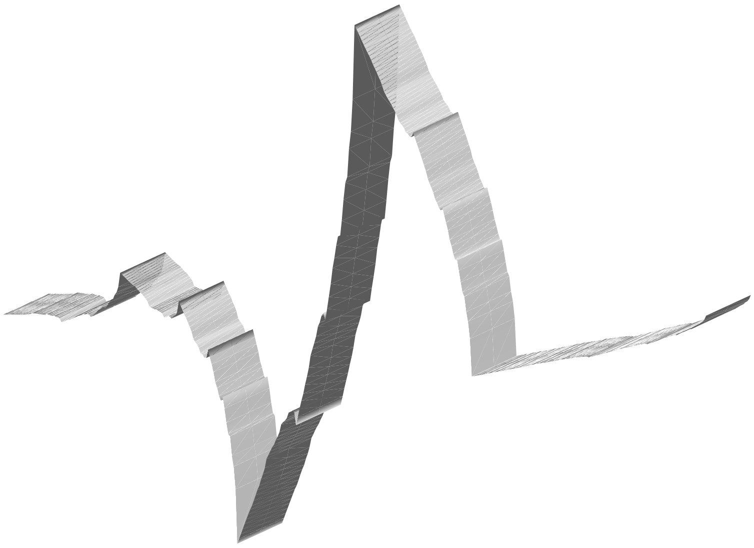
Wavelets in Scientific Computing

Ph.D. Dissertation

by

Ole Møller Nielsen

uniomni@uni-c.dk
http://www.imm.dtu.dk/~omni



Department of Mathematical Modelling
Technical University of Denmark
DK-2800 Lyngby
Denmark

UNI•C
Technical University of Denmark
DK-2800 Lyngby
Denmark

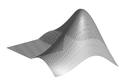
Preface

This Ph.D. study was carried out at the Department of Mathematical Modelling, Technical University of Denmark and at UNI•C, the Danish Computing Centre for Research and Education. It has been jointly supported by UNI•C and the Danish Natural Science Research Council (SNF) under the program Efficient Parallel Algorithms for Optimization and Simulation (EPOS).

The project was motivated by a desire in the department to generate knowledge about wavelet theory, to develop and analyze parallel algorithms, and to investigate wavelets' applicability to numerical methods for solving partial differential equations. Accordingly, the report falls into three parts:

Part I: Wavelets: Basic Theory and Algorithms.
Part II: Fast Wavelet Transforms on Supercomputers.
Part III: Wavelets and Partial Differential Equations.

Wavelet analysis is a young and rapidly expanding field in mathematics, and there are already a number of excellent books on the subject. Important examples are [SN96, Dau92, HW96, Mey93, Str94]. However, it would be almost impossible to give a comprehensive account of wavelets in a single Ph.D. study, so we have limited ourselves to one particular wavelet family, namely the compactly supported orthogonal wavelets. This family was first described by Ingrid Daubechies [Dau88], and it is particularly attractive because there exist fast and accurate algorithms for the associated transforms, the most prominent being the pyramid algorithm which was developed by Stephane Mallat [Mal89].



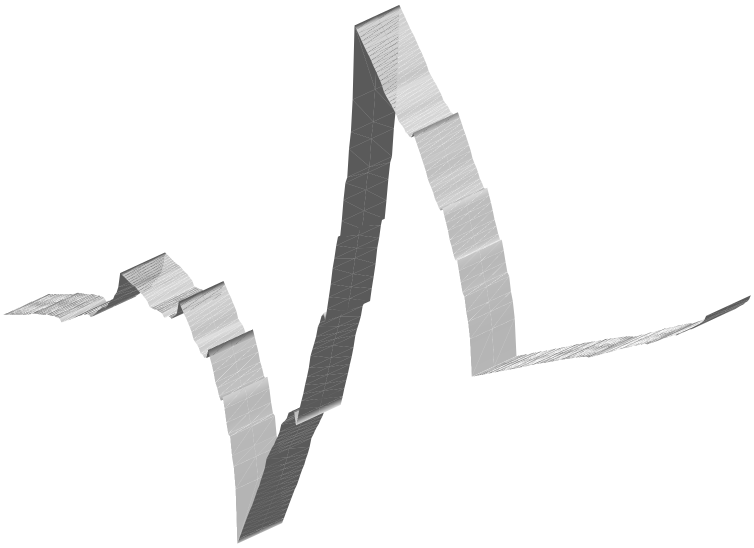
Our focus is on algorithms and we provide Matlab programs where applicable. This will be indicated by the margin symbol shown here. The Matlab package is available on the World Wide Web at

```
http://www.imm.dtu.dk/~omni/wapa20.tgz
```

and its contents are listed in Appendix E.

We have tried our best to make this exposition as self-contained and accessible as possible, and it is our sincere hope that the reader will find it a help for understanding the underlying ideas and principles of wavelets as well as a useful collection of recipes for applied wavelet analysis.

I would like to thank the following people for their involvement and contributions to this study: My advisors at the Department of Mathematical Modelling, Professor Vincent A. Barker, Professor Per Christian Hansen, and Professor Mads Peter Sørensen. In addition, Dr. Markus Hegland, Computer Sciences Laboratory, RSISE, Australian National University, Professor Lionel Watkins, Department of Physics, University of Auckland, Mette Olufsen, Math-Tech, Denmark, Iestyn Pierce, School of Electronic Engineering and Computer Systems, University of Wales, and last but not least my family and friends.



Lyngby, March 1998

Ole Møller Nielsen

Abstract

Wavelets in Scientific Computing

Wavelet analysis is a relatively new mathematical discipline which has generated much interest in both theoretical and applied mathematics over the past decade. Crucial to wavelets are their ability to analyze different parts of a function at different scales and the fact that they can represent polynomials up to a certain order exactly. As a consequence, functions with fast oscillations, or even discontinuities, in localized regions may be approximated well by a linear combination of relatively few wavelets. In comparison, a Fourier expansion must use many basis functions to approximate such a function well. These properties of wavelets have lead to some very successful applications within the field of signal processing. This dissertation revolves around the role of wavelets in scientific computing and it falls into three parts:

Part I gives an exposition of the theory of orthogonal, compactly supported wavelets in the context of multiresolution analysis. These wavelets are particularly attractive because they lead to a stable and very efficient algorithm, namely the fast wavelet transform (FWT). We give estimates for the approximation characteristics of wavelets and demonstrate how and why the FWT can be used as a front-end for efficient image compression schemes.

Part II deals with vector-parallel implementations of several variants of the Fast Wavelet Transform. We develop an efficient and scalable parallel algorithm for the FWT and derive a model for its performance.

Part III is an investigation of the potential for using the special properties of wavelets for solving partial differential equations numerically. Several approaches are identified and two of them are described in detail. The algorithms developed are applied to the nonlinear Schrödinger equation and Burgers' equation. Numerical results reveal that good performance can be achieved provided that problems are large, solutions are highly localized, and numerical parameters are chosen appropriately, depending on the problem in question.

Resume på dansk

Wavelets i Scientific Computing

Waveletteori er en forholdsvis ny matematisk disciplin, som har vakt stor interesse indenfor både teoretisk og anvendt matematik i løbet af det seneste årti. De altafgørende egenskaber ved wavelets er at de kan analysere forskellige dele af en funktion på forskellige skalatrin, samt at de kan repræsentere polynomier nøjagtigt op til en given grad. Dette fører til, at funktioner med hurtige oscillationer eller singulariteter indenfor lokaliserede områder kan approksimeres godt med en linearkombination af forholdsvis få wavelets. Til sammenligning skal man medtage mange led i en Fourierrække for at opnå en god tilnærmelse til den slags funktioner. Disse egenskaber ved wavelets har med held været anvendt indenfor signalbehandling. Denne afhandling omhandler wavelets rolle indenfor scientific computing og den består af tre dele:

Del I giver en gennemgang af teorien for ortogonale, kompakt støttede wavelets med udgangspunkt i multiskala analyse. Sådanne wavelets er særligt attraktive, fordi de giver anledning til en stabil og særdeles effektiv algoritme, kaldet den hurtige wavelet transformation (FWT). Vi giver estimater for approksimationsegenskaberne af wavelets og demonstrerer, hvordan og hvorfor FWT-algoritmen kan bruges som første led i en effektiv billedkomprimerings metode.

Del II omhandler forskellige implementeringer af FWT algoritmen på vektorcomputere og parallelle datamater. Vi udvikler en effektiv og skalerbar parallel FWT algoritme og angiver en model for dens ydeevne.

Del III omfatter et studium af mulighederne for at bruge wavelets særlige egenskaber til at løse partielle differentialligninger numerisk. Flere forskellige tilgange identificeres og to af dem beskrives detaljeret. De udviklede algoritmer anvendes på den ikke-lineære Schrödinger ligning og Burgers ligning. Numeriske undersøgelser viser, at algoritmerne kan være effektive under forudsætning af at problemerne er store, at løsningerne er stærkt lokaliserede og at de forskellige numeriske metode-parametre kan vælges på passende vis afhængigt af det pågældende problem.

Notation

Symbol	Page	Description
$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$		Generic matrices
$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$		Generic vectors
$\overline{a}_p$	110	$\overline{a}_p = \sum_r a_r a_{r-p}$
$A(\xi)$	25	$A(\xi) = 2^{-1/2} \sum_{k=0}^{D-1} a_k e^{-ik\xi}$
a_k	16	Filter coefficient for ϕ
$B(\xi)$	29	$B(\xi) = 2^{-1/2} \sum_{k=0}^{D-1} b_k e^{-ik\xi}$
$B(\boldsymbol{D})$	116	Bandwidth of matrix $\boldsymbol{D}$
b_k	16	Filter coefficient for ψ
$\boldsymbol{c}$	49	Vector containing scaling function coefficients
$\boldsymbol{c}_u$	163	Vector containing scaling function coefficients with respect to the function u
$c_{j,k}$	6	Scaling function coefficient
c_k	3	Coefficient in Fourier expansion
C_P	23	$1/(P!) \int_0^{D-1} \left\| y^P \psi(y) \right\| \, dy$
C_ψ	62	$C_\psi = \max_{y \in [0,D-1]} \|\psi(y)\|$
$CC^{i,j}, CD^{i,j}, DC^{i,j}, DD^{i,j}$	122	Shift-circulant block matrices
$cc^{i,j}, cd^{i,j}, dc^{i,j}, dd^{i,j}$	128	Vectors representing shift-circulant block matrices
D	16	Wavelet genus - number of filter coefficients
$\boldsymbol{D}_F^{(d)}$	216	Fourier differentiation matrix
$\boldsymbol{D}^{(d)}$	113	Scaling function differentiation matrix
$\breve{\boldsymbol{D}}^{(d)}$	119	Wavelet differentiation matrix
$\boldsymbol{d}$	57	Vector containing wavelet coefficients
$\boldsymbol{d}_u$	166	Vector containing wavelet coefficients with respect to the function u
$d_{j,k}$	6	Wavelet coefficient
$\boldsymbol{E}$	175	$\boldsymbol{E} = \exp\left(\frac{\Delta t}{2} \boldsymbol{L}\right)$
$e_J(x)$	61	Pointwise approximation error
$\tilde{e}_J(x)$	63	Pointwise (periodic) approximation error
f, g, h, u, v	3	Generic functions
$\boldsymbol{F}_N$	210	Fourier matrix
$\hat{f}(\xi)$	25	Continuous Fourier transform $\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-i\xi x} \, dx$

Symbol	Page	Description
$\boldsymbol{H}$	121	Wavelet transform of a circulant matrix
$\boldsymbol{H}^{i,j}$	136	Block matrix of $\boldsymbol{H}$
$\boldsymbol{I}_{j,k}$	17	Support of $\phi_{j,k}$ and $\psi_{j,k}$
J_0	6	Integer denoting coarsest approximation
J	6	Integer denoting finest approximation
L	146	Bandwidth (only in Chapter 8)
L	174	Length of period (only in Chapter 9)
$L^{i,j}$	136	Bandwidth of block $\boldsymbol{H}^{i,j}$ (only in Chapter 8)
$\mathcal{L}$	174	Linear differential operator
$\boldsymbol{L}$	174	Matrix representation of $\mathcal{L}$
M_k^p	19	pth moment of $\phi(x-k)$
N	5	Length of a vector
$\mathbf{N}$	47	Set of positive integers
$\mathbf{N}_0$	27	Set of non-negative integers
$N_r(\varepsilon)$	65	Number of insignificant elements in wavelet expansion
$N_s(\varepsilon)$	64	Number of significant elements in wavelet expansion
$\boldsymbol{N}$	174	Matrix representation of nonlinearity
P	19	Number of vanishing moments $P = D/2$
P	85	Number of processors (only in Chapter 6)
$P_{V_j} f$	15	Orthogonal projection of f onto V_j
$P_{W_j} f$	15	Orthogonal projection of f onto W_j
$P_{\tilde{V}_j} f$	41	Orthogonal projection of f onto $\tilde{V}_j$
$P_{\tilde{W}_j} f$	41	Orthogonal projection of f onto $\tilde{W}_j$
$\mathbf{R}$	3	Set of real numbers
S_i	58	$S_i = N/2^i$, size of a subvector at depth i
S_i^P	88	$S_i^P = S_i/P$, size of a subvector at depth i on one of P processors
$\boldsymbol{T}$	49	Matrix mapping scaling function coefficients to function values: $\boldsymbol{f} = \boldsymbol{T}\boldsymbol{c}$
u_J	163	Approximation to the function u
V_j	11	jth approximation space, $V_j \in L^2(\mathbf{R})$
$\tilde{V}_j$	7	jth periodized approximation space, $\tilde{V}_j \in L^2([0,1])$
W_j	11	jth detail space, $W_j \perp V_j$ and $W_j \in L^2(\mathbf{R})$
$\tilde{W}_j$	7	jth periodized detail space, $\tilde{W}_j \perp \tilde{V}_j$ and $\tilde{W}_j \in L^2([0,1])$
$\boldsymbol{W}^{\lambda}$	58	Wavelet transform matrix: $\boldsymbol{d} = \boldsymbol{W}^{\lambda}\boldsymbol{c}$
$\breve{\boldsymbol{X}}$	60	Wavelet transform of matrix $\boldsymbol{X}$: $\breve{\boldsymbol{X}} = \boldsymbol{W}^{\lambda_M}\boldsymbol{X}\boldsymbol{W}^{\lambda_N}$
$\breve{\boldsymbol{X}}^{\varepsilon}$	69	Wavelet transformed and truncated matrix
$\breve{\boldsymbol{x}}$	59	Wavelet transform of vector $\boldsymbol{x}$: $\breve{\boldsymbol{x}} = \boldsymbol{W}^{\lambda}\boldsymbol{x}$
$\mathbf{Z}$	11	Set of integers

Symbol	Page	Description
α	163	Constant in Helmolz equation
β_2, β_3	174	Dispersion constants
Γ_l^d	108	Connection coefficient
$\mathbf{\Gamma}^d$	109	Vector of connection coefficients
γ	174	Nonlinearity factor
$\delta_{k,l}$	14	Kronecker delta
ε_V	171	Threshold for vectors
ε_M	171	Threshold for matrices
ε_D	187	Special fixed threshold for differentiation matrix
$\mathbf{\Lambda}_{\boldsymbol{a}}$	212	Diagonal matrix
$\lambda, \lambda_M, \lambda_N$	15	Depth of wavelet decomposition
ν	168	Diffusion constant
ξ	23	Substitution for x or
	25	Variable in Fourier transform
ρ	186	Advection constant
$\phi(x)$	11, 13	Basic scaling function
$\phi_{j,k}(x)$	13	$\phi_{j,k}(x) = 2^{j/2}\phi(2^j x - k)$
$\phi_k(x)$	13	$\phi_k = \phi_{0,k}(x)$
$\tilde{\phi}_{j,k}(x)$	6, 34	Periodized scaling function
$\psi(x)$	13	Basic wavelet
$\psi_{j,k}(x)$	13	$\psi_{j,k}(x) = 2^{j/2}\phi(2^j x - k)$
$\psi_k(x)$	13	$\psi_k = \psi_{0,k}(x)$
$\tilde{\psi}_{j,k}(x)$	6, 34	Periodized wavelet
ω_N	210	$\omega_N = e^{i2\pi/N}$

Symbols parametrized by algorithms

Symbol	Page	Description
$F_{\mathcal{A}}(N)$	59	Complexity of algorithm $\mathcal{A}$
$C_{\mathcal{A}}$	93	Communication time for algorithm $\mathcal{A}$
$T_{\mathcal{A}}(N)$	76	Execution time for Algorithm $\mathcal{A}$
$T_{\mathcal{A}}^{P}(N)$	93	Execution time for Algorithm $\mathcal{A}$ on P processors
$T_{\mathcal{A}}^{0}(N)$	93	Sequential execution time for Algorithm $\mathcal{A}$
$E_{\mathcal{A}}^{P}(N)$	94	Efficiency of Algorithm $\mathcal{A}$ on P processors
$S_{\mathcal{A}}^{P}(N)$	97	Speedup of Algorithm $\mathcal{A}$ on P processors

The symbol $\mathcal{A}$ is one the following algorithms:

Algorithm ($\mathcal{A}$)	Page	Description
PWT	53	Partial Wavelet Transform; one step of the FWT
FWT	53	Fast Wavelet Transform
FWT2	60	2D FWT
MFWT	79	Multiple 1D FWT
RFWT	95	Replicated FWT algorithm
CFWT	97	Communcation-efficient FWT algorithm
CIRPWT1	134	Circulant PWT (diagonal block)
CIRPWT2	134	Circulant PWT (off-diagonal blocks)
CIRFWT	139	Circulant 2D FWT
CIRMUL	158	Circulant matrix multiplication in a wavelet basis

Miscellaneous

Symbol	Page	Description
$\|u\|_\infty$	63	Infinity norm for functions. $\|u\|_\infty = \max_x \|u(x)\|$
$\|u\|_{J,\infty}$	165	Pointwise infinity norm for functions. $\|u\|_{J,\infty} = \max_k \left\|u(k/2^J)\right\|$
$\|\boldsymbol{u}\|_\infty$	173	Infinity norm for vectors. $\|\boldsymbol{u}\|_\infty = \max_k \|u_k\|$
$\|u\|_2$	11	2-norm for functions. $\|u\|_2 = (\int \|u(x)\|^2 \, dx)^{1/2}$
$\|\boldsymbol{u}\|_2$		2 norm for vectors. $\|\boldsymbol{u}\|_2 = (\sum_k \|u_k\|^2)^{1/2}$
$\lceil x \rceil$	36	The smallest integer greater than x
$\lfloor x \rfloor$	39	The largest integer smaller than x
$[x]$	205	Nearest integer towards zero
$\langle n \rangle_p$	205	Modulus operator (n mod p)
$[\boldsymbol{x}]_n, x_n$	212	The nth element in vector $\boldsymbol{x}$
$[\boldsymbol{X}]_{m,n}, X_{m,n}$	209	The m, nth element in matrix $\boldsymbol{X}$

Contents

Part I

Wavelets: Basic Theory and Algorithms

Chapter 1

Motivation

This section gives an introduction to wavelets accessible to non-specialists and serves at the same time as an introduction to key concepts and notation used throughout this study.

