_x.

3.1 The Standard Form

Following [6] and [8], we are concerned with the representation of Calderón-Zygmund and pseudo-differential operators, defined below.

Definition 3.1 ([27, p. 47]) *A* **standard kernel** *is a continuous function* $K : \Delta^c \to \mathrm{C}^0(\mathbf{R})$, *where* $\Delta^c := \{(x,y) \in \mathbf{R}^2 : x \neq y\}$, *for which there exists a constant* $C > 0$ *such that for all* $(x,y) \in \Delta^c$ *and some* $n \geq 1$,

$$|K(x,y)| \leq \frac{C}{|x-y|^n}$$

and

$$|\partial_x K(x,y)| + |\partial_y K(x,y)| \leq \frac{C}{1+|x-y|^{n+1}}.$$

Definition 3.2 ([27, p. 48]) *An operator* $T : \mathrm{C}^\infty(\mathbf{R}) \to \mathbf{L}^1(\mathbf{R})$ *is called a* **Calderón-Zygmund operator** *if and only if:*

(1) T extends to a bounded linear operator on $\mathbf{L}^2(\mathbf{R})$, and

(2) there exists a standard kernel K such that for every $f \in \mathbf{L}^\infty(\mathbf{R})$,

$$Tf(x) = \int K(x,y) f(y)\, dy$$

almost everywhere on the support of f.

A pseudo-differential operator is a generalization of differential operators and is defined by a formula of the form

$$Tf(x) = \int e^{2\pi i x\xi} \sigma(x,\xi) \hat{f}(\xi)\, d\xi,$$

for T the operator and $\sigma(x,\xi)$, the *symbol* of T, a polynomial in ξ whose coefficients are functions in x. A pseudo-differential operator is in fact a Calderón-Zygmund operator under certain boundedness conditions on the symbol $\sigma(x,\xi)$, as given in the theorem on page 75 of [27].

As stated in [8, p. 160], we additionally require the *weak cancelation condition*, namely

$$\left| \int_{I \times I} K(x,y)\, dx\, dy \right| \leq C\, |I|\,,$$

for all *dyadic intervals* I. (A dyadic interval $I_{j,k}$ is of the form $I_{j,k} := [2^{-j}(k-1), 2^{-j}k]$.)

By property (1) of Definition 3.2 and the nesting sub-spaces property of M.R.A., we may write T as a *telescopic series* (see, for instance, [5, p. 145]) giving

$$T = \sum_{m \in \mathbf{Z}} \left(Q_m T Q_m + \sum_{m' \leq m-1} (Q_{m'} T Q_m + Q_m T Q_{m'}) \right). \tag{3.1}$$

Projecting $K(x,y)$ onto a basis generated by the tensor product of wavelets (*i.e.*, by $\psi_{j,k}\psi_{j',k'}$) gives

$$K(x,y) = \sum_{j,k} \sum_{j',k'} \alpha_{jj'kk'} \psi_{j,k}(x) \psi_{j',k'}(y),$$

where

$$\alpha_{jj'kk'} := \langle K(\cdot,\cdot), \psi_{j,k}(\cdot)\psi_{j',k'}(\cdot)\rangle = \int_{\mathbf{R}}\int_{\mathbf{R}} \psi_{j,k}(x)K(x,y)\psi_{j',k'}(y)\,dx\,dy.$$

Then for $f \in \mathbf{L}^2(\mathbf{R})$, as $\oplus_j W_j = \mathbf{L}^2(\mathbf{R})$ (see Section 2.2) so that $f(x) = \sum_{j,k} f_{jk}\psi_{j,k}$, and assuming that the interchanges of summation and integration are justified (see the discussions in Sections 4.1 and 4.5 of [8]), we have

$$\begin{aligned} Tf(x) &= \int_{\mathbf{R}} \sum_{j,k}\sum_{j',k'} \alpha_{jj'kk'}\psi_{j,k}(x)\psi_{j',k'}(y)\sum_{s,t} f_{st}\psi_{s,t}(y)\,dy \\ &= \sum_{s,t}\sum_{j,k}\sum_{j',k'} \alpha_{jj'kk'} f_{st}\psi_{j,k}(x)\int_{\mathbf{R}} \psi_{j',k'}(y)\psi_{s,t}(y)\,dy \\ &= \sum_{j,k}\sum_{j',k'} \alpha_{jj'kk'} f_{j'k'}\psi_{j,k}(x). \end{aligned} \tag{3.2}$$

Now the projection of f onto subspace W_m is given by $Q_m f(x) = \sum_k f_{mk}\psi_{m,k}(x)$, where $f_{mk} = \langle f, \psi_{m,k}\rangle = \int_{\mathbf{R}} f(x)\psi_{m,k}(x)\,dx$. Then a calculation similar to (3.2) yields

$$\begin{aligned} T(Q_m f)(x) &= \int_{\mathbf{R}} \sum_{l}\sum_{j,k}\sum_{j',k'} \alpha_{jj'kk'}\psi_{j,k}(x)\psi_{j',k'}(y) f_{ml}\psi_{m,l}(y)\,dy \\ &= \sum_{l}\sum_{j,k}\sum_{j',k'} \alpha_{jj'kk'} f_{ml}\psi_{j,k}(x)\int_{\mathbf{R}} \psi_{j',k'}(y)\psi_{m,l}(y)\,dy \\ &= \sum_{j,k}\sum_{k'} \alpha_{jmkk'} f_{mk'}\psi_{j,k}(x). \end{aligned}$$

Applying the Q_m projection again gives

$$\begin{aligned} Q_m(TQ_m f)(x) &= \sum_n \int_{\mathbf{R}} \left(\sum_{j,k}\sum_{k'} \alpha_{jmkk'} f_{mk'}\psi_{j,k}(y)\right)\psi_{m,n}(y)\,dy\,\psi_{m,n}(x) \\ &= \sum_n\sum_{j,k}\sum_{k'} \alpha_{jmkk'} f_{mk'}\psi_{m,n}(x)\int_{\mathbf{R}} \psi_{j,k}(y)\psi_{m,n}(y)\,dy \\ &= \sum_n\sum_{k'} \alpha_{mmnk'} f_{mk'}\psi_{m,n}(x). \end{aligned}$$

This implies that

$$\left(Q_m T Q_m\right)(f)(x) = \sum_k \left(\sum_j \alpha^m_{k,j} f_{mj}\right)\psi_{m,k}(x),$$

where $\alpha_{kj}^m := \alpha_{mmkj}$, so we have a wavelet expansion with respect to $\psi_{m,k}$ (*i.e.*, in W_m), with coefficients given by $\sum_j \alpha_{kj}^m f_{mj}$.

Similarly for

$$\begin{aligned} \beta_{mm'jk} &:= \langle K(\cdot,\cdot), \psi_{m,j}(\cdot)\psi_{m',k}(\cdot)\rangle \quad \text{and} \\ \gamma_{m'mjk} &:= \langle K(\cdot,\cdot), \psi_{m',j}(\cdot)\psi_{m,k}(\cdot)\rangle\,, \end{aligned}$$

we find

$$\begin{aligned} \left(Q_m T Q_{m'}\right)(f)(x) &= \sum_k \left(\sum_j \beta_{mm'jk} f_{m'j}\right) \psi_{m,k}(x) \quad \text{and} \\ \left(Q_{m'} T Q_m\right)(f)(x) &= \sum_k \left(\sum_j \gamma_{m'mjk} f_{mj}\right) \psi_{m',k}(x). \end{aligned}$$

Hence we have all of the components of the expansion (3.1).

Define $\mathbf{u}^m := \{\langle f, \psi_{m,n}\rangle\}_n$ (the coefficients from the projection of f onto W_m) and operators

$$\begin{aligned} A_m &:= Q_m T Q_m &&: W_m \to W_m, \\ B_m^{m'} &:= Q_m T Q_{m'} &&: W_{m'} \to W_m, \quad \text{and} \\ \Gamma_m^{m'} &:= Q_{m'} T Q_m &&: W_m \to W_{m'}. \end{aligned} \tag{3.3}$$

The elements of the matrices representing these operators, with respect to the bases $\{\psi_{m,j}\}_j$ and $\{\psi_{m',j}\}_j$, are given by

$$\begin{aligned} [A_m]_{jk} &:= \alpha_{jk}^m &&= \int_{\mathbf{R}}\int_{\mathbf{R}} \psi_{m,j}(x) K(x,y) \psi_{m,k}(y)\, dx\, dy, \\ \left[B_m^{m'}\right]_{jk} &:= \beta_{jk}^{m'm} &&= \int_{\mathbf{R}}\int_{\mathbf{R}} \psi_{m',j}(x) K(x,y) \psi_{m,k}(y)\, dx\, dy, \quad \text{and} \\ \left[\Gamma_m^{m'}\right]_{jk} &:= \gamma_{jk}^{mm'} &&= \int_{\mathbf{R}}\int_{\mathbf{R}} \psi_{m,j}(x) K(x,y) \psi_{m',k}(y)\, dx\, dy. \end{aligned}$$

So following (3.1), applying the s-form of T to a vector $\mathbf{u}$ results in an equation of the form

$$\begin{bmatrix} \ddots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & A_{m+1} & B^{m}_{m+1} & B^{m-1}_{m+1} & B^{m-2}_{m+1} & \cdots \\ \cdots & \Gamma^{m}_{m+1} & A_m & B^{m-1}_{m} & B^{m-2}_{m} & \cdots \\ \cdots & \Gamma^{m-1}_{m+1} & \Gamma^{m-1}_{m} & A_{m-1} & B^{m-2}_{m-1} & \cdots \\ \cdots & \Gamma^{m-2}_{m+1} & \Gamma^{m-2}_{m} & \Gamma^{m-2}_{m-1} & A_{m-2} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ \mathbf{u}^{m+1} \\ \mathbf{u}^{m} \\ \mathbf{u}^{m-1} \\ \mathbf{u}^{m-2} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \mu^{m+1} \\ \mu^{m} \\ \mu^{m-1} \\ \mu^{m-2} \\ \vdots \end{bmatrix}, \tag{3.4}$$

for μ the resultant vector. Thus the sub-vectors μ^m are given by

$$\mu^m = \cdots + \underbrace{\Gamma^{m}_{m+1}\mathbf{u}^{m+1}}_{\in W_m} + \underbrace{A_m\mathbf{u}^{m}}_{\in W_m} + \underbrace{B^{m-1}_{m}\mathbf{u}^{m-1}}_{\in W_m} + \cdots. \tag{3.5}$$

By (3.5) one can see that application of the wavelet-projection of the operator T to the vector $\mathbf{u}$ of coefficients of the wavelet expansion of a function also produces a vector of wavelet expansion coefficients μ (*i.e.*, we can multiply Mallat-projected vectors directly). Hence T may be applied within a wavelet basis without extra computation. This is a considerable advantage, as it is therefore unnecessary to convert into a physical domain to apply the operator, and then convert back to the frequency domain.

In practice we do not use the infinite expansion of (3.1) (which gives the bi-infinite matrix in (3.4)). For finitely many scales $m = -J, \ldots, 0$, where $-J$ corresponds to the coarsest scale and 0 to the finest, (3.1) becomes

$$\begin{aligned} T \approx T_0 &:= Q_0 T Q_0 \\ &= \sum_{m=-J}^{-1} \left(Q_m T Q_m + \sum_{m'=-J}^{m-1} (Q_{m'} T Q_m + Q_m T Q_{m'}) \right) + T_{-J}. \end{aligned} \tag{3.6}$$

This results in a block matrix similar to that given in (3.4), but with finitely many sub-matrices, and a small block matrix corresponding to $T_{-J} = P_{-J}TP_{-J}$ in the lower right corner ([5], [9], [10], [28]).

The standard form is used in Section 4.3 in the solution of ordinary differential equations.

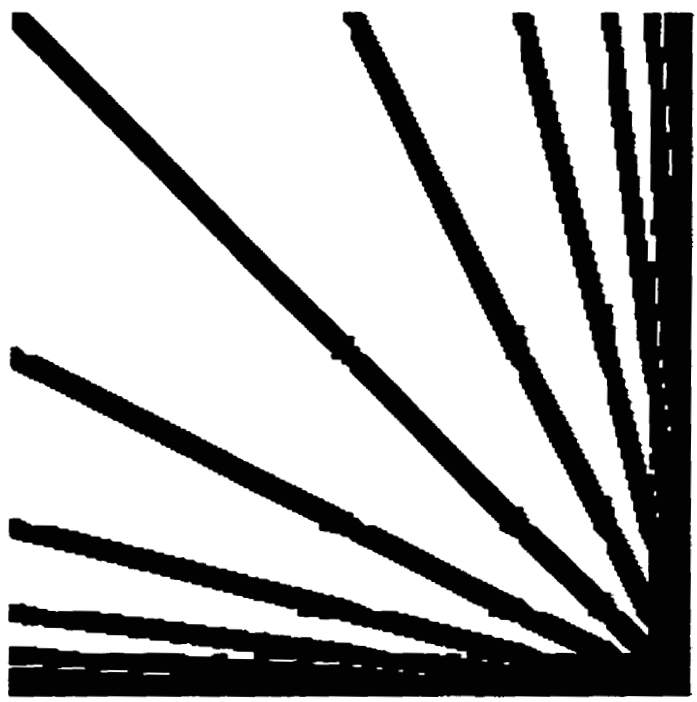


Figure 3.1: An example of the "fingers" present in the s-form.

While the standard form has the advantage of direct application in wavelet bases, it is relatively dense overall due to scale interactions. This scale interaction is seen as banding in each of the sub-blocks, manifested as "fingers" in the matrix. An example is presented in Figure 3.1. The result is slow matrix-vector multiplication, as the relative density requires a large number of operations. Further, as indicated in [5], the s-form

does not reproduce convolution operators. That is to say, the s-form of a convolution operator is not itself a convolution operator. A sparser matrix representation that does reproduce convolutions is obtained from the non-standard form.

3.2 The Non-Standard Form

We may avoid the problems introduced by scale interactions in the standard form by using the non-standard form (briefly, the *ns-form*). Recall from Section 2.2 that since $V_{j+1} = V_j \oplus W_j$, we have

$$V_i \supset W_j \quad \text{for} \quad i > j. \tag{3.7}$$

Hence for $P_m : f \to V_m$ we expand T as the telescopic series

$$\begin{aligned} T &= \sum_m (P_{m+1}TP_{m+1} - P_mTP_m) \\ &= \sum_m ((P_{m+1} - P_m)T(P_{m+1} - P_m) + (P_{m+1} - P_m)TP_m + P_mT(P_{m+1} - P_m)) \\ &= \sum_m (Q_mTQ_m + Q_mTP_m + P_mTQ_m) \end{aligned} \tag{3.8}$$

(as $Q_m = P_{m+1} - P_m$), or for finitely many scales $m = -J, \ldots, 0$,

$$T \approx T_0 := \sum_{m=-J}^{-1} (Q_mTQ_m + Q_mTP_m + P_mTQ_m) + P_{-J}TP_{-J}. \tag{3.9}$$

Due to (3.7), we have the s-form implicitly, but the scales are decoupled. In other words, by the nesting of subspaces, the interaction between W_j spaces is still implicitly represented but without explicit values for scale coupling (the $\beta_{jk}^{m'm}$ and $\gamma_{jk}^{mm'}$). Based on (3.8) and (3.9) we construct the operators analogous to (3.3) as

$$\begin{aligned} A_m &:= Q_mTQ_m &&: W_m \to W_m, \\ B_m &:= Q_mTP_m &&: V_m \to W_m, \quad \text{and} \\ \Gamma_m &:= P_mTQ_m &&: W_m \to V_m, \end{aligned}$$

where the entries for the matrix representing A_m are the same as for the s-form, and those for B_m, Γ_m, and $P_{-J}TP_{-J}$ are given by

$$\begin{aligned}
[B_m]_{jk} &:= \beta_{jk}^m = \int_{\mathbf{R}}\int_{\mathbf{R}} \psi_{m,j}(x)K(x,y)\phi_{m,k}(y)\,dx\,dy, \\
[\Gamma_m]_{jk} &:= \gamma_{jk}^m = \int_{\mathbf{R}}\int_{\mathbf{R}} \phi_{m,j}(x)K(x,y)\psi_{m,k}(y)\,dx\,dy, \quad \text{and} \\
[P_{-J}TP_{-J}]_{jk} &:= \sigma_{jk}^{-J} = \int_{\mathbf{R}}\int_{\mathbf{R}} \psi_{-J,j}(x)K(x,y)\psi_{-J,k}(y)\,dx\,dy.
\end{aligned}$$

In practice we use finitely many scales. In order to remain in a finite wavelet basis (via the finite Mallat expansion, see Section 2.4), for a function f we need

$$T_0 f(x) = \sum_{m=-J}^{-1} \sum_{k\in\mathbb{F}_{2^{n+m}}} \widehat{d_k^m}\psi_{m,k}(x) + \sum_{k\in\mathbb{F}_{2^{n-J}}} \widehat{s_k^{-J}}\phi_{-J,k}(x), \tag{3.10}$$

where V_0 is of dimension 2^n, $n \geq J$, $\mathbb{F}_{2^{n+m}} = \{0,\ldots,2^{n+m}-1\}$, and $\widehat{d_k^m}$ and $\widehat{s_k^{-J}}$ are some set of Mallat expansion coefficients. Unfortunately the ns-form does not produce such an expansion. We must in fact apply the ns-form to a vector composed of all of the M.R.A. coefficients of f, namely $\{\mathrm{d}^m,\mathrm{s}^m\}_{m=-J}^0$, and not just $\left\{\{\mathrm{d}^m\}_{m=-J}^0,\mathrm{s}^{-J}\right\}$. To make things worse, the result of applying this form does not directly produce a standard Mallat expansion (Mallat wavelet decomposition), as explicitly seen in the expansion:

$$\begin{aligned}
T_0 f(x) &= \sum_{m=-J}^{-1} \sum_{j\in\mathbb{F}_{2^{n+m}}} \sum_{k\in\mathbb{F}_{2^{n+m}}} \left(\alpha_{jk}^m d_k^m \psi_{m,k}(x) + \beta_{jk}^m s_k^m \psi_{m,k}(x) + \gamma_{jk}^m d_k^m \phi_{j,k}(x)\right) \\
&\qquad + \sum_{k\in\mathbb{F}_{2^{n+m}}} \sigma_{jk}^{-J} s_k^{-J} \phi_{-J,k}(x) \\
&= \sum_{m=-J}^{-1} \sum_{j\in\mathbb{F}_{2^{n+m}}} \sum_{k\in\mathbb{F}_{2^{n+m}}} \left(\alpha_{jk}^m d_k^m + \beta_{jk}^m s_k^m\right)\psi_{m,k}(x) \\
&\qquad + \sum_{m=-J+1}^{-1} \sum_{j\in\mathbb{F}_{2^{n+m}}} \sum_{k\in\mathbb{F}_{2^{n+m}}} \gamma_{jk}^m d_k^m \phi_{m,k}(x) \\
&\qquad + \sum_{j\in\mathbb{F}_{2^{n-J}}} \sum_{k\in\mathbb{F}_{2^{n-J}}} \left(\gamma_{jk}^{-J} d_k^{-J} + \sigma_{jk}^{-J} s_k^{-J}\right)\phi_{-J,k}(x) \\
&= \sum_{m=-J}^{-1} \sum_{k\in\mathbb{F}_{2^{n+m}}} \varrho_k^m \psi_{m,k}(x) + \zeta_k^m \phi_{m,k}(x),
\end{aligned} \tag{3.11}$$

where

$$\varrho_k^m := \sum_{j \in \mathbb{F}_{2^{n+m}}} \left(\alpha_{jk}^m d_k^m + \beta_{jk}^m s_k^m \right),$$

$$\zeta_k^m := \begin{cases} \displaystyle\sum_{j \in \mathbb{F}_{2^{n+m}}} \gamma_{jk}^m d_k^m & \text{if } m = -J+1, \ldots, -1 \\ \displaystyle\sum_{j \in \mathbb{F}_{2^{n-J}}} \left(\gamma_{jk}^{-J} d_k^{-J} + \sigma_{jk}^{-J} s_k^{-J} \right) & \text{if } m = -J. \end{cases}$$

An inspection of these formulas reveals that we have a matrix equation (*cf.* Figure 3.2) of the form

$$\begin{bmatrix} A_{-1} & B_{-1} & 0 & 0 & 0 & 0 & 0 \\ \Gamma_{-1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & A_{-2} & B_{-2} & 0 & 0 & 0 \\ 0 & 0 & \Gamma_{m-2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & A_{-J} & B_{-J} \\ 0 & 0 & 0 & 0 & 0 & \Gamma_{-J} & T_{-J} \end{bmatrix} \begin{bmatrix} \mathbf{d}^{-1} \\ \mathbf{s}^{-1} \\ \mathbf{d}^{-2} \\ \mathbf{s}^{-2} \\ \vdots \\ \mathbf{d}^{-J} \\ \mathbf{s}^{-J} \end{bmatrix} = \begin{bmatrix} \varrho^{-1} \\ \zeta^{-1} \\ \varrho^{-2} \\ \zeta^{-2} \\ \vdots \\ \varrho^{-J} \\ \zeta^{-J} \end{bmatrix} \tag{3.12}$$

(ϱ^m is the vector with entries ϱ_k^m, given above; similarly for ζ^m).

This result may be written as a Mallat expansion by rewriting (3.11) as

$$T_0 f(x) = \sum_{m=-J}^{-1} \sum_{k \in \mathbb{F}_{2^{n-m}}} \left(\varrho_k^m + \delta_k^m \right) \psi_{m,k}(x) + \sum_{k \in \mathbb{F}_{2^{n-J}}} \eta_k \phi_{-J,k}(x),$$

where we take

$$\delta_k^m = \left\langle \sum_{j=-J}^{-1} \sum_{l \in \mathbb{F}_{2^{n+m}}} \zeta_l^j \phi_{j,l}(\cdot),\, \psi_{m,k}(\cdot) \right\rangle \quad \text{and} \tag{3.13}$$

$$\eta_k = \left\langle \sum_{j=-J}^{-1} \sum_{l \in \mathbb{F}_{2^{n-J}}} \zeta_l^j \phi_{j,l}(\cdot),\, \phi_{-J,k}(\cdot) \right\rangle. \tag{3.14}$$

By the orthogonality of the $\phi_{j,k}$, (3.14) may be simplified to

$$\eta_k = \zeta_k^{-J}.$$

Simplifying (3.13) requires some more work, as follows:

$$\begin{aligned}
\delta_k^m &= \sum_{j=-J}^{-1} \sum_{l \in \mathbb{F}_{2^{n-J}}} \zeta_l^j \int_{\mathbf{R}} \phi_{j,l}(x) \psi_{m,k}(x)\, dx \\
&= 2^{m/2} \sum_{j=-J}^{-1} 2^{j/2} \sum_{l \in \mathbb{F}_{2^{n-J}}} \zeta_l^j \int_{\mathbf{R}} \phi(2^j x - l) \psi(2^m x - k)\, dx \\
&= 2^{1/2} 2^{m/2} \sum_{j=-J}^{-1} 2^{j/2} \sum_{l \in \mathbb{F}_{2^{n-j}}} \zeta_l^j \sum_{n=0}^{L-1} g_n \\
&\qquad \times \int_{\mathbf{R}} \phi(2^j x - l) \phi(2^{m+1} x - 2k - n)\, dx \qquad (3.15) \\
&= \begin{cases} 0 & \text{if } j \neq m+1 \\ 2^{m+1} \sum_{l \in \mathbb{F}_{2^{n+m+1}}} \zeta_l^{m+1} \sum_{n=0}^{L-1} g_n & \\ \quad \times \int_{\mathbf{R}} \phi(2^{m+1} x - l) \phi(2^{m+1} x - 2k - n)\, dx & \text{if } j = m+1. \end{cases} \qquad (3.16)
\end{aligned}$$

Equation (3.15) results from the application of the two-scale relation (2.10), and (3.16) holds by orthogonality of the $\phi_{j,k}$. As $m = -J, \ldots, -1$ and $j = -J, \ldots, -1$, it follows that $\delta_k^{-1} = 0$. For $m < -1$ and $j = m+1$ we have

$$\begin{aligned}
\delta_k^m &= \sum_{l \in \mathbb{F}_{2^{n+m+1}}} \zeta_l^{m+1} \sum_{n=0}^{L-1} g_n \int_{\mathbf{R}} \phi(x - (l - 2k - n)) \phi(x)\, dx \\
&= \sum_{l \in \mathbb{F}_{2^{n+m+1}}} \zeta_l^{m+1} \tilde{g}_{l-2k}, \qquad (3.17)
\end{aligned}$$

where

$$\tilde{g}_{l-2k} = \begin{cases} g_{l-2k} & \text{if } 0 \leq l - 2k \leq L-1 \\ 0 & \text{else} \end{cases}$$

and (3.17) again holds by orthogonality of the $\phi_{j,k}$. Hence we have

$$\delta_k^m = \begin{cases} 0 & \text{if } m = -1 \\ \sum_{l \in \mathbb{F}_{2n+m+1}} \zeta_l^{m+1} \tilde{g}_{l-2k} & \text{if } m = -J, \ldots, -2. \end{cases}$$

So we may define $\widehat{d}_k^m := \varrho_k^m + \delta_k^m$ and $\widehat{s}_k^{-J} := \eta_k$, giving a finite Mallat wavelet expansion of the form (3.10) with coefficients $\left\{ \left\{ \widehat{\mathbf{d}}^m \right\}_{m=-J}^{-1}, \widehat{\mathbf{s}}^{-J} \right\}$.