The wavelets considered in this introduction are called *periodized Daubechies wavelets of genus four* and they constitute a specific but representative example of wavelets in general. For the notation to be consistent with the rest of this work, we write our wavelets using the symbol $\tilde{\psi}_{j,l}$, the tilde signifying periodicity.

1.1 Fourier expansion

Many mathematical functions can be represented by a sum of fundamental or simple functions denoted basis functions. Such representations are known as expansions or series, a well-known example being the Fourier expansion

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{i2\pi kx}, \quad x \in \mathbf{R} \tag{1.1}$$

which is valid for any reasonably well-behaved function f with period 1. Here, the basis functions are complex exponentials $e^{i2\pi kx}$ each representing a particular frequency indexed by k. The Fourier expansion can be interpreted as follows: If f is a periodic signal, such as a musical tone, then (1.1) gives a decomposition of f as a superposition of harmonic modes with frequencies k (measured by cycles per time unit). This is a good model for vibrations of a guitar string or an air column in a wind instrument, hence the term “harmonic modes”.

The coefficients c_k are given by the integral

$$c_k = \int_0^1 f(x) e^{-i2\pi kx}\, dx$$

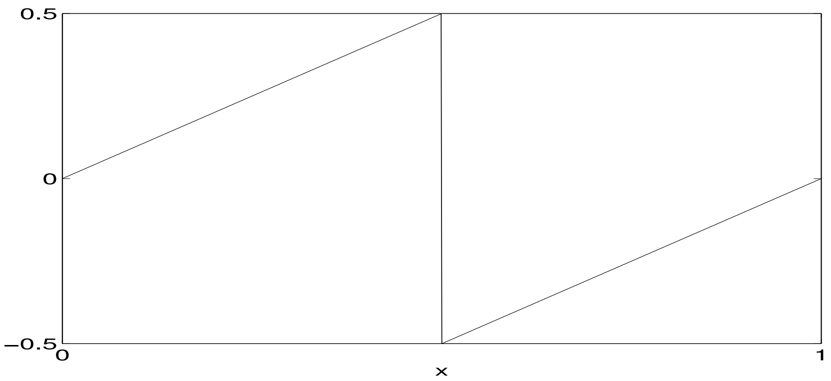


Figure 1.1: The function f.

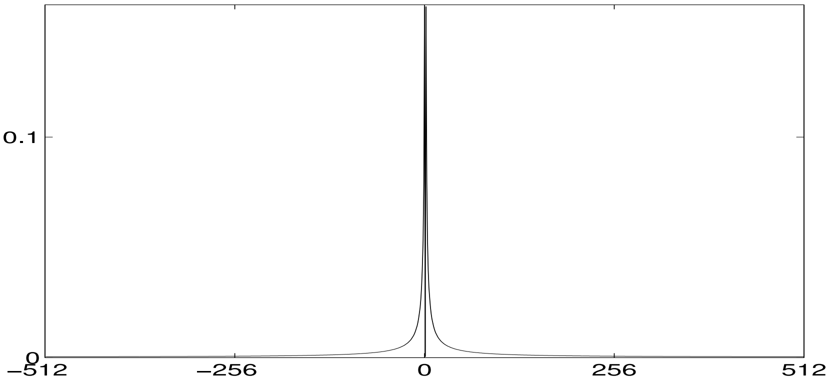


Figure 1.2: Fourier coefficients of f.

Each coefficient c_k can be conceived as the average harmonic content (over one period) of f at frequency k. The coefficient c_0 is the average at frequency 0, which is just the ordinary average of f. In electrical engineering this term is known as the "DC" term. The computation of c_k is called the *decomposition* of f and the series on the right hand side of (1.1) is called the *reconstruction* of f.

In theory, the reconstruction of f is exact, but in practice this is rarely so. Except in the occasional event where (1.1) can be evaluated analytically it must be truncated in order to be computed numerically. Furthermore, one often wants to save computational resources by discarding many of the *smallest* coefficients c_k. These measures naturally introduce an approximation error.

To illustrate, consider the sawtooth function

$$f(x) = \begin{cases} x & 0 \leq x < 0.5 \\ x - 1 & 0.5 \leq x < 1 \end{cases}$$

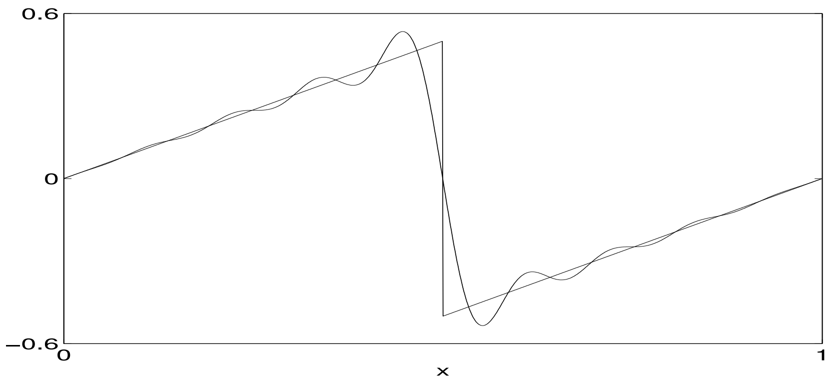


Figure 1.3: A function f and a truncated Fourier expansion with only 17 terms

which is shown in Figure 1.1. The Fourier coefficients c_k of the truncated expansion

$$\sum_{k=-N/2+1}^{N/2} c_k e^{i2\pi kx}$$

are shown in Figure 1.2 for $N = 1024$.

If, for example, we retain only the 17 largest coefficients, we obtain the truncated expansion shown in Figure 1.3. While this approximation reflects *some* of the behavior of f, it does not do a good job for the discontinuity at $x = 0.5$. It is an interesting and well-known fact that such a discontinuity is perfectly resolved by the series in (1.1), even though the individual terms themselves are continuous. However, with only a finite number of terms this will not be the case. In addition, and this is very unfortunate, the approximation error is not restricted to the discontinuity but spills into much of the surrounding area. This is known as the Gibbs phenomenon.

The underlying reason for the poor approximation of the discontinuous function lies in the nature of complex exponentials, as they all cover the entire interval and differ only with respect to frequency. While such functions are fine for representing the behavior of a guitar string, they are not suitable for a discontinuous function. Since each of the Fourier coefficient reflects the *average* content of a certain frequency, it is impossible to see where a singularity is located by looking only at individual coefficients. The information about position can be recovered only by computing all of them.

1.2 Wavelet expansion

The problem mentioned above is one way of motivating the use of wavelets. Like the complex exponentials, wavelets can be used as basis functions for the expansion of a function f. Unlike the complex exponentials, they are able to capture the positional information about f as well as information about scale. The latter is essentially equivalent to frequency information. A wavelet expansion for a 1-periodic function f has the form

$$\boxed{f(x) = \sum_{k=0}^{2^{J_0}-1} c_{J_0,k}\tilde{\phi}_{J_0,k}(x) + \sum_{j=J_0}^{\infty}\sum_{k=0}^{2^j-1} d_{j,k}\tilde{\psi}_{j,k}(x), \quad x \in \mathbf{R}} \tag{1.2}$$

where J_0 is a non-negative integer. This expansion is similar to the Fourier expansion (1.1): It is a linear combination of a set of basis functions, and the wavelet coefficients are given by

$$\begin{aligned} c_{J_0,k} &= \int_0^1 f(x)\tilde{\phi}_{J_0,k}(x)\,dx \\ d_{j,k} &= \int_0^1 f(x)\tilde{\psi}_{j,k}(x)\,dx \end{aligned}$$

One immediate difference with respect to the Fourier expansion is the fact that now we have two types of basis functions and that both are indexed by two integers. The $\tilde{\phi}_{J_0,k}$ are called scaling functions and the $\tilde{\psi}_{j,k}$ are called wavelets. Both have compact support such that

$$\tilde{\phi}_{j,k}(x) = \tilde{\psi}_{j,k}(x) = 0 \quad \text{for} \quad x \notin \left[\frac{k}{2^j}, \frac{k+3}{2^j}\right]$$

We call j the **scale parameter** because it scales the width of the support, and k the **shift parameter** because it translates the support interval. There are generally no explicit formulas for $\tilde{\phi}_{j,k}$ and $\tilde{\psi}_{j,k}$ but their function values are computable and so are the above coefficients. The scaling function coefficient $c_{J_0,k}$ can be interpreted as a local weighted average of f in the region where $\tilde{\phi}_{J_0,k}$ is non-zero. On the other hand, the wavelet coefficients $d_{j,k}$ represent the opposite property, namely the details of f that are lost in the weighted average.

In practice, the wavelet expansion (like the Fourier expansion) must be truncated at some finest scale which we denote $J-1$: The truncated wavelet expansion is

$$\sum_{k=0}^{2^{J_0}-1} c_{J_0,k}\tilde{\phi}_{J_0,k}(x) + \sum_{j=J_0}^{J-1}\sum_{k=0}^{2^j-1} d_{j,k}\tilde{\psi}_{j,k}(x)$$

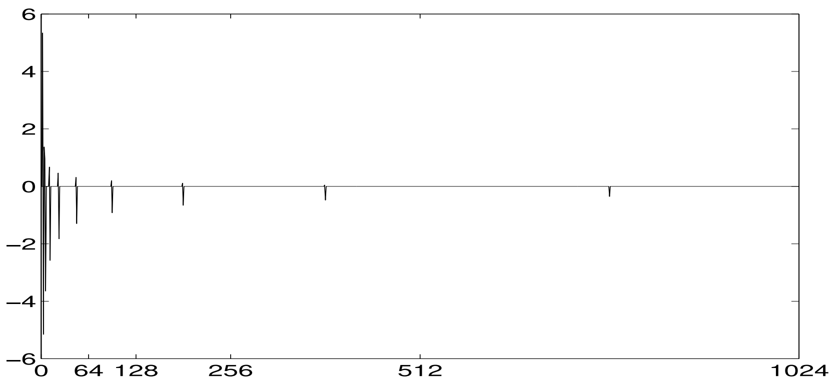


Figure 1.4: Wavelet coefficients of f.

and the wavelet coefficients ordered as

$$\left\{ \{c_{J_0,k}\}_{k=0}^{2^{J_0}-1}, \quad \{\{d_{j,k}\}_{k=0}^{2^j-1}\}_{j=J_0}^{J-1} \right\}$$

are shown in Figure 1.4. The wavelet expansion (1.2) can be understood as follows: The first sum is a coarse representation of f, where f has been replaced by a linear combination of 2^{J_0} translations of the scaling function $\tilde{\phi}_{J_0,0}$. This corresponds to a Fourier expansion where only low frequencies are retained. The remaining terms are refinements. For each j a layer represented by 2^j translations of the wavelet $\tilde{\psi}_{j,0}$ is added to obtain a successively more detailed approximation of f. It is convenient to define the approximation spaces

$$\boxed{\begin{aligned} \tilde{V}_j &= \text{span}\{\tilde{\phi}_{j,k}\}_{k=0}^{2^j-1} \\ \tilde{W}_j &= \text{span}\{\tilde{\psi}_{j,k}\}_{k=0}^{2^j-1} \end{aligned}}$$

These spaces are related such that

$$\tilde{V}_J = \tilde{V}_{J_0} \oplus \tilde{W}_{J_0} \oplus \cdots \oplus \tilde{W}_{J-1}$$

The coarse approximation of f belongs to the space $\tilde{V}_{J_0}$ and the successive refinements are in the spaces $\tilde{W}_j$ for $j = J_0, J_0 + 1, \ldots, J - 1$. Together, all of these contributions constitute a refined approximation of f. Figure 1.5 shows the scaling functions and wavelets corresponding to $\tilde{V}_2$, $\tilde{W}_2$ and $\tilde{W}_3$.

Figure 1.5: There are four scaling functions in $\tilde{V}_2$ and four wavelets in $\tilde{W}_2$ but eight more localized wavelets in $\tilde{W}_3$.

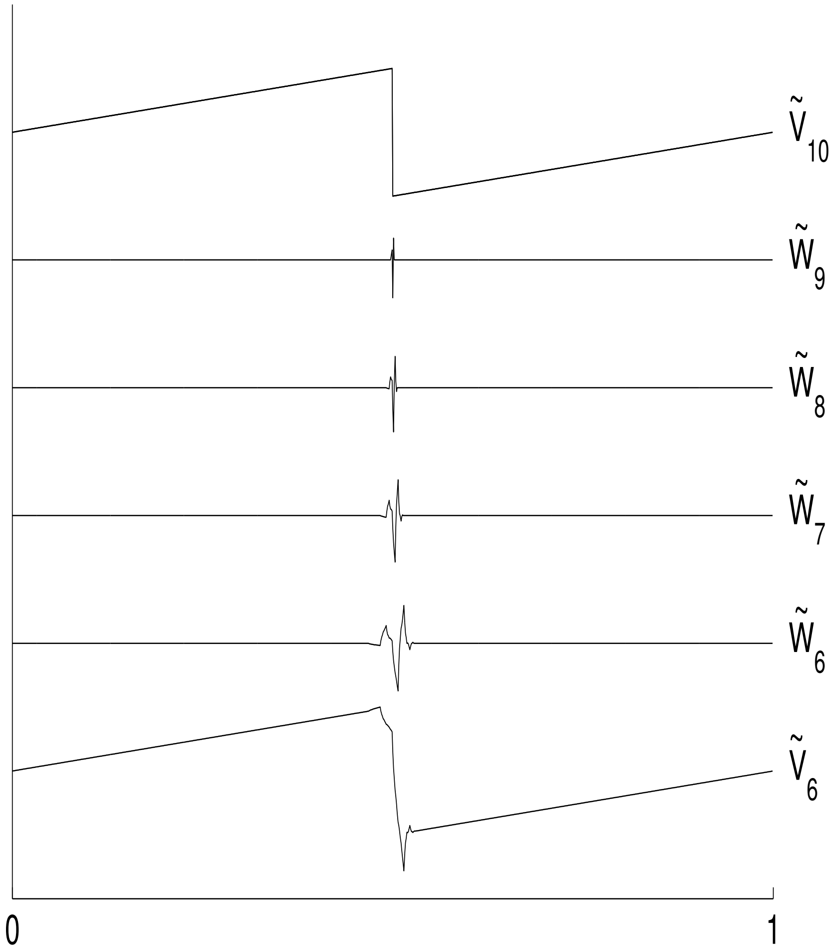


Figure 1.6: The top graph is the sum of its projections onto a coarse space $\tilde{V}_6$ and a sequence of finer spaces $\tilde{W}_6$–$\tilde{W}_9$.

Figure 1.6 shows the wavelet decomposition of f organized according to scale: Each graph is a projection of f onto one of the approximation spaces mentioned above. The bottom graph is the coarse approximation of f in $\tilde{V}_6$. Those labeled $\tilde{W}_6$ to $\tilde{W}_9$ are successive refinements. Adding these projections yields the graph labeled $\tilde{V}_{10}$.

Figure 1.4 and Figure 1.6 suggest that many of the wavelet coefficients are zero. However, at all scales there are some non-zero coefficients, and they reveal the position where f is discontinuous. If, as in the Fourier case, we retain only the 17 largest wavelet coefficients, we obtain the approximation shown in Figure 1.7. Because of the way wavelets work, the approximation error is much smaller than that of the truncated Fourier expansion and, very significantly, is highly localized at the point of discontinuity.

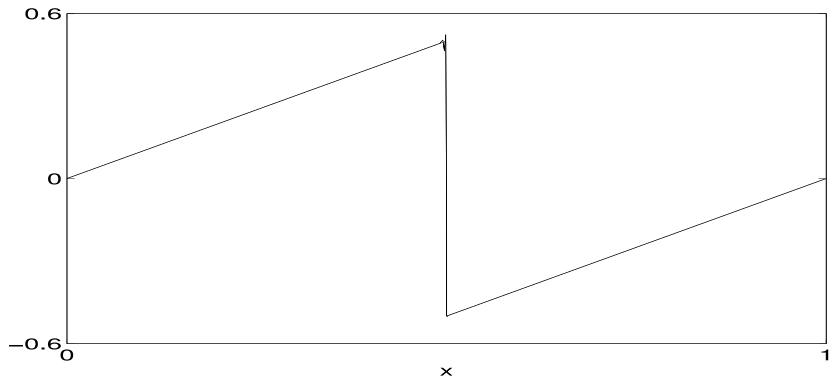


Figure 1.7: A function f and a truncated wavelet expansion with only 17 terms

1.2.1 Summary

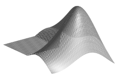
There are three important facts to note about the wavelet approximation:

1. The good resolution of the discontinuity is a consequence of the large wavelet coefficients appearing at the fine scales. The local high frequency content at the discontinuity is captured much better than with the Fourier expansion.

2. The fact that the error is restricted to a small neighborhood of the discontinuity is a result of the "locality" of wavelets. The behavior of f at one location affects only the coefficients of wavelets close to that location.

3. Most of the linear part of f is represented *exactly*. In Figure 1.6 one can see that the linear part of f is approximated exactly even in the coarsest approximation space $\tilde{V}_6$ where only a few scaling functions are used. Therefore, no wavelets are needed to add further details to these parts of f.

The observation made in 3 is a manifestation of a property called *vanishing moments* which means that the scaling functions can locally represent low order polynomials exactly. This property is crucial to the success of wavelet approximations and it is described in detail in Sections 2.1.5 and 2.1.6.



The Matlab function `wavecompare` conducts this comparison experiment and the function `basisdemo` generates the basis functions shown in Figure 1.5.

Chapter 2

Multiresolution analysis

2.1 Wavelets on the real line

A natural framework for wavelet theory is **multiresolution analysis** (MRA) which is a mathematical construction that characterizes wavelets in a general way. MRA yields fundamental insights into wavelet theory and leads to important algorithms as well. The goal of MRA is to express an arbitrary function $f \in L^2(\mathbf{R})$ at various levels of detail. MRA is characterized by the following axioms:

$$\begin{aligned}
&\{0\} \subset \cdots \subset V_{-1} \subset V_0 \subset V_1 \subset \cdots \subset L^2(\mathbf{R}) \quad && (a) \\
&\overline{\bigcup_{j=-\infty}^{\infty} V_j} = L^2(\mathbf{R}) && (b) \\
&\{\phi(x-k)\}_{k\in\mathbf{Z}} \text{ is an orthonormal basis for } V_0 && (c) \\
&f \in V_j \quad \Leftrightarrow \quad f(2\cdot) \in V_{j+1} && (d)
\end{aligned} \tag{2.1}$$

This describes a sequence of nested approximation spaces V_j in $L^2(\mathbf{R})$ such that the closure of their union equals $L^2(\mathbf{R})$. Projections of a function $f \in L^2(\mathbf{R})$ onto V_j are approximations to f which converge to f as $j \to \infty$. Furthermore, the space V_0 has an orthonormal basis consisting of integral translations of a certain function ϕ. Finally, the spaces are related by the requirement that a function f moves from V_j to V_{j+1} when rescaled by 2. From (2.1c) we have the normalization (in the L^2-norm)

$$\|\phi\|_2 \equiv \left(\int_{-\infty}^{\infty} |\phi(x)|^2 \, dx \right)^{1/2} = 1$$

and it is also required that ϕ has unit area [JS94, p. 383], [Dau92, p. 175], i.e.

$$\int_{-\infty}^{\infty} \phi(x)\, dx = 1 \tag{2.2}$$

Remark 2.1 *A fifth axiom is often added to (2.1), namely*

$$\bigcap_{j=-\infty}^{\infty} V_j = \{0\}$$

However, this is not necessary as it follows from the other four axioms in (2.1) [HW96].

Remark 2.2 *The nesting given in (2.1a) is also used by [SN96, HW96, JS94, Str94] and many others. However, some authors e.g. [Dau92, Bey93, BK97, Kai94] use the reverse ordering of the subspaces, making*

$$\{0\} \subset \cdots \subset V_1 \subset V_0 \subset V_{-1} \subset \cdots \subset L^2(\mathbf{R})$$

2.1.1 The detail spaces W_j

Given the nested subspaces in (2.1), we define W_j to be the orthogonal complement of V_j in V_{j+1}, i.e. $V_j \perp W_j$ and

$$V_{j+1} = V_j \oplus W_j \tag{2.3}$$

Consider now two spaces V_{J_0} and V_J, where $J > J_0$. Applying (2.3) recursively we find that

$$V_J = V_{J_0} \oplus \left(\bigoplus_{j=J_0}^{J-1} W_j \right) \tag{2.4}$$

Thus any function in V_J can be expressed as a linear combination of functions in V_{J_0} and $W_j,\ j = J_0, J_0+1, \ldots, J-1$; hence it can be analyzed separately at different scales. Multiresolution analysis has received its name from this separation of scales.

Continuing the decomposition in (2.4) for $J_0 \to -\infty$ and $J \to \infty$ yields in the limits

$$\bigoplus_{j=-\infty}^{\infty} W_j = L^2(\mathbf{R})$$

It follows that all W_j are mutually orthogonal.

Remark 2.3 *W_j can be chosen such that it is not orthogonal to V_j. In that case MRA will lead to the so-called bi-orthogonal wavelets [JS94]. We will not address this point further but only mention that bi-orthogonal wavelets are more flexible than orthogonal wavelets. We refer to [SN96] or [Dau92] for details.*

2.1.2 Basic scaling function and basic wavelet

Since the set $\{\phi(x-k)\}_{k\in\mathbf{Z}}$ is an orthonormal basis for V_0 by axiom (2.1c) it follows by repeated application of axiom $(2.1d)$ that

$$\{\phi(2^j x - k)\}_{k\in\mathbf{Z}} \tag{2.5}$$

is an orthogonal basis for V_j. Note that (2.5) is the function $\phi(2^j x)$ translated by $k/2^j$, i.e. it becomes narrower and translations get smaller as j grows. Since the squared norm of one of these basis functions is

$$\int_{-\infty}^{\infty} \left|\phi(2^j x - k)\right|^2 \, dx = 2^{-j} \int_{-\infty}^{\infty} |\phi(y)|^2 \, dy = 2^{-j} \, \|\phi\|_2^2 = 2^{-j}$$

it follows that

$$\boxed{\{2^{j/2}\phi(2^j x - k)\}_{k\in\mathbf{Z}} \text{ is an orthonormal basis for } V_j}$$

Similarly, it is shown in [Dau92, p. 135] that there exists a function $\psi(x)$ such that

$$\boxed{\{2^{j/2}\psi(2^j x - k)\}_{k\in\mathbf{Z}} \text{ is an orthonormal basis for } W_j}$$

We call ϕ the **basic scaling function** and ψ the **basic wavelet**[1]. It is generally not possible to express either of them explicitly, but, as we shall see, there are efficient and elegant ways of working with them, regardless. It is convenient to introduce the notations

$$\begin{aligned} \phi_{j,k}(x) &= 2^{j/2}\phi(2^j x - k) \\ \psi_{j,k}(x) &= 2^{j/2}\psi(2^j x - k) \end{aligned} \tag{2.6}$$

and

$$\begin{aligned} \phi_k(x) &= \phi_{0,k}(x) \\ \psi_k(x) &= \psi_{0,k}(x) \end{aligned} \tag{2.7}$$

[1] In the literature ψ is often referred to as the **mother wavelet**.

We will use the long and short forms interchangeably depending on the given context.

Since $\psi_{j,k} \in W_j$ it follows immediately that $\psi_{j,k}$ is orthogonal to $\phi_{j,k}$ because $\phi_{j,k} \in V_j$ and $V_j \perp W_j$. Also, because all W_j are mutually orthogonal, it follows that the wavelets are orthogonal across scales. Therefore, we have the orthogonality relations

$$\int_{-\infty}^{\infty} \phi_{j,k}(x)\phi_{j,l}(x)\,dx = \delta_{k,l} \tag{2.8}$$

$$\int_{-\infty}^{\infty} \psi_{i,k}(x)\psi_{j,l}(x)\,dx = \delta_{i,j}\delta_{k,l} \tag{2.9}$$

$$\int_{-\infty}^{\infty} \phi_{i,k}(x)\psi_{j,l}(x)\,dx = 0, \qquad j \geq i \tag{2.10}$$

where $i, j, k, l \in \mathbf{Z}$ and $\delta_{k,l}$ is the **Kronecker delta** defined as

$$\delta_{k,l} = \begin{cases} 0 & k \neq l \\ 1 & k = l \end{cases}$$

2.1.3 Expansions of a function in V_J

A function $f \in V_J$ can be expanded in various ways. For example, there is the pure scaling function expansion

$$\boxed{f(x) = \sum_{l=-\infty}^{\infty} c_{J,l}\phi_{J,l}(x), \quad x \in \mathbf{R}} \tag{2.11}$$

where

$$c_{J,l} = \int_{-\infty}^{\infty} f(x)\phi_{J,l}(x)\,dx \tag{2.12}$$

For any $J_0 \leq J$ there is also the wavelet expansion

$$\boxed{f(x) = \sum_{l=-\infty}^{\infty} c_{J_0,l}\phi_{J_0,l}(x) + \sum_{j=J_0}^{J-1}\sum_{l=-\infty}^{\infty} d_{j,l}\psi_{j,l}(x), \quad x \in \mathbf{R}} \tag{2.13}$$

where

$$\begin{aligned} c_{J_0,l} &= \int_{-\infty}^{\infty} f(x)\phi_{J_0,l}(x)\,dx \\ d_{j,l} &= \int_{-\infty}^{\infty} f(x)\psi_{j,l}(x)\,dx \end{aligned} \tag{2.14}$$

Note that the choice $J_0 = J$ in (2.13) yields (2.11) as a special case. We define

$$\boxed{\lambda = J - J_0} \tag{2.15}$$

and denote λ the **depth** of the wavelet expansion. From the orthonormality of scaling functions and wavelets we find that

$$\|f\|_2^2 = \sum_{k=-\infty}^{\infty} |c_{J,k}|^2 = \sum_{k=-\infty}^{\infty} |c_{J_0,k}|^2 + \sum_{j=J_0}^{J-1} \sum_{k=-\infty}^{\infty} |d_{j,k}|^2$$

which is Parseval's equation for wavelets.

Definition 2.1 *Let P_{V_j} and P_{W_j} denote the operators that project any $f \in L^2(\mathbf{R})$ orthogonally onto V_j and W_j, respectively. Then*

$$\begin{aligned}
(P_{V_j} f)(x) &= \sum_{l=-\infty}^{\infty} c_{j,l}\phi_{j,l}(x) \\
(P_{W_j} f)(x) &= \sum_{l=-\infty}^{\infty} d_{j,l}\psi_{j,l}(x)
\end{aligned}$$

where

$$\begin{aligned}
c_{j,l} &= \int_{-\infty}^{\infty} f(x)\phi_{j,l}(x)\,dx \\
d_{j,l} &= \int_{-\infty}^{\infty} f(x)\psi_{j,l}(x)\,dx
\end{aligned}$$

and

$$P_{V_J} f = P_{V_{J_0}} f + \sum_{j=J_0}^{J-1} P_{W_j} f$$

2.1.4 Dilation equation and wavelet equation

Since $V_0 \subset V_1$, any function in V_0 can be expanded in terms of basis functions of V_1. In particular, $\phi(x) = \phi_{0,0}(x) \in V_0$ so

$$\phi(x) = \sum_{k=-\infty}^{\infty} a_k \phi_{1,k}(x) = \sqrt{2} \sum_{k=-\infty}^{\infty} a_k \phi(2x - k)$$

where

$$a_k = \int_{-\infty}^{\infty} \phi(x)\phi_{1,k}(x)\,dx \tag{2.16}$$

For compactly supported scaling functions only finitely many a_k will be nonzero and we have [Dau92, p. 194]

$$\boxed{\phi(x) = \sqrt{2}\sum_{k=0}^{D-1} a_k\phi(2x-k)} \tag{2.17}$$

Equation (2.17) is fundamental for wavelet theory and it is known as the **dilation equation**. D is an even positive integer called the **wavelet genus** and the numbers $a_0, a_1, \ldots, a_{D-1}$ are called **filter coefficients**. The scaling function is uniquely characterized (up to a constant) by these coefficients.

In analogy to (2.17) we can write a relation for the basic wavelet ψ. Since $\psi \in W_0$ and $W_0 \subset V_1$ we can expand ψ as

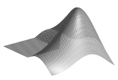
$$\boxed{\psi(x) = \sqrt{2}\sum_{k=0}^{D-1} b_k\phi(2x-k)} \tag{2.18}$$

where the filter coefficients are

$$b_k = \int_{-\infty}^{\infty} \psi(x)\phi_{1,k}(x)\,dx \tag{2.19}$$

We call (2.18) the **wavelet equation**.

Although the filter coefficients a_k and b_k are formally defined by (2.16) and (2.19), they are not normally computed that way because we do not know ϕ and ψ explicitly. However, they can found indirectly from properties of ϕ and ψ, see [SN96, p. 164–173] and [Dau92, p. 195] for details.



The Matlab function `daubfilt(D)` returns a vector containing the filter coefficients $a_0, a_1, \ldots, a_{D-1}$.

It turns out that b_k can be expressed in terms of a_k as follows:

Theorem 2.1

$$\boxed{b_k = (-1)^k a_{D-1-k}, \quad k = 0, 1, \ldots, D-1}$$

Proof: It follows from (2.10) that $\int_{-\infty}^{\infty} \phi(x)\psi(x)\,dx = 0$. Using (2.17) and (2.18) we then have

$$\begin{aligned}
\int_{-\infty}^{\infty} \phi(x)\psi(x)\,dx &= 2\int_{-\infty}^{\infty} \sum_{k=0}^{D-1} a_k\phi(2x-k) \sum_{l=0}^{D-1} b_l\phi(2x-l)\,dx \\
&= \sum_{k=0}^{D-1}\sum_{l=0}^{D-1} a_k b_l \underbrace{\int_{-\infty}^{\infty} \phi(y-k)\phi(y-l)\,dy}_{=\delta_{k,l}} \\
&= \sum_{k=0}^{D-1} a_k b_k \;=\; 0
\end{aligned}$$

This relation is fulfilled if either $a_k = 0$ or $b_k = 0$ for all k, the trivial solutions, or if $b_k = (-1)^k a_{m-k}$ where m is an odd integer provided that we set $a_{m-k} = 0$ for $m - k \notin [0, D-1]$. In the latter case the terms $a_p b_p$ will cancel with the terms $a_{m-p} b_{m-p}$ for $p = 0, 1, \ldots, (m+1)/2 - 1$. An obvious choice is $m = D - 1$. □

The Matlab function `low2hi` computes $\{b_k\}_{k=0}^{D-1}$ from $\{a_k\}_{k=0}^{D-1}$.

One important consequence of (2.17) and (2.18) is that $\text{supp}(\phi) = \text{supp}(\psi) = [0, D-1]$ (see e.g. [Dau92, p. 176] or [SN96, p. 185]). It follows immediately that

$$\boxed{\text{supp}(\phi_{j,l}) = \text{supp}(\psi_{j,l}) = I_{j,l}} \tag{2.20}$$

where

$$I_{j,l} = \left[\frac{l}{2^j}, \frac{l+D-1}{2^j}\right] \tag{2.21}$$

Remark 2.4 *The formulation of the dilation equation is not the same throughout the literature. We have identified three versions:*

1. $\phi(x) = \sum_k a_k\phi(2x-k)$
2. $\phi(x) = \sqrt{2}\sum_k a_k\phi(2x-k)$
3. $\phi(x) = 2\sum_k a_k\phi(2x-k)$

*The first is used by e.g. [WA94, Str94], the second by e.g. [Dau92, Bey93, Kai94], and the third by e.g. [HW96, JS94]. We have chosen the second formulation, partly because it comes directly from the MRA expansion of ϕ in terms of $\phi_{1,k}$ but also because it leads to ortho*normality *of the wavelet transform matrices, see Section 3.3.2.*

2.1.5 Filter coefficients

In this section we will use properties of ϕ and ψ to derive a number of relations satisfied by the filter coefficients.