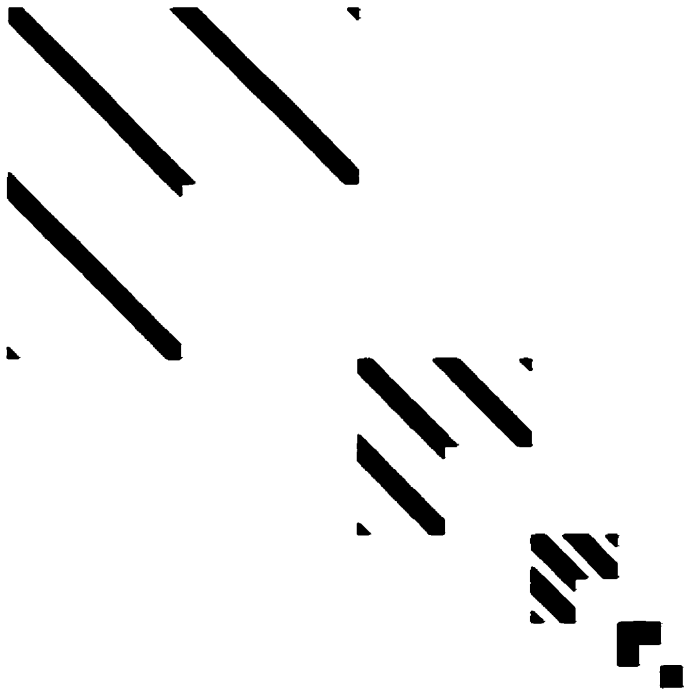


Figure 3.2: An example representation of a matrix in the ns-form.

The key advantage in using the non-standard form over the standard form is the sparseness of the representation of operators, as seen in the example in Figure 3.2. First we have the decoupling of scales, producing a block-banded matrix without fingers. Further, due the the relative losslessness of wavelet representations (*i.e.*, the ability to

reproduce information with little loss of detail, via Theorem 2.6), we may make the matrix sparser by using ϵ-thresholding with little effect on the result. The next two sections discuss the ns-form representation of differential operators. Chapter 5 makes full use of the ns-form for the solution of partial differential equations.

3.3 The ns-form Representation of Differential Operators

The majority of the operators considered in Chapter 5 are spatial differential operators, of the form $T = \partial_x^p \equiv d^p/dx^p$. As such operators do not have a kernel of integration (as in Definition 3.2), the process of determining the representation is slightly different, but the end result is as before.

For $Q_m f(x) := \sum_k f_{mk}\psi_{m,k}(x)$, we have

$$TQ_m f(x) = \frac{d^p}{dx^p} Q_m f(x) = \sum_k f_{mk}\psi^{(p)}_{m,k}(x),$$

so

$$\begin{aligned} Q_m T Q_m f(x) &= \sum_n \left\langle \sum_k f_{mk}\psi^{(p)}_{m,k}(\cdot), \psi_{m,n}(\cdot) \right\rangle \psi_{m,n}(x) \\ &= \sum_n \left(\sum_k \alpha^m_{nk} f_{mk} \right) \psi_{m,n}(x), \end{aligned}$$

where now

$$\alpha^m_{nk} = \int_{\mathbf{R}} \psi_{m,n}(x)\psi^{(p)}_{m,k}(x)\,dx.$$

Similarly, we have

$$\begin{aligned} \beta^m_{nk} &= \int_{\mathbf{R}} \psi_{m,n}(x)\phi^{(p)}_{m,k}(x)\,dx, \\ \gamma^m_{nk} &= \int_{\mathbf{R}} \phi_{m,n}(x)\psi^{(p)}_{m,k}(x)\,dx, \quad \text{and} \\ \sigma^m_{nk} &= \int_{\mathbf{R}} \phi_{m,n}(x)\phi^{(p)}_{m,k}(x)\,dx. \end{aligned} \tag{3.18}$$

Note that in general $B \neq \Gamma^T$ (whereas $B = \Gamma^T$ in Section 3.2).

The α_{nk}^m may be simplified as

$$\begin{aligned}
\alpha_{nk}^m &= \int_{\mathbf{R}} \frac{d^p}{dx^p}\left(2^{m/2}\psi(2^m x - k)\right) 2^{m/2}\psi(2^m x - n)\, dx \\
&= 2^{pm}\int_{\mathbf{R}} \psi^{(p)}(s-k)\psi(s-n)\, ds \\
&= 2^{pm}\int_{\mathbf{R}} \psi^{(p)}(v)\psi(v-(n-k))\, dv \\
&= 2^{pm}\alpha_{n-k,0}^0 \\
&= 2^{pm}\alpha_l^0, \qquad l = n-k.
\end{aligned}$$

Similarly, with $l = n - k$,

$$\begin{aligned}
\beta_{nk}^m &= 2^{pm}\beta_l^0, \\
\gamma_{nk}^m &= 2^{pm}\gamma_l^0, \quad \text{and} \\
\sigma_{nk}^m &= 2^{pm}\sigma_l^0.
\end{aligned}$$

It should be noted that we need only concern ourselves with $l = 0, \ldots, L-2$. This holds as $\text{supp}(\phi) = \text{supp}(\psi) = \text{supp}(\phi^{(p)}) = \text{supp}(\psi^{(p)}) = [0, L-1]$, so for $p \geq 1$,

$$\begin{aligned}
\int_{\mathbf{R}} \phi^{(p)}(x)\phi(x-l)\, dx &= \int_0^{L-1} \phi^{(p)}(x)\phi(x-l)\, dx \\
&= 0
\end{aligned}$$

for $l \leq 1 - L$ and $l \geq L - 1$. The same result holds for ψ.

Using the relations (2.9) and (2.10) we may further simplify the above results. For α_l^m,

$$\begin{aligned}
\alpha_l^m &= 2^{pm}\int_{\mathbf{R}} \psi^{(p)}(x)\psi(x-l)\, dx \\
&= 2^{pm}\int_{\mathbf{R}} \left(\sqrt{2}\sum_{k'=0}^{L-1} g_{k'}\frac{d^p}{dx^p}\left(\phi(2x-k')\right)\right)\left(\sqrt{2}\sum_{k=0}^{L-1} g_k\phi(2x-2l-k)\right) dx
\end{aligned}$$

$$
\begin{aligned}
&= 22^{mp}2^{p}\sum_{k=0}^{L-1}\sum_{k'=0}^{L-1} g_k g_{k'} \int_{\mathbf{R}} \phi^{(p)}(\nu)\phi(\nu+k'-2l-k)\frac{1}{2}\,d\nu \\
&= \sum_{k=0}^{L-1}\sum_{k'=0}^{L-1} g_k g_{k'} 2^{p(m+1)} \int_{\mathbf{R}} \phi^{(p)}(\nu)\phi(\nu+k'-2l-k)\,d\nu \\
&= \sum_{k=0}^{L-1}\sum_{k'=0}^{L-1} g_k g_{k'} \sigma^{m+1}_{2l+k-k'} \qquad (3.19) \\
&= 2^{p(m+1)}\sum_{k=0}^{L-1}\sum_{k'=0}^{L-1} g_k g_{k'} \sigma^{0}_{2l+k-k'}.
\end{aligned}
$$

Similarly for β_l^m, γ_l^m and σ_l^m:

$$
\begin{aligned}
\beta_l^m &= 2^{p(m+1)}\sum_{k=0}^{L-1}\sum_{k'=0}^{L-1} g_k h_{k'} \sigma^{0}_{2l+k-k'}, \\
\gamma_l^m &= 2^{p(m+1)}\sum_{k=0}^{L-1}\sum_{k'=0}^{L-1} h_k g_{k'} \sigma^{0}_{2l+k-k'}, \qquad \text{and} \\
\sigma_l^m &= 2^{p(m+1)}\sum_{k=0}^{L-1}\sum_{k'=0}^{L-1} h_k h_{k'} \sigma^{0}_{2l+k-k'}. \qquad (3.20)
\end{aligned}
$$

The relation between α^m and σ^{m+1} in (3.19) differs from that given in [9, p. 158], [10, p. 293], and [28, p. 153]. These papers erroneously state this relation with an additional factor of 2 appearing before the double summation.

By Parseval's relation (see [14]) we also have

$$
\begin{aligned}
\sigma_l^0 &= \left\langle \phi_{0,l}, \phi^{(p)} \right\rangle \\
&= \left\langle \widehat{\phi}_{0,l}, \widehat{\phi^{(p)}} \right\rangle \\
&= (-1)^p \sigma^0_{-l},
\end{aligned}
$$

by a straightforward calculation.

So we can define α, β, and γ in terms of σ. However we still need a way to find the σ_l^0. This is a problem of solving for connection coefficients, which in general are denoted $\Lambda_l^{d_1,d_2} := \int \phi_l^{(d_1)}(x)\phi^{(d_2)}(x)\,dx$; here $\sigma_l^0 \equiv \Lambda_l^{0,p}$. (An approach to solving for more general

$\Lambda_l^{d_1,d_2}$ is presented in Section 4.2.) We start by setting $m = 0$ in (3.20), then substitute $k' = k - n$. This gives

$$\begin{aligned}
\sigma_l^0 &= 2^p \sum_{k=0}^{L-1} \sum_{n=k-L+1}^{k} h_k h_{k-n} \sigma_{2l+n}^0 \\
&= 2^p \sum_{n=-L+1}^{-1} \sum_{k=0}^{n+L-1} h_k h_{k-n} \sigma_{2l+n}^0 + 2^p \sum_{k=0}^{L-1} h_k^2 \sigma_{2l}^0 \\
&\quad + 2^p \sum_{n=1}^{L-1} \sum_{k=n}^{L-1} h_k h_{k-n} \sigma_{2l+n}^0 \\
&= 2^p \sum_{m=-L+1}^{-1} \sigma_{2l+m}^0 \sum_{k=0}^{m+L-1} h_k h_{k-m} + 2^p \sigma_{2l}^0 \sum_{k=0}^{L-1} h_k^2 \\
&\quad + 2^p \sum_{m=1}^{L-1} \sigma_{2l+m}^0 \sum_{k=m}^{L-1} h_k h_{k-m}.
\end{aligned} \tag{3.21}$$

Additionally we have *quadrature mirror connection coefficients* a_n (*a.k.a.* autocorrelation coefficients) which are defined to be

$$\begin{aligned}
a_n &:= 2 \sum_{i=0}^{L-1-n} h_i h_{i+n} \\
&= 2 \sum_{j=n}^{L-1} h_{j-n} h_j.
\end{aligned}$$

Noting that L is always even for Coiflets (Section 2.3) and $a_n = 0$ for even $n > 0$ (see [5, p. 1720] and [18, p. 137]), for $2 - L \leq l \leq L - 2$ equation (3.21) becomes

$$\begin{aligned}
\sigma_l^0 &= 2^p \sum_{m=-L+1}^{-1} \sigma_{2l+m}^0 \frac{1}{2} a_{-m} + 2^p \sigma_{2l}^0 \sum_{k=0}^{L-1} h_k^2 \\
&\quad + 2^p \sum_{k=1}^{L-1} \sigma_{2l+m}^0 \frac{1}{2} a_m \\
&= 2^{p-1} \sum_{k=-L/2}^{-1} \sigma_{2l+2k+1}^0 a_{-2k-1} + 2^{p-1} \sum_{k=-(L-2)/2}^{-1} \sigma_{2l+2k}^0 a_{-2k} \\
&\quad + 2^p \sigma_{2l}^0 \sum_{k=0}^{L-1} h_k^2 + 2^{p-1} \sum_{k=1}^{L/2} \sigma_{2l+2k-1}^0 a_{2k-1}
\end{aligned}$$

$$+ 2^{p-1} \sum_{k=1}^{L-2/2} \sigma^0_{2l+2k} a_{2k}$$

$$= 2^p \sigma^0_{2l} + 2^{p-1} \sum_{k=1}^{L/2} a_{2k-1} \left(\sigma^0_{2l-2k+1} + \sigma^0_{2l+2k-1} \right), \tag{3.22}$$

as $\sum_{k=0}^{L-1} h_k^2 = 1$ (see equation (2.11)).

The result (3.22) does not guarantee a unique solution—we require one further equation, equivalent to a normalization condition ([5], [30], [37]). Define the moments of $\phi(x)$ as

$$\mathcal{M}_l^\phi := \int_{\mathbf{R}} \phi(x) x^l \, dx.$$

By [5] and Theorem 4 of [34, p. 33] it can then be shown that

$$\sum_{k \in \mathbf{Z}} k^m \phi(x-k) = x^m + \sum_{l=1}^{m} (-1)^l \binom{m}{l} \mathcal{M}_l^\phi x^{m-l},$$

which allows us to write

$$\sum_{l=-L+2}^{L-2} l^p \sigma_l^0 = (-1)^p p!. \tag{3.23}$$

The solution to (3.22) and (3.23) provides a unique set of σ_l^0, which in turn may be used to determine the α, β and γ entries quickly and efficiently.

We extend this result a little further in the next section.

3.4 Analytic Functions of Differential Operators

Here we consider the ns-form representation of analytic, $\mathbf{L}^2(\mathbf{R})$ functions of the differential operators ∂_x. The use of such projections is of prime importance in Chapter 5.

There are two approaches, both valid due to the analyticity of the function f, used in [9], [10], and [28] to approximate $f(\partial_x)$:

$$(1) \quad P_0 f(\partial_x) P_0 \quad \text{and} \tag{3.24}$$

$$(2) \quad f(P_0 \partial_x P_0). \tag{3.25}$$

The choice of (3.24) versus (3.25) depends on the behaviour of a cutoff function found in the expansion of (3.24) (see equation (3.28) and the subsequent discussion below).

First we consider the representation given by (3.24), $P_0 f(\partial_x) P_0$. From Sections 3.2 and 3.3, we must find $\sigma^m_{kk'}$ of the form

$$\sigma^m_{kk'} = 2^m \int_{\mathbf{R}} \phi(2^m x - k) f(\partial_x) \phi(2^m x - k')\, dx.$$

Define the Fourier and inverse Fourier transforms as

$$\begin{aligned} (\mathcal{F}f)(z) &= \widehat{f}(z) = \int_{\mathbf{R}} f(x) e^{-2\pi \imath x z}\, dx \quad \text{and} \\ (\mathcal{F}^{-1}\widehat{f})(x) &= f(x) = \int_{\mathbf{R}} \widehat{f}(z) e^{2\pi \imath z x}\, dz. \end{aligned}$$

For f analytic and $f, g \in \mathbf{L}^2(\mathbf{R})$, [9, p. 160] uses the relation $\mathcal{F}(f(\partial_x) g(x))(\xi) = f(2\pi\imath\xi)\widehat{g}(\xi)$ (this can be shown by first considering f a polynomial and then extending this result to the space of analytic functions by Taylor's Theorem). Hence

$$\mathcal{F}\left(f(\partial_x)(\phi(2^m x - k'))\right)(\xi) = f(2\pi\imath\xi) e^{-2\pi\imath k'\xi} 2^{-m} \widehat{\phi}(2^{-m}\xi),$$

which implies

$$f(\partial_x)\phi(2^m x - k') = \int_{\mathbf{R}} 2^{-m} f(2^m 2\pi\imath\xi) e^{-2\pi\imath 2^{-m} k'\xi} \widehat{\phi}(2^{-m}\xi) e^{2\pi\imath x\xi}\, d\xi. \tag{3.26}$$

We may then rewrite $\sigma^m_{kk'}$ in terms of (3.26) (because $\widehat{\phi} f \in \mathbf{L}^1$, Fubini's theorem applies) as

$$\begin{aligned} \sigma^m_{kk'} &= 2^m \int_{\mathbf{R}} \phi(2^m x - k) \int_{\mathbf{R}} f(2^m 2\pi\imath\xi) e^{-2\pi\imath 2^{-m} k'\xi} \widehat{\phi}(2^{-m}\xi) e^{2\pi\imath x\xi}\, d\xi\, dx \\ &= 2^m \int_{\mathbf{R}} \phi(2^m x - k) \int_{\mathbf{R}} f(2^{2m} 2\pi\imath\eta) e^{-2\pi\imath k'\eta} \widehat{\phi}(\eta) e^{2\pi\imath 2^m x\eta}\, d\eta\, dx \\ &= 2^m 2^{-m} \int_{\mathbf{R}} \int_{\mathbf{R}} \phi(\gamma) \widehat{\phi}(\eta) e^{-2\pi\imath k'\eta} e^{2\pi\imath(\gamma + k)\eta}\, d\gamma f(2^{2m} 2\pi\imath\eta)\, d\eta \end{aligned}$$

$$= \int_{\mathbf{R}} \left(\int_{\mathbf{R}} \phi(\gamma) e^{2\pi\imath\gamma\eta}\, d\gamma \right) \widehat{\phi}(\eta) e^{2\pi\imath(k-k')\eta} f(2^{2m} 2\pi\imath\eta)\, d\eta$$
$$= \int_{\mathbf{R}} \overline{\widehat{\phi}(\eta)} \widehat{\phi}(\eta) e^{2\pi\imath(k-k')\eta} f(2^{2m} 2\pi\imath\eta)\, d\eta.$$

Setting $l = k - k'$ gives us

$$\begin{aligned} \sigma_l^m &= \int_{\mathbf{R}} f(2^{2m} 2\pi\imath\eta) \left|\widehat{\phi}(\eta)\right|^2 e^{2\pi\imath l\eta}\, d\eta \\ &= \int_0^1 \sum_{j\in\mathbf{Z}} f(2^{2m} 2\pi\imath(\eta+j)) \left|\widehat{\phi}(\eta+j)\right|^2 e^{2\pi\imath l(\eta+j)}\, d\eta \\ &= \int_0^1 g(\eta) e^{2\pi\imath l\eta}\, d\eta, \end{aligned} \tag{3.27}$$

where we define $g(\eta)$ to be

$$g(\eta) := \sum_{j\in\mathbf{Z}} f(2^{2m} 2\pi\imath(\eta+j)) \left|\widehat{\phi}(\eta+j)\right|^2. \tag{3.28}$$

As a consequence of the Riemann-Lebesgue lemma we have for every $\varepsilon > 0$ there exists $K \in \mathbf{Z}, K > 0$, such that $\left|\widehat{\phi}(\eta)\right|^2 < \varepsilon$ for $|\eta| < K$. Hence we may use $\left|\widehat{\phi}(\eta)\right|^2$ as a *cutoff function*, to obtain an estimate $\widetilde{g}$ of g:

$$\widetilde{g}(\eta) := \sum_{j=-K}^{K} f(2^{2m} 2\pi\imath(\eta+j)) \left|\widehat{\phi}(\eta+j)\right|^2. \tag{3.29}$$

which gives us an approximation to (3.27) as

$$\widetilde{\sigma}_l^m := \frac{1}{N} \sum_{n=0}^{N-1} \widetilde{g}(\eta_n) e^{2\pi\imath l\eta_n} \quad \text{where } \eta_n = \frac{n}{N-1}. \tag{3.30}$$

Here N is the number of quadrature points used in approximating the integral. Hence by (3.30), $\widetilde{\sigma}_l^m$ may be obtained by an inverse fast Fourier transform of the $\widetilde{g}(\eta_n)$ sequence. Use of (3.24) via (3.29) is reasonable if K is relatively small and hence abbreviates the calculation of (3.29).

We now proceed to discuss the approach of (3.25), representing $f(\partial_x)$ by $f(P_0\partial_x P_0)$. Begin by considering the bi-infinite Toeplitz matrix M representing the operator $P_0\partial_x P_0$ (see (3.18)) given by

$$M = \begin{bmatrix} \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \cdots & 0 & \sigma_{L-2} & \cdots & \sigma_1 & \sigma_0 & \sigma_{-1} & \cdots & \sigma_{-L+2} & 0 & 0 & \cdots \\ \cdots & 0 & 0 & \sigma_{L-2} & \cdots & \sigma_1 & \sigma_0 & \sigma_{-1} & \cdots & \sigma_{-L+2} & 0 & \cdots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}.$$

Then M is a convolution matrix, as $(M\mathbf{c})_i = (\sum_j M_{ij}c_j)_i = (\sum_j \sigma_{i-j}c_j)_i =: (\sigma * \mathbf{c})_i$. It can be shown that for $\mathbf{v}_k$ defined as

$$\mathbf{v}_k = \begin{bmatrix} \vdots \\ e^{\frac{6k\pi\imath}{2L-3}} \\ e^{\frac{4k\pi\imath}{2L-3}} \\ e^{\frac{2k\pi\imath}{2L-3}} \\ 1 \\ e^{-\frac{2k\pi\imath}{2L-3}} \\ e^{-\frac{4k\pi\imath}{2L-3}} \\ e^{-\frac{6k\pi\imath}{2L-3}} \\ \vdots \end{bmatrix}, \tag{3.31}$$

one has $M\mathbf{v}_k = \lambda_k\mathbf{v}_k$ for

$$\lambda_k = \sigma_0 + \sum_{j=1}^{L-2}\left(\sigma_j e^{2\pi\imath kj/(2L-3)} + \sigma_{-j}e^{-2\pi\imath kj/(2L-3)}\right), \quad k = 1, \ldots, 2L-3. \tag{3.32}$$

Hence we have a set of eigenvalues λ_k and eigenvectors $\mathbf{v}_k$ of M. We wish to use this result in the same manner as used in [9], [10], [28], and [43], a full theorem and proof for which we give below.