Orthonormality property

Using the dilation equation (2.17) we can transform the orthonormality of the translates of ϕ, (2.1c) into a condition on the filter coefficients a_k. From (2.8) we have the orthonormality property

$$\begin{aligned}
\delta_{0,n} &= \int_{-\infty}^{\infty} \phi(x)\phi(x-n)\,dx \\
&= \int_{-\infty}^{\infty} \left(\sqrt{2}\sum_{k=0}^{D-1} a_k \phi(2x-k)\right)\left(\sqrt{2}\sum_{l=0}^{D-1} a_l \phi(2x-2n-l)\right)\,dx \\
&= \sum_{k=0}^{D-1}\sum_{l=0}^{D-1} a_k a_l \int_{-\infty}^{\infty} \phi(y)\phi(y+k-2n-l)\,dy, \qquad y = 2x-k \\
&= \sum_{k=0}^{D-1}\sum_{l=0}^{D-1} a_k a_l \delta_{k-2n,l} \\
&= \sum_{k=k1(n)}^{k_2(n)} a_k a_{k-2n}, \quad n \in \mathbf{Z}
\end{aligned}$$

where $k_1(n) = \max(0, 2n)$ and $k_2(n) = \min(D-1, D-1+2n)$. Although this holds for all $n \in \mathbf{Z}$, it will only yield $D/2$ distinct equations corresponding to $n = 0, 1, \dots, D/2-1$ because the sum equals zero trivially for $n \geq D/2$ as there is no overlap of the nonzero a_ks. Hence we have

$$\boxed{\sum_{k=k1(n)}^{k_2(n)} a_k a_{k-2n} = \delta_{0,n}, \quad n = 0, 1, \dots, D/2-1} \tag{2.22}$$

Similarly, it follows from Theorem 2.1 that

$$\boxed{\sum_{k=k1(n)}^{k_2(n)} b_k b_{k-2n} = \delta_{0,n}, \quad n = 0, 1, \ldots, D/2 - 1} \tag{2.23}$$

Conservation of area

Recall that $\int_{-\infty}^{\infty} \phi(x)\,dx = 1$. Integration of both sides of (2.17) then gives

$$\int_{-\infty}^{\infty} \phi(x)\,dx = \sqrt{2} \sum_{k=0}^{D-1} a_k \int_{-\infty}^{\infty} \phi(2x-k)\,dx = \frac{1}{\sqrt{2}} \sum_{k=0}^{D-1} a_k \int_{-\infty}^{\infty} \phi(y)\,dy$$

or

$$\boxed{\sum_{k=0}^{D-1} a_k = \sqrt{2}} \tag{2.24}$$

The name “conservation of area” is suggested by Newland [New93, p. 308].

Property of vanishing moments

Another important property of the scaling function is its ability to represent polynomials exactly up to some degree $P-1$. More precisely, it is required that

$$x^p = \sum_{k=-\infty}^{\infty} M_k^p \phi(x-k), \quad x \in \mathbf{R}, \;\; p = 0, 1, \ldots, P-1 \tag{2.25}$$

where

$$M_k^p = \int_{-\infty}^{\infty} x^p \phi(x-k)\,dx, \quad k \in \mathbf{Z}, \;\; p = 0, 1, \ldots, P-1 \tag{2.26}$$

We denote M_k^p the pth moment of $\phi(x-k)$ and it can be computed by a procedure which is described in Appendix A.

Equation (2.25) can be translated into a condition involving the wavelet by taking the inner product with $\psi(x)$. This yields

$$\int_{-\infty}^{\infty} x^p \psi(x)\,dx = \sum_{k=-\infty}^{\infty} M_k^p \int_{-\infty}^{\infty} \phi(x-k)\psi(x)\,dx = 0$$

since ψ and ϕ are orthonormal. Hence, we have the property of P vanishing moments:

$$\boxed{\int_{-\infty}^{\infty} x^p \psi(x)\,dx = 0, \quad x \in \mathbf{R},\ p = 0, 1, \ldots, P-1} \tag{2.27}$$

The property of vanishing moments can be expressed in terms of the filter coefficients as follows. Substituting the wavelet equation (2.18) into (2.27) yields

$$\begin{aligned}
0 &= \int_{-\infty}^{\infty} x^p \psi(x)\,dx \\
&= \sqrt{2}\sum_{k=0}^{D-1} b_k \int_{-\infty}^{\infty} x^p \phi(2x-k)\,dx \\
&= \frac{\sqrt{2}}{2^{p+1}}\sum_{k=0}^{D-1} b_k \int_{-\infty}^{\infty} (y+k)^p \phi(y)\,dy, \qquad y = 2x-k \\
&= \frac{\sqrt{2}}{2^{p+1}}\sum_{k=0}^{D-1} b_k \sum_{n=0}^{p} \binom{p}{n} k^n \int_{-\infty}^{\infty} y^{p-n}\phi(y)\,dy \\
&= \frac{\sqrt{2}}{2^{p+1}}\sum_{n=0}^{p} \binom{p}{n} M_0^{p-n} \sum_{k=0}^{D-1} b_k k^n
\end{aligned} \tag{2.28}$$

where we have used (2.26) and the binomial formula

$$(y+k)^p = \sum_{n=0}^{p} \binom{p}{n} y^{p-n} k^n$$

For $p = 0$ relation (2.28) becomes $\sum_{k=0}^{D-1} b_k = 0$, and using induction on p we obtain P moment conditions on the filter coefficients, namely

$$\sum_{k=0}^{D-1} b_k k^p = \sum_{k=0}^{D-1} (-1)^k a_{D-1-k} k^p = 0, \quad p = 0, 1, \ldots, P-1$$

This expression can be simplified further by the change of variables $l = D-1-k$. Then

$$0 = \sum_{l=0}^{D-1} (-1)^{D-1-l} a_l (D-1-l)^p$$

and using the binomial formula again, we arrive at

$$\boxed{\sum_{l=0}^{D-1} (-1)^l a_l l^p = 0, \quad p = 0, 1, \ldots, P-1} \tag{2.29}$$

Other properties

The conditions (2.22), (2.24) and (2.29) comprise a system of $D/2+1+P$ equations for the D filter coefficients $a_k, k = 0, 1, \ldots, D-1$. However, it turns out that one of the conditions is redundant. For example (2.29) with $p = 0$ can be obtained from the others, see [New93, p. 320]. This leaves a total of $D/2 + P$ equations for the D filter coefficients. Not surprisingly, it can be shown [Dau92, p. 194] that the highest number of vanishing moments for this type of wavelet is

$$\boxed{P = D/2}$$

yielding a total of D equations that must be fulfilled.

This system can be used to determine filter coefficients for compactly supported wavelets or used to validate coefficients obtained otherwise.

The Matlab function `filttest` checks if a vector of filter coefficients fulfils (2.22), (2.24) and (2.29) with $P = D/2$.

Finally we note two other properties of the filter coefficients.

Theorem 2.2

$$\boxed{\sum_{k=0}^{D/2-1} a_{2k} = \sum_{k=0}^{D/2-1} a_{2k+1} = \frac{1}{\sqrt{2}}}$$

Proof: Adding (2.24) and (2.29) with $p = 0$ yields the first result. Subtracting them yields the second. □

Theorem 2.3

$$\boxed{\sum_{l=0}^{D/2-1} \sum_{n=0}^{D-2l-2} a_n a_{n+2l+1} = \frac{1}{2}}$$

Proof: Using (2.29) twice we write

$$\begin{aligned} 0 &= \left(\sum_{k=0}^{D-1}(-1)^k a_k\right)\left(\sum_{l=0}^{D-1}(-1)^{-l} a_l\right) = \sum_{k=0}^{D-1}\sum_{l=0}^{D-1}(-1)^{k-l} a_k a_l \\ &= \underbrace{\sum_{k=0}^{D-1} a_k^2}_{=1} + \sum_{k=0}^{D-1}\sum_{l=0}^{k-1}(-1)^{k-l} a_k a_l + \sum_{k=0}^{D-1}\sum_{l=k+1}^{D-1}(-1)^{l-k} a_k a_l \end{aligned}$$

Using (2.22) with $n = 0$ in the first sum and rearranging the other sums yields

$$\begin{aligned} 0 &= 1 + \sum_{p=1}^{D-1} \sum_{l=0}^{D-1-p} (-1)^p a_{l+p} a_l + \sum_{p=1}^{D-1} \sum_{k=0}^{D-1-p} (-1)^p a_k a_{k+p} \\ &= 1 + 2 \sum_{p=1}^{D-1} \sum_{n=0}^{D-1-p} (-1)^p a_n a_{n+p} \end{aligned}$$

All sums where p is even vanish by (2.22), so we are left with the "odd" terms ($p = 2l + 1$)

$$\begin{aligned} 0 &= 1 + 2 \sum_{l=0}^{D/2-1} (-1)^{2l+1} \sum_{n=0}^{D-2l-2} a_n a_{n+2l+1} \\ &= 1 - 2 \sum_{l=0}^{D/2-1} \sum_{n=0}^{D-2l-2} a_n a_{n+2l+1} \end{aligned}$$

from which the result follows. □

Equation (2.24) can now be derived directly from (2.22) and (2.29) and have the following Corollary of Theorem 2.2.

Corollary 2.4

$$\sum_{k=0}^{D-1} = \sqrt{2}$$

Proof: By a manipulation similar to that of Theorem 2.2 we write

$$\begin{aligned} \left(\sum_{k=0}^{D-1} a_k \right) \left(\sum_{l=0}^{D-1} a_l \right) &= \sum_{k=0}^{D-1} \sum_{l=0}^{D-1} a_k a_l \\ &= 1 + 2 \sum_{l=0}^{D/2-1} \sum_{n=0}^{D-2l-2} a_n a_{n+2l+1} \\ &= 2 \end{aligned}$$

where Theorem 2.2 was used for the last equation. Taking the square root on each side yields the result. □

2.1.6 Decay of wavelet coefficients

The P vanishing moments have an important consequence for the wavelet coefficients $d_{j,k}$ (2.14): They decrease rapidly for a smooth function. Furthermore, if a function has a discontinuity in one of its derivatives then the wavelet coefficients will decrease slowly *only* close to that discontinuity and maintain fast decay where the function is smooth. This property makes wavelets particularly suitable for representing *piecewise smooth functions*. The decay of wavelet coefficients is expressed in the following theorem:

Theorem 2.5 *Let $P = D/2$ be the number of vanishing moments for a wavelet $\psi_{j,k}$ and let $f \in C^P(\mathbf{R})$. Then the wavelet coefficients given in (2.14) decay as follows:*

$$\boxed{|d_{j,k}| \leq C_P \, 2^{-j(P+\frac{1}{2})} \max_{\xi \in I_{j,k}} \left| f^{(P)}(\xi) \right|}$$

where C_P is a constant independent of j, k, and f and $I_{j,k} = \mathrm{supp}\{\psi_{j,k}\} = [k/2^j, (k+D-1)/2^j]$.

Proof: For $x \in I_{j,k}$ we write the Taylor expansion for f around $x = k/2^j$.

$$f(x) = \left(\sum_{p=0}^{P-1} f^{(p)}\left(k/2^j\right) \frac{(x - \frac{k}{2^j})^p}{p!} \right) + f^{(P)}(\xi) \frac{(x - \frac{k}{2^j})^P}{P!} \tag{2.30}$$

where $\xi \in [k/2^j, x]$.

Inserting (2.30) into (2.14) and restricting the integral to the support of $\psi_{j,k}$ yields

$$\begin{aligned}
d_{j,k} &= \int_{I_{j,k}} f(x)\psi_{j,k}(x)\,dx \\
&= \left(\sum_{p=0}^{P-1} f^{(p)}\left(k/2^j\right) \frac{1}{p!} \int_{I_{j,k}} \left(x - \frac{k}{2^j}\right)^p \psi_{j,k}(x)\,dx \right) + \\
&\quad \frac{1}{P!} \int_{I_{j,k}} f^{(P)}(\xi) \left(x - \frac{k}{2^j}\right)^P \psi_{j,k}(x)\,dx
\end{aligned}$$

Recall that ξ depends on x, so $f^{(P)}(\xi)$ is not constant and must remain under the last integral sign.

Consider the integrals where $p = 0, 1, \ldots, P-1$. Using (2.21) and letting $y = 2^j x - k$ we obtain

$$\begin{aligned}
&\int_{k/2^j}^{(k+D-1)/2^j} \left(x - \frac{k}{2^j}\right)^p 2^{j/2}\psi(2^j x - k)\,dx \\
&= 2^{j/2} \int_0^{D-1} \left(\frac{y}{2^j}\right)^p \psi(y)\,2^{-j}\,dy \\
&= 2^{-j(p+1/2)} \int_0^{D-1} y^p \psi(y)\,dy \\
&= 0, \quad p = 0, 1, \ldots, P-1
\end{aligned}$$

because of the P vanishing moments (2.27). Therefore, the wavelet coefficient is determined from the remainder term alone. Hence,

$$\begin{aligned}
|d_{j,k}| &= \frac{1}{P!}\left| \int_{I_{j,k}} f^{(P)}(\xi) \left(x - \frac{k}{2^j}\right)^P 2^{j/2}\psi(2^j x - k)\,dx \right| \\
&\le \frac{1}{P!} \max_{\xi \in I_{j,k}} \left|f^{(P)}(\xi)\right| \int_{I_{j,k}} \left| \left(x - \frac{k}{2^j}\right)^P 2^{j/2}\psi(2^j x - k)\right| dx \\
&= 2^{-j(P+1/2)} \frac{1}{P!} \max_{\xi \in I_{j,k}} \left|f^{(P)}(\xi)\right| \int_0^{D-1} \left|y^P \psi(y)\right| dy
\end{aligned}$$

Defining

$$C_P = \frac{1}{P!} \int_0^{D-1} \left|y^P \psi(y)\right| dy$$

we obtain the desired inequality. □

From Theorem 2.5 we see that if f behaves like a polynomial of degree less than P in the interval $I_{j,k}$ then $f^{(P)} \equiv 0$ and the corresponding wavelet coefficient $d_{j,k}$ is zero. If $f^{(P)}$ is different from zero, coefficients will decay exponentially with respect to the scale parameter j. If f has a discontinuity in a derivative of order less than or equal to P, then Theorem 2.5 does not hold for wavelet coefficients located at the discontinuity[2]. However, coefficients away from the discontinuity are not affected. The coefficients in a wavelet expansion thus reflect *local* properties of f and isolated discontinuities do not ruin the convergence away from the discontinuities. This means that functions that are piecewise smooth have many small wavelet coefficients in their expansions and may thus be represented

[2] There are $D-1$ affected wavelet coefficients at each level

well by relatively few wavelet coefficients. This is the principle behind wavelet based data compression and one of the reasons why wavelets are so useful in e.g. signal processing applications. We saw an example of this in Chapter 1 and Section 4.4 gives another example. The consequences of Theorem 2.5 with respect to approximation errors of wavelet expansions are treated in Chapter 4.

2.2 Wavelets and the Fourier transform

It is often useful to consider the behavior of the Fourier transform of a function rather than the function itself. Also in case it gives rise to some intriguing relations and insights about the basic scaling function and the basic wavelet.
We define the (continuous) Fourier transform as

$$\hat{\phi}(\xi) = \int_{-\infty}^{\infty} \phi(x) e^{-i\xi x}\, dx, \quad \xi \in \mathbf{R}$$

The requirement that ϕ has unit area (2.2) immediately translates into

$$\hat{\phi}(0) = \int_{-\infty}^{\infty} \phi(x)\, dx = 1 \tag{2.31}$$

Our point of departure for expressing $\hat{\phi}$ at other values of ξ is the dilation equation (2.17). Taking the Fourier transform on both sides yields

$$\begin{aligned}
\hat{\phi}(\xi) &= \sqrt{2} \sum_{k=0}^{D-1} a_k \int_{-\infty}^{\infty} \phi(2x-k) e^{-i\xi x}\, dx \\
&= \sqrt{2} \sum_{k=0}^{D-1} a_k \int_{-\infty}^{\infty} \phi(y) e^{-i\xi(y+k)/2}\, dy/2 \\
&= \frac{1}{\sqrt{2}} \sum_{k=0}^{D-1} a_k e^{-ik\xi/2} \int_{-\infty}^{\infty} \phi(y) e^{-i(\xi/2)y}\, dy \\
&= A\left(\frac{\xi}{2}\right) \hat{\phi}\left(\frac{\xi}{2}\right)
\end{aligned} \tag{2.32}$$

where

$$\boxed{A(\xi) = \frac{1}{\sqrt{2}} \sum_{k=0}^{D-1} a_k e^{-ik\xi}, \quad \xi \in \mathbf{R}} \tag{2.33}$$

$A(\xi)$ is a 2π-periodic function with some interesting properties which can be derived directly from the conditions on the filter coefficients established in Section 2.1.