Theorem 3.3 *For M a discrete convolution operator, Λ the diagonal eigenvalue matrix of M, f a function analytic in a neighbourhood of the spectrum of M, and $\mathcal{F}$ the discrete Fourier transform, we have that*

$$f(M) \equiv \mathcal{F} f(\Lambda) \mathcal{F}^{-1}.$$

Proof: We begin by giving the definition of the discrete Fourier transform for a vector of length N as

$$\begin{aligned} (\mathcal{F}f)_n &= \widehat{f}_n = \sum_{j=0}^{N-1} f_j e^{-2\pi \imath nj/N} \quad \text{and} \\ (\mathcal{F}^{-1}\widehat{f})_j &= f_j = \frac{1}{N} \sum_{n=0}^{N-1} \widehat{f}_n e^{2\pi \imath jn/N}. \end{aligned}$$

First we consider $f = I$, the identity, so that $f(M) = M$. For any vector b of length $N = 2L - 3$ and making use of (3.32) we have

$$\begin{aligned} \mathcal{F}\Lambda\mathcal{F}^{-1}\mathbf{b} &= \mathcal{F}\Lambda \left\{ \frac{1}{N} \sum_{n=-L-2}^{L-2} b_n e^{2\pi \imath nk/N} \right\}_k \\ &= \mathcal{F} \left\{ \frac{1}{N} \sum_{n=-L+2}^{L-2} \lambda_k b_n e^{2\pi \imath nk/N} \right\}_k \\ &= \left\{ \sum_{s=-L+2}^{L-2} \left(\frac{1}{N} \sum_{n=-L+2}^{L-2} \lambda_s b_n e^{2\pi \imath ns/N} \right) e^{-2\pi \imath sj/N} \right\}_j \\ &= \left\{ \sum_{s=-L+2}^{L-2} \sum_{n=-L+2}^{L-2} b_n e^{2\pi \imath ns/N} e^{-2\pi \imath sj/N} \sum_{t=-L+2}^{L-2} \sigma_t e^{2\pi \imath st/N} \right\}_j \\ &= \left\{ \frac{1}{N} \sum_{t=-L+2}^{L-2} \sigma_t \sum_{n=-L+2}^{L-2} b_n \sum_{s=-L+2}^{L-2} e^{2\pi \imath s(n-(j-t))/N} \right\}_j \\ &= \left\{ \frac{1}{N} \sum_{t=-L+2}^{L-2} \sigma_t \sum_{n=-L+2}^{L-2} b_n \sum_{s=-L+2}^{L-2} \omega^{s(n-(j-t))} \right\}_j \\ &= \left\{ \frac{1}{N} \sum_{t=-L+2}^{L-2} \sigma_t \sum_{n=-L+2}^{L-2} b_n \sum_{s=-L+2}^{L-2} \mu_{njt}^s \right\}_j, \end{aligned} \tag{3.33}$$

where $\omega := e^{2\pi\imath/N}$ and $\mu_{njt} := \omega^{n-(j-t)}$.

For $n-(j-t) \neq 0$,

$$\begin{aligned}\sum_{s=-L+2}^{L-2} \mu_{njt}^{s} &= \mu_{njt}^{-L+2}\left(1+\cdots+\mu_{njt}^{2L-4}\right) \\ &= \mu_{njt}^{-L+2}\left(\frac{\mu_{njt}^{2L-3}-1}{\mu_{njt}-1}\right) \\ &= 0,\end{aligned}$$

as $N = 2L-3$ implies that $\mu_{njt}^{2L-3} = e^{2\pi\imath} = 1$.

For $n-(j-t) = 0$,

$$\sum_{s=-L+2}^{L-2} \mu_{njt}^{s} = \sum_{s=-L+2}^{L-2} 1 = 2L-3 = N.$$

So we have that (3.33) gives

$$\mathcal{F}\Lambda\mathcal{F}^{-1}\mathbf{b} = \left\{\sum_{t=-L+2}^{L-2} \sigma_t b_{j-t}\right\}_j = M\mathbf{b}. \tag{3.34}$$

Thus for $f = I$, we have $f(M) = M = \mathcal{F}\Lambda\mathcal{F}^{-1}$.

Now we consider a general functions f, analytic in a neighbourhood of the spectrum of M. As f is analytic, we may use the Taylor expansion and the usual definition of analytic functions of a matrix to obtain

$$\begin{aligned}f(M) &= \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} M^i, \\ f(\Lambda) &= \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} \Lambda^i \\ &= \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} \operatorname{diag}\left(\left\{\lambda_k^i\right\}_k\right) \\ &= \operatorname{diag}\left(\sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} (\lambda_k)^i\right) \\ &= \operatorname{diag}(f(\lambda_k)),\end{aligned}$$

where "diag" refers to the diagonal matrix.

Also

$$f(M)\mathbf{b} = I\mathbf{b} + M\mathbf{b} + \sum_{i=2}^{\infty} \frac{f^{(i)}(0)}{i!} M^{i-1} M\mathbf{b}.$$

Due to (3.34) we have

$$\begin{aligned} M^{i-1}M\mathbf{b} &= M^{i-1}\mathcal{F}\Lambda\mathcal{F}^{-1}\mathbf{b} \\ &= M^{i-2}\mathcal{F}\Lambda\mathcal{F}^{-1}\mathcal{F}\Lambda\mathcal{F}^{-1}\mathbf{b} \\ &= M^{i-2}\mathcal{F}\Lambda^2\mathcal{F}^{-1}\mathbf{b} \\ &= \cdots \\ &= \mathcal{F}\Lambda^i\mathcal{F}^{-1}\mathbf{b}, \end{aligned}$$

using the convention $M^0 = I$. This allows us to write the following:

$$\begin{aligned} f(M)\mathbf{b} &= \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} \mathcal{F}\Lambda^i\mathcal{F}^{-1}\mathbf{b} \\ &= \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} \mathcal{F}\Lambda^i \left\{ \frac{1}{N} \sum_{j=-L+2}^{L-2} b_j e^{2\pi\imath kj/N} \right\}_k \\ &= \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} \mathcal{F} \left\{ \frac{1}{N} \sum_{j=-L+2}^{L-2} b_j \lambda_k^i e^{2\pi\imath kj/N} \right\}_k \\ &= \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} \left\{ \frac{1}{N} \sum_{k=-L+2}^{L-2} \sum_{j=-L}^{L} b_j \lambda_k^i e^{2\pi\imath k(j-m)/N} \right\}_m \\ &= \left\{ \frac{1}{N} \sum_{j=-L+2}^{L-2} b_j \sum_{k=-L+2}^{L-2} e^{2\pi\imath k(j-m)/N} \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} \lambda_k^i \right\}_m \\ &= \left\{ \frac{1}{N} \sum_{j=-L+2}^{L-2} b_j \sum_{k=-L+2}^{L-2} e^{2\pi\imath k(j-m)/N} f(\lambda_k) \right\}_m . \end{aligned} \tag{3.35}$$

Also,

$$\mathcal{F}f(\Lambda)\mathcal{F}^{-1}\mathbf{b} = \mathcal{F}f(\Lambda) \left\{ \frac{1}{N} \sum_{j=-L+2}^{L-2} b_j e^{2\pi\imath kj/N} \right\}_k$$

$$= \mathcal{F}\left\{\frac{1}{N}\sum_{j=-L+2}^{L-2} b_j f(\lambda_k) e^{2\pi\imath kj/N}\right\}_k$$

$$= \left\{\frac{1}{N}\sum_{j=-L+2}^{L-2} b_j \sum_{k=-L+2}^{L-2} f(\lambda_k) e^{2\pi\imath k(j-m)/N}\right\}_m . \quad (3.36)$$

We can see by comparing (3.35) and (3.36) that we indeed have $f(M) = \mathcal{F} f(\Lambda)\mathcal{F}^{-1}$. □

As we have eigenvalues λ_k given by (3.32) for the matrix M illustrated in (3.31), it follows that we may use this theorem to obtain

$$\tilde{\sigma}^0_{-l} = \frac{1}{2L-3}\sum_{k=-L+2}^{L-2} f(\lambda_k) e^{2\pi\imath kl/(2L-3)} = \left\{\mathcal{F}^{-1} f(\bar{\lambda})\right\}_l \quad (3.37)$$

for $\tilde{\sigma}^0_l$ the entries of the convolution matrix corresponding to $f(M)$.

Thus we have two methods of determining the convolution matrix entries $\tilde{\sigma}^0_l$ for the representation of $f(M)$, as given by (3.30) and (3.37).

3.5 Conclusions

The above results may be summarized as a series of pros and cons.

The advantages provided by the s-form are:

(1) ease of construction (by way of simple applications of the Mallat transform);

(2) all operations (notably matrix-vector multiplication) occur within a wavelet basis;

(3) Calderón-Zygmund operators are represented as banded matrices.

The disadvantages are:

(1) convolution operators are not reproduced;

(2) explicit scale interactions result in matrix representations which are not sparse.

In comparison, the key advantages of the ns-form are:

(1) convolution operators are reproduced;

(2) scale interactions are implicit, resulting in a sparse representation;

(3) Calderón-Zygmund operators are represented as banded block matrices.

However, the disadvantages of the ns-form are:

(1) construction of ns-form representations is more difficult than the corresponding s-form representations;

(2) the resulting matrices are larger then those of the s-form;

(3) operations require an additional projection step (albeit a relatively inexpensive one).

The s-form is used in the next chapter in Bertoluzza and Naldi's method. The ns-form is made use of in Chapter 5.

Chapter 4

Ordinary Differential Equations

In this chapter we are interested in the approaches of Bertoluzza and Naldi in [4] and Amaratunga *et al.* in [1] to the wavelet-based solution of ordinary differential equations. Our consideration is limited to boundary value problems of the form

$$\begin{cases} \mathcal{L}u(x) = f(x), \quad x_\ell \le x \le x_r \\ u(x_\ell) = a, \;\; u(x_r) = b, \end{cases} \tag{4.1}$$

where $\mathcal{L}$ is (at most) a second-order linear differential operator in one space dimension, $x_\ell,\ x_r \in \mathbf{R}$, and $a, b \in \mathbf{R}$.

In Section 4.1 we present the classical collocation and Galerkin techniques for the solution of ODEs, which are the basis of the wavelet methods presented. Section 4.2 considers the problem of evaluating connection coefficients of scaling functions. In Section 4.3 we consider the wavelet collocation method of Bertoluzza and Naldi, while in Sections 4.4 we consider the Galerkin approach of Amaratunga *et al.*

4.1 The Collocation and Galerkin Techniques

"Classic" finite element projection methods for the solution of (4.1) are based on its variational form by the method of weighted residuals. Two such methods are the *collocation method* and the *Galerkin method* (see [2], [12], and [41]).

Take $a = b = 0$ in (4.1) and assume an "appropriate" (a term we will leave somewhat ambiguous) finite dimensional space with basis $\{\phi_i(x)\}_{i=1}^n$ such that $\phi_i(x_\ell) = \phi_i(x_r) = 0$,

$i = 1, \ldots, n$. Note that we use a uniformly spaced mesh with $h := (x_r - x_\ell)/(n+1)$ and $x_i := x_\ell + ih$, such that $x_0 = x_\ell$ and $x_{n+1} = x_r$. Then the exact solution of (4.1) may be approximated by a projection of the form

$$u(x) \approx U(x) := \sum_{i=1}^{n} c_i \phi_i(x) \tag{4.2}$$

for some set of coefficients $\{c_i\}$.

Consider the residual

$$\begin{aligned} \Delta(x) &:= \mathcal{L}U(x) - f(x) \\ &= \sum_{i=1}^{n} c_i \mathcal{L}\phi_i(x) - f(x), \end{aligned}$$

which expresses the degree to which U fails to satisfy the approximation in (4.2). One selects a set of independent, normalized weight functions $\{\omega_j\}_{j=1}^{n}$, requiring for $j = 1, \ldots, n$,

$$\int_{x_\ell}^{x_r} \omega_j(x)\Delta(x)\,dx = 0,$$

so

$$\sum_{i=1}^{n} c_i \int_{x_\ell}^{x_r} \omega_j(x)\mathcal{L}\phi_i(x)\,dx = \int_{x_\ell}^{x_r} \omega_j(x) f(x)\,dx.$$

In other words, we require the residual to be orthogonal to the weight functions. (Hence the term "method of weighted residuals".) This may of course be written in terms of matrices as

$$M\mathbf{c} = \mathbf{b} \tag{4.3}$$

where

$$\begin{aligned} M_{ij} &= \int_{x_\ell}^{x_r} \omega_i(x)\mathcal{L}\phi_j(x)\,dx \quad \text{and} \\ b_i &= \int_{x_\ell}^{x_r} \omega_i(x) f(x)\,dx. \end{aligned} \tag{4.4}$$

The collocation method chooses the weight functions to be Dirac delta functions, namely $\omega_i(x) = \delta(x - x_i)$, for which (4.4) becomes

$$\begin{aligned} M_{ij} &= \mathcal{L}\phi_j(x_i) \quad \text{and} \\ b_i &= f(x_i), \end{aligned} \tag{4.5}$$

which is equivalent to requiring that $U(x)$ satisfies (4.1) at each of the mesh points x_i, $i = 1, \ldots, n$.

The Galerkin method forces the residual to be orthogonal to the basis functions and therefore uses (4.3) with $\{\omega_i\}_{i=1}^n \equiv \{\phi_i\}_{i=1}^n$. This results in *connection coefficients* M_{ij} in (4.4) (*i.e.*, a quantitative representation of a connection between the basis functions), which must be evaluated separately.

The scaling functions that arise in multi-resolution analysis are especially well-suited to acting as basis functions, due to very good localization properties in both space and frequency (*i.e.*, the windowing property). Hence their use in Sections 4.3 and 4.4.

4.2 Evaluating Connection Coefficients

For now we put aside the solution methodologies for numerically solving differential equations, and consider the evaluation of connection coefficients. This is a common problem among wavelet projection methods. Here we consider the approach of Latto *et al.* in [30] (discussed further in [37] and [38]). A similar approach is given by Keiser in [28] and Beylkin and Keiser in [9] and [10], as previously seen in Sections 3.3, especially equations (3.22) and (3.23).

Although [30] considers more general connection coefficients, the numerical methods

investigated in this chapter only require us to consider connection coefficients of the form

$$\Lambda_l^{d_1,d_2} := \int_{\mathbf{R}} \phi_l^{(d_1)}(x)\phi^{(d_2)}(x)\,dx, \tag{4.6}$$

where $\phi^{(k)}(x) = d^k\phi(x)/dx^k$ and ϕ is some scaling function with compact support $[0, L-1]$. Direct numerical approximation of (4.6) is difficult, due the oscillatory nature of the integrand for most scaling functions. The $\Lambda_l^{d_1,d_2}$ can however be evaluated exactly. Due to the compact support of ϕ, we need only consider $l = 2-L, \ldots, L-2$.

By the two-scale relation (2.9) we may write

$$\phi^{(d)}(x) = \sqrt{2}\sum_{k=0}^{L-1} h_k 2^d \phi^{(d)}(2x-k).$$

Then

$$\begin{aligned}
\Lambda_l^{d_1,d_2} &= 2\int_{\mathbf{R}} \left(2^{d_1}\sum_{k=1}^{L-1} h_k \phi^{(d_1)}(2x-2l-k)\right)\left(2^{d_2}\sum_{j=0}^{L-1} h_j\phi^{(d_2)}(2x-j)\right)dx \\
&= 2^{d_1+d_2+1}\sum_{k=0}^{L-1}\sum_{j=0}^{L-1} h_k h_j \int_{\mathbf{R}} \phi^{(d_1)}(2x-2l-k)\phi^{(d_2)}(2x-j)\,dx \\
&= 2^{d_1+d_2}\sum_{k=0}^{L-1}\sum_{j=0}^{L-1} h_{k-2l+j}h_j \int_{\mathbf{R}} \phi^{(d_1)}(x-k)\phi^{(d_2)}(x)\,dx.
\end{aligned}$$

Hence we may write

$$\frac{1}{2^{d_1+d_2}}\Lambda^{d_1,d_2} = A\Lambda^{d_1,d_2} \tag{4.7}$$

where

$$[A]_{l,k} = \sum_{j=0}^{L-1} h_j h_{k-2l+j}.$$

It is noted in [30, p. 8] that (4.7) requires an additional inhomogeneous equation in order to find a unique solution. This additional equation is derived from the moment conditions of the scaling functions and essentially imposes a normalization on Λ. If we

consider wavelets with M vanishing moments (see Section 2.3), then

$$x^k = \sum_{l \in \mathbf{Z}} \mathcal{M}_l^k \phi_{0,l}(x) \quad \text{for } 0 \le k \le M-1, \tag{4.8}$$

where

$$\mathcal{M}_l^k := \int_{\mathbf{R}} x^k \phi_{0,l}(x)\, dx.$$

Differentiating (4.8) k times gives

$$k! = \sum_{l \in \mathbf{Z}} \mathcal{M}_l^k \phi_{0,l}^{(k)}(x).$$

Taking the inner product of both sides with $\phi(x)$ gives

$$\int_{\mathbf{R}} k! \phi(x)\, dx = \sum_{l \in \mathbf{Z}} \mathcal{M}_l^k \int_{\mathbf{R}} \phi(x) \phi_{0,l}^{(k)}(x)\, dx,$$

and hence

$$k! = \sum_{l \in \mathbf{Z}} \mathcal{M}_l^k \Lambda_l^{0,k}. \tag{4.9}$$

If we let $k = d_1 + d_2$ then (4.9) gives

$$\begin{aligned} (d_1+d_2)! &= \sum_{l \in \mathbf{Z}} \mathcal{M}_l^{d_1+d_2} \Lambda_l^{0,d_1+d_2} \\ &= (-1)^{d_1} \sum_{l \in \mathbf{Z}} \mathcal{M}_l^{d_1+d_2} \Lambda_l^{d_1,d_2}, \end{aligned}$$

since $\Lambda_l^{0,d_1+d_2} = (-1)^{d_1} \Lambda_l^{d_1,d_2}$ (an integration by parts reveals this). Hence

$$(-1)^{d_1}(d_1+d_2)! = \sum_{l \in \mathbf{Z}} \mathcal{M}_l^{d_1+d_2} \Lambda_l^{d_1,d_2} \tag{4.10}$$

(*cf.* equation (3.23)). In order to apply (4.10) we must use scaling functions with $M > d_1 + d_2$ vanishing moments.

Equations (4.7) and (4.10) are adequate to uniquely determine the $\Lambda_i^{d_1,d_2}$. Note that for the numerical methods in the next two sections, d_1 is taken to be 0, further simplifying the above.

4.3 The Method of Bertoluzza and Naldi

In [4], Bertoluzza and Naldi expand on [3] and consider a collocation method for the solution of (4.1). We refer to this as a pseudo-collocation method, as one in fact obtains a Galerkin-type method of solution (see below). For the purposes of this section and without loss of generality we limit our consideration to the unit interval $[0, 1]$.

As with the collocation method in Section 4.1, we require interpolation at each of the mesh points. Hence [4] uses functions θ such that $\theta(n) = \delta_{0n}$, where θ is generated by an *autocorrelation function* of Daubechies wavelets (see (4.11) below). Then we take our approximate solution at the dyadic points $n2^{-m}$ to be

$$u_m(x) = \sum_{n=0}^{2^m} u_m(n2^{-m})\theta(2^m x - n),$$

requiring θ such that

$$u_m(0) = a, \quad u_m(1) = b,$$

(no longer requiring $a = b = 0$). In other words, problem (4.1) is satisfied exactly at the mesh points.

The function θ is derived as follows. We start with a compactly supported multiresolution analysis $\{V_j\}$, with scaling functions ϕ with M vanishing moments. Then define

$$\theta(x) = (\phi * \phi(-\cdot))(x) = \int_{\mathbf{R}} \phi(y)\phi(y - x)\, dy \tag{4.11}$$

and take $\widetilde{V}_j = \text{span}\{\theta(2^j x - k) : k \in \mathbf{Z}\}$, which produces a non-orthonormal multiresolution analysis. Due to the orthogonality of the integer translates of ϕ, we have $\theta(0) = 1$ and $\theta(n) = 0$ if $n \neq 0$ (the *interpolation property*). By integration by parts

for integers l and s satisfying $0 \leq l \leq s$, we have

$$\theta^{(s)}(x) = (-1)^{s-l} \int_{\mathbf{R}} \phi^{(l)}(y)\phi^{(s-l)}(y-x)\,dy. \tag{4.12}$$

Hence by the collocation method (4.5) our problem becomes one of determining $\mathbf{c} \equiv \{u_m(x_n)\}_n$, $x_n := n2^{-m}$, as in (4.3). Given the definition of the θ in (4.11) and substituting (4.12) in (4.5), we in fact have a Galerkin problem where $\phi_{m,k}$ are the basis and weight functions (*cf.* Section 4.1, especially (4.4)). So the M_{ij} entries are then autocorrelation coefficients, as calculated in Section 4.2.

It should be noted that during implementation (unlike Amaratunga *et al.* in [1]), Bertoluzza and Naldi use the standard form of the M matrix, which allows the use of preconditioning. This is obtained by defining $\widetilde{M} = PMP$, where P is a diagonal block matrix, for which the i^{th} block is of dimension $2^{m-i} \times 2^{m-i}$ and diagonal entries 2^i (assuming finitely many scales, $i = 1, \ldots, m$).

So far we have not imposed the boundary conditions. This may pose a problem, due to the compact support and continuity properties of the scaling functions. Two possible approaches, in the context of Bertoluzza and Naldi's method, are as follows.

First, we may introduce functions $\tilde{\theta}_{m,0} = \sum_{k=-\infty}^{0} \theta_{m,k}$ and $\tilde{\theta}_{m,2^m} = \sum_{k=2^m}^{\infty} \theta_{m,k}$, where $\theta_{m,k}(x) = 2^{m/2}\theta(2^m x - k)$. Then we solve for u_m of the form

$$u_m(x) = u_m(0)\tilde{\theta}_{m,0}(x) + \sum_{k=1}^{2^m-1} u_m(x_k)\theta(2^m x - k) + u_m(1)\tilde{\theta}_{m,2^m}(x)$$

which satisfies

$$u_m(0) = a, \quad u_m(1) = b, \quad \text{and}$$

$$\mathcal{L}u_m(x_n) = f(x_n), \quad n = 1, \ldots, 2^m - 1.$$

This deals with inhomogeneous boundary conditions, but gives an error of order $2^{-m/n}$ (see [4, p. 5]) which is computationally prohibitive.

The second approach is to define $u_m(x) := \sum_{k=-L+1}^{2^m+L-1} \alpha_k \theta_{m,k}$, hence using 2^m+2L-1 collocation points. As we already have 2^m+1 dyadic mesh points $x_k = k2^{-m}$, $k = 0,\ldots,2^m$, the additional $2L-2$ points are placed near the boundary. For the left boundary these points are given by $x_k = (2k+1)2^{-(m+1)}$, $k = 0,\ldots,L-2$, and for the right boundary they are given by $x_k = 1-(2k+1)2^{-(m+1)}$, $k = 0,\ldots,L-2$. Hence we will have improved convergence near the boundary points, but in so doing we do not obtain *end-point interpolation* (*i.e.*, at the endpoints the boundary conditions are not satisfied exactly, but the solution will approach the boundary condition values near the endpoints).

Unfortunately, neither of these approaches are particularly easy to implement when using the standard form to obtain preconditioning.

Examples of the solution of an ordinary differential equations via [4] is given in Section 6.1.

4.4 The Galerkin Method of Amaratunga *et al.*

The Galerkin approach to (4.1) considered in [1] uses V_m and its associated scaling functions as a basis. Additional conditions on the problem are imposed (some of which are relaxed later): $u(x)$ and $f(x)$ are considered periodic with period $d \in \mathbb{Z}$; the problem is solved on the entire real line; $0 \leq x_\ell < x_R \leq d$; and the boundary conditions satisfy $u(x_\ell) = u(x_r)$. So on scale m we consider the projection

$$P_m u(x) = \sum_{k\in\mathbb{Z}} c_k \phi_{m,k}(x)$$

$$= \sum_{k\in\mathbf{Z}} c_k 2^{m/2}\phi(2^m x - k).$$

At the dyadic points $y = 2^m x$ and defining $\tilde{c}_k := 2^{m/2}c_k$, this gives

$$u(x) = U(y) := \sum_{k=0}^{2^m d-1} \tilde{c}_k \phi(y-k). \tag{4.13}$$

This is due to $u(x)$ having period d, which implies $U(y)$ has period $2^m d \in \mathbf{Z}$. Thus $\tilde{c}_k$ likewise has period $2^m d$. Letting y take only integer values gives $u(x)$ at the dyadic points $x = y/2^m$, and for $U_i := U(i)$, $i = 0, \ldots, 2^m d - 1$, (4.13) becomes

$$U_i = \sum_{k=0}^{2^m d-1} \tilde{c}_k \phi(i-k) = \sum_{k=0}^{2^m d-1} \tilde{c}_{i-k}\phi(k).$$

This is a discrete convolution, as $\tilde{\mathbf{c}}$ is periodic.

Likewise define $P_m f(x) = F(y) := \sum_{k\in\mathbf{Z}} \tilde{g}_k \phi(y-k)$, giving

$$F_i = \sum_{k=0}^{2^m d-1} \tilde{g}_{i-k}\phi(k).$$

Thus problem (4.1) becomes

$$\mathcal{L}\left(\sum_{k=0}^{2^m d-1} \tilde{c}_{i-k}\phi(k)\right) = \sum_{k=0}^{2^m d-1} \tilde{g}_{i-k}\phi(k) \tag{4.14}$$

for $i = 0, \ldots, 2^m d - 1$.

Following Section 4.1 we solve via the Galerkin approach to (4.14) as

$$M\tilde{\mathbf{c}} = \mathbf{b} \tag{4.15}$$

where

$$M_{ij} = \Omega_{i-j} := \int_{\mathbf{R}} \phi(y-(i-j))\mathcal{L}\phi(y)\,dy \quad \text{and}$$
$$b_j = \tilde{g}_j,$$

for $i, j = 0, \ldots, 2^m d - 1$. Note that $\Omega_{i-j} = 0$ for $i - j \le 1 - L$ or $i - j \ge L - 1$.

We define the discrete Fourier and inverse Fourier transforms by

$$\begin{aligned} \widehat{f}_j &= \sum_{k=0}^{N-1} f_k e^{-2\pi\imath jk/N} \quad \text{and} \\ f_k &= \frac{1}{N}\sum_{j=0}^{N-1} \widehat{f}_j e^{2\pi\imath jk/N}. \end{aligned} \tag{4.16}$$

By (4.14) we have $b_j = \sum_i \Omega_{i-j}\tilde{c}_i$, which implies that

$$\begin{aligned} \widehat{b}_j &= \sum_k \sum_i \Omega_{i-k}\tilde{c}_i e^{-2\pi\imath jk/N} \\ &= \sum_k \sum_i \Omega_k \tilde{c}_i e^{-2\pi\imath j(k+i)/N} \\ &= \widehat{\Omega}_j \cdot \widehat{\tilde{c}}_j, \end{aligned} \tag{4.17}$$

and likewise

$$\widehat{U}_j = \widehat{\Phi}_j \cdot \widehat{\tilde{c}}_j \quad \text{and} \tag{4.18}$$

$$\widehat{F}_j = \widehat{\Phi}_j \cdot \widehat{\tilde{g}}_j, \tag{4.19}$$

where $\cdot$ indicates point-wise multiplication and $\Phi_j = \phi(j)$.

Hence by (4.15), (4.17), (4.18), and (4.19),

$$\begin{aligned} \widehat{U} &= \widehat{\Phi} \cdot ((\widehat{F} \not/ \widehat{\Phi}) \not/ \widehat{\Omega}) \\ &= \widehat{F} \not/ \widehat{\Omega}, \end{aligned}$$

where $\not/$ indicates pointwise division. So by (4.16),

$$U_j = \frac{1}{2^m d}\sum_{k=0}^{2^m d-1} \frac{\widehat{F}_k}{\widehat{\Omega}_k} e^{2\pi\imath kj/(2^m d-1)}. \tag{4.20}$$

The key advantage of this approach is the use of scaling functions, *i.e.*, V_m as the basis of our solution. Due to the two-scale relations (2.9) and (2.10) we can refine the

solution by recomputing (4.20) in a finer basis V_n, $n > m$. The key difficulty lies in the evaluation of the connection coefficients Ω_{i-j} given in (4.15).

So far we have only dealt with the periodic boundary condition version of problem (4.1). However, inhomogeneous Dirichlet conditions may be dealt with efficiently (despite the fact that scaling functions by their nature do not directly support inhomogeneous conditions) by the *capacitance matrix method*. Here we present a generalized version of that found in [1].

Consider, without loss of generality, problem (4.1) with $0 < x_\ell < x_r < d$, $d \in \mathbb{Z}$, and $u(x)$ and $f(x)$ functions of period d. Let $u(x) = v(x) + w(x)$, such that

$$\begin{cases} \mathcal{L}v(x) = f(x), & x \in (x_\ell, x_r) \\ v(0) = v(d) \end{cases} \tag{4.21}$$

and

$$\begin{cases} \mathcal{L}w(x) = 0, & x \in (x_\ell, x_r) \\ \mathcal{L}w(x) = X(x), & x \in (0, d)\,, \end{cases}$$

where

$$X(x) := X_{x_\ell}\delta(x - x_\ell) + X_{x_r}\delta(x - x_r)$$

and $\delta(x)$ is the Dirac delta function. We must find constants X_{x_ℓ} and X_{x_r} such that the original problem for $u(x)$ is satisfied at x_ℓ and x_r.

For the Green's function given by

$$\begin{cases} \mathcal{L}G(x) = \delta(x) \\ G(0) = G(d), \end{cases} \tag{4.22}$$

we have

$$w(x) = G(x) * X(x) = X_{x_\ell}G(x - x_\ell) + X_{x_r}G(x - x_r). \tag{4.23}$$

Problems (4.21) and (4.22) for periodic $v(x)$ and $G(x)$ (respectively) can be solved by the method outlined above for periodic functions. As $w(x)$ is given by (4.23), we need only solve for X_{x_ℓ} and X_{x_r}, by

$$\begin{cases} w(x_\ell) &= X_{x_\ell}G(0) + X_{x_r}G(x_\ell - x_r) &= u(x_\ell) - v(x_\ell) \\ w(x_r) &= X_{x_\ell}G(x_r - x_\ell) + X_{x_r}G(0) &= u(x_r) - v(x_r), \end{cases}$$

so

$$\begin{bmatrix} G(0) & G(x_\ell - x_r) \\ G(x_r - x_\ell) & G(0) \end{bmatrix} \begin{bmatrix} X_{x_\ell} \\ X_{x_r} \end{bmatrix} = \begin{bmatrix} u(x_\ell) - v(x_\ell) \\ u(x_r) - v(x_r) \end{bmatrix}.$$

There is, however, a problem in solving (4.22) by the above method. This is due to the wavelet projection of the delta function, resulting in L non-zero coefficients (the size of the support of the scaling function), which in turn leads to large residual errors (a discussion of this is found in [1, p. 2711]). We may avoid this problem by introducing offset boundary sources:

$$\alpha := x_\ell - s, \quad \beta := x_r + s,$$

where $L/2^m < s < \max(x_\ell, d - x_r)$. Then

$$X(x) = X_{x_\ell}\delta(x - \alpha) + X_{x_r}\delta(x - \beta),$$

and X_{x_ℓ} and X_{x_r} are determined from

$$\begin{bmatrix} G(x_\ell - \alpha) & G(x_\ell - \beta) \\ G(x_r - \alpha) & G(x_r - \beta) \end{bmatrix} \begin{bmatrix} X_{x_\ell} \\ X_{x_r} \end{bmatrix} = \begin{bmatrix} u(x_\ell) - v(x_\ell) \\ u(x_r) - v(x_r) \end{bmatrix}.$$

Examples of solving some ordinary differential equation via the method of Amaratunga *et al.* using offset boundary sources is presented in Section 6.2.

4.5 Conclusions

Although a computational comparison is left to Chapter 6, a few points about the advantages and disadvantages of each method can be made at this point.

Bertoluzza and Naldi's method contains a double edged sword—on the one hand, the use of the s-form allows for preconditioning, but at the same time, this creates a difficulty in dealing with boundary conditions.

Amaratunga, Williams, Qian, and Weiss' method has a few advantages:

(1) the method is relatively easy to program and use for computation;

(2) the scale of computation (number of mesh points) can be readily changed;

(3) the capacitance matrix method handles inhomogeneous boundary conditions well.

The two primary disadvantages, albeit minor ones, are:

(1) use of the capacitance matrix method leads to three problems from one;

(2) one must impose an artificial period on the problem and make use of shifted boundary sources.

Sections 6.1 and 6.2 provide numerical examples for each of these respective methods, and the above advantages and disadvantages are considered from a practical standpoint.

Chapter 5

Partial Differential Equations

This chapter considers the work of Keiser in [28], Beylkin and Keiser in [9] and [10], and Beylkin, Keiser, and Vozovoi in [11]. Their work focuses on a wavelet projection method for the solution of non-linear partial differential equations, although this method is just as valid for linear equations. It must be noted that the following assumes the use of Coiflets as our wavelet basis (see Chapter 2). Section 5.1 gives an overview of the problem at hand, and the semigroup approach to redefining the problem. Section 5.2 shows how the results of Section 5.1 can be reconsidered using a quadrature formulation. Finally Section 5.3 presents the evaluation of analytic functions in our wavelet basis.

5.1 The Problem and a Semigroup Solution

The general non-linear partial differential equation considered in [9], [10], [11], and [28] is the evolution equation

$$\begin{cases} u_t(x,t) = \mathcal{L}u(x,t) + \mathcal{N}u(x,t) & \\ u(x,0) = u_0(x), & x \in [x_\ell, x_r] \\ u(x_\ell,t) = u(x_r,t), & t \in [0,T], \end{cases} \tag{5.1}$$

where $\mathcal{L}$ is a linear (spatial) differential operator representing the linear part of the equation, $\mathcal{N}$ is an (spatial) operator representing the non-linear part, and our domain of consideration is $[x_\ell, x_r] \times [0,T]$.

By [22] and [36] problems of the type (5.1) may be dealt with by the semigroup ap-

proach. Define a spatial linear operator $\mathcal{A}$ as the (infinitesimal) generator of a semigroup $T(t)$ (where the *semigroup property* is $T(t+s) = T(t)T(s)$, $t > 0$, $s > 0$, and $T(0) = I$ for I the identity operator). For $f \in D(\mathcal{A})$, $\mathcal{A}$ and $T(t)$ are related by the identity

$$\mathcal{A}f = \lim_{t \to 0} \frac{T(t)f - f}{t},$$

which implies that

$$T(t) = e^{t\mathcal{A}} \quad \text{and} \quad \mathcal{A} = \frac{d}{dt}T(t)\bigg|_{t=0}.$$

For equation (5.1) we restrict ourselves to a linear operator $\mathcal{L}$ such that the conditions for the existence of a semigroup T, for which the corresponding $\mathcal{A} = \mathcal{L}$, are satisfied. Hence we may proceed as follows. Define

$$g(s) := T(t-s)u(x,s), \qquad 0 < s < t.$$

Then

$$\begin{aligned} \frac{dg}{ds} &= -T(t-s)\mathcal{L}u(x,s) + T(t-s)\frac{d}{ds}u(x,s) \\ &= -T(t-s)\mathcal{L}u(x,s) + T(t-s)\left(\mathcal{L}u(x,s) + \mathcal{N}u(x,s)\right) \\ &= T(t-s)\mathcal{N}u(x,s). \end{aligned} \tag{5.2}$$

Integrating (5.2) from 0 to t gives

$$\int_0^t \frac{dg}{ds}\,ds = \int_0^t T(t-s)\mathcal{N}u(x,s)\,ds.$$

Therefore,

$$g(t) = g(0) + \int_0^t T(t-s)\mathcal{N}u(x,s)\,ds.$$

By the definition of $g(s)$ we then have

$$\begin{aligned} u(x,t) &= T(t)u(x,0) + \int_0^t T(t-s)\mathcal{N}u(x,s)\,ds \\ &= e^{t\mathcal{L}}u(x,0) + \int_0^t e^{(t-s)\mathcal{L}}\mathcal{N}u(x,s)\,ds. \end{aligned} \tag{5.3}$$

Hence (5.3) gives an integral solution of (5.1) in terms of exponentials of operators. There is nonetheless a difficulty posed by this approach—matrix representations of operators of the form $e^{t\mathcal{L}}$ are usually dense, and hence computationally prohibitive. However this is often not the case when using the ns-form. The next section considers a quadrature formulation of (5.3), after which we look at the application of wavelet bases to generating a solution.

5.2 A Quadrature Formulation

A quadrature approach to solving (5.3) is given in [28] (and discussed briefly in [9] and [10]). Here we review the *Exact Linear Part* (E.L.P.) scheme presented in [11], as it appears to be more thorough than that found in [28] and has been subjected to stability analysis.

A more general form of (5.3) evaluates the linear part of the right-hand side using some previous time value $(t-\eta)$, where $0 \leq \eta \leq t$. This give us

$$u(x,t) = e^{(t-\eta)\mathcal{L}}u(x,\eta) + \int_{\eta}^{t} e^{(t-s)\mathcal{L}}\mathcal{N}u(x,s)\,ds.$$

Discretizing this equation for a fixed time mesh of width Δt and taking $\eta = t_{n+1-i}$ gives a quadrature formulation of the form

$$u_{n+1} = e^{i\Delta t\mathcal{L}}u_{n+1-i} + \Delta t\left(\gamma N_{n+1} + \sum_{s=0}^{S-1}\beta_s N_{n-s}\right), \tag{5.4}$$

where $t_n := n\Delta t$, $u_n := u(x,t_n)$, $N_n := \mathcal{N}u(x,t_n)$, $0 < i \leq S-1$, S is the number of time steps used in the approximation of the integral, and $\gamma \equiv \gamma(i,\Delta t\mathcal{L})$ and $\beta_s \equiv \beta_s(i,\Delta t\mathcal{L})$ are coefficient operators dependent on i and must be determined. Hence we are computing the solution at the time t_n using some previous time t_{n+1-i}. This can be

generalized further as

$$u_{n+1} = \sum_{i=1}^{d} c_i \left[e^{i\Delta t \mathcal{L}} u_{n+1-i} + \Delta t \left(\gamma_i N_{n+1} \sum_{s=0}^{S-1} \beta_{i,s} N_{n-s} \right) \right], \tag{5.5}$$

where the c_i are weight parameters such that $\sum_{i=1}^{d} c_i = 1$, and d is the number of previous time steps being used with $0 < d \leq S \leq n+1$. Thus we have an explicit method if $\gamma_i \equiv 0$, $i = 1, \ldots, d$, and an implicit method otherwise. This is referred to as an "Exact Linear Part" scheme by Beylkin, Keiser, and Vozovoi in [11]. We consider the expansion of (5.5) for the general case of $d \geq 1$ (whereas [11] only presents (5.4) for $d = 1$).

We start by expanding u_{n+1} as a Taylor series about t_{n+1-i}:

$$u_{n+1} = \sum_{k=0}^{\infty} u_{n+1-i}^{(k)} \frac{(i\Delta t)^k}{k!}, \tag{5.6}$$

where $u^{(k)} \equiv \partial^k u / \partial t^k$. Now by (5.1) we also have that

$$\begin{aligned}
u^{(1)} &= \mathcal{L}u + N \\
u^{(2)} &= \mathcal{L}u^{(1)} + N^{(1)} \\
&= \mathcal{L}(\mathcal{L}u + N) + N^{(1)} \\
&= \mathcal{L}^2 u + \mathcal{L}N + N^{(1)} \\
&\cdots \\
u^{(k)} &= \mathcal{L}^k u + \sum_{j=0}^{k-1} \mathcal{L}^{k-1-j} N^{(j)},
\end{aligned} \tag{5.7}$$

where $N := \mathcal{N}u(x,t)$ and $N^{(j)} = \partial^j N / \partial t^j$.

Substituting (5.7) into (5.6) gives

$$u_{n+1} = \sum_{i=1}^{d} c_i \left[\sum_{k=0}^{\infty} \left(\mathcal{L}^k u_{n+1-i} + \sum_{j=0}^{k-1} \mathcal{L}^{k-1-j} N_{n+1-i}^{(j)} \right) \frac{(i\Delta t)^k}{k!} \right]$$

$$= \sum_{i=1}^{d} c_i \left[\sum_{k=0}^{\infty} \mathcal{L}^k u_{n+1-i} \frac{(i\Delta t)^k}{k!} + \underbrace{\sum_{k=0}^{\infty} \sum_{j=0}^{k-1} \mathcal{L}^{k-1-j} N^{(j)}_{n+1-i} \frac{(i\Delta t)^{k+1}}{k!}}_{\mathrm{I}} \right]. \quad (5.8)$$

Limiting our consideration to I of (5.8), reversing the order of summation and re-indexing slightly gives

$$\begin{aligned} \mathrm{I} &= \sum_{j=0}^{\infty} \sum_{k=j}^{\infty} \mathcal{L}^{k-j} N^{(j)}_{n+1-i} \frac{(i\Delta t)^{k+1}}{(k+1)!} \\ &= \sum_{j=0}^{\infty} \left((i\Delta t \mathcal{L})^{-(j+1)} \sum_{k=j+1}^{\infty} \mathcal{L}^k \frac{(i\Delta t)^k}{k!} \right) N^{(j)}_{n+1-i} (i\Delta t)^{j+1} \\ &= \Delta t \sum_{j=0}^{\infty} i^{j+1} j! D_{j+1}(i\Delta t \mathcal{L}) N^{(j)}_{n+1-i} \frac{\Delta t^j}{j!}, \end{aligned}$$

where

$$\begin{aligned} D_j(x) &:= (e^x - E_j(x))\, x^{-j} \quad \text{and} \\ E_j(x) &:= \sum_{k=0}^{j-1} \frac{x^j}{j!}. \end{aligned} \quad (5.9)$$

Then (5.8) becomes

$$u_{n+1} = \sum_{i=1}^{k} c_i \left[e^{i\Delta t \mathcal{L}} u_{n+1-i} + \Delta t \sum_{j=0}^{\infty} \underbrace{i^{j+1} j! D_{j+1}(i\Delta t \mathcal{L})}_{\mathrm{II}} N^{(j)}_{n+1-i} \frac{\Delta t^j}{j!} \right]. \quad (5.10)$$

Returning now to the discretization (5.5), we expand N_{n+1} and N_{n-s} as Taylor series about t_{n+1-i}:

$$\begin{aligned} N_{n+1} &= \sum_{k=0}^{\infty} N^{(k)}_{n+1-i} \frac{(i\Delta t)^k}{k!} \\ N_{n-s} &= \sum_{k=0}^{\infty} N^{(k)}_{n+1-i} \frac{((i-s-1)\Delta t)^k}{k!}. \end{aligned}$$

Substituting into (5.5) gives

$$u_{n+1} = \sum_{i=1}^{d} c_i \left[e^{i\Delta t \mathcal{L}} u_{n+1-i} \right.$$

$$+ \Delta t \left(\gamma_i \sum_{k=0}^{\infty} N_{n+1-i}^{(k)} \frac{(i\Delta t)^k}{k!} + \sum_{s=0}^{S-1} \beta_{i,s} \sum_{k=0}^{\infty} N_{n+1-i}^{(k)} \frac{(i-s-1)^k (\Delta t)^k}{k!} \right) \Bigg]$$

$$= \sum_{i=1}^{d} c_i \Bigg[e^{i\Delta t \mathcal{L}} u_{n+1-i}$$

$$+ \Delta t \sum_{k=0}^{\infty} \underbrace{\left(\gamma_i i^k + \sum_{s=0}^{S-1} (i-s-1)^k \beta_{i,s} \right)}_{\text{III}} N_{n+1-i}^{(k)} \frac{(\Delta t)^k}{k!} \Bigg]. \tag{5.11}$$

Thus (5.10) and (5.11) give us two equations for u_{n+1}. Equating II and III allows us to solve for γ_i and $\beta_{i,m}$ in terms of the D_j. This results in

$$D_{j+1}(i\Delta t\mathcal{L}) = \frac{1}{ij!}\gamma_i + \frac{1}{i^{j+1}j!} \sum_{s=0}^{S-1} (i-s-1)^j \beta_{i,s} \tag{5.12}$$

which holds for $j = 0, \ldots, S$ in the implicit case, and $j = 0, \ldots, S-1$ in the explicit case ($\gamma_i \equiv 0$).

As an illustration we have for $d = 1$, $S = 3$, and $j = 0, \ldots, 3$ that

$$D_{j+1}(\Delta t\mathcal{L}) = \frac{1}{j!}\gamma_1 + \frac{1}{j!} \sum_{s=0}^{2} \beta_{1,s}(-s)^j$$

This gives us a matrix problem of the form

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & -1 & -2 \\ \frac{1}{2} & 0 & \frac{1}{2} & 2 \\ \frac{1}{6} & 0 & \frac{-1}{6} & \frac{-4}{3} \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \beta_{1,0} \\ \beta_{1,1} \\ \beta_{1,2} \end{bmatrix} = \begin{bmatrix} D_1(\Delta t\mathcal{L}) \\ D_2(\Delta t\mathcal{L}) \\ D_3(\Delta t\mathcal{L}) \\ D_4(\Delta t\mathcal{L}) \end{bmatrix},$$

so that we have the solution

$$\begin{aligned} \gamma_1 &= \tfrac{1}{3}D_2(\Delta t\mathcal{L}) + D_3(\Delta t\mathcal{L}) + D_4(\Delta t\mathcal{L}) \\ \beta_{1,0} &= D_1(\Delta t\mathcal{L}) + \tfrac{1}{2}D_2(\Delta t\mathcal{L}) - 2D_3(\Delta t\mathcal{L}) - 3D_4(\Delta t\mathcal{L}) \\ \beta_{1,1} &= -1D_2(\Delta t\mathcal{L}) + D_3(\Delta t\mathcal{L}) + 3D_4(\Delta t\mathcal{L}) \\ \beta_{1,2} &= \tfrac{1}{6}D_2(\Delta t\mathcal{L}) - D_4(\Delta t\mathcal{L}). \end{aligned}$$

Now that we have a solution for the γ and β_s we can make use of (5.5). There remains the difficulty of computing the $D_j(x)$ operators, due to the presence of an inverse (see (5.9)). In [11], a *scaling and squaring method* is used, as x may have large singular values.

Noting that $D_0(x) = e^x$ so that $D_0(2x) = (D_0(x))^2$ and $\mathcal{L}$ is represented in its non-standard form, [11] starts by finding a large enough n such that the largest singular value of $2^{-n}\Delta t\mathcal{L}$ is less than one, to ensure the Taylor expansion of $D_0(2^{-n}\Delta t\mathcal{L})$ converges. The matrix resulting from this Taylor expansion is then squared n times to obtain $D_0(\Delta t\mathcal{L})$.

To compute $D_j(x)$ for $j = 1$ and 2, one may note that

$$D_1(2x) = \frac{1}{2}(D_0(x)D_1(x) + D_1(x)) \tag{5.13}$$

$$D_2(2x) = \frac{1}{2}(D_1(x)D_1(x) + 2D_2(x)) \tag{5.14}$$

Similar to the above, one finds an n such that the largest singular value of all of the $D_j(2^{-n}\Delta t\mathcal{L})$, $j = 0, 1, 2$ is less than 1. Then one applies (5.13) and (5.14) n times to obtain the desired D_j. Unfortunately similar relations for $j > 2$ are not as easily constructed.