Lemma 2.6 *If ψ has P vanishing moments then*

$$\boxed{\begin{aligned} A(0) &= 1 \\ \frac{d^p}{d\xi^p}A(\xi)\mid_{\xi=\pi} &= 0, \quad p = 0, 1, \ldots, P-1 \end{aligned}}$$

Proof: Let $\xi = 0$ in (2.33). Using (2.24) we find immediately

$$A(0) = \frac{1}{\sqrt{2}}\sum_{k=0}^{D-1} a_k = \frac{\sqrt{2}}{\sqrt{2}} = 1$$

Now let $\xi = \pi$. Then by (2.29)

$$\begin{aligned} \frac{d^p}{d\xi^p}A(\xi)\mid_{\xi=\pi} &= \frac{1}{\sqrt{2}}\sum_{k=0}^{D-1}(-ik)^p a_k e^{-ik\pi} \\ &= \frac{1}{\sqrt{2}}(-i)^p \sum_{k=0}^{D-1} k^p a_k (-1)^k \\ &= 0, \quad p = 0, 1, \ldots, P-1 \end{aligned}$$

□

Putting $p = 0$ in Lemma 2.6 and using the 2π periodicity of $A(\xi)$ we obtain

Corollary 2.7

$$\boxed{A(n\pi) = \begin{cases} 1 & n \text{ even} \\ 0 & n \text{ odd} \end{cases}}$$

Equation (2.32) can be repeated for $\hat{\phi}(\xi/2)$ yielding

$$\hat{\phi}(\xi) = A\left(\frac{\xi}{2}\right) A\left(\frac{\xi}{4}\right) \hat{\phi}\left(\frac{\xi}{4}\right)$$

After N such steps we have

$$\hat{\phi}(\xi) = \prod_{j=1}^{N} A\left(\frac{\xi}{2^j}\right) \; \hat{\phi}\left(\frac{\xi}{2^N}\right)$$

It follows from (2.33) and (2.24) that $|A(\xi)| \leq 1$ so the product converges for $N \to \infty$ and we get the expression

$$\hat{\phi}(\xi) = \prod_{j=1}^{\infty} A\left(\frac{\xi}{2^j}\right) \hat{\phi}(0)$$

Using (2.31) we arrive at the product formula

$$\boxed{\hat{\phi}(\xi) = \prod_{j=1}^{\infty} A\left(\frac{\xi}{2^j}\right), \quad \xi \in \mathbf{R}} \tag{2.34}$$

Lemma 2.8

$$\boxed{\hat{\phi}(2\pi n) = \delta_{0,n}, \quad n \in \mathbf{Z}}$$

Proof: The case $n = 0$ follows from (2.31). Therefore, let $n \in \mathbf{Z} \setminus \{0\}$ be expressed in the form $n = 2^i K$ where $i \in \mathbf{N}_0$ and $K \in \mathbf{Z}$ with K odd, i.e. $\langle K \rangle_2 = 1$. Then using (2.34) we get

$$\begin{aligned}
\hat{\phi}(2\pi n) &= \prod_{j=1}^{\infty} A\left(\frac{2\pi n}{2^j}\right) \\
&= \prod_{j=1}^{\infty} A\left(\frac{2^{i+1} K\pi}{2^j}\right) \\
&= A(2^i K\pi) A(2^{i-1} K\pi) \ldots A(K\pi) \ldots \\
&= 0
\end{aligned}$$

since $A(K\pi) = 0$ by Corollary 2.7. $\square$

A consequence of Lemma 2.8 is the following basic property of ϕ:

Theorem 2.9

$$\boxed{\sum_{n=-\infty}^{\infty} \phi_{j,0}(x+n) = 2^{-j/2}, \quad j \leq 0, \quad x \in \mathbf{R}}$$

Proof: Let

$$S_j(x) = \sum_{n=-\infty}^{\infty} \phi_{j,0}(x+n)$$

We observe that $S_j(x)$ is a 1-periodic function. Hence it has the Fourier series expansion

$$S_j(x) = \sum_{k=-\infty}^{\infty} c_k e^{i2\pi kx}, \quad x \in \mathbf{R} \tag{2.35}$$

where the Fourier coefficients c_k are defined by

$$\begin{aligned}
c_k &= \int_0^1 S_j(x) e^{-i2\pi kx}\, dx \\
&= \int_0^1 \sum_{n=-\infty}^{\infty} \phi_{j,0}(x+n) e^{-i2\pi kx}\, dx \\
&= \sum_{n=-\infty}^{\infty} \int_n^{n+1} \phi_{j,0}(y) e^{-i2\pi ky} \underbrace{e^{i2\pi kn}}_{=1}\, dy, \qquad y = x+n \\
&= \int_{-\infty}^{\infty} \phi_{j,0}(y) e^{-i2\pi ky}\, dy \\
&= 2^{j/2} \int_{-\infty}^{\infty} \phi(2^j y) e^{-i2\pi ky}\, dy \\
&= 2^{j/2} \int_{-\infty}^{\infty} \phi(z) e^{-i2\pi k2^{-j}z}\, 2^{-j} dz, \qquad z = 2^j y \\
&= 2^{-j/2} \hat{\phi}(2\pi k 2^{-j}), \quad k \in \mathbf{Z}
\end{aligned}$$

We know from Lemma 2.8 that $\hat{\phi}(2\pi k) = \delta_{0,k}$. Therefore, since $j \leq 0$ by assumption,

$$c_k = 2^{-j/2} \hat{\phi}(2\pi k 2^{-j}) = 2^{-j/2} \delta_{0,k}$$

so the Fourier series collapses into one term, namely the “DC” term c_0, i.e.

$$S_j(x) = c_0 = 2^{-j/2}$$

from which the result follows. $\square$

Theorem 2.9 states that if π is a zero of the function $A(\xi)$ then the constant function can be represented by a linear combination of the translates of $\phi_{j,0}(x)$, which again is equivalent to the zeroth vanishing moment condition (2.27). If the number of vanishing moments P is greater than one, then a similar argument can be used to show the following more general statement [SN96, p. 230]

Theorem 2.10 *If π is a zero of $A(\xi)$ of multiplicity P, i.e. if*

$$\frac{d^p}{d\xi^p} A(\xi) \mid_{\xi=\pi} \;=\; 0, \quad p = 0, 1, \ldots, P-1$$

then:

1. *The integral translates of $\phi(x)$ can reproduce polynomials of degree less than P.*

2. *The wavelet $\psi(x)$ has P vanishing moments.*

3. $\hat{\phi}^{(p)}(2\pi n) = 0$ *for* $n \in \mathbf{Z}$, $n \neq 0$ *and* $p < P$.

4. $\hat{\psi}^{(p)}(0) = 0$ *for* $p < P$.

2.2.1 The wavelet equation

In the beginning of this section we obtained a relation (2.32) for the scaling function in the frequency domain. Using (2.18) we can obtain an analogous expression for $\hat{\psi}$.

$$\begin{aligned}
\hat{\psi}(\xi) &= \sqrt{2} \sum_{k=0}^{D-1} b_k \int_{-\infty}^{\infty} \phi(2x-k) e^{-i\xi x}\, dx \\
&= \frac{1}{\sqrt{2}} \sum_{k=0}^{D-1} b_k e^{-ik\xi/2} \int_{-\infty}^{\infty} \phi(y) e^{-i(\xi/2)y}\, dy \\
&= B\left(\frac{\xi}{2}\right) \hat{\phi}\left(\frac{\xi}{2}\right)
\end{aligned}$$

where

$$B(\xi) = \frac{1}{\sqrt{2}} \sum_{k=0}^{D-1} b_k e^{-ik\xi} \tag{2.36}$$

Using Theorem 2.1 we can express $B(\xi)$ in terms of $A(\xi)$.

$$
\begin{aligned}
B(\xi) &= \frac{1}{\sqrt{2}} \sum_{k=0}^{D-1} (-1)^k a_{D-1-k} e^{-ik\xi} \\
&= \frac{1}{\sqrt{2}} \sum_{k=0}^{D-1} a_{D-1-k} e^{-ik(\xi+\pi)} \\
&= \frac{1}{\sqrt{2}} \sum_{l=0}^{D-1} a_l e^{-i(D-1-l)(\xi+\pi)} \\
&= e^{-i(D-1)(\xi+\pi)} \frac{1}{\sqrt{2}} \sum_{l=0}^{D-1} a_l e^{il(\xi+\pi)} \\
&= e^{-i(D-1)(\xi+\pi)} \overline{A(\xi+\pi)}
\end{aligned}
$$

This leads to the wavelet equation in the frequency domain.

$$
\boxed{\hat{\psi}(\xi) = e^{-i(D-1)(\xi/2+\pi)} \overline{A(\xi/2+\pi)} \hat{\phi}(\xi/2)} \tag{2.37}
$$

An immediate consequence of (2.37) is the following lemma:

Lemma 2.11

$$
\boxed{\hat{\psi}(4\pi n) = 0, \quad n \in \mathbf{Z}}
$$

Proof: Letting $\xi = 4\pi n$ in (2.37) yields

$$
\hat{\psi}(4\pi n) = e^{-i(D-1)(2\pi n+\pi)} \overline{A(2\pi n+\pi)} \hat{\phi}(2\pi n)
$$

which is equal to zero by Lemma 2.8 for $n \neq 0$. For $n = 0$ we have

$$
\hat{\psi}(0) = -\overline{A(\pi)}
$$

but this is also zero by Corollary 2.7. □

2.2.2 Orthonormality in the frequency domain

The inner products of ϕ with its integral translates also have an interesting formulation in the frequency domain. By Plancherel's identity [HW96, p. 4] the inner product in the physical domain equals the inner product in the frequency domain (except for a factor of 2π). Hence

$$\begin{aligned} f_k &= \int_{-\infty}^{\infty} \phi(x)\phi(x-k)\,dx \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{\phi}(\xi)\overline{\hat{\phi}(\xi)e^{-i\xi k}}\,d\xi \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \left|\hat{\phi}(\xi)\right|^2 e^{i\xi k}\,d\xi \\ &= \frac{1}{2\pi}\int_{0}^{2\pi} \sum_{n=-\infty}^{\infty} \left|\hat{\phi}(\xi+2\pi n)\right|^2 e^{i\xi k}\,d\xi \end{aligned} \tag{2.38}$$

Define

$$\boxed{F(\xi) = \sum_{n=-\infty}^{\infty} \left|\hat{\phi}(\xi+2\pi n)\right|^2, \quad \xi \in \mathbf{R}} \tag{2.39}$$

Then we see from (2.38) that f_k is the k'th Fourier coefficient of $F(\xi)$. Thus

$$F(\xi) = \sum_{k=-\infty}^{\infty} f_k e^{-ik\xi} \tag{2.40}$$

Since ϕ is orthogonal to its integral translations we know from (2.8) that

$$f_k = \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases}$$

hence (2.40) evaluates to

$$F(\xi) = 1, \quad \xi \in \mathbf{R}$$

Thus we have proved the following lemma.

Lemma 2.12 *The translates $\phi(x-k)$, $k \in \mathbf{Z}$ are orthonormal if and only if*

$$\boxed{F(\xi) \equiv 1}$$

We can now translate the condition on F into a condition on $A(\xi)$. From (2.32) and (2.39) we have

$$\begin{aligned} F(2\xi) &= \sum_{n=-\infty}^{\infty} \left|\hat{\phi}(2\xi + 2\pi n)\right|^2 \\ &= \sum_{n=-\infty}^{\infty} \left|\hat{\phi}(\xi + \pi n)\right|^2 |A(\xi + \pi n)|^2 \end{aligned}$$

Splitting the sum into two sums according to whether n is even or odd and using the periodicity of $A(\xi)$ yields

$$\begin{aligned} F(2\xi) &= \sum_{n=-\infty}^{\infty} \left|\hat{\phi}(\xi + 2\pi n)\right|^2 |A(\xi + 2\pi n)|^2 + \\ &\quad \sum_{n=-\infty}^{\infty} \left|\hat{\phi}(\xi + \pi + 2\pi n)\right|^2 |A(\xi + \pi + 2\pi n)|^2 \\ &= |A(\xi)|^2 \sum_{n=-\infty}^{\infty} \left|\hat{\phi}(\xi + 2\pi n)\right|^2 + |A(\xi + \pi)|^2 \sum_{n=-\infty}^{\infty} \left|\hat{\phi}(\xi + \pi + 2\pi n)\right|^2 \\ &= |A(\xi)|^2 F(\xi) + |A(\xi + \pi)|^2 F(\xi + \pi) \end{aligned}$$

If $F(\xi) = 1$ then $|A(\xi)|^2 + |A(\xi + \pi)|^2 \equiv 1$ and the converse is also true [SN96, p. 205–206], [JS94, p. 386]. For this reason we have

Lemma 2.13

$$\boxed{F(\xi) \equiv 1 \quad \Leftrightarrow \quad |A(\xi)|^2 + |A(\xi + \pi)|^2 \equiv 1}$$

Finally, Lemma 2.12 and Lemma 2.13 yields the following theorem:

Theorem 2.14 *The translates $\phi(x - k)$, $k \in \mathbf{Z}$ are orthonormal if and only if*

$$\boxed{|A(\xi)|^2 + |A(\xi + \pi)|^2 \equiv 1}$$

2.2.3 Overview of conditions

We summarize now various formulations of the orthonormality property, the property of vanishing moments and the conservation of area.

Orthonormality property:

$$\int_{-\infty}^{\infty} \phi(x)\phi(x-k)\,dx = \delta_{0,k}, \quad k \in \mathbf{Z}$$

$$\sum_{k=0}^{D-1} a_k a_{k+2n} = \delta_{0,n}, \quad n \in \mathbf{Z}$$

$$|A(\xi)|^2 + |A(\xi+\pi)|^2 \equiv 1, \quad \xi \in \mathbf{R}$$

Property of vanishing moments:

For $p = 0, 1, \ldots, P-1$ and $P = D/2$ we have

$$\int_{-\infty}^{\infty} \psi(x) x^p\,dx = 0$$

$$\sum_{k=0}^{D-1} (-1)^k a_k k^p = 0$$

$$\frac{d^p}{d\xi^p} A(\xi) \mid_{\xi=\pi} = 0$$

Conservation of area:

$$\phi(x) = \sqrt{2} \sum_{k=0}^{D-1} a_k \phi(2x-k), \quad x \in \mathbf{R}$$

$$\sum_{k=0}^{D-1} a_k = \sqrt{2}$$

$$A(0) = 1$$

2.3 Periodized wavelets

So far our functions have been defined on the entire real line, e.g. $f \in L^2(\mathbf{R})$. There are applications, such as the processing of audio signals, where this is a reasonable model because audio signals can be arbitrarily long and the total length may be unknown until the moment when the audio signal stops. However, in most practical applications such as image processing, data fitting, or problems involving differential equations, the space domain is a finite interval. Many of these cases can be dealt with by introducing periodized scaling functions and wavelets which we define as follows:

Definition 2.2 *Let $\phi \in L^2(\mathbf{R})$ and $\psi \in L^2(\mathbf{R})$ be the basic scaling function and the basic wavelet from a multiresolution analysis as defined in (2.1). For any $j, l \in \mathbf{Z}$ we define the 1-periodic scaling function*

$$\tilde{\phi}_{j,l}(x) = \sum_{n=-\infty}^{\infty} \phi_{j,l}(x+n) = 2^{j/2} \sum_{n=-\infty}^{\infty} \phi(2^j(x+n)-l), \quad x \in \mathbf{R} \tag{2.41}$$

and the 1-periodic wavelet

$$\tilde{\psi}_{j,l}(x) = \sum_{n=-\infty}^{\infty} \psi_{j,l}(x+n) = 2^{j/2} \sum_{n=-\infty}^{\infty} \psi(2^j(x+n)-l), \quad x \in \mathbf{R} \tag{2.42}$$

The 1 periodicity can be verified as follows

$$\tilde{\phi}_{j,l}(x+1) = \sum_{n=-\infty}^{\infty} \phi_{j,l}(x+n+1) = \sum_{m=-\infty}^{\infty} \phi_{j,l}(x+m) = \tilde{\phi}_{j,l}(x)$$

and similarly $\tilde{\psi}_{j,l}(x+1) = \tilde{\psi}_{j,l}(x)$.

2.3.1 Some important special cases

1. $\tilde{\phi}_{j,l}(x)$ is constant for $j \leq 0$. To see this note that (2.41) yields

$$\begin{aligned} \tilde{\phi}_{j,l}(x) &= 2^{j/2} \sum_{n=-\infty}^{\infty} \phi(2^j(x+n-2^{-j}l)) \\ &= 2^{j/2} \sum_{m=-\infty}^{\infty} \phi(2^j(x+m)) = \tilde{\phi}_{j,0}(x), \quad l \in \mathbf{Z} \end{aligned}$$

   where $m = n - 2^{-j}l$ is an integer because $2^{-j}l$ is an integer. Hence, by (2.41) and Theorem 2.9 we have $\tilde{\phi}_{j,0}(x) = \sum_{n=-\infty}^{\infty} \phi_{j,0}(x+n) = 2^{-j/2}$, so

$$\boxed{\tilde{\phi}_{j,l}(x) = 2^{-j/2}, \quad j \leq 0, \;\; l \in \mathbf{Z}, \;\; x \in \mathbf{R}} \tag{2.43}$$

2. $\tilde{\psi}_{j,l}(x) = 0$ for $j \leq -1$. To see this we note first that, by an analysis similar to the above, $\tilde{\psi}_{j,l}(x) = \tilde{\psi}_{j,0}(x)$ for $j \leq 0$ and $\tilde{\psi}_{j,0}(x)$ has a Fourier expansion of the form (2.35). The same manipulations as in Theorem 2.9 yield the following Fourier coefficients for $\tilde{\psi}_{j,0}(x)$:

$$c_k = 2^{-j/2}\hat{\psi}(2\pi k 2^{-j}), \quad k \in \mathbf{Z} \tag{2.44}$$

From Lemma 2.11 we have that $\hat{\psi}(4\pi k) = 0,\ k \in \mathbf{Z}$. Hence $c_k = 0$ for $j \leq -1$ which means that

$$\boxed{\tilde{\psi}_{j,l}(x) = 0, \quad j \leq -1, \;\; l \in \mathbf{Z}, \;\; x \in \mathbf{R}} \tag{2.45}$$

When $j = 0$ in (2.44), one finds from (2.37) that $\hat{\psi}(2\pi k) \neq 0$ for k odd, so $\tilde{\psi}_{0,k}(x)$ is neither 0 nor another constant for such value of k. This is as expected since the role of $\tilde{\psi}_{0,k}(x)$ is to represent the details that are lost when projecting a function from an approximation at level 1 to level 0.