5.3 Function Evaluation

We now have the following to deal with each component of the discretization of (5.1): a semigroup approach to create an integral equation, (5.3); a quadrature formulation to generate a discretization, (5.5), and a means to compute the coefficients of this discretization, (5.12); and the wavelet transform to project vectors, and the ns-form to project the operators D_i, into a wavelet basis. However we have not yet dealt with the N_i

terms in (5.5). Recall that the N_i arise from the application of a non-linear operator $\mathcal{N}$. For our purposes we assume that $\mathcal{N}$ is some combination of linear (spatial) differential operators and analytic functions f of u. It then remains to show how to compute $f(u)$ in a wavelet basis, as per [7], [9], [10], and [28]. We would like to evaluate $f(u)$ by

$$f(u) = \sum_k f(s_k^0)\phi(x-k), \tag{5.15}$$

which would require s_k^0 to be interpolating (*i.e.*, $s_k^0 = u(k)$). In a numerical context, it is enough to have the above evaluation of $f(u)$ be a quantifiable approximation. To begin we consider the calculation of u^2, and then extend this to general $f(u)$.

Given $u \in \mathbf{L}^2(\mathbf{R})$, the projections $P_j u \in V_j$ and $Q_j u \in W_j$, and finitely many scales $-J, \ldots, 0$, define j_f, $-J \leq j_f \leq -1$, to be the finest scale for which u has significant wavelet coefficients (*i.e.*, coefficients greater than some ϵ in the ℓ^∞-norm). Then we define

$$(P_0 u)_\epsilon(x) := \sum_{j=-J}^{j_f} \sum_{k:|d_k^j|>\epsilon} d_k^j \psi_{j,k}(x) + \sum_{k \in \mathbb{F}_{2^{n-J}}} s_k^{-J} \phi_{-J,k}(x),$$

where $\mathbb{F}_{2^{n-J}} := \{0, 1, \ldots, 2^{n-J}-1\}$. For now we assume that the functions u and u^2 are in V_0 so that

$$(P_0 u)_\epsilon^2 - (P_{-J} u)^2 = \sum_{j=-J}^{j_f} \left((P_{j+1} u)^2 - (P_j u)^2 \right).$$

Using the fact that $P_{j+1} = P_j + Q_j$ gives

$$(P_0 u)_\epsilon^2 = (P_{-J} u)^2 + \sum_{j=-J}^{j_f} \left(2 (P_j u)(Q_j u) + (Q_j u)^2 \right). \tag{5.16}$$

We are then faced with the problem of evaluating $(P_{-J}u)^2$, $(P_j u)(Q_j u)$, and $(Q_j u)^2$ for $j = -J, \ldots, j_f$. Note that $(P_j u)(Q_j u)$ and $(Q_j u)^2$ may not belong to the same subspaces as their respective multiplicands. Nonetheless, due to the definition of the the

V_j and W_j (see Section 2.2), we may consider $P_j u \in V_j \subset V_{j+j_0}$ and $Q_j u \in W_j \subset V_{j+j_0}$ for some $j_0 \geq 1$. More importantly there exists a j_0 so that to within our tolerance ϵ, $(P_{-J}u)^2$, $(P_j u)(Q_j u)$, and $(Q_j u)^2$ all belong to V_{j+j_0} (see [9, p. 170]). Hence we project V_j and W_j into V_{j+j_0} and evaluate $(P_{-J})^2$, $(P_j u)(Q_j u)$, and $(Q_j u)^2$ in this space using (5.15). For the sake of simplicity we consider the case $j = 0$ and assume $u \in V_0$.

Recall from Section 2.4 that the coefficients $s_l^{j_0}$ of the projection of u on V_{j_0} are

$$\begin{aligned} s_l^{j_0} &:= 2^{j_0/2} \int_{\mathbf{R}} u(x)\phi(2^{j_0}x - l)\,dx \\ &= 2^{j_0/2} \int_{\mathbf{R}} \widehat{u}(2^{j_0}\xi)\overline{\widehat{\phi}(\xi)}e^{-2\pi\imath\xi l}\,d\xi \\ &= 2^{j_0/2} \sum_{k\in\mathbf{Z}} \int_{-\pi}^{\pi} \widehat{u}(2^{j_0}(\xi + 2\pi k))\overline{\widehat{\phi}(\xi + 2\pi k)}e^{-2\pi\imath\xi l}\,d\xi. \end{aligned} \tag{5.17}$$

Further,

$$\begin{aligned} \widehat{u}(\xi) &= \int_{\mathbf{R}} \sum_{k\in\mathbb{Z}/2^n\mathbb{Z}} s_k^0 \phi(x-k)e^{-2\pi\imath\xi x}\,dx \\ &= \sum_{k\in\mathbb{Z}/2^n\mathbb{Z}} s_k^0 \int_{\mathbf{R}} \phi(x-k)e^{-2\pi\imath\xi x}\,dx \\ &= \sum_{k\in\mathbb{Z}/2^n\mathbb{Z}} s_k^0 e^{-2\pi\imath\xi k}\widehat{\phi}(\xi), \end{aligned} \tag{5.18}$$

so substituting (5.18) into (5.17) gives

$$s_l^{j_0} = 2^{j_0/2} \sum_{k\in\mathbf{Z}} \int_{-\pi}^{\pi} \left(\sum_{m\in\mathbb{Z}/2^n\mathbb{Z}} s_m^0 e^{-2\pi\imath 2^{j_0} m(\xi+2\pi k)} \widehat{\phi}(2^{j_0}(\xi + 2\pi k))\overline{\widehat{\phi}(\xi + 2\pi k)}e^{-2\pi\imath\xi l} \right) d\xi.$$

As $\left|\text{supp}\left(\widehat{\phi}(2^{j_0}(\xi + 2\pi k)) \cap \widehat{\phi}(\xi + 2\pi k)\right)\right|$ is small for $k > 1$ and j_0 sufficiently large (see [9, p. 172]), we use the $k = 0$ term of (5.17) as an approximation:

$$s_l^{j_0} \approx 2^{j_0/2} \int_{-\pi}^{\pi} \widehat{u}(2^{j_0}\xi)\overline{\widehat{\phi}(\xi)}e^{-2\pi\imath\xi l}\,d\xi. \tag{5.19}$$

As we are using Coiflets we have for $\chi := \int_{\mathbf{R}} x\phi(x)\,dx$ that $\int_{\mathbf{R}} (x-\chi)^m \phi(x)\,dx = 0$, $m = 1, \ldots, M$. Then we have

$$\int_R (x-\chi)^m \phi(x)\,dx = \frac{1}{(-2\pi\imath)^m} \frac{\partial^m}{\partial\xi^m} e^{2\pi\imath\chi\xi} \int_{\mathbf{R}} \phi(x)e^{-2\pi\imath\xi x}\,dx\Big|_{\xi=0} = 0$$

so

$$\frac{\partial^m}{\partial \xi^m} \overline{\widehat{\phi}(\xi)} e^{2\pi i x \xi} \Big|_{\xi=0} = 0.$$

Expanding $\overline{\widehat{\phi}(\xi)} e^{2\pi i x \xi}$ as a Taylor series about $\xi = 0$ gives

$$\overline{\widehat{\phi}(\xi)} e^{2\pi i x \xi} = 1 + \frac{\xi^{M+1}}{(M+1)!} \frac{\partial^{M+1}}{\partial \xi^{M+1}} \left(\overline{\widehat{\phi}(\xi)} e^{2\pi i x \xi} \right) \Big|_{\xi=z}, \tag{5.20}$$

for some $z \in (0, \xi)$.

Noting that $\left| \text{supp} \left(\widehat{u}(2^{j_0}(\xi)) \cap \widehat{\phi}(\xi) \right) \right|$ decreases as j_0 increases, there is a sufficiently large j_0 such that we may approximate the $s_l^{j_0}$ in (5.19) by considering a small neighbourhood about $\xi = 0$ via (5.20). Then (5.19) becomes

$$s_l^{j_0} \approx 2^{j_0/2} \int_{-\pi}^{\pi} \widehat{u}(2^{j_0}\xi) e^{-2\pi i \xi (l+x)} \, d\xi + E_{M,j_0},$$

with

$$E_{M,j_0} := \frac{2^{j_0/2}}{(M+1)!} \int_{-\pi}^{\pi} \widehat{u}(2^{j_0}\xi) e^{-2\pi i \xi (l+x)} \xi^{M+1} \frac{\partial^{M+1}}{\partial \xi^{M+1}} \left(\overline{\widehat{\phi}(\xi)} e^{2\pi i x \xi} \right) \Big|_{\xi=z} d\xi.$$

The magnitude of the error E_{M,j_0} is then dependent on our choice of j_0.

It remains to calculate $(P_{-J})^2$, $(P_j u)(Q_j u)$, and $(Q_j u)^2$ in V_{j+j_0}. First we define operators $\mathcal{V}$ and $\mathcal{W}$ by

$$\mathcal{V}_j^{j_0} \; : \; V_j \to V_{j+j_0} \quad \text{and}$$
$$\mathcal{W}_j^{j_0} \; : \; W_j \to W_{j+j_0},$$

both of which may be calculated directly by the Mallat algorithm (Section 2.4). Recalling that $P_j : \mathbf{L^2(R)} \to V_j$, $Q_j : \mathbf{L^2(R)} \to W_j$, and working on V_{j+j_0}, the components of (5.16) can be computed for $j = -J+1, \ldots, j_f$ using

$$P_{j+j_0}(u^2) \; = \; 2 \left(\mathcal{V}_j^{j_0}(P_j u) \right) \left(\mathcal{W}_j^{j_0}(Q_j u) \right) + \left(\mathcal{W}_j^{j_0}(Q_j u) \right)^2 \quad \text{and} \tag{5.21}$$

$$P_{-J+j_0}(u^2) = \left(\mathcal{V}_{-J}^{j_0}(P_{-J}u)\right)^2 + 2\left(\mathcal{V}_{-J}^{j_0}(P_{-J}u)\right)\left(\mathcal{W}_{-J}^{j_0}(Q_{-J}u)\right) + \left(\mathcal{W}_{-J}^{j_0}(Q_{-J}u)\right)^2. \tag{5.22}$$

That is to say, we evaluate the right-hand sides of (5.21) and (5.22) in V_{j+j_0} to obtain the values of $P_{j+j_0}(u^2)$, and then project this result back into our starting multiresolution analysis $j = -J, \ldots, j_f$ (again inexpensively done via the Mallat wavelet projection). We note that this process is analogous to the decomposition done after applying the ns-form of an operator (*cf.* Section 3.2).

For more general analytic $f(u)$, the method is similar. One makes the assumption that $f(P_0u) \in V_0$ and uses the expansion

$$f(P_0u) - f(P_{-J}) = \sum_{j=-J}^{-1} f(P_{j+1}u) - f(P_ju).$$

Using $P_{j+1} = P_j + Q_j$ and the Taylor expansion for analytic f gives

$$f(Q_ju + P_ju) = \sum_{n=0}^{N} \frac{1}{n!} f^{(n)}(P_ju)(Q_ju)^n + E_{j,N}(f, u),$$

and hence

$$f(P_0u) = f(P_{-J}u) + \sum_{j=-J}^{-1} \sum_{n=1}^{N} \frac{1}{n!} f^{(n)}(P_ju)(Q_ju)^n + E_{j,N}(f, u).$$

For a given accuracy ϵ one can find an N such that $|E_{j,N}(f, u)| < \epsilon$. Then one uses the approach described above, *i.e.*, using $f^{(n)}(\mathcal{V}_j^{j_0}(P_ju))$ and $\left(\mathcal{W}_j^{j_0}(Q_ju)\right)^n$.

Thus we have all that is necessary to evaluate the quadrature formation of section 5.1 in a wavelet space. The numerical results of this approach applied to some example problems is presented in Section 6.3.

Chapter 6

Numerical Experiments

This final chapter presents results of and conclusions based on a selection of examples for each of the wavelet-based methods presented above. While this experimentation is not exhaustive, it does shed light on the relative practicality and usefulness of each method.

6.1 The Method of Bertoluzza and Naldi

The first example using Bertoluzza and Naldi's method, as presented in Section 4.3, is the simple homogeneous problem

$$\begin{cases} -u_{xx} = f(x), \quad 0 \leq x \leq 1 \\ u(0) = u(1) = 0, \end{cases} \tag{6.1}$$

where $f(x)$ is chosen such that the exact solution is

$$u(x) = x(x-1)\sin^2(6x). \tag{6.2}$$

A plot of (6.2) is seen in Figure 6.1.

Error calculations and the total number of floating point operations ($FLOPS$) required to find a solution are given in Figure 6.3. The number of discretization points used is given by 2^m, so the mesh size is $h = 2^{-m}$. Bertoluzza and Naldi's method, using Daubechies wavelets with $L = 8$ (D4) and $L = 20$ (D10), is compared to a simple finite difference scheme (based on [12, pp. 442-443]). Considering the absolute ℓ^∞, ℓ^1, and ℓ^2 errors, it can be seen that Bertoluzza and Naldi's method performs progressively

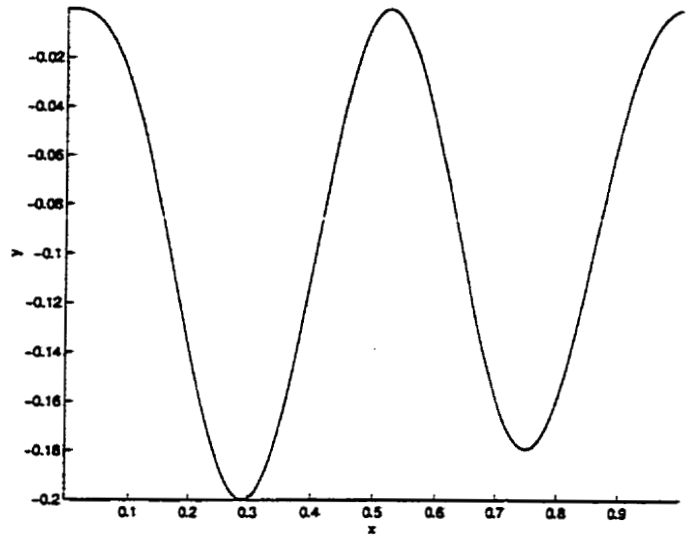


Figure 6.1: The exact solution of problem (6.1).

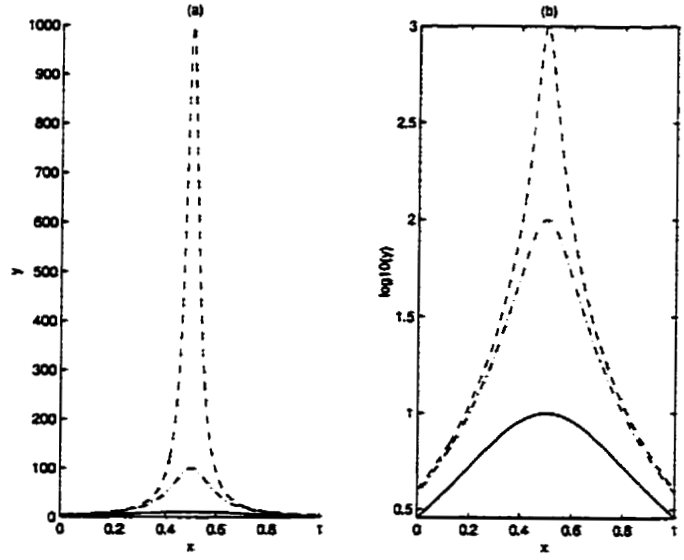


Figure 6.2: The exact solution of problem (6.3), for $\epsilon = 0.1$ (—), $\epsilon = 0.01$ (-·-), and $\epsilon = 0.001$ (– –). (a) x-y plot; (b) x-$\log_{10} y$ plot.

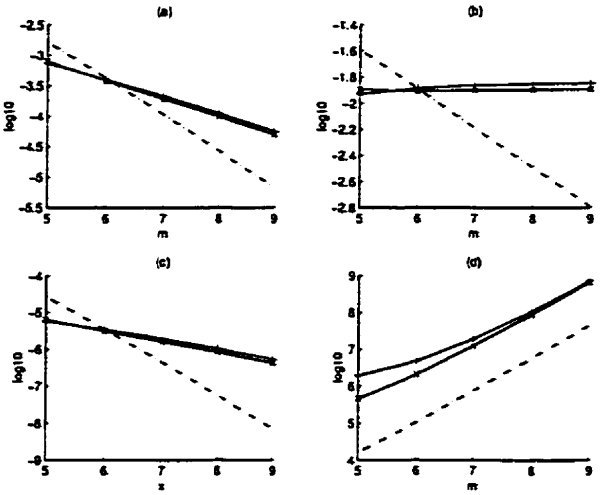


Figure 6.3: Results of solving problem (6.1) using a finite difference method (–·–) and Bertoluzza and Naldi's method (D4 –×–, D10 –+–), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

worse than the finite difference method for $m \geq 6$. To add insult to injury, 100 times as many FLOPS are required for the wavelet-based method for each choice of m. Clearly Bertoluzza and Naldi's method is not well-suited to this problem.

The second example is

$$\begin{cases} -\frac{1}{2\epsilon}u_{xx} + u_x = f(x), \quad 0 \leq x \leq 1 \\ u(0) = u(1) = \frac{1}{\epsilon + 1/4}; \end{cases} \tag{6.3}$$

where $\epsilon > 0$ and $f(x)$ is chosen such that the solution is

$$u(x) = \frac{1}{\epsilon + (x - 1/2)^2}. \tag{6.4}$$

The Matlab code used to implement this problem is seen in Appendix A; the exact solution is seen in Figure 6.2. The numerical solution is considered for $\epsilon = 0.1$, 0.01, and 0.001, using $m = 5$, 7, and 9 for each choice of ϵ. Results for $\epsilon = 0.1$ are plotted in Figure 6.4; for $\epsilon = 0.01$ in Figure 6.5; and for $\epsilon = 0.001$ in Figure 6.6.

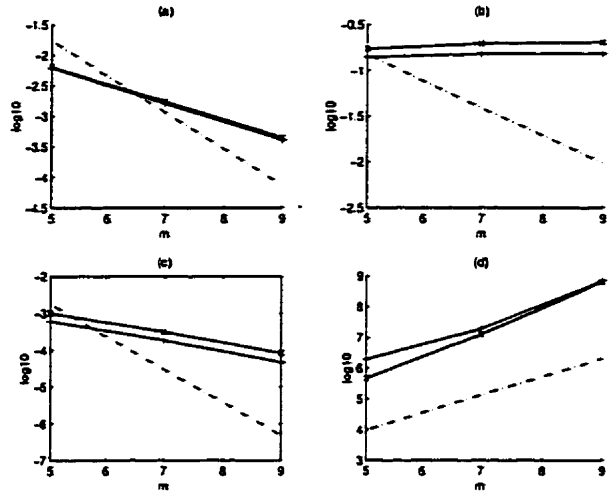


Figure 6.4: Results of solving problem (6.3) with $\epsilon = 0.1$, using a finite difference method (–·–) and Bertoluzza and Naldi's method (D4 –×–, D10 –+–), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

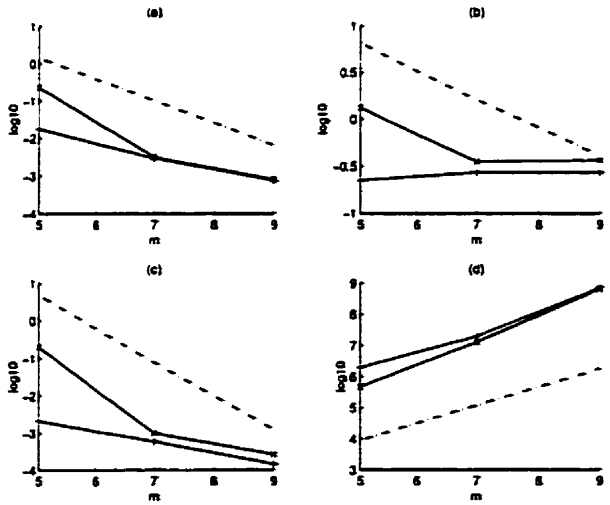


Figure 6.5: Results of solving problem (6.3) with $\epsilon = 0.01$, using a finite difference method (–·–) versus Bertoluzza and Naldi's method (D4 –×–, D10 –+–), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

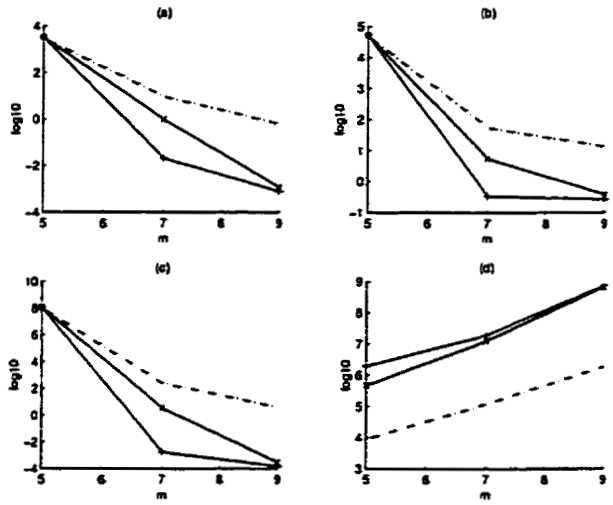


Figure 6.6: Results of solving problem (6.1) with $\epsilon = 0.001$, using a finite difference method (–·–) versus Bertoluzza and Naldi's method (D4 —×—, D10 —+—), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

As with the first example, Bertoluzza and Naldi's method performs worse than the finite difference method for $\epsilon = 0.1$. This changes, however, as the "spike" generated at $x = 1/2$ increases as ϵ decreases. In Figure 6.5 one sees that the absolute ℓ^∞ errors are an order of magnitude smaller for Bertoluzza and Naldi's method; in Figure 6.6, the absolute ℓ^∞ errors are as many as three orders of magnitude smaller. It must nonetheless be noted that the finite difference method remains considerably more efficient as m increases, using approximately 100 times less floating point operations to generate a solution.

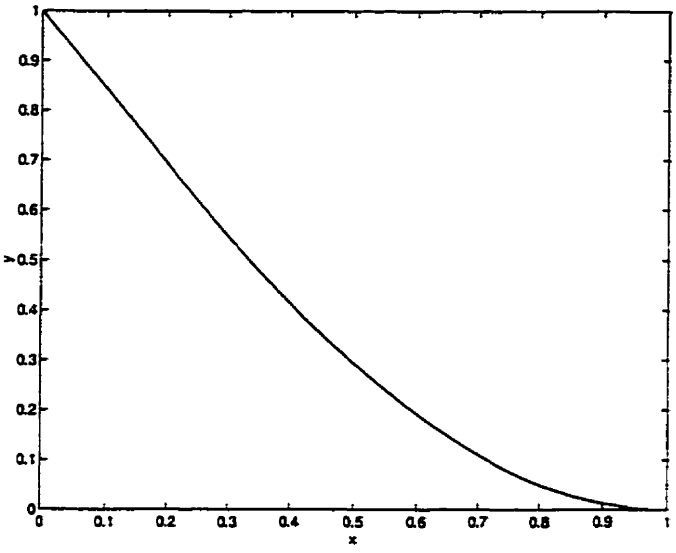


Figure 6.7: The exact solution of problem (6.5).

6.2 The Method of Amaratunga *et al.*

We again start with a relatively simple ordinary differential equation,

$$\begin{cases} u_{xx} + \frac{\pi^2}{4}u = \frac{\pi^2}{4}, & 0 \leq x \leq 1 \\ u(0) = 1, \quad u(1) = 0. \end{cases} \tag{6.5}$$

The exact solution is

$$u(x) = 1 - \sin\left(\frac{\pi x}{2}\right), \tag{6.6}$$

and is presented in Figure 6.7. The Matlab code used to implement this problem is given in Appendix B.

The method of Amaratunga *et al.* is compared to a finite difference method and a Galerkin method (using hat functions as the basis functions) in Figure 6.8. Errors and total floating point operations are plotted for 2^m discretization points, for $m = 5, 6, 7,$ and 8.

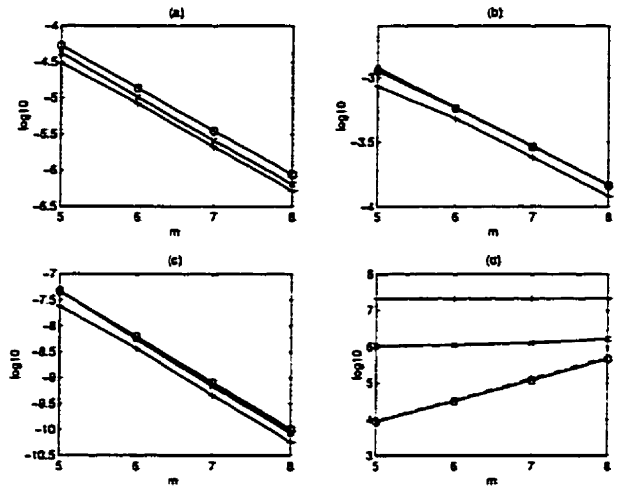


Figure 6.8: Results of solving problem (6.5) using a finite difference method (–·–), a Galerkin method using hat functions (—o—), and the method of Amaratunga *et al.* (D4 —×—, D10 —+—), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

As seen in the plots of the error calculations, this approach generates slightly better results than the finite difference and hat function-based Galerkin methods. Computational complexity (FLOPS) is however much more expensive—at $m = 7$, approximately ten times as many FLOPS are required using Daubechies-4 wavelets, and 100 times as many using Daubechies-10. Extrapolating these results, though, would indicate that this cost differential is overcome for larger choices of m.

The second and third examples arise from the problem

$$\begin{cases} -\frac{1}{2\epsilon}u_{xx} + \frac{1}{2\epsilon}u_x = f(x), \quad 0 \le x \le 1 \\ u(0) = u_0, \quad u(1) = u_1, \end{cases} \tag{6.7}$$

where $\epsilon > 0$ and $f(x)$ is chosen such that the exact solution is

$$u(x) = (\epsilon + 1/4)\,\frac{(u_1 - u_0)x + u_0}{\epsilon + (x - 1/2)^2}. \tag{6.8}$$

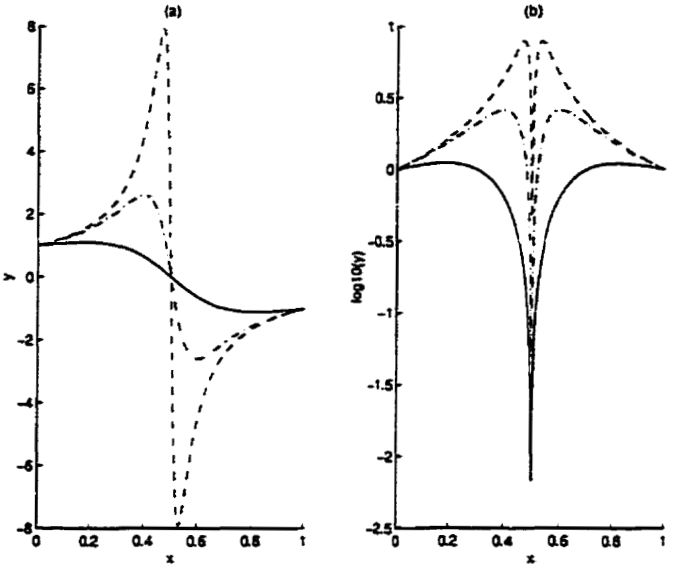


Figure 6.9: The exact solution of problem (6.7) for $u_0 = 1$, $u_1 = -1$, and $\epsilon = 0.1$ (—), $\epsilon = 0.01$ (-·-), and $\epsilon = 0.001$ (- -). (a) x-y plot; (b) x-$\log_{10} y$ plot.

We first consider $u_0 = 1$, $u_1 = -1$, and solve the resulting problem for $\epsilon = 0.1$, 0.01, and 0.001, using $m = 5$, 7, and 9 for each ϵ. The exact solution is seen in Figure 6.9. Results comparing the method of Amaratunga *et al.* (using D4 and D10 wavelets) to the finite difference method are presented in Figures 6.10, 6.11, and 6.12 for the respective ϵ values.

The second choice of boundary conditions is $u_0 = u_1 = 1/(\epsilon + 1/4)$. This problem is similar to the second example for Bertoluzza and Naldi's method (*cf.* (6.7)). As the exact solution is nearly identical, we do not include its graph (see Figure 6.2). Results for the different choices of ϵ (as before) are seen in Figures 6.13, 6.14, and 6.15.

Unlike Bertoluzza and Naldi's method, for $m = 9$ and using D4 wavelets, the method of Amaratunga *et al.* is not much more computationally expensive than the finite differ-

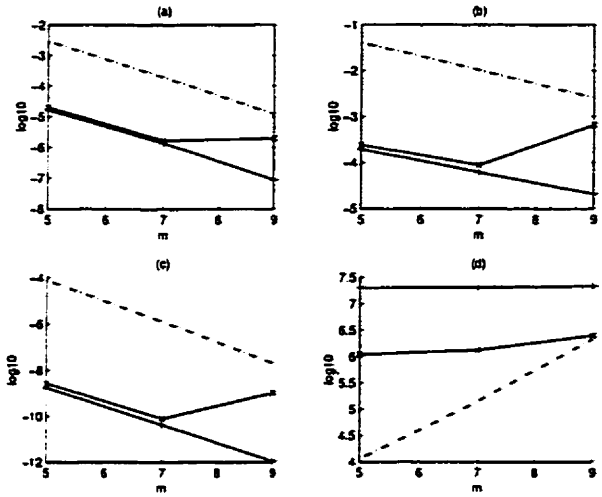


Figure 6.10: Results of solving problem (6.7) for $u_0 = 1$, $u_1 = -1$ with $\epsilon = 0.1$, using a finite difference method (–·–) and the method of Amaratunga *et al.* (D4 –×–, D10 –+–), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

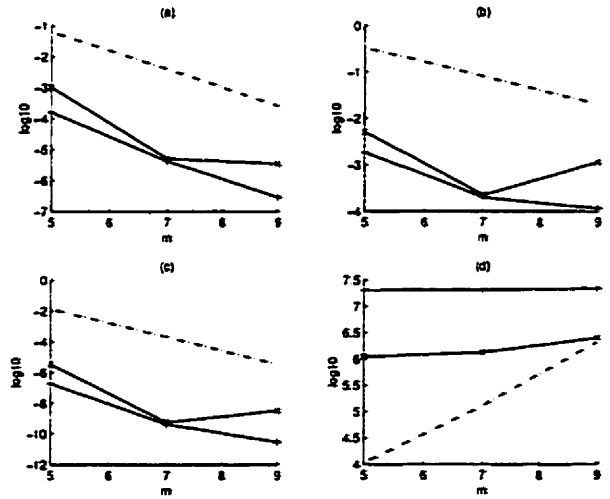


Figure 6.11: Results of solving problem (6.7) for $u_0 = 1$, $u_1 = -1$ with $\epsilon = 0.01$, using a finite difference method (–·–) and the method of Amaratunga *et al.* (D4 –×–, D10 –+–), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

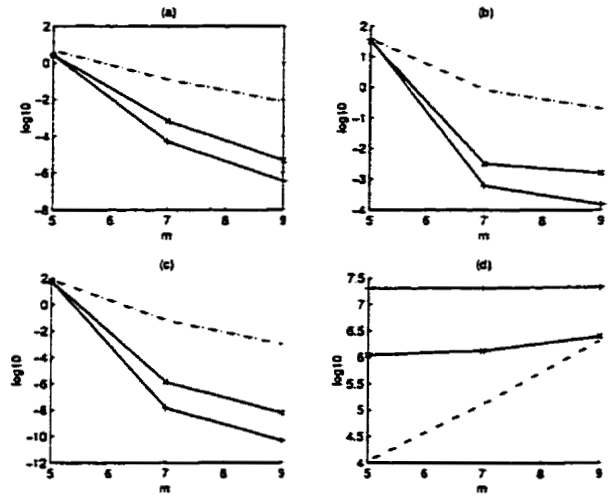


Figure 6.12: Results of solving problem (6.7) for $u_0 = 1$, $u_1 = -1$ with $\epsilon = 0.001$, using a finite difference method (–·–) and the method of Amaratunga *et al.* (D4 –×–, D10 –+–), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

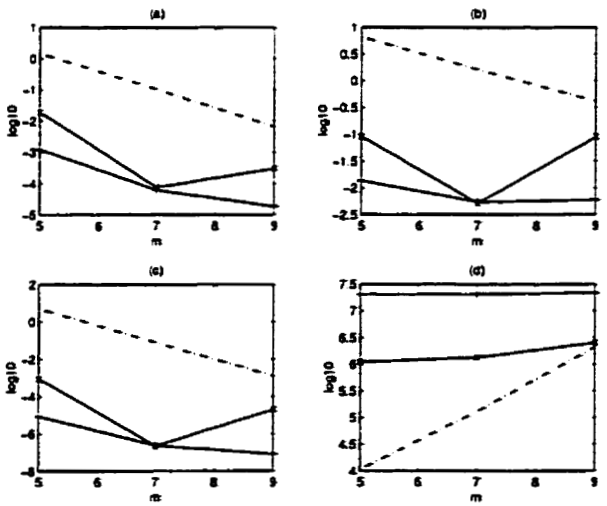


Figure 6.13: Results of solving problem (6.7) for $u_0 = u_1 = 1/(\epsilon + 1/4)$, with $\epsilon = 0.1$, using a finite difference method (–·–) and the method of Amaratunga *et al.* (D4 –×–, D10 –+–), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

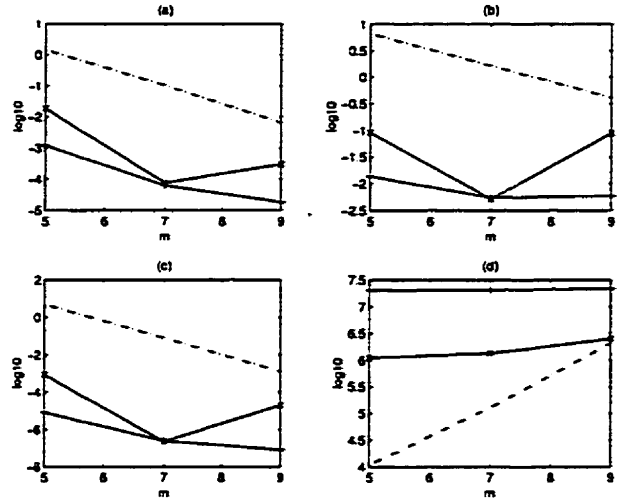


Figure 6.14: Results of solving problem (6.7) for $u_0 = u_1 = 1/(\epsilon + 1/4)$, with $\epsilon = 0.01$, using a finite difference method (–·–) and the method of Amaratunga *et al.* (D4 –×–, D10 –+–), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

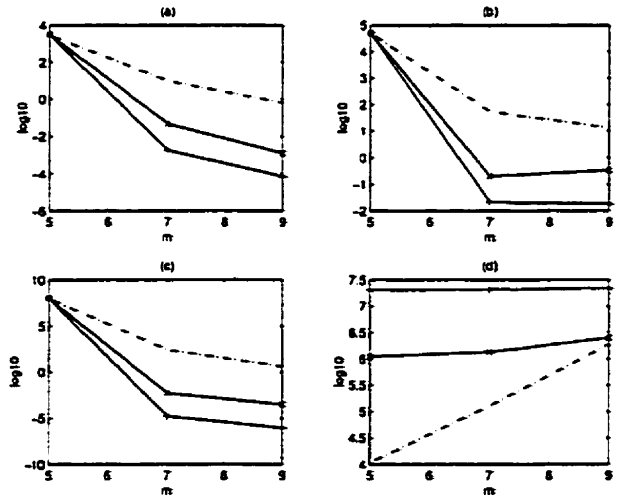


Figure 6.15: Results of solving problem (6.7) for $u_0 = u_1 = 1/(\epsilon + 1/4)$, with $\epsilon = 0.001$, using a finite difference method (–·–) and the method of Amaratunga *et al.* (D4 –×–, D10 –+–), for 2^m discretization points. (a) absolute ℓ^∞ error; (b) absolute ℓ^1 error; (c) absolute ℓ^2 error; (d) FLOPS.

ence method. Further, the number of FLOPS required by this method remains relatively constant as m increases, whereas the FLOPS required by the finite difference method increases exponentially as m increases.

As with the first example, it is hoped that for larger m, the method of Amaratunga *et al.* will provide better approximations of the exact solution than those of the finite difference method, at lesser computational cost. The odd behaviour of the various error computations for D4 wavelets should, however, be noted. At the time of writing, the cause of this behaviour remains unknown—it may be the result of an inherent problem in the method, or simply (and more likely) an error in the implementation.

6.3 Beylkin and Keiser's Method

We begin the examples of Beylkin and Keiser's method for partial differential equations with the diffusion (heat equation) problem,

$$\begin{cases} u_t = \nu u_{xx}, & 0 \le x \le 1,\ 0 \le t \\ u(x,0) = u_0(x), & 0 \le x \le 1, \\ u(0,t) = u(1,t), & 0 \le t. \end{cases} \tag{6.9}$$

(It should be noted that in order to obtain a unique solution, we are actually considering the problem on the entire real line by using the periodic extension, then limiting our consideration to the interval $[0,1]$.) While the diffusion problem does not contain any nonlinear behaviour, it does provide a relatively easy means of testing the method (and code), as the exact solution can readily be computed for many choices of $u_0(x)$.

The first example uses

$$u_0(x) = \sin(2\pi x), \tag{6.10}$$

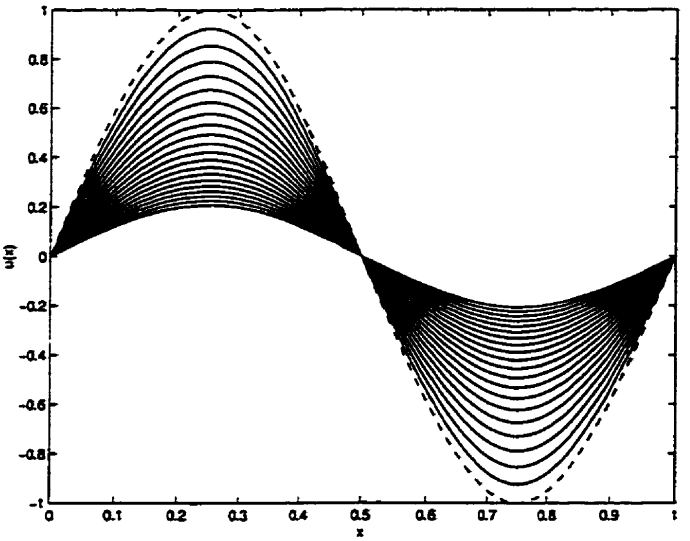


Figure 6.16: Results of solving problem (6.10) using Coiflet-8, $m = 9$, $J = -4$, $\nu = 1$, $\epsilon = 0.0001$, and $dt = 0.0001$, for 400 time steps, plotting every 20^{th} step. The initial condition is given by the dashed line.

for which the exact solution for $\nu = 1$ is

$$u(x,t) = \sin(2\pi x)e^{(-4\pi^2 t)}.$$

The computed solution is plotted in Figure 6.16. The associated absolute ℓ^∞, ℓ^1, and ℓ^2 error calculations are plotted in Figures 6.17, 6.18, and 6.19, respectively. It is worth noting that all of the errors reach a maximum around 260 time steps, then begin to decrease.

Additional examples of the heat equation are provided, but without error calculations. (Computation of the exact solution proved to be prohibitive in Matlab.) The second example, the computed solution of which is plotted in Figure 6.20, corresponds to the

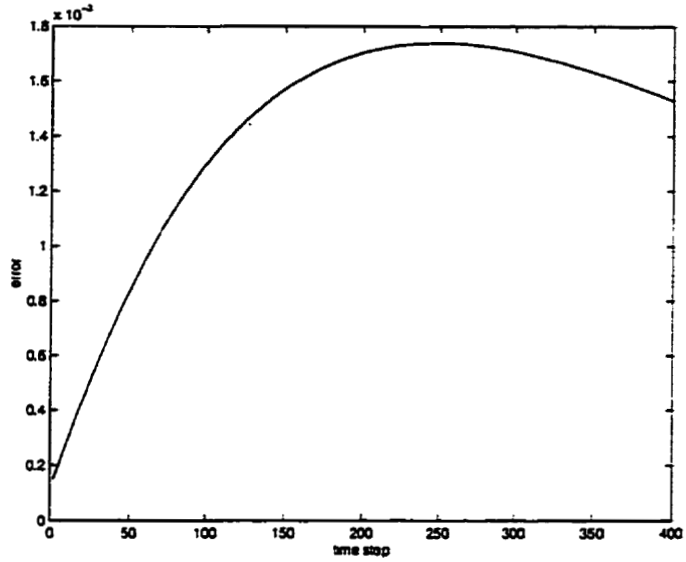


Figure 6.17: Absolute ℓ^∞ error corresponding to Figure 6.16, calculated every second time step.

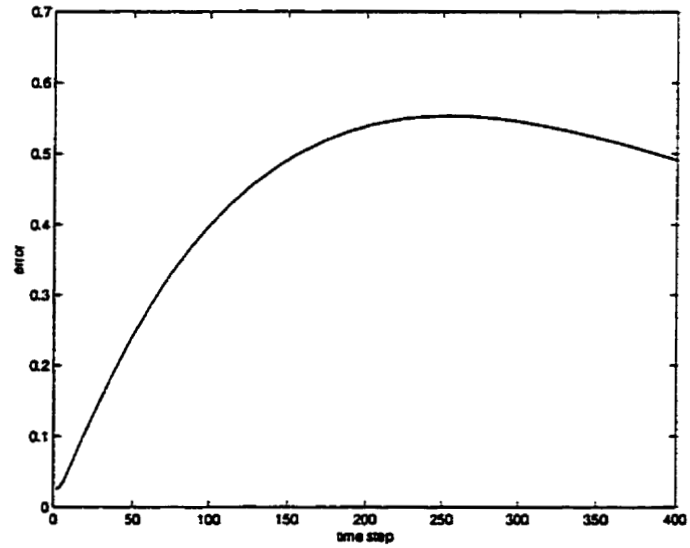


Figure 6.18: Absolute ℓ^1 error corresponding to Figure 6.16, calculated every second time step.

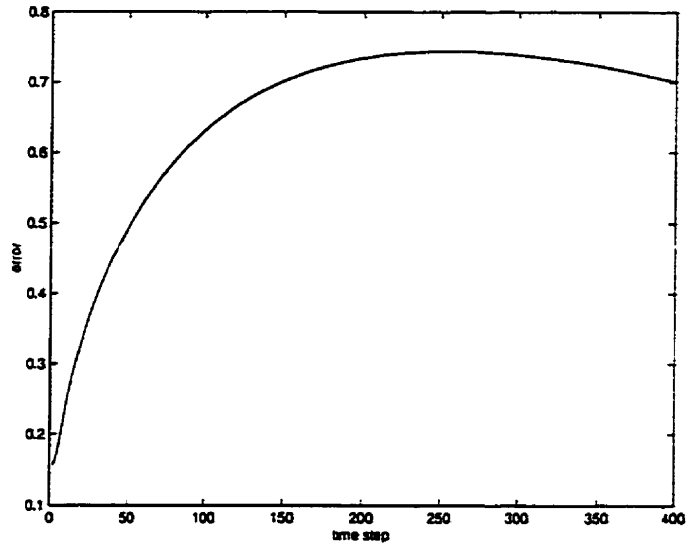


Figure 6.19: Absolute ℓ^2 error corresponding to Figure 6.16, calculated every second time step.

initial condition

$$u_0(x) = \begin{cases} x, & 0 \le x \le \frac{1}{2} \\ 1-x, & \frac{1}{2} \le x \le 1. \end{cases} \tag{6.11}$$

As pointed out in [9, pp. 178-179], behaviour of the computed solution should be noted at $x = 0$, $1/2$, and 1. Methods such as the Crank-Nicolson scheme will often generate a slowly decaying peak (cusp) at these points, rather than producing the correct, smooth behaviour. This failing in such schemes is attributed to an inability to properly resolve high frequency components in the initial conditions.

The final example using the heat equation is for the initial condition

$$u_0(x) = \begin{cases} -\frac{1}{2} - x, & -1 \le x \le -\frac{1}{2} \\ x + \frac{1}{2}, & -\frac{1}{2} \le x \le \frac{1}{2} \\ \frac{3}{2} - x, & \frac{1}{2} \le x \le 1. \end{cases} \tag{6.12}$$

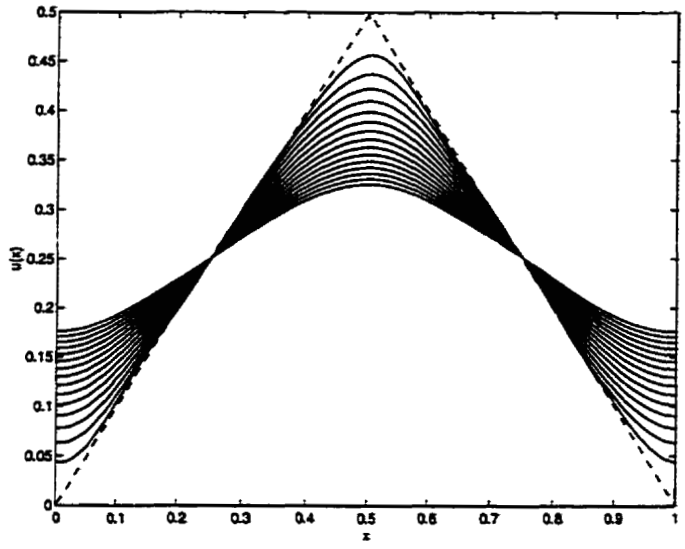


Figure 6.20: Results of solving problem (6.11) using Coiflet-8, $m = 8$, $J = -6$, $\nu = 1$, $\epsilon = 0.0001$, and $dt = 0.001$, for 30 time steps, plotting every second step. The initial condition is given by the dashed line.

This is similar to taking the odd periodic extension of the hat function in (6.11), in order to impose homogeneous boundary condition.

We now proceed to a nonlinear example, Burgers' equation, of the form

$$\begin{cases} u_t = \nu u_{xx} + u u_x, & 0 \le x \le 1, \ 0 \le t \\ u(0,t) = u(1,t), & 0 \le t \\ u(x,0) = u_0(x), & 0 \le x \le 1. \end{cases} \tag{6.13}$$

The computed solution, for $\nu = 1/1000$, $u_0(x) = \sin(2\pi x)$, and times $t = 0.024$, 0.026, 0.028, and 0.030, is seen in Figures 6.22, 6.23, 6.24, and 6.25. This value of ν has been chosen to emphasize the non-linear behaviour of the solution. The Matlab code used for the solution of (6.13) is listed in Appendix C.

Unfortunately the analytic solution of Burgers' equation for $\nu = 1/1000$ requires

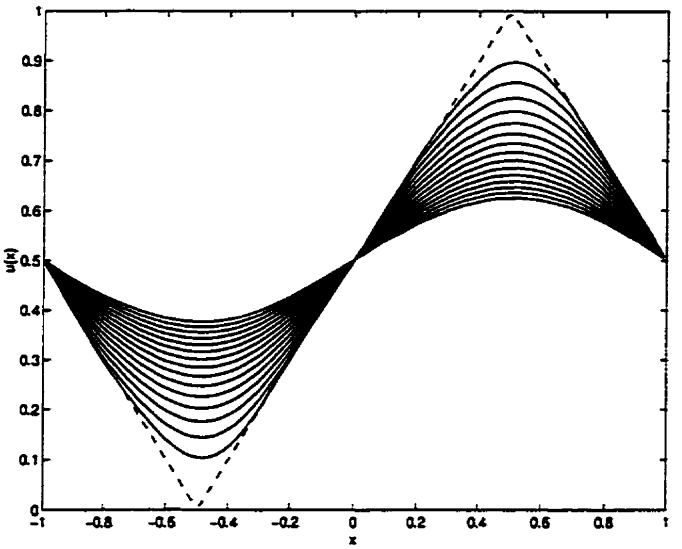


Figure 6.21: Results of solving problem (6.12) using Daubechies-10, $m = 7$, $J = -6$, $\nu = 1$, $\epsilon = 0.0001$, and $dt = 0.001$, for 30 times steps, plotting every second step. The initial condition is given by the dashed line.

an integration too complicated for computation by Matlab's built-in functions. Error calculations are therefore necessarily omitted.