3. $\tilde{\phi}_{j,l}(x)$ and $\tilde{\psi}_{j,l}(x)$ are periodic in the shift parameter k with period 2^j for $j > 0$. We will show this only for $\tilde{\psi}_{j,l}$ since the proof is the same for $\tilde{\phi}_{j,l}$. Let $j > 0$, $p \in \mathbf{Z}$ and $0 \leq l \leq 2^j - 1$; then

$$\begin{aligned}
\tilde{\psi}_{j,l+2^j p}(x) &= \sum_{m=-\infty}^{\infty} \psi_{j,l+2^j p}(x+m) \\
&= 2^{j/2} \sum_{m=-\infty}^{\infty} \psi(2^j(x+m) - l - 2^j p) \\
&= 2^{j/2} \sum_{m=-\infty}^{\infty} \psi(2^j(x+m-p) - l) \\
&= 2^{j/2} \sum_{n=-\infty}^{\infty} \psi(2^j(x+n) - l) \\
&= \sum_{n=-\infty}^{\infty} \psi_{j,l}(x+n) \\
&= \tilde{\psi}_{j,l}(x), \quad x \in \mathbf{R}
\end{aligned}$$

Hence there are only 2^j distinct periodized wavelets:

$$\boxed{\left\{\tilde{\psi}_{j,l}\right\}_{l=0}^{2^j-1} \qquad j > 0}$$

4. Let $2^j \geq D - 1$: Rewriting (2.42) as follows yields an alternative formulation of the periodization process:

$$\tilde{\psi}_{j,l}(x) = 2^{j/2} \sum_{n=-\infty}^{\infty} \psi(2^j x + 2^j n - l) = \sum_{n=-\infty}^{\infty} \psi_{j,l-2^j n}(x) \tag{2.46}$$

Because ψ is compactly supported, the supports of the terms in this sum do not overlap provided that 2^j is sufficiently large. Let J_0 be the smallest integer such that

$$2^{J_0} \geq D - 1 \tag{2.47}$$

Then we see from (2.20) that for $j \geq J_0$ the width of $I_{j,l}$ is smaller than 1 and (2.46) implies that $\tilde{\psi}_{j,l}$ does not wrap in such a way that it overlaps itself. Consequently, the periodized scaling functions and wavelets can be described for $x \in [0,1]$ in terms of their non-periodic counterparts:

$$\tilde{\psi}_{j,l}(x) = \begin{cases} \psi_{j,l}(x), & x \in I_{j,l} \cap [0,1] \\ \psi_{j,l}(x+1), & x \in [0,1], \quad x \notin I_{j,l} \end{cases}$$

The above results are summarized in the following theorem:

Theorem 2.15 *Let the basic scaling function ϕ and wavelet ψ have support $[0, D-1]$, and let $\tilde{\phi}_{j,l}$ and $\tilde{\psi}_{j,l}$ be defined as in Definition 2.2. Then*

- $j \leq 0, \quad l \in \mathbf{Z}, \quad x \in \mathbf{R}$*:*

$$\begin{aligned} \tilde{\phi}_{j,l}(x) &= 2^{-j/2} \\ \tilde{\psi}_{j,l}(x) &= 0, \quad j \leq -1 \end{aligned}$$

- $j > 0, \quad x \in \mathbf{R}$*:*

$$\begin{aligned} \tilde{\phi}_{j,l+2^j p}(x) &= \tilde{\phi}_{j,l}(x) \\ \tilde{\psi}_{j,l+2^j p}(x) &= \tilde{\psi}_{j,l}(x) \end{aligned}$$

- $j > J_0 \geq \lceil \log_2(D-1) \rceil, \quad x \in [0,1]$*:*

$$\tilde{\phi}_{j,l}(x) = \begin{cases} \phi_{j,l}(x), & x \in I_{j,l} \\ \phi_{j,l}(x+1), & x \notin I_{j,l} \end{cases}$$

and

$$\tilde{\psi}_{j,l}(x) = \begin{cases} \psi_{j,l}(x), & x \in I_{j,l} \\ \psi_{j,l}(x+1), & x \notin I_{j,l} \end{cases}$$

2.3.2 Periodized MRA in $L^2([0,1])$

Many of the properties of the non-periodic scaling functions and wavelets carry over to the periodized versions restricted to the interval $[0,1]$. Wavelet orthonormality, for example, is preserved for the scales $i,j>0$:

$$\begin{aligned}
\int_0^1 \tilde{\psi}_{i,k}(x)\tilde{\psi}_{j,l}(x)\,dx &= \int_0^1 \sum_{m=-\infty}^{\infty} \psi_{i,k}(x+m)\tilde{\psi}_{j,l}(x)\,dx \\
&= \sum_{m=-\infty}^{\infty} \int_m^{m+1} \psi_{i,k}(y)\tilde{\psi}_{j,l}(y-m)\,dy \\
&= \sum_{m=-\infty}^{\infty} \int_m^{m+1} \psi_{i,k}(y)\tilde{\psi}_{j,l}(y)\,dy \\
&= \int_{-\infty}^{\infty} \psi_{i,k}(y)\tilde{\psi}_{j,l}(y)\,dy
\end{aligned}$$

Using (2.46) for the second function and invoking the orthogonality relation for non-periodic wavelets (2.9) gives

$$\int_0^1 \tilde{\psi}_{i,k}(x)\tilde{\psi}_{j,l}(x)\,dx = \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} \psi_{i,k}(x)\psi_{j,l-2^j n}(x)\,dx = \delta_{i,j} \sum_{n=-\infty}^{\infty} \delta_{k,l-2^j n}$$

If $i=j$ then $\delta_{i,j}=1$ and $\delta_{k,l-2^j n}$ contributes only when $n=0$ and $k=l$ because $k,l \in [0, 2^j-1]$. Hence,

$$\boxed{\int_0^1 \tilde{\psi}_{i,k}(x)\tilde{\psi}_{j,l}(x)\,dx = \delta_{i,j}\delta_{k,l}} \tag{2.48}$$

as desired. By a similar analysis one can establish the relations

$$\begin{aligned}
\int_0^1 \tilde{\phi}_{j,k}(x)\tilde{\phi}_{j,l}(x)\,dx &= \delta_{k,l}, \quad j \geq 0 \\
\int_0^1 \tilde{\phi}_{i,k}(x)\tilde{\psi}_{j,l}(x)\,dx &= 0, \quad j \geq i \geq 0
\end{aligned}$$

The periodized wavelets and scaling functions restricted to $[0,1]$ generate a multiresolution analysis of $L^2([0,1])$ analogous to that of $L^2(\mathbf{R})$. The relevant subspaces are given by

Definition 2.3

$$\begin{aligned}
\tilde{V}_j &= \text{span}\left\{\tilde{\phi}_{j,l}, \quad x \in [0,1]\right\}_{l=0}^{2^j-1} \\
\tilde{W}_j &= \text{span}\left\{\tilde{\psi}_{j,l}, \quad x \in [0,1]\right\}_{l=0}^{2^j-1}
\end{aligned}$$

It turns out [Dau92, p. 305] that the $\tilde{V}_j$ are nested as in the non-periodic MRA,

$$\tilde{V}_0 \subset \tilde{V}_1 \subset \tilde{V}_2 \subset \cdots \subset L^2([0,1])$$

and that the $\overline{\bigcup_{j=0}^{\infty} \tilde{V}_j} = L^2([0,1])$. In addition, the orthogonality relations imply that

$$\tilde{V}_j \oplus \tilde{W}_j = \tilde{V}_{j+1} \tag{2.49}$$

so we have the decomposition

$$L^2([0,1]) = \tilde{V}_0 \oplus \left(\bigoplus_{j=0}^{\infty} \tilde{W}_j \right) \tag{2.50}$$

From Theorem 2.15 and (2.50) we then see that the system

$$\left\{ 1, \quad \left\{ \left\{ \tilde{\psi}_{j,k} \right\}_{k=0}^{2^j-1} \right\}_{j=0}^{\infty} \right\} \tag{2.51}$$

is an orthonormal basis for $L^2([0,1])$. This basis is canonical in the sense that the space $L^2([0,1])$ is fully decomposed as in (2.50); i.e. the orthogonal decomposition process cannot be continued further because, as stated in (2.45), $\tilde{W}_j = \{0\}$ for $j \leq -1$. Note that the scaling functions no longer appear explicitly in the expansion since they have been replaced by the constant 1 according to (2.43).

Sometimes one wants to use the basis associated with the decomposition

$$L^2([0,1]) = \tilde{V}_{J_0} \oplus \left(\bigoplus_{j=J_0}^{\infty} \tilde{W}_j \right)$$

for some $J_0 > 0$. We recall that if $J_0 \geq \log_2(D-1)$ then the non-periodic basis functions do not overlap. This property is exploited in the parallel algorithm described in Chapter 6.

2.3.3 Expansions of periodic functions

Let $f \in \tilde{V}_J$ and let J_0 satisfy $0 \leq J_0 \leq J$. The decomposition

$$\tilde{V}_J = \tilde{V}_{J_0} \oplus \left(\bigoplus_{j=J_0}^{J-1} \tilde{W}_j \right)$$

which is obtained from (2.49), leads to two expansions of f, namely the pure periodic scaling function expansion

$$f(x) = \sum_{l=0}^{2^J-1} c_{J,l}\tilde{\phi}_{J,l}(x), \quad x \in [0,1] \tag{2.52}$$

and the periodic wavelet expansion

$$f(x) = \sum_{l=0}^{2^{J_0}-1} c_{J_0,l}\tilde{\phi}_{J_0,l}(x) + \sum_{j=J_0}^{J-1}\sum_{l=0}^{2^j-1} d_{j,l}\tilde{\psi}_{j,l}(x), \quad x \in [0,1] \tag{2.53}$$

If $J_0 = 0$ then (2.53) becomes

$$\boxed{f(x) = c_{0,0} + \sum_{j=0}^{J-1}\sum_{l=0}^{2^j-1} d_{j,l}\tilde{\psi}_{j,l}(x)} \tag{2.54}$$

corresponding to a truncation of the canonical basis (2.51).

Let now $\tilde{f}$ be the periodic extension of f, i.e.

$$\tilde{f}(x) = f(x - \lfloor x \rfloor), \quad x \in \mathbf{R} \tag{2.55}$$

Then $\tilde{f}$ is 1-periodic

$$\tilde{f}(x+1) = f(x+1-(\lfloor x+1 \rfloor)) = f(x - \lfloor x \rfloor) = \tilde{f}(x), \quad x \in \mathbf{R}$$

Because $\lfloor x \rfloor$ is an integer, we have $\tilde{\phi}(x - \lfloor x \rfloor) = \tilde{\phi}(x)$ and $\tilde{\psi}(x - \lfloor x \rfloor) = \tilde{\psi}(x)$ for $x \in \mathbf{R}$. Using (2.55) in (2.52) we obtain

$$\tilde{f}(x) = f(x - \lfloor x \rfloor) = \sum_{l=0}^{2^J-1} c_{J,l}\tilde{\phi}_{J,l}(x - \lfloor x \rfloor) = \sum_{l=0}^{2^J-1} c_{J,l}\tilde{\phi}_{J,l}(x), \quad x \in \mathbf{R} \tag{2.56}$$

and by a similar argument

$$\tilde{f}(x) = \sum_{l=0}^{2^{J_0}-1} c_{J_0,l}\tilde{\phi}_{J_0,l}(x) + \sum_{j=J_0}^{J-1}\sum_{l=0}^{2^j-1} d_{j,l}\tilde{\psi}_{j,l}(x), \quad x \in \mathbf{R} \tag{2.57}$$

The coefficients in (2.52) and (2.53) are defined by

$$\begin{aligned} c_{j,l} &= \int_0^1 f(x)\tilde{\phi}_{j,l}(x)\,dx \\ d_{j,l} &= \int_0^1 f(x)\tilde{\psi}_{j,l}(x)\,dx \end{aligned}$$

but it turns out that they are, in fact, the *same* as those of the non-periodic expansions. To see this we use the fact that $f(x) = \tilde{f}(x), \ \ x \in [0,1]$ and write

$$
\begin{aligned}
d_{j,l} &= \int_0^1 \tilde{f}(x)\tilde{\psi}_{j,l}(x)\,dx \\
&= \sum_{n=-\infty}^{\infty} \int_0^1 \tilde{f}(x)\psi_{j,l}(x+n)\,dx \\
&= \sum_{n=-\infty}^{\infty} \int_n^{n+1} \tilde{f}(y-n)\psi_{j,l}(y)\,dy \\
&= \sum_{n=-\infty}^{\infty} \int_n^{n+1} \tilde{f}(y)\psi_{j,l}(y)\,dy \\
&= \int_{-\infty}^{\infty} \tilde{f}(y)\psi_{j,l}(y)\,dy
\end{aligned}
\tag{2.58}
$$

which is the coefficient in the non-periodic case. Similarly, we find that

$$
c_{j,l} = \int_0^1 \tilde{f}(x)\tilde{\phi}_{j,l}(x)\,dx = \int_{-\infty}^{\infty} \tilde{f}(x)\psi_{j,l}(x)\,dx
$$

However, periodicity in $\tilde{f}$ induces periodicity in the wavelet coefficients:

$$
\begin{aligned}
d_{j,l+2^j p} &= \int_{-\infty}^{\infty} \tilde{f}(x)\psi_{j,l+2^j p}(x)\,dx \\
&= \int_{-\infty}^{\infty} \tilde{f}(x)2^{j/2}\psi(2^j(x-p)-l)\,dx \\
&= \int_{-\infty}^{\infty} \tilde{f}(y+p)2^{j/2}\psi(2^j y-l)\,dx \\
&= \int_{-\infty}^{\infty} \tilde{f}(y)\psi_{j,l}(y)\,dx \\
&= d_{j,l}
\end{aligned}
\tag{2.59}
$$

Similarly,

$$
c_{j,l+2^j p} = c_{j,l} \tag{2.60}
$$

For simplicity we will drop the tilde and identify f with its periodic extension $\tilde{f}$ throughout this study. Finally, we define

Definition 2.4 *Let $P_{\tilde{V}_j}$ and $P_{\tilde{W}_j}$ denote the operators that project any $f \in L^2([0,1])$ orthogonally onto $\tilde{V}_j$ and $\tilde{W}_j$, respectively. Then*

$$
\begin{aligned}
(P_{\tilde{V}_j} f)(x) &= \sum_{l=-\infty}^{\infty} c_{j,l} \tilde{\phi}_{j,l}(x) \\
(P_{\tilde{W}_j} f)(x) &= \sum_{l=-\infty}^{\infty} d_{j,l} \tilde{\psi}_{j,l}(x)
\end{aligned}
$$

where

$$
\begin{aligned}
c_{j,l} &= \int_0^1 f(x) \tilde{\phi}_{j,l}(x)\, dx \\
d_{j,l} &= \int_0^1 f(x) \tilde{\psi}_{j,l}(x)\, dx
\end{aligned}
$$

and

$$
P_{\tilde{V}_J} f = P_{\tilde{V}_{J_0}} f + \sum_{j=J_0}^{J-1} P_{\tilde{W}_j} f
$$

Chapter 3

Wavelet algorithms

3.1 Numerical evaluation of ϕ and ψ

Generally, there are no explicit formulas for the basic functions ϕ and ψ. Hence most algorithms concerning scaling functions and wavelets are formulated in terms of the filter coefficients. A good example is the task of computing the function values of ϕ and ψ. Such an algorithm is useful when one wants to make plots of scaling functions and wavelets or of linear combinations of such functions.

3.1.1 Computing ϕ at integers

The scaling function ϕ has support on the interval $[0, D-1]$, with $\phi(0) = 0$ and $\phi(D-1) = 0$ because it is continuous for $D \geq 4$ [Dau92, p. 232]. We will discard $\phi(D-1)$ in our computations, but, for reasons explained in the next section, we keep $\phi(0)$.

Putting $x = 0, 1, \ldots, D-2$ in the dilation equation (2.17) yields a homogeneous linear system of equations, shown here for $D = 6$.

$$\begin{bmatrix} \phi(0) \\ \phi(1) \\ \phi(2) \\ \phi(3) \\ \phi(4) \end{bmatrix} = \sqrt{2} \begin{bmatrix} a_0 & & & & \\ a_2 & a_1 & a_0 & & \\ a_4 & a_3 & a_2 & a_1 & a_0 \\ & a_5 & a_4 & a_3 & a_2 \\ & & & a_5 & a_4 \end{bmatrix} \begin{bmatrix} \phi(0) \\ \phi(1) \\ \phi(2) \\ \phi(3) \\ \phi(4) \end{bmatrix} = \boldsymbol{A}_0 \Phi(0) \tag{3.1}$$

where we have defined the vector valued function

$$\Phi(x) = [\phi(x), \phi(x+1), \ldots, \phi(x+D-2)]^T$$

Consider then the eigenvalue problem for $\boldsymbol{A}_0$,

$$\boldsymbol{A}_0 \Phi(0) = \lambda \Phi(0) \tag{3.2}$$

Equation (3.1) has a solution if $\lambda = 1$ is among the eigenvalues of $\boldsymbol{A}_0$. Hence the computational problem amounts to finding the eigensolutions of (3.2). It is shown in [SN96, p. 199] that the eigenvalues of $\boldsymbol{A}_0$ include

$$\lambda = 2^{-m}, \quad m = 0, 1, \ldots, D/2 - 1$$

The present case (3.1) does, indeed, have an eigensolution corresponding to the eigenvalue 1, and we use the result from Theorem 2.9 to fix the inherent multiplicative constant. Consequently, we choose the solution where

$$\sum_{k=0}^{D-1} \phi(k) = 1$$

3.1.2 Computing ϕ at dyadic rationals.

Given $\Phi(0)$ from (3.1) we can use (2.17) again to obtain ϕ at all *midpoints* between integers in the interval, namely the vector $\Phi(1/2)$. Substituting $x = \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \ldots$ into the dilation equation yields another matrix equation of the form (still shown for $D = 6$):

$$\Phi\left(\frac{1}{2}\right) = \begin{bmatrix} \phi(\frac{1}{2}) \\ \phi(\frac{3}{2}) \\ \phi(\frac{5}{2}) \\ \phi(\frac{7}{2}) \\ \phi(\frac{9}{2}) \end{bmatrix} = \sqrt{2} \begin{bmatrix} a_1 & a_0 & & & \\ a_3 & a_2 & a_1 & a_0 & \\ a_5 & a_4 & a_3 & a_2 & a_1 \\ & & a_5 & a_4 & a_3 \\ & & & & a_5 \end{bmatrix} \begin{bmatrix} \phi(0) \\ \phi(1) \\ \phi(2) \\ \phi(3) \\ \phi(4) \end{bmatrix} = \mathbf{A}_1 \Phi(0) \tag{3.3}$$

Equation (3.3) is an explicit formula, hence

$$\begin{aligned} \Phi(0) &= \mathbf{A}_0 \Phi(0) \\ \Phi(1/2) &= \mathbf{A}_1 \Phi(0) \end{aligned}$$

This pattern continues to integers of the form $k/4$, where k is odd, and we get the following system

$$\begin{bmatrix} * & \phi(\frac{1}{4}) \\ \circ & \phi(\frac{3}{4}) \\ * & \phi(\frac{5}{4}) \\ \circ & \phi(\frac{7}{4}) \\ * & \phi(\frac{9}{4}) \\ \circ & \phi(\frac{11}{4}) \\ * & \phi(\frac{13}{4}) \\ \circ & \phi(\frac{15}{4}) \\ * & \phi(\frac{17}{4}) \\ \circ & \phi(\frac{19}{4}) \end{bmatrix} = \sqrt{2} \begin{bmatrix} a_0 & & & & \\ a_1 & a_0 & & & \\ a_2 & a_1 & a_0 & & \\ a_3 & a_2 & a_1 & a_0 & \\ a_4 & a_3 & a_2 & a_1 & a_0 \\ a_5 & a_4 & a_3 & a_2 & a_1 \\ & a_5 & a_4 & a_3 & a_2 \\ & & a_5 & a_4 & a_3 \\ & & & a_5 & a_4 \\ & & & & a_5 \end{bmatrix} \begin{bmatrix} \phi(\frac{1}{2}) \\ \phi(\frac{3}{2}) \\ \phi(\frac{5}{2}) \\ \phi(\frac{7}{2}) \\ \phi(\frac{9}{2}) \end{bmatrix} \tag{3.4}$$

Equation (3.4) could be used as it is but we observe that if we split it in two systems, one with the equations marked with $*$ and one with the equations marked with $\circ$, we can reuse the matrices $\mathbf{A}_0$ and $\mathbf{A}_1$. This pattern repeats itself as follows:

$$
\begin{aligned}
\Phi(\tfrac{1}{4}) &= \mathbf{A}_0\Phi(\tfrac{1}{2}) \\
\Phi(\tfrac{3}{4}) &= \mathbf{A}_1\Phi(\tfrac{1}{2})
\end{aligned}
$$

$$
\begin{aligned}
\Phi(\tfrac{1}{8}) &= \mathbf{A}_0\Phi(\tfrac{1}{4}) & \Phi(\tfrac{5}{8}) &= \mathbf{A}_1\Phi(\tfrac{1}{4}) \\
\Phi(\tfrac{3}{8}) &= \mathbf{A}_0\Phi(\tfrac{3}{4}) & \Phi(\tfrac{7}{8}) &= \mathbf{A}_1\Phi(\tfrac{3}{4})
\end{aligned}
$$

$$
\begin{aligned}
\Phi(\tfrac{1}{16}) &= \mathbf{A}_0\Phi(\tfrac{1}{8}) & \Phi(\tfrac{9}{16}) &= \mathbf{A}_1\Phi(\tfrac{1}{8}) \\
\Phi(\tfrac{3}{16}) &= \mathbf{A}_0\Phi(\tfrac{3}{8}) & \Phi(\tfrac{11}{16}) &= \mathbf{A}_1\Phi(\tfrac{3}{8}) \\
\Phi(\tfrac{5}{16}) &= \mathbf{A}_0\Phi(\tfrac{5}{8}) & \Phi(\tfrac{13}{16}) &= \mathbf{A}_1\Phi(\tfrac{5}{8}) \\
\Phi(\tfrac{7}{16}) &= \mathbf{A}_0\Phi(\tfrac{7}{8}) & \Phi(\tfrac{15}{16}) &= \mathbf{A}_1\Phi(\tfrac{7}{8})
\end{aligned}
$$

This is the reason why we keep $\phi(0)$ in the initial eigenvalue problem (3.1): We can use the same two matrices for all steps in the algorithm and we can continue as follows until a desired resolution 2^q is obtained:

for $j = 2, 3, \ldots, q$
    for $k = 1, 3, 5, \ldots, 2^{j-1} - 1$

$$
\begin{aligned}
\Phi\left(\frac{k}{2^j}\right) &= \mathbf{A}_0\Phi\left(\frac{k}{2^{j-1}}\right) \\
\Phi\left(\frac{k}{2^j} + \frac{1}{2}\right) &= \mathbf{A}_1\Phi\left(\frac{k}{2^{j-1}}\right)
\end{aligned}
$$

3.1.3 Function values of the basic wavelet:

Function values of ψ follows immediately from the computed values of ϕ by the wavelet equation (2.18). However, function values of ϕ are only needed at even numerators:

$$\psi(m/2^q) = \sqrt{2}\sum_{k=0}^{D-1} b_k \phi(2m/2^q - k)$$

The Matlab function `cascade(D,q)` computes $\phi(x)$ and $\psi(x)$ at $x = k/2^q$, $k = 0,\ 1/2^q,\ \ldots,\ (D-1)/2^q$.

Figure 3.1 shows ϕ and ψ for different values of D.

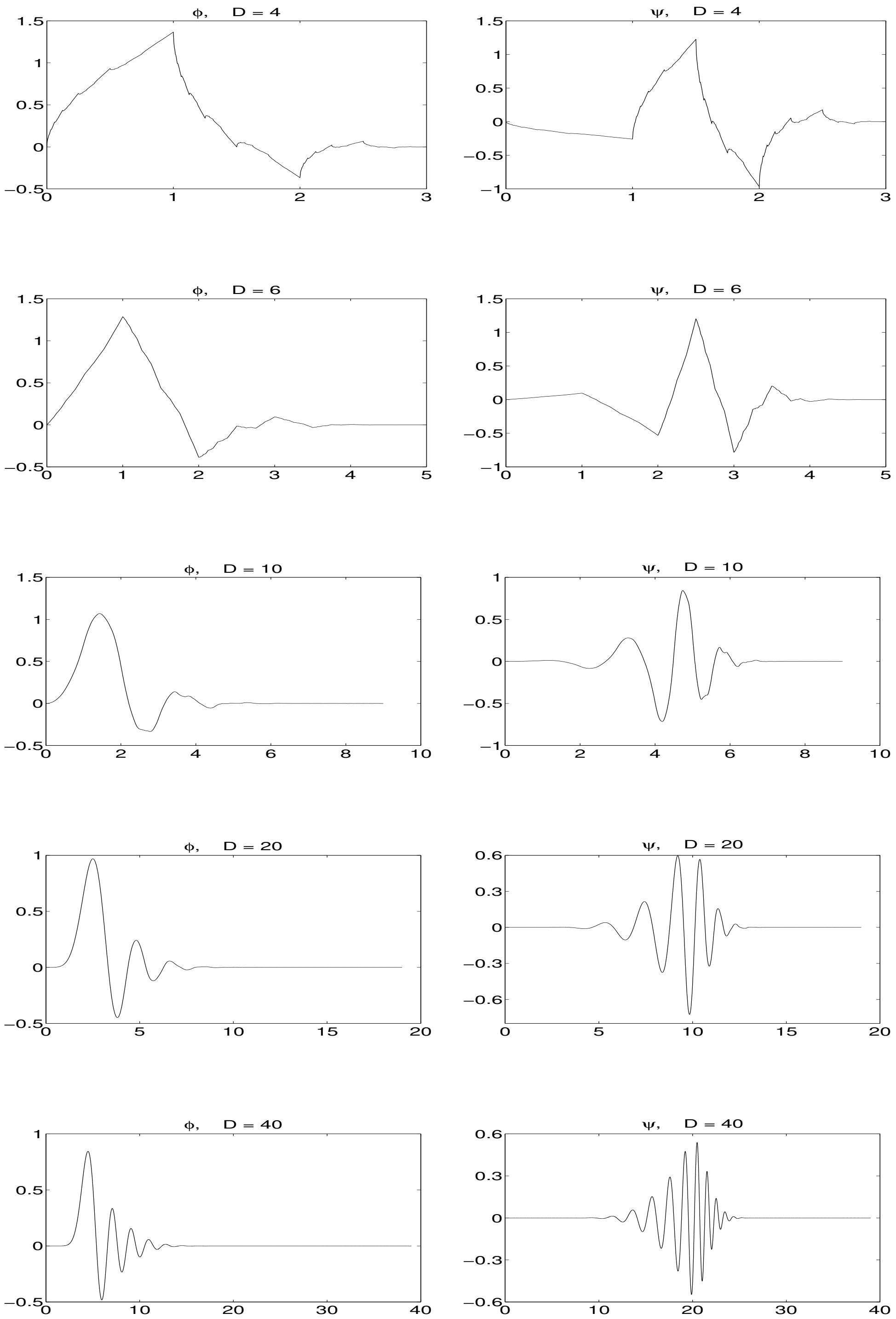


Figure 3.1: Basic scaling functions and wavelets plotted for $D = 4, 6, 10, 20, 40$.

3.2 Evaluation of scaling function expansions

3.2.1 Nonperiodic case

Let ϕ be the basic scaling function of genus D and assume that ϕ is known at the dyadic rationals $m/2^q$, $m = 0, 1, \ldots, (D-1)2^q$, for some chosen $q \in \mathbf{N}$. We want to compute the function

$$f(x) = \sum_{l=-\infty}^{\infty} c_{j,l}\phi_{j,l}(x) \tag{3.5}$$

at the grid points

$$x = x_k = k/2^r, \quad k \in \mathbf{Z} \tag{3.6}$$

where $r \in \mathbf{N}$ corresponds to some chosen (dyadic) resolution of the real line. Using (2.6) we find that

$$\begin{aligned} \phi_{j,l}(k/2^r) &= 2^{j/2}\phi(2^j(k/2^r) - l) \\ &= 2^{j/2}\phi(2^{j-r}k - l) \\ &= 2^{j/2}\phi((2^{j+q-r}k - 2^q l)/2^q) \\ &= 2^{j/2}\phi(m(k,l)/2^q) \end{aligned} \tag{3.7}$$

where

$$m(k,l) = k2^{j+q-r} - l2^q \tag{3.8}$$

Hence, $m(k,l)$ serves as an index into the vector of pre-computed values of ϕ. For this to make sense $m(k,l)$ must be an integer, which leads to the restriction

$$j + q - r \geq 0 \tag{3.9}$$

Only $D-1$ terms of (3.5) can be nonzero for any given x_k. From (3.7) we see that these terms are determined by the condition

$$0 < \frac{m(k,l)}{2^q} < D - 1$$

Hence, the relevant values of l are $l = l_0(k), l_0(k)+1, \ldots, l_0(k)+D-2$, where

$$l_0(k) = \lceil k2^{j-r} \rceil - D + 1 \tag{3.10}$$

The sum (3.5), for x given by (3.6), can therefore be written as

$$\boxed{f\left(\frac{k}{2^r}\right) = 2^{j/2} \sum_{l=l_0(k)}^{l_0(k)+D-2} c_{j,l}\phi\left(\frac{m(k,l)}{2^q}\right), \quad k \in \mathbf{Z}} \tag{3.11}$$

3.2.2 Periodic case

We want to compute the function

$$f(x) = \sum_{l=0}^{2^j-1} c_{j,l}\tilde{\phi}_{j,l}(x), \quad x \in [0,1] \tag{3.12}$$

for $x = x_k = k/2^r$, $k = 0, 1, \ldots, 2^r - 1$ where $r \in \mathbf{N}$. Hence we have

$$\begin{aligned} f\left(\frac{k}{2^r}\right) &= \sum_{l=0}^{2^j-1} c_{j,l}\tilde{\phi}_{j,l}\left(\frac{k}{2^r}\right) \\ &= \sum_{l=0}^{2^j-1} c_{j,l} \sum_{n\in\mathbf{Z}} \phi_{j,l}\left(\frac{k}{2^r}+n\right) \\ &= 2^{j/2}\sum_{l=0}^{2^j-1} c_{j,l} \sum_{n\in\mathbf{Z}} \phi\left(\frac{m(k,l)+2^{j+q}n}{2^q}\right) \end{aligned}$$

with $m(k,l) = k2^{j+q-r} - l2^q$ by the same manipulation as in (3.7). Now, assuming that $j \geq J_0$ where J_0 is given by (2.47) we have $2^j \geq D-1$. Using Lemma 3.1 (proved below), we obtain the expression

$$\boxed{f\left(\frac{k}{2^r}\right) = 2^{j/2}\sum_{l=0}^{2^j-1} c_{j,l}\phi\left(\frac{\langle m(k,l)\rangle_{2^{j+q}}}{2^q}\right), \quad k = 0, 1, \ldots, 2^r - 1} \tag{3.13}$$

Lemma 3.1 *Let ϕ be a scaling function with support $[0, D-1]$ and let $m, n, j, q \in \mathbf{Z}$ with $q \geq 0$ and $2^j \geq D-1$. Then*

$$\sum_{n=-\infty}^{\infty} \phi\left(\frac{m+2^{j+q}n}{2^q}\right) = \phi\left(\frac{\langle m\rangle_{2^{j+q}}}{2^q}\right)$$

Proof: Since $2^j \geq D-1$, only one term of the sum contributes to the result and there exists a unique $n = n^*$ such that

$$\frac{m+2^{j+q}n^*}{2^q} \in [0, 2^j]$$

Then we know that $m + 2^{j+q}n^* \in [0, 2^{j+q}]$ but this interval is precisely the range of the modulus operator defined in Appendix B, so

$$m + 2^{j+q}n^* = \langle m + 2^{j+q}n\rangle_{2^{j+q}} = \langle m\rangle_{2^{j+q}}, \quad n \in \mathbf{Z}$$

from which the result follows. □

3.2.3 DST and IDST - matrix formulation

Equation (3.13) is a linear mapping from 2^j scaling function coefficients to 2^r samples of f, so it has a matrix formulation. Let $\boldsymbol{c}_j = [c_{j,0}, c_{j,1}, \ldots, c_{j,2^j-1}]^T$ and $\boldsymbol{f}_r = [f(0), f(1/2^r), \ldots, f((2^r-1)/2^r)]^T$. We denote the mapping

$$\boldsymbol{f}_r = \boldsymbol{T}_{r,j}\boldsymbol{c}_j \tag{3.14}$$

When $r = j$ then (3.14) becomes

$$\boldsymbol{f}_j = \boldsymbol{T}_{j,j}\boldsymbol{c}_j \tag{3.15}$$

$\boldsymbol{T}_{j,j}$ is a square matrix of order $N = 2^j$. In the case of (3.15) we will often drop the subscripts and write simply

$$\boxed{\boldsymbol{f} = \boldsymbol{T}\boldsymbol{c}} \tag{3.16}$$

This has the form (shown here for $j = 3$, $D = 4$)

$$\begin{pmatrix} f(0) \\ f(\frac{1}{8}) \\ f(\frac{2}{8}) \\ f(\frac{3}{8}) \\ f(\frac{4}{8}) \\ f(\frac{5}{8}) \\ f(\frac{6}{8}) \\ f(\frac{7}{8}) \end{pmatrix} = 2^{\frac{3}{2}} \begin{pmatrix} \phi(0) & & & & & & \phi(2) & \phi(1) \\ \phi(1) & \phi(0) & & & & & & \phi(2) \\ \phi(2) & \phi(1) & \phi(0) & & & & & \\ & \phi(2) & \phi(1) & \phi(0) & & & & \\ & & \phi(2) & \phi(1) & \phi(0) & & & \\ & & & \phi(2) & \phi(1) & \phi(0) & & \\ & & & & \phi(2) & \phi(1) & \phi(0) & \\ & & & & & \phi(2) & \phi(1) & \phi(0) \end{pmatrix} \begin{pmatrix} c_{3,0} \\ c_{3,1} \\ c_{3,2} \\ c_{3,3} \\ c_{3,4} \\ c_{3,5} \\ c_{3,6} \\ c_{3,7} \end{pmatrix}$$

Note that only values of ϕ at the integers appear in $\boldsymbol{T}$. The matrix T is non-singular and we can write

$$\boldsymbol{c} = \boldsymbol{T}^{-1}\boldsymbol{f} \tag{3.17}$$

We denote (3.17) the **discrete scaling function transform** (DST)[1] and (3.16) the **inverse discrete scaling function transform** (IDST) .