It can be seen that up to $t = 0.024$, Beylkin and Keiser's method (correctly) reproduces non-linear behaviour, manifested as the development of a shock at $x = 1/2$. The method is not infallible, however. Further time steps indicate the development of Gibbs phenomena, as the gradient of the shock increases. Similar examples generated in [9, p. 184] indicate that this problem can be overcome by increasing the size of the initial space V_0 (and hence decreasing the mesh size).

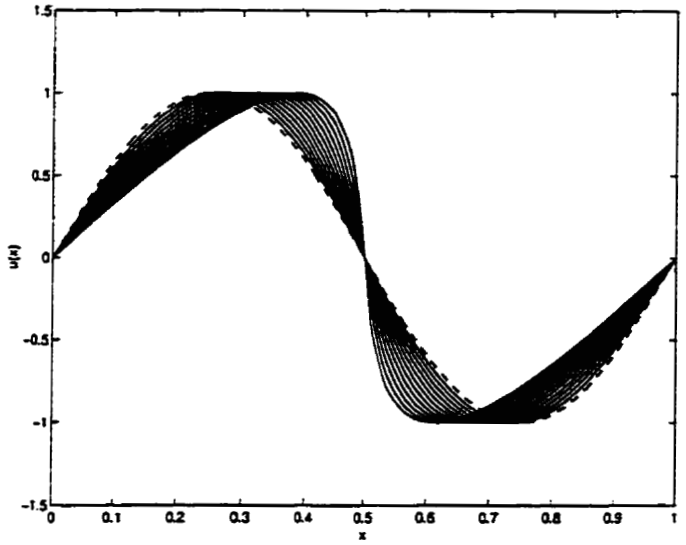


Figure 6.22: Results of solving problem (6.13) to $t = 0.024$ using Coiflet-8, $m = 11$, $J = -5$, $\nu = 0.001$, $\epsilon = 0.00001$, and $dt = 0.001$, plotting every second time step.

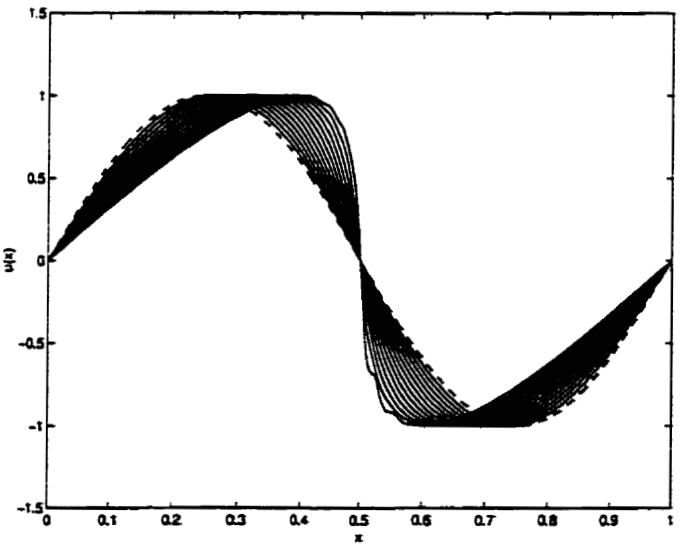


Figure 6.23: Results of solving problem (6.13) to $t = 0.026$. Gibbs phenomena starts to develop.

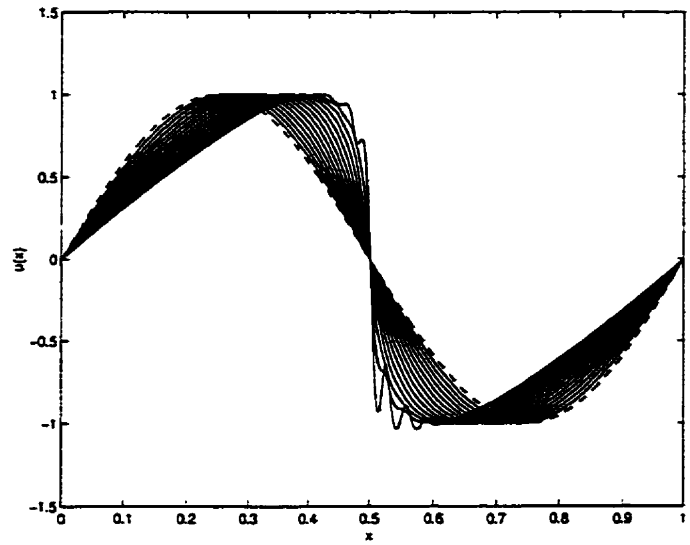


Figure 6.24: Results of solving problem (6.13) to $t = 0.028$. Note that the Gibbs phenomena worsens.

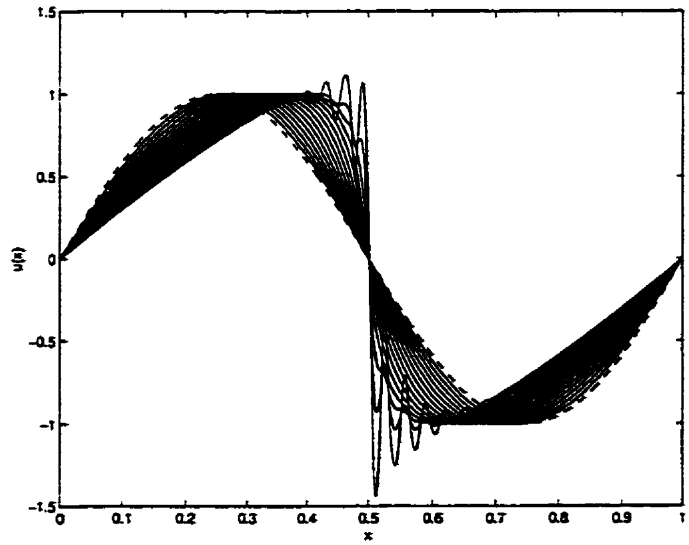


Figure 6.25: Results of solving problem (6.13) to $t = 0.030$. Note that the Gibbs phenomena grows and propagates.

6.4 Conclusions

Wavelet theory is a rather new branch of mathematics that many feel holds considerable promise in a variety of applications. The methods presented herein are some of the first attempts at the numerical solution of differential equations using this new theory.

Our contributions to this topic have primarily consisted of the following: a generalisation of the capacitance matrix method (consideration in [1] is limited to problems of the form $u_{xx} + au = b$); more robust testing of the method of Amaratunga *et al.*; a generalisation of Beylkin, Keiser, and Vozovoi's quadrature formulation to include weighted time steps; and the discovery of an error in Beylkin and Keiser's derivation of the matrix elements in the representation of spatial derivative operators, d^p/dx^p.

The method of Amaratunga *et al.* appears to be a good choice for the solution of ordinary differential equations. For problems with low frequency behaviour, this method produces results similar to the finite difference method, but without exponential growth in computational complexity as the mesh is refined. The true strength of this method is seen when considering problems containing high frequency features.

Unfortunately the same cannot be said of the method of Bertoluzza and Naldi. While this method can approximate solutions with greater accuracy than the finite difference method, it does so at a considerably greater computational cost.

Beylkin and Keiser's approach to the solution on non-linear partial differential equations is somewhat novel, and appears to be promising. Of greatest importance, for the problems considered, high frequency behaviour is correctly resolved in both linear and non-linear problems.

Overall, wavelet-based methods for the numerical solution of differential equations

appear to be viable. Their practicality, by way of a thorough comparison to existing, specialized techniques, remains to be investigated.

Bibliography

[1] Amaratunga, K., J.R. Williams, S. Qian, and J. Weiss. Wavelet-Galerkin Solutions for One-Dimensional Partial Differential Equations. *International Journal for Numerical Methods in Engineering*, Vol. 37, John Wiley & Sons, 1994.

[2] Ascher, U.M., R.M.M. Mattheji, and R.D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, SIAM, 1995.

[3] Bertoluzza, S., Y. Maday, and J.C. Ravel. A Dynamically Adaptive Wavelet Method for Solving Partial Differential Equations. *Computational Methods of Applied Mechanical Engineering*, Vol. 116, North-Holland, 1994.

[4] Bertoluzza, S. and G. Naldi. A Wavelet Method for the Numerical Solution of Partial Differential Equations. *Applied and Computational Harmonic Analysis* **3**, Academic Press, 1996.

[5] Beylkin, G. On the Representation of Operators in Bases of Compactly Supported Wavelets. *SIAM Journal on Numerical Analysis*, Vol. 29 No. 6, SIAM, 1992.

[6] Beylkin, G. On Wavelet-based Algorithms for Solving Differential Equations. *Wavelets: Mathematics and Applications*, CRC Press, 1994.

[7] Beylkin, G. Wavelets and Fast Numerical Algorithms. Lecture Notes for Short Course, AMS-93. *Proceedings of Symposia in Applied Mathematics*, Vol. 47, American Mathematical Society, 1993.

[8] Beylkin, G., R. Coifman, and V. Rokhlin. Fast Wavelet Transforms and Numerical

Algorithms I. *Communications on Pure and Applied Mathematics*, Vol. XLIV, John Wiley & Sons, 1991.

[9] Beylkin, G. and J.M. Keiser. An Adaptive Pseudo-Wavelet Approach for Solving Nonlinear Partial Differential Equations. In W. Dahmen, A. Kurdila, and P. Oswald, editors, *Multiscale Wavelet Methods for Partial Differential Equations*, Academic Press, 1997.

[10] Beylkin, G. and J.M. Keiser. On the Adaptive Numerical Solution of Nonlinear Partial Differential Equations in Wavelet Bases. *Journal of Computational Physics*, Vol. 132 No. 2, Academic Press, 1997.

[11] Beylkin, G., J.M. Keiser, and L. Vozovoi. A New Class of Stable Time Discretization Schemes for the Solution of Nonlinear PDEs. *Journal of Computational Physics*, Vol. 147 No. 2, Academic Press, 1998. 1998.

[12] Borse, G.J., *Numerical Methods with MATLAB*, P.W.S. Publishing Co., 1997.

[13] Buckheit, J., S. Chen, D. Donoho, I. Johnstone, and J. Scargle. WaveLab 0.701. *http://www-stat.stanford.edu/~wavelab*, 1996.

[14] Chui, C.K., *An Introduction to Wavelets*, Academic Press, 1992.

[15] Courant, R. and D. Hilbert, *Methods of Mathematical Physics*, Vol. I, John Wiley & Sons, 1989.

[16] Daalhuis, A.B.O. Computing with Daubechies Wavelets. In [29], 1993.

[17] Dahlke, S. and A. Kunoth. A Biorthogonal Wavelet Approach for Solving Boundary Value Problems. *Institut für Geometrie und Praktische Mathematik Bericht Nr. 85*, Rheinisch–Westfälische Technische Hochschule Aachen, 1993.

[18] Daubechies, I. Orthonormal Bases of Compactly Supported Wavelets. *Communications on Pure and Applied Mathematics*, Vol. XLI, John Wiley & Sons, 1988.

[19] Daubechies, I. Orothonormal Bases of Compactly Supported Wavelets II. Variations on a Theme. *SIAM Journal of Mathematical Analysis*, Vol. 24, SIAM, 1993.

[20] Daubechies, I. *Ten Lectures on Wavelets*, SIAM, 1992.

[21] Farge, M. Wavelet Transforms and Their Applications to Turbulence. *Annual Review of Fluid Mechanics*, Vol. 24, Annual Reviews, 1992.

[22] Goldstein, J.A., *Semigroups of Linear Operators and Applications*, Oxford University Press, 1985.

[23] Hasham, Z., *An Overview of One-Dimensional Wavelet Theory*, M.Sc. thesis, University of Calgary, 1993.

[24] Heijmans, H.J.A.M. Discrete Wavelets and Multiresolution Analysis. In [29], 1993.

[25] Hemker, P.W., T.H. Koornwinder, and N.M. Temme. Wavelets: Mathematical Preliminaries. In [29], 1993.

[26] Jawerth, B. and W. Sweldens. An Overview of Wavelet Based Multiresolution Analysis. *SIAM Review*, Vol. 36 No. 3, SIAM, 1994.

[27] Journé, J.-L. Calderón-Zygmund Operators, Pseudo-Differential Operators and the Cauchy Integral of Calderón. In A. Dold and B. Eckmann, editors, *Lecture Notes in Mathematics*, No. 994, Springer-Verlag, 1983.

[28] Keiser, J.M., *I. Wavelet Based Approach to Numerical Solution of Nonlinear Partial Differential Equations and II. Nonlinear Waves in Fully Discrete Dynamical Systems*, Ph.D. thesis, University of Colorado, 1995.

[29] Koornwinder, T.H., editor, *Wavelets: An Elementary Treatment of Theory and Applications*, World Scientific, 1993.

[30] Latto, A., H.L. Resnikoff, and E. Tenenbaum. The Evaluation of Connection Coefficients of Compactly Supported Wavelets. In Y. Meyer, editor, *Proceedings of the French-U.S.A. Workshop on Wavelets and Turbulence , Princeton University, June 1991*, Springer-Verlag, 1992.

[31] Liandrat, J., V. Perrier, and PH. Tchamitchian. Numerical Resolution of Nonlinear Partial Differential Equations Using the Wavelet Approach. In C.K. Chui, editor, *Wavelets and Their Applications*, Academic Press, 1990.

[32] Mallat, S. and W.L. Hwang. Singularity Detection and Processing with Wavelets. *IEEE Transactions on Information Theory*, IEEE, Vol. 38 No.2, 1992.

[33] Meneveau, C. Analysis of Turbulence in the Orthonormal Wavelet Representation. *Journal of Fluid Mechanics*, Vol. 232, Cambridge University Press, 1991.

[34] Meyer, Y. *Wavelets and Operators*, Cambridge University Press, 1992.

[35] Monzón, L.A., G. Beylkin, and W. Hereman. On Almost Interpolating and Nearly Linear Phase Compactly Supported Wavelets (Coiflets). *PAM Report 343*, University of Colorado, 1997.

[36] Pazy, A., *Semigroups of Linear Operators and Applications to Partial Differential Equations*, Springer-Verlag, 1983.

[37] Restrepo, F.M. and G.K. Leaf. Inner Product Computations Using Periodized Daubechies Wavelets. *International Journal of Numerical Methods in Engineering*, Vol. 40 No. 19, Wiley, 1997.

[38] Schlossnagle, G., J.M. Restrepo, and G.K. Leaf, *Technical Report ANL-93/34: Periodized Wavelets*, Argonne National Labratory, 1994.

[39] Stein, E.M. and G. Weiss, *Introduction to Fourier Analysis on Euclidean Spaces*, Princeton Book Co., 1971.

[40] Temme, N.M. Wavelets, First Steps. In [29], 1993.

[41] Vichnevetsky, R., *Computer Methods for Partial Differential Equations, Volume I*, Prentice-Hall, 1981.

[42] Xu, J.C. and W-C. Shann. Galerkin-Wavelet Methods for Two-Point Boundary Value Problems. *Numerische Mathematik*, Vol. 63, Springer-Verlag, 1992.

[43] Yosida, K., *Functional Analysis*, Springer-Verlag, 1980.

Appendix A

Bertoluzza and Naldi Code

The following lists the routines used to implement the second example of Bertoluzza and Naldi's method for the numerical solution of ordinary differential equations. The code is Matlab 5.2 compliant, and makes use of the wavelet routines provided in the WaveLab 0.701 package ([13]).

```
%--- BN_ex2.m -----|---------|---------|---------|---------|
% Corresponds to the problem (e := epsilon):
%     -{1 \over 2e} u^{(2)} + u^{(1)} =
%        {-16 ( 8ex^3 + (-12e+12)x^2 + (-12+8e^2+6e)x -
%        4e^2+3-5e )} \over {64e (e-(x-{1 \over 2})^2)^3}
%     u(0) = {1 \over {e + {1 \over 4}}} = u(1)
%
%   ==> u(x) = {1 \over {e + ( x - {1 \over 2} )^2 }}
%
% Arguments:
%    m         - $log_2$ of the dimension of $V_0$
%    par       - the length of the Daubechies wavelet filter
%                to use
%    p_bool    - value to indicate if plots should be
%                generated
%    epsilon   - variable parameter for problem

function BN_ex2(m, par, p_bool, epsilon)

   % Specify the differentiation operator as a vector.
   sL = [0 1 -1/(2*epsilon)];

   flops(0);
   % Solve the problem.
   u = BN_solver(sL,m,'BN_ex2_f',par,epsilon);
   f_count = flops;

   % The exact solution (symmetric interval in [0,1]).
   x = [2^(-m-1):2^(-m):1-2^(-m-1)];
   f = 1 ./ (epsilon + (x-1/2).^2);
```

```
   % Plot the results.
   if p_bool == 1
      figure
      plot(x,u,'-.');
      hold on
      plot(x,f);
      hold off
   end;

   % Output error values;
   sprintf('linf error: %d',max(abs(u'-f)))
   sprintf('l1 error:   %d',sum(abs(u'-f)))
   sprintf('l2 error:   %d',sum((u'-f).^2))
   sprintf('FLOPS:      %d',f_count)


%--- BN_ex2_f.m ---|---------|---------|---------|---------|
% Evaluates the RHS for example 2. Note the use of a
% symmetric interval within [0,1].

function v = BN_ex2_f(m,e)

   x = [2^(-m-1):2^(-m):1-2^(-m-1)];
   v = -16*(8*e*x.^3+(-12*e+12)*x.^2+(-12+8*e^2+6*e)*x-...
            4*e^2+3-5*e)/e./(4*e+4*x.^2-4*x+1).^3;


%--- BN_solver.m --|---------|---------|---------|---------|
% This function is used to solve a second order ODE problem
% by the method of Bertoluzza and Naldi.
%
% Arguments:
%   sL      - the operator $\bf \cal L$  as a vector;
%             the i^th element contains the coefficient
%             associated with $d^{i-1} \over dx^{i-1}$.
%   m       - log2 of the size of the basis of $V_0$
%   f_prob  - the name of the function to compute the RHS
%             values of the problem
%   par     - the length of the low-pass filter of the
%             DB wavelet to use; the number of vanishing
%             moments is par/2
%   epsilon - variable parameter of the problem

function U = BN_solver( sL, m, f_prob, par, epsilon )

   % Get the low-pass filter coefficients:
   h = MakeONFilter('Daubechies', par);
```

```
L = length(h);
D = 2^m;

lsL = length(sL);              % length of the sL vector

% Build the needed matrices for each derivative
% operator in sL; each matrix will be added to make the
% "full" operator matrix for the problem.
M = zeros(D);                  % operator matrix
for p = 1:lsL                  % calculate for each deriv
   if sL(p) ~= 0               % ensure operator present
      N = zeros(D);
      if p == 1                % identity operator
         N(:,:) = sL(p) * eye(D);
      else                     % p-1 derivative
         R = auto_val(h,p-1); % Find auto-correl vals
         for i = 1:D
            for k = 1:D
               if 2-L <= i-k & i-k <= L-2
                 N(i,k) = sL(p)*2^((p-1)*m)*R(i-k+L-2+1);
                                % Op matrix for deriv p-1
               end;
            end;
         end;
      end;
      M = M + N;               % add operators into sL
   end;
end;


% Get the function values at the dyadic points for
% the RHS of the problem.
F = feval(f_prob,m,epsilon)';

% Create the pre-conditioning matrix.
P = zeros(D);
s_ind = -1;
e_ind = 0;
for k = 0:m
   s_ind = e_ind + 1;
   if k == 0 | k == 1
      e_ind = e_ind + 1;
   else
      e_ind = e_ind+2^(k-1);
   end;
   for i = s_ind:e_ind
      if k == 0 | k == 1
```

```
            P(i,i) = 2^(m);
         else
            P(i,i) = 2^(m-k+1);
         end;
      end;
   end;

   % Generate the standard form of M by applying the Fast
   % Wavelet Transform to the rows, and then to the columns.
   % Here we are using the periodized, orthogonal wavelet
   % transform provided in WaveLab ver 0.701.
   for i = 1:D
      Mw(i,1:D) = FWT_PO(M(i,1:D),0,h);
   end;
   for j = 1:D
      Mw(1:D,j) = FWT_PO(Mw(1:D,j),0,h);
   end;

   Fw = FWT_PO(F,0,h);

   % Solve for the standard form of U, using the
   % pre-conditioned matrix operator.
   Uw = P * ( (P * Mw * P) \ (P * Fw) );

   % Take the inverse wavelet transform to get our solution.
   U = IWT_PO(Uw,0,h);


%--- auto_val.m ---|---------|---------|---------|---------|
% This function is used to compute the $\sigma$ values used
% in constructing standard and non-standard matrices for
% derivative operators. Used by both the Bertolluza & Naldi
% code and the Beylkin & Keiser code.
%
% Arguments:
%    h          - a QMF low pass (father) filter
%    p          - the order of the derivative which we
%                 are working with

function R = auto_val( h, p )

   L = length(h);

   % Determine the autocorrelation coefficients of the
   % low-pass Q.M.F.
   a = [2 zeros(1,L-1)];
   for k = 1:2:L-1
```

```
   a(k+1) = 0;
   for i = 0:L-k-1
      a(k+1) = a(k+1) + h(i+1)*h(i+1+k);
   end;
   a(k+1) = 2*a(k+1);
end;

% We will now solve for the $\sigma^0_l$ values. Note
% that $\sigma^p_l \equiv r^p_l = 2^{p} r^0_l$ for the
% $p^{th}$ order derivative.  We will use the method
% outlined in G. Beylkin, "Wavelets and Fast Numerical
% Algorithms", AMS Short Course AMS-93, 1994.

% First we need to build the eigenproblem matrix that
% relates the auto-correlation values to the $\sigma_l$
% values.
M = zeros(2*L-3);
for i = 2-L:L-2
   for j = 1-L:L-1
      if j == 0 & 2-L <= 2*i & 2*i <= L-2
         M(i+L-2+1,2*i+L-2+1) = 2^p;
      elseif 2-L <= 2*i+j & 2*i+j <= L-2
         M(i+L-2+1,2*i+j+L-2+1) = 2^(p-1)*a(abs(j)+1);
      end;
   end;
end;

[v,dd] = eig(M,'nobalance');
                        % - v the eigenvector matrix of M
                        % - dd the diag matrix of eigvals
for i = 1:2*L-3
   if abs(dd(i,i) - 1) < 10^(-5)
                        % - find the eigenvalue near 1
      in = i;           % - the relevant column of v
   end;
end;

% Build the final $\sigma$ vector.
R = v(:,in)';

% Impose the summation condition for uniqueness.
l_vec = [2-L:L-2].^p;
s = sum(R(:) .* l_vec(:));
R = ((-1)^p * prod([1:p]) / s) * R;
```

Appendix B

Amaratunga *et al.* Code

The following list the routines used to implement the first example of the method of Amaratunga *et al.* The code is Matlab 5.2 compliant, and makes use of the wavelet routines in the Wavelet Toolbox package in Matlab.

```
%--- AM_ex1.m -----|---------|---------|---------|---------|
% Uses the method of Amaratunga, Williams, Qian, and Weiss
% to solve the 2nd order ODE problem corresponding to:
%      u^{(2)} + {\pi^2 \over 4} u = {\pi^2 \over 4}
%      u(0) = u0, u(1) = u1
%
% ==> u(x) = (u0-1) cos(\pi x/2) + (u1-1) sin(\pi x/2) + 1
%
% Arguments:
%    m         - log2 of the dimension of V_0
%    s         - amount of offset for boundary sources
%    par       - order of Daubechies wavelet to use
%    u0        - value at left boundary (x=0)
%    u1        - value at right boundary (x=1)
%    p_bool    - indicate if plots should be generated

function AM_ex1( m, s, par, u0, u1, p_bool )

   flops(0);

   % Solve the problem.
   U = AM_solver_ex1(m,s,par,u0,u1);

   % Extract the V_0 part.
   U = U(2^m+1:2^(m+1)+1);

   fcount = flops;

   % Calculate the exact solution.
   [x,E] = AM_ex1_sol(m,u0,u1);

   % Plot the computed and exact solutions.
   if p_bool == 1
```

```
    figure;
    plot(x,U,'r');
    hold on
    plot(x,E(:),'-');
    title('Ex1 - exact blue, computed red');
    hold off
end;

% Calculate the errors.
v = abs(U-E);
v2 = v.^2;
sprintf('linf error: %d',max(v))
sprintf('l1 error:   %d',sum(v))
sprintf('l2 error:   %d',sum(v2))
sprintf('FLOPS:      %d',fcount)
```

```
%--- AM_solver_ex1.m --------|---------|---------|---------|
% Implements Amaratunga et al.'s method for the 1st example.
%
% Arguments:
%    m         - we are working on V_m
%    s         - value for the boundary source offset
%    par       - order of Daubechies wavelet to use
%    u0        - left boundary value
%    u1        - right boundary value

function U = AM_solver_ex1( m, s, par, u0, u1 )

   % Set the shifted boundaries, the amount of the problem
   % shift, and the assumed period.
   a = 1;                % shifted left boundary
   b = 2;                % shifted right boundary
   ps = 1;               % amount of right shift
   d = 3;                % artificial period for the problem

   % Get low-pass Daubechies filter coefficients.
   h = wfilters(strcat('db',int2str(par)),'l');

   % First we solve the problem $ \sL v = \tilde{f}$, for
   % $\tilde{f}$ the shifted version of f.
   F = AM_ex1_f(m, a, b, ps, u0, u1);

   % Take Fast Fourier Transform.
   F_hat = fft(F);

   % Create the operator matrix.
```

```
    Omega = AM_makeL_ex1(m, h, d);

    % Take Fast Fourier Transform.
    Omega_hat = fft(Omega);

    % Solve for $v$.
    v_hat = F_hat ./ Omega_hat;
    v = ifft(v_hat);

    % Now solve the problem $ \sL G = \delta(x) $
    D = [1 zeros(1,2^m*d-1)];
    D_hat = fft(D);

    G_hat = D_hat ./ Omega_hat;
    G = ifft(G_hat);

    % Solve for the $X_{x_i}$ and $X_{x_f}$ values.
    G_matrix = [G(mod(2^m*s,2^m*d)+1), ...
                  G(mod(2^m*(a-b-s),2^m*d)+1); ...
                G(mod(2^m*(b-a+s),2^m*d)+1), ...
                  G(mod(2^m*(-s),2^m*d)+1) ];
    val_vec  = [u0-v(2^m*a+1); u1-v(2^m*b+1)];
    X_vec    = G_matrix \ val_vec;

    % Create w solution.
    for i = 1:2^m*d
       w(i) = X_vec(1) * G(mod(i-2^m*(a-s)-1,2^m*d)+1) + ...
              X_vec(2) * G(mod(i-2^m*(b+s)-1,2^m*d)+1);
    end;

    % Generate the final solution u = v + w.
    U = real(v + w);


%--- AM_makeL_ex1.m ---------|---------|---------|---------|
% Generates the discrete L operator for the first example.
%
% Arguments:
%    m        - we are working on V_m
%    h        - low-pass Q.M.F.
%    d        - (imposed) period of the problem

function dL = AM_makeL_ex1( m, h, d )

   L = length(h);
   A   = CCmatrix(h);      % connection coefficient matrix
   Iden   = eye(L);        % identity matrix
```

```
    % Create La, corresponding to $2^{2*m} {d^2 \over dx^2}$.
    M = Mji(h,2,0);          % find the vec of moment numbers
    [v,dd] = eig(4*A,'nobalance');
                             % v is the eigenvector matrix;
                             % dd the diag matrix of eigenvals
    for i = 1:2*L-3
       if abs(dd(i,i) - 1) < 10^(-5)
          in = i;
       end;
    end;
    s = sum( M(:) .* v(:,in) );
    adj = 2 / s;
    La  = 2^(2*m) * adj * v(:,in);

    % Create Lb, corresponding to ${\pi^2 \over 4}$.
    M = Mji(h,0,0);          % find the vec of moment numbers
    [v,dd] = eig(A,'nobalance');        % as above
    for i = 1:2*L-3
       if abs(dd(i,i) - 1) < 10^(-5)
          in = i;
       end;
    end;
    s = sum( M(:) .* v(:,in) );
    adj = 1 / s;
    Lb  = pi^2 / 4 * adj * v(:,in);

    % ccL is the full form of the discrete L operator
    ccL = La + Lb;

    % Return the convolution kernal.
    vv = zeros(1,2^m*d-2*L+3);
    dL = [ccL(L-1:2*L-3)' vv(1,:) ccL(1:L-2)'];


%--- AM_ex1_f.m ---|---------|---------|---------|---------|
% Evaluates the extended & shifted $f$ function. Note that
% all we really need to do is evalute it on the original
% interval and add enough zeroes on each side.
%
% Arguments:
%    m         - we are working on V_m
%    a         - shifted left boundary
%    b         - shifted right boundary
%    ps        - problem shift parameter
%    u0        - left boundary value
%    u1        - right boundary value
```

```
function f = AM_ex1_f(m, a, b, ps, u0, u1)

   x = [a-ps:2^(-m):b-ps];
   ff = pi^2 / 4 * ones(1,length(x));
   f = [zeros(1,2^m) ff(1,:) zeros(1,2^m-1)];


%--- CCmatrix.m ---|---------|---------|---------|---------|
% This routine creates the connection coefficient matrix
% for the method of Amaratunga et al.  Note that this matrix
% is of dimensions 2L-3 x 2L-3 as we will need 2L-3
% connection coefficients.
%
% Arguments:
%    h         - low-pass Q.M.F.

function A = CCmatrix( h )

   L = length(h);

   M = zeros(2*L-3);
   for l = -L+2:L-2
      for k = -L+2:L-2
         s = 0;
         for p = 0:L-1
            if (k-2*l+p) >= 0 & (k-2*l+p) <= L-1
               s = s + h(p+1)*h(k-2*l+p+1);
            end;
         end;
         A(l+L-2+1,k+L-2+1) = s;
      end;
   end;


%--- ex2.m --------|---------|---------|---------|---------|
% Generates the vector of moment numbers.
%
% Arguments:
%    h         - low-pass Q.M.F.
%    j         - jth moment vector
%    m         - we are working on V_m

function ret = Mji(h, j, m)

   L = length(h);
   if j == 0
```

```
      ret = 1 / sqrt(2^m) * ones(1,2*L-3);
   else
      for k = 2-L:L-2
         M(1,k+L-1) = 1;
      end;

   % Calculate the $M_0^k$  entries for k = 1:j. Note the
   % shift in indices due to matrix indexing in Matlab.
      for k = 1:j
         s = 0;
         for n = 0:k-1
            s_sub = 0;
            for p = 1:L-1
               s_sub = s_sub + h(p+1)*p^(k-n);
            end;
            s = s + prod([1:k]) / (prod([1:n]) * ...
                   prod([1:k-n])) * M(n+1,L-1) * s_sub;
         end;
         M(k+1,L-1) =  s * sqrt(2) / 2 / (2^k - 1);
      end;


   % Calculate $M_n^k$ for k = 1:j and n = 2-L:L-2. Again
   % note shift due to Matlab indexing. Return the vector.
      for k = 1:j
         for n = 2-L:L-2
            s = 0;
            for p = 0:k
              s = s + prod([1:k]) / ...
                      (prod([1:p])*prod([1:k-p])) * ...
                      n^(k-p)*M(p+1,L-1);
            end;
            M(k+1,n+L-1) = s * 2^(-m*(k+.5));
         end;
      end;
      ret = M(j+1,:);
   end;


%--- AM_ex1_sol.m -|---------|---------|---------|---------|
% Calculates the exact solution for this example.
%
% Arguments:
%    m         - we are working in V_m
%    u0        - left boundary value
%    u1        - right boundary value
```

```
function [x,res] = AM_ex1_sol(m, u0, u1)

   x = [0:2^(-m):1];
   res = (u0-1)*cos(pi*x/2) + (u1-1)*sin(pi*x/2) + 1;
```

Appendix C

Beylkin, Keiser, and Vozovoi Code

The following lists the routines used to solve Burgers' equation by Beylkin, Keiser, and Vozovoi's method. The code is Matlab 5.2 compliant, and makes use of the wavelet routines provided in the Matlab Wavelet Toolbox. It should be noted that the routine *auto_val.m*, given in Appendix A, is also used in *NS_func_deriv_op.m*.

```
%--- Burgers.m ----|---------|---------|---------|---------|
% Used to evaluate Burgers' equation by the method of
% Beylkin, Keiser, and Vozovoi.
%
% Arguments:
%    wavelet  - the wavelet basis to use
%    u0       - a vector of length $2^{m_0}$ representing
%               the initial conditions
%    m0       - $log_2$ of the dimension of $V_0$
%    J        - number of sub-spaced to be considered
%    eta      - parameter for the problem
%    dt       - size of time-step to use
%    N        - number of time steps to take
%    E        - epsilon - threshold cutoff value for
%               coefficients in operator matrix
%    plot_s   - number of steps to allow between plots
%    dt       - size of time step to use

function Burgers( wavelet, m0, J, eta, N, E, plot_s, dt )

   % Get the low- and high-pass QMF coefficients
   [h, g] = wfilters(wavelet);

   [u0,x] = feval('Burger_u0',m0);

   % Get the NS-form of our semi-group operator for the
   % linear part.
   L = NS_func_deriv_op( h, g, 'Burgers_L', m0, J, ...
                         eta, dt, E );

   % Get the NS-form operators for the non-linear part.
```

```
Q1 = NS_func_deriv_op( h, g, 'Burgers_Q1', m0, J, ...
                       eta, dt, E );
Q2 = NS_func_deriv_op( h, g, 'Burgers_Q2', m0, J, ...
                       eta, dt, E );
Q3 = NS_func_deriv_op( h, g, 'Burgers_Q3', m0, J, ...
                       eta, dt, E );

N1 =  -1 * NS_func_deriv_op( h, g, 'Burgers_N', m0, ...
                       J, eta, dt, E );

% Use an explicit quadrature method of Beylkin, Keiser,
% and Vozovoi, for d = 1, S = 3.
op1 = dt * ( Q1 + 3/2 * Q2 + Q3 );
op2 = dt * (-2) * ( Q2 + Q3 );
op3 = dt * ( 1/2 * Q2 + Q3 );

% Plot the initial condition.
figure;
plot(x,u0,'r--')
hold on;

% Project u0 into its non-standard form.
u_ns = decomp(h,g,u0,m0,J,0);
u_ns = u_ns';

% First step:
u_1 = u_ns;
u_ns = L * u_1 + ...
       dt * Q1 * N1 * (0.5 * u_1 .* u_1);
if plot_s == 1
   u_plot = NS_recomp(h,g,u_ns',m0,J);
   plot(x,u_plot,'b')
end;


% Second step:
u_11 = u_1;
u_1 = u_ns;
u_ns = L * u_1 + ...
       dt * (Q1+Q2) * N1 * (0.5 * u_1 .* u_1) - ...
       dt * Q2 * N1 * (0.5 * u_11 .* u_11);
if plot_s == 1 | plot_s == 2
   u_plot = NS_recomp(h,g,u_ns',m0,J);
   plot(x,u_plot,'b')
end;

% Successive steps:
```

```
   for j = 3:N
      u_lll = u_ll;
      u_ll = u_l;
      u_l = u_ns;
      u_ns = L * u_l;
      u_ns = L * u_l  + ...
             op1 * N1 * ( 0.5 * u_l .* u_l ) + ...
             op2 * N1 * ( 0.5 * u_ll .* u_ll ) + ...
             op3 * N1 * ( 0.5 * u_lll .* u_lll );

      if mod(j,plot_s) == 0
         u_plot = NS_recomp(h,g,u_ns',m0,J);
         plot(x,u_plot,'b')
      end;
   end;
   hold off;


%--- decomp.m -----|---------|---------|---------|---------|
% This function decomposes a given vector $u$ of length
% $2^{m0}$ into either the usual Mallat wavelet expansion,
% or the non-standard expansion needed for the
% multiplication of NS-form operators with vectors in $V_0$.
%
% Arguments:
%    h       - Q.M.F. low-pass filter coefficients
%    g       - Q.M.F. high-pass filter coefficients
%    u       - the $V_0$ vector to be projected into the
%              desired basis
%    m0      - log2 of the dimension of $V_0$; u should be
%              of length $2^{m_0}$
%    J       - the number of nested subspaces considered
%    Mallat  - flag to indicate if the projected vector
%              should be returned as a Mallat wavelet
%              projection (1), or a so-called NS-form
%              projection (0)

function R = decomp( h, g, u, m0, J, Mallat)

   L = length(g);

   c = zeros(J,2^m0);
   c(1,1:2^m0) = u(1:2^m0);          % - c(1,:) as V0 vec

   for m = 1:J
      [c(m+1,1:2^(m0-m)),d(m+1,1:2^(m0-m))] = ...
           dwtper(c(m,1:2^(m0-m+1)),h,g);
```

```
   end;

   if Mallat == 1          % - return vector in usual form
      R = [];
      for j = 2:J+1
         R = [R, d(j,1:2^(m0-j+1))];
      end;
      R = [R, c(J+1,1:2^(m0-J))];
   else                    % - return vector in NS form
      R = [];
      for j = 2:J+1
         R = [R, d(j,1:2^(m0-j+1)), c(j,1:2^(m0-j+1))];
      end;
   end;


%--- NS_recomp.m --|---------|---------|---------|---------|
% This function recomposes a given vector $u$ (assumed to be
% a NS-form wavelet expansion) into a vector on $V_0$. Hence
% $u$ is assumed to be of the form $\left[ d_{-1} c_{-1}
% d_{-2} c_{-2} \ldots d_{-J} c_{-J} \right]$. So $u$ should
% be % $2^{m_0+1}-2^{m_0-J+1}$ in length.
%
% Arguments:
%    h         - Q.M.F. low-pass filter coefficients
%    g         - Q.M.F. high-pass filter coefficients
%    u         - the NS-form vector to be projected
%    m0        - $log_2$ of the dimension of V0
%    J         - number of nested subspaces considered

function R = NS_recomp( h, g, u, m0, J)

   L = length(h);

   % Extract $d_{-J}$ and $c_{-J}$ from $u$
   ind = 2^(m0+1) - 2^(m0-J+1) - 2^(m0-J) + 1;
   c( J, 1:2^(m0-J) ) = u( ind:ind+2^(m0-J)-1 );
   ind = ind - 2^(m0-J);
   dhat( J, 1:2^(m0-J) ) = u( ind:ind+2^(m0-J)-1 );

   % Here we will follow the approach described in the
   % Beylkin and Keiser paper, "An Adaptive Pseudo-
   % Wavelet Approach for Solving Nonlinear Partial
   % Differential Equations," pp. 156-157.

   for m = J-1:-1:1
      ind = ind - 2^(m0-m);
```

```
      chat(m,1:2^(m0-m)) = u(ind:ind+2^(m0-m)-1);
      ind = ind - 2^(m0-m);
      dhat(m,1:2^(m0-m)) = u(ind:ind+2^(m0-m)-1);

      ctilde(m,1:2^(m0-m)) = idwtper(c(m+1,1:2^(m0-m-1)),...
         dhat(m+1,1:2^(m0-m-1)),reverse(h),reverse(g));
      c(m,1:2^(m0-m)) = ctilde(m,1:2^(m0-m)) + ...
                        chat(m,1:2^(m0-m));
   end;

   R = idwtper(c(1,1:2^(m0-1)),dhat(1,1:2^(m0-1)),...
               reverse(h),reverse(g));


%--- NS_func_deriv_op.m -----|---------|---------|---------|
% Generates the NS-form matrix for the function $F$ of the
% first order spacial derivative operator. This code follows
% the second approach taken in Section 3.4 of the thesis.
%
% Arguments:
%    h      - the low-pass Q.M.F. coefficients
%    g      - the high-pass Q.M.F. coefficients
%    F      - the function in question
%    m0     - log2 of the dimension of $V_0$
%    J      - the number of scales involved ($V_{-J}$
%             corresponds to the coarsest sub-space)
%    eta    - parameter for the problem
%    dt     - size of the time step
%    E      - epsilon - threshold cutoff value
%
% Note: NS form of op should have dimensions
%  $2^{m_0+1} - 2^{m_0+1-J} \times
%                          2^{m_0+1} - 2^{m_0+1-J}$

function M = NS_func_deriv_op( h, g, F, m0, J, eta, dt, E )

   L = length(h);
   N = 2*L-3;

   % Get the $\sigma_0$ values of the first order derivative
   sig = auto_val(h,1);

   % Now set the eigenvalues needed as per Section 3.4.
   sum_vec = 2*pi*sqrt(-1)/N*[2-L:L-2];
   lambda = exp( sum_vec' * [2-L:L-2] ) * sig';

   % Determine $\sigma^0$ for $f(P_0 d_x P_0)$
```

```
% Take the real part (okay, since the imaginary part is
% zero anyway, but machine error introduces a 0i term
% that creates trouble in Matlab later).
f_vec = feval(F,lambda,eta,dt,m0);

sigma_0 = real( exp(sum_vec'*[2-L:L-2]) * f_vec ) / N;

% Build the NS-form matrix use sparse matrices.
M = zeros(1,1);
M = sparse(M);
M(2^(m0+1)-2^(m0+1-J),2^(m0+1)-2^(m0+1-J)) = 0;

start = 1;
shift = 0;
sigma_mp1 = sigma_0;

for m = 1:J                    % - sub-block number

   % Create the sub-block entry values
   sigma_m = zeros(N,1);
   alpha_m = zeros(N,1);
   beta_m  = zeros(N,1);
   gamma_m = zeros(N,1);
   for l = 2-L:L-2
      for k = 0:L-1
         for kp = 0:L-1
            ind = 2*l+k-kp;
            if (2-L <= ind & ind <= L-2)
               sigma_m(l+L-2+1) = sigma_m(l+L-2+1) + ...
                  h(k+1)*h(kp+1)*sigma_mp1(ind+L-2+1);
               alpha_m(l+L-2+1) = alpha_m(l+L-2+1) + ...
                  g(k+1)*g(kp+1)*sigma_mp1(ind+L-2+1);
               beta_m(l+L-2+1)  = beta_m(l+L-2+1) + ...
                  g(k+1)*h(kp+1)*sigma_mp1(ind+L-2+1);
               gamma_m(l+L-2+1) = gamma_m(l+L-2+1) + ...
                  h(k+1)*g(kp+1)*sigma_mp1(ind+L-2+1);
            end;
         end;
      end;
   end;

   sigma_mp1 = sigma_m;

   start = start + 2*shift;    % - starting index for
                               %   this set of sub-blocks

   shift = 2^(m0-m);           % - shift position for B
```

```
                                        %   and G sub-blocks
        end_ = start+shift-1;

        % Place values into the sub-blocks
        for i = 1:shift
           for j = 2-L:L-2
              ind = mod( -j + mod(i-1,shift), shift ) + 1;
             % $\Alpha$ block
              if abs(alpha_m(j+L-2+1)) >= E
                 M(start+i-1,start+ind-1) = ...
                      M(start+i-1,start+ind-1) + ...
                                        alpha_m(j+L-2+1);
              end;
             % $\Beta$ block
              if abs(beta_m(j+L-2+1)) >= E
                 M(start+i-1,start+ind-1+shift) = ...
                      M(start+i-1,start+ind-1+shift) + ...
                                        beta_m(j+L-2+1);
              end;
             % $\Gamma$ block
              if abs(gamma_m(j+L-2+1)) >= E
                 M(start+i-1+shift,start+ind-1) = ...
                      M(start+i-1+shift,start+ind-1) + ...
                                        gamma_m(j+L-2+1);
              end;
             % $\T$ block
              if m == J
                 if abs(sigma_m(j+L-2+1)) >= E
                M(start+i-1+shift,start+ind-1+shift) = ...
                  M(start+i-1+shift,start+ind-1+shift) + ...
                                        sigma_m(j+L-2+1);
                 end;
              end;
           end;
        end;
     end;


%--- Burgers_u0.m -|---------|---------|---------|---------|
% Generates the initial conditions for the given example
% of solving Burgers' equation.

function [u0,x] = Burgers_u0( m0 )

   x = [2^(-m0-1):2^(-m0):1-2^(-m0-1)];
   u0 = sin(2*pi*x);
```

```
%--- Burgers_L.m --|---------|---------|---------|---------|
% This function computes the exact-part exponential operator
% at the point $x$ for parameter $\eta$ and time step
% $\delta t$.

% Arguments:
%    x        - point to be considered
%    eta      - parameter from the problem
%    dt       - size of time step

function R = Burgers_L( x, eta, dt, m0 )

   R = exp( eta * dt * ((2^m0 - 1) * x).^2 );


%--- Burgers_Q1.m -|---------|---------|---------|---------|
% This function computes the Q_1 operator at the point $x$
% for time step $\delta t$. Uses a Taylor expansion
% approximation, truncated when machine calculation limit
% is reached in the l1 norm.

% Arguments:
%    x        - point to be considered
%    eta      - parameter from problem
%    dt       - size of time step

function R = Burgers_Q1( x, eta, dt, m0 )

   v = eta * dt * ((2^m0 - 1) * x).^2;

   s_new = 0;
   s_old = 1;

   k = 0;

   while abs(sum(norm(s_new - s_old))) > 0
      s_old = s_new;
      s_new = s_new + v.^k / prod([1:k+1]);
      k = k+1;
   end;
   R = s_new;


%--- Burgers_Q2.m -|---------|---------|---------|---------|

function R = Burgers_Q2( x, eta, dt, m0 )
```

```
   v = eta * dt * ((2^m0 - 1) * x).^2;

   s_old = 1;
   s_new = 0;
   k = 0;

   while abs(sum(norm(s_new - s_old))) > 0
      s_old = s_new;
      s_new = s_new + v.^k / prod([1:k+2]);
      k = k+1;
   end;
   R = s_new;


%--- Burgers_Q3.m -|---------|---------|---------|---------|

function R = Burgers_Q3( x, eta, dt, m0 )

   v = eta * dt * ((2^m0 - 1) * x).^2;

   s_old = 1;
   s_new = 0;
   k = 0;
   while abs(sum(norm(s_new - s_old))) > 0
      s_old = s_new;
      s_new = s_new + v.^k / prod([1:k+3]);
      k = k+1;
   end;
   R = s_new;


%--- Burgers_N.m --|---------|---------|---------|---------|
% This function is used in computing the first order
% derivative at the point $s$ for parameter $\eta$ and
% time step $\delta t$.
%
% Arguments:
%    x        - point to be considered
%    eta      - parameter from the problem
%    dt       - size of time step

function R = Burgers_N( x, eta, dt, m0 )

   R = (2^m0 - 1) * x;
```