The Matlab function `dst(`$\boldsymbol{f}$`,`D`)` computes (3.17) and `idst(`$\boldsymbol{c}$`,`D`)` computes (3.16).

---

[1]DST should not be confused with the discrete sine transform.

We now consider the the computational problem of interpolating a function $f \in C([0,1])$ between samples at resolution r. That is, we want to use the function values $f(k/2^r), k = 0, 1, \dots, 2^r - 1$ to compute approximations to $f(k/2^{r'}), k = 0, 1, \dots, 2^{r'} - 1$ for some $r' > r$. There are two steps. The first is to solve the system

$$\boldsymbol{T}_{r,r}\boldsymbol{c}_r = \boldsymbol{f}_r$$

for $\boldsymbol{c}_r$. The second is to compute the vector $\boldsymbol{f}_{r'}$ defined by

$$\boldsymbol{f}_{r'} = \boldsymbol{T}_{r',r}\boldsymbol{c}_r \tag{3.18}$$

Equation (3.18) is illustrated below for the case $r = 3, r' = 4$.

$$\begin{pmatrix} f(0) \\ f(\frac{1}{16}) \\ f(\frac{2}{16}) \\ f(\frac{3}{16}) \\ f(\frac{4}{16}) \\ f(\frac{5}{16}) \\ f(\frac{6}{16}) \\ f(\frac{7}{16}) \\ f(\frac{8}{16}) \\ f(\frac{9}{16}) \\ f(\frac{10}{16}) \\ f(\frac{11}{16}) \\ f(\frac{12}{16}) \\ f(\frac{13}{16}) \\ f(\frac{14}{16}) \\ f(\frac{15}{16}) \end{pmatrix} = 2^{\frac{3}{2}} \begin{pmatrix} \phi(0) & & & & & & \phi(2) & \phi(1) \\ \phi(\frac{1}{2}) & & & & & & \phi(\frac{5}{2}) & \phi(\frac{3}{2}) \\ \phi(1) & \phi(0) & & & & & & \phi(2) \\ \phi(\frac{3}{2}) & \phi(\frac{1}{2}) & & & & & & \phi(\frac{5}{2}) \\ \phi(2) & \phi(1) & \phi(0) & & & & & \\ \phi(\frac{5}{2}) & \phi(\frac{3}{2}) & \phi(\frac{1}{2}) & & & & & \\ & \phi(2) & \phi(1) & \phi(0) & & & & \\ & \phi(\frac{5}{2}) & \phi(\frac{3}{2}) & \phi(\frac{1}{2}) & & & & \\ & & \phi(2) & \phi(1) & \phi(0) & & & \\ & & \phi(\frac{5}{2}) & \phi(\frac{3}{2}) & \phi(\frac{1}{2}) & & & \\ & & & \phi(2) & \phi(1) & \phi(0) & & \\ & & & \phi(\frac{5}{2}) & \phi(\frac{3}{2}) & \phi(\frac{1}{2}) & & \\ & & & & \phi(2) & \phi(1) & \phi(0) & \\ & & & & \phi(\frac{5}{2}) & \phi(\frac{3}{2}) & \phi(\frac{1}{2}) & \\ & & & & & \phi(2) & \phi(1) & \phi(0) \\ & & & & & \phi(\frac{5}{2}) & \phi(\frac{3}{2}) & \phi(\frac{1}{2}) \end{pmatrix} \begin{pmatrix} c_{3,0} \\ c_{3,1} \\ c_{3,2} \\ c_{3,3} \\ c_{3,4} \\ c_{3,5} \\ c_{3,6} \\ c_{3,7} \end{pmatrix}$$

Figure 3.2 show how scaling functions can interpolate between samples of a sine function.

3.2.4 Periodic functions on the interval $[a, b]$

Consider the problem of expressing a periodic function f defined on the interval $[a, b]$, where $a, b \in \mathbf{R}$ instead of the unit interval. This can be accomplished by mapping the interval $[a, b]$ linearly to the unit interval and then use the machinery derived in Section 3.2.3.

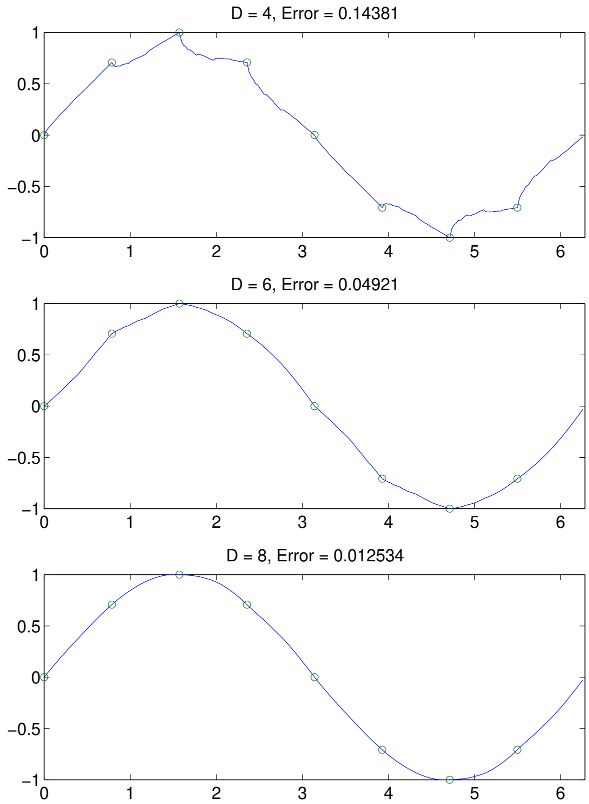


Figure 3.2: A sine function is sampled in 8 points ($r = 3$). Scaling functions of genus $D = 4, 6, 8$ are then used to interpolate in 128 points ($r' = 7$).

We impose the resolution 2^r on the interval $[a, b[$, i.e.

$$x_k = k\frac{b-a}{2^r} + a, \quad k = 0, 1, \dots, 2^r - 1$$

The linear mapping of the interval $a \le x \le b$ to the interval $0 \le y \le 1$ is given by

$$y = \frac{x-a}{b-a}, \qquad a \le x < b$$

hence $y_k = k/2^r, k = 0, 1, \dots, 2^r - 1$. Let

$$g(y) = f(x) = f((b-a)y + a), \qquad 0 \le y < 1$$

Then we have from (3.13)

$$g(y_k) = g(k/2^r) = 2^{j/2} \sum_{l=0}^{2^j-1} c_{j,l} \phi \left(\frac{\langle m(k,l) \rangle_{2^{j+q}}}{2^q} \right)$$

and transforming back to the interval $[a, b[$ yields $f(x_k) = g(y_k)$. Thus we have effectively obtained an expansion of $f \in [a, b[$ in terms of scaling functions "stretched" to fit this interval at its dyadic subdivisions.

3.3 Fast Wavelet Transforms

The orthogonality of scaling functions and wavelets together with the dyadic coupling between MRA spaces lead to a relation between scaling function coefficients and wavelet coefficients on different scales. This yields a fast and accurate algorithm due to Mallat [Mal89] denoted the **pyramid algorithm** or the **fast wavelet transform** (FWT). We use the latter name.

Let $f \in L^2(\mathbf{R})$ and consider the projection

$$(P_{V_j} f)(x) = \sum_{l=-\infty}^{\infty} c_{j,l} \phi_{j,l}(x) \tag{3.19}$$

which is given in terms of scaling functions only. We know from Definition 2.1 that $P_{V_j} f = P_{V_{j-1}} f + P_{W_{j-1}} f$ so the projection also has a formulation in terms of scaling functions *and* wavelets:

$$(P_{V_j} f)(x) = \sum_{l=-\infty}^{\infty} c_{j-1,l} \phi_{j-1,l}(x) + \sum_{l=-\infty}^{\infty} d_{j-1,l} \psi_{j-1,l}(x) \tag{3.20}$$

Our goal here is to derive a mapping between the sequence $\{c_{j,l}\}_{l \in \mathbf{Z}}$ and the sequences

$$\{c_{j-1,l}\}_{l \in \mathbf{Z}}, \quad \{d_{j-1,l}\}_{l \in \mathbf{Z}}$$

The key to the derivations is the dilation equation (2.17) and the wavelet equation (2.18). Using (2.17) we derive the identity

$$\begin{aligned} \phi_{j-1,l}(x) &= 2^{(j-1)/2} \phi(2^{j-1} x - l) \\ &= 2^{j/2} \sum_{k=0}^{D-1} a_k \phi(2^j x - 2l - k) \\ &= \sum_{k=0}^{D-1} a_k \phi_{j,2l+k}(x) \end{aligned} \tag{3.21}$$

and from (2.18) we have

$$\psi_{j-1,l}(x) = \sum_{k=0}^{D-1} b_k \phi_{j,2l+k}(x) \tag{3.22}$$

Substituting the first identity into the definition of $c_{j,l}$ (2.12) we obtain

$$\begin{aligned} c_{j-1,l} &= \int_{-\infty}^{\infty} f(x) \sum_{k=0}^{D-1} a_k \phi_{j,2l+k}(x)\, dx \\ &= \sum_{k=0}^{D-1} a_k \int_{-\infty}^{\infty} f(x)\phi_{j,2l+k}(x)\, dx \\ &= \sum_{k=0}^{D-1} a_k c_{j,2l+k} \end{aligned}$$

and similarly for $d_{j-1,l}$. Thus, we obtain the relations

$$c_{j-1,l} = \sum_{k=0}^{D-1} a_k c_{j,2l+k} \tag{3.23}$$

$$d_{j-1,l} = \sum_{k=0}^{D-1} b_k c_{j,2l+k} \tag{3.24}$$

which define a linear mapping from the coefficients in (3.19) to the coefficients in (3.20). We will refer to this as the **partial wavelet transform** (PWT) . To decompose the space V_j further, one applies the mapping to the sequence $\{c_{j-1,l}\}_{l\in\mathbf{Z}}$ to obtain the new sequences $\{c_{j-2,l}\}_{l\in\mathbf{Z}}$ and $\{d_{j-2,l}\}_{l\in\mathbf{Z}}$. This pattern can be further repeated yielding the full FWT: Applying (3.23) and (3.24) recursively for $j = J, J-1, \ldots, J_0+1$, starting with the initial sequence $\{c_{J,l}\}_{l\in\mathbf{Z}}$, will then produce the wavelet coefficients in the expansion given in (2.13). Note that once the elements $d_{j-1,l}$ have been computed, they are not modified in subsequent steps. This means that the FWT is very efficient in terms of computational work.

The inverse mapping can be derived in a similar fashion. Equating (3.19) with (3.20) and using (3.21), (3.22) again we get

$$\begin{aligned} \sum_{l=-\infty}^{\infty} c_{j,l}\phi_{j,l}(x) &= \sum_{n=-\infty}^{\infty} c_{j-1,n}\phi_{j-1,n}(x) + \sum_{n=-\infty}^{\infty} d_{j-1,n}\psi_{j-1,n}(x) \\ &= \sum_{n=-\infty}^{\infty} c_{j-1,n} \sum_{k=0}^{D-1} a_k\phi_{j,2n+k}(x) + \sum_{n=-\infty}^{\infty} d_{j-1,n} \sum_{k=0}^{D-1} b_k\phi_{j,2n+k}(x) \\ &= \sum_{k=0}^{D-1} \sum_{n=-\infty}^{\infty} \left[c_{j-1,n}a_k + d_{j-1,n}b_k\right] \phi_{j,2n+k}(x) \end{aligned}$$

We now introduce the variable $l = 2n+k$ in the last expression. Since $k = l-2n$ and $k \in [0, D-1]$, we find for given l the following bounds on n:

$$\left\lceil \frac{l-D+1}{2} \right\rceil \equiv n_1(l) \leq n \leq n_2(l) \equiv \left\lfloor \frac{l}{2} \right\rfloor \tag{3.25}$$

Hence

$$\sum_{l=-\infty}^{\infty} c_{j,l}\phi_{j,l}(x) = \sum_{l=-\infty}^{\infty} \sum_{n=n_1(l)}^{n_2(l)} \left[c_{j-1,n}a_{l-2n} + d_{j-1,n}b_{l-2n}\right] \phi_{j,l}(x)$$

and equating coefficients we obtain the reconstruction formula

$$c_{j,l} = \sum_{n=n_1(l)}^{n_2(l)} c_{j-1,n}a_{l-2n} + d_{j-1,n}b_{l-2n} \tag{3.26}$$

We will call this the **inverse partial wavelet transform** (IPWT). Consequently, the **inverse fast wavelet transform** (IFWT) is obtained by repeated application of (3.26) for $j = J_0+1, J_0+2, \ldots, J$.

3.3.1 Periodic FWT

If the underlying function f is periodic we also have periodicity in the scaling function and wavelet coefficients; $c_{j,l} = c_{j,l+2^j p}$ by (2.60) and $d_{j,l} = d_{j,l+2^j p}$ by (2.59). Hence, it is enough to consider 2^j coefficients of either type at level j. The periodic PWT is thus

$$c_{j-1,l} = \sum_{k=0}^{D-1} a_k c_{j,\langle 2l+k\rangle_{2^j}} \tag{3.27}$$

$$d_{j-1,l} = \sum_{k=0}^{D-1} b_k c_{j,\langle 2l+k\rangle_{2^j}} \tag{3.28}$$

where

$$l = 0, 1, \ldots, 2^{j-1}-1$$

Hence the periodic FWT is obtained by repeated application of (3.27) and (3.28) for $j = J, J-1, \ldots, J_0+1$.

If we let $J_0 = 0$ then (3.27) and (3.28) can be applied until all coefficients in the (canonical) expansion (2.54) have been computed. There is then only one scaling function coefficient left, namely $c_{0,0}$ (the “DC” term), the rest will be wavelet coefficients. Figure 3.3 shows an example of the periodic FWT.

$c_{4,0}$	$\rightarrow$	$c_{3,0}$	$\rightarrow$	$c_{2,0}$	$\rightarrow$	$c_{1,0}$	$\rightarrow$	$c_{0,0}$
$c_{4,1}$		$c_{3,1}$		$c_{2,1}$		$c_{1,1}$		$d_{0,1}$
$c_{4,2}$		$c_{3,2}$		$c_{2,2}$		$d_{1,0}$		$d_{1,0}$
$c_{4,3}$		$c_{3,3}$		$c_{2,3}$		$d_{1,1}$		$d_{1,1}$
$c_{4,4}$		$c_{3,4}$		$d_{2,0}$		$d_{2,0}$		$d_{2,0}$
$c_{4,5}$		$c_{3,5}$		$d_{2,1}$		$d_{2,1}$		$d_{2,1}$
$c_{4,6}$		$c_{3,6}$		$d_{2,2}$		$d_{2,2}$		$d_{2,2}$
$c_{4,7}$		$c_{3,7}$		$d_{2,3}$		$d_{2,3}$		$d_{2,3}$
$c_{4,8}$		$d_{3,0}$		$d_{3,0}$		$d_{3,0}$		$d_{3,0}$
$c_{4,9}$		$d_{3,1}$		$d_{3,1}$		$d_{3,1}$		$d_{3,1}$
$c_{4,10}$		$d_{3,2}$		$d_{3,2}$		$d_{3,2}$		$d_{3,2}$
$c_{4,11}$		$d_{3,3}$		$d_{3,3}$		$d_{3,3}$		$d_{3,3}$
$c_{4,12}$		$d_{3,4}$		$d_{3,4}$		$d_{3,4}$		$d_{3,4}$
$c_{4,13}$		$d_{3,5}$		$d_{3,5}$		$d_{3,5}$		$d_{3,5}$
$c_{4,14}$		$d_{3,6}$		$d_{3,6}$		$d_{3,6}$		$d_{3,6}$
$c_{4,15}$		$d_{3,7}$		$d_{3,7}$		$d_{3,7}$		$d_{3,7}$

Figure 3.3: The periodic FWT. The shaded elements $d_{j,l}$ obtained at each level remain the same at subsequent levels. Here the depth is taken to be as large as possible, i.e. $\lambda = J = 4$.

The reconstruction algorithm (IPWT) for the periodic problem is similar to the non-periodic version:

$$c_{j,l} = \sum_{n=n_1(l)}^{n_2(l)} c_{j-1,\langle n\rangle_{2^{j-1}}} a_{l-2n} + d_{j-1,\langle n\rangle_{2^{j-1}}} b_{l-2n} \tag{3.29}$$

with $n_1(l)$ and $n_2(l)$ defined as in (3.25) and

$$l = 0, 1, \ldots, 2^j - 1$$

Hence the periodic IFWT is obtained by applying (3.29) for $j = J_0 + 1, J_0 + 2, \ldots, J$. We restrict ourselves to algorithms for periodic problems throughout this report. Hence we will consider only the *periodic* cases of FWT, PWT, IFWT, and IPWT.

3.3.2 Matrix representation of the FWT

Let $\boldsymbol{c}_j = [c_{j,0}, c_{j,1}, \ldots, c_{j,2^j-1}]^T$ and similarly, $\boldsymbol{d}_j = [d_{j,0}, d_{j,1}, \ldots, d_{j,2^j-1}]^T$. Since (3.27) and (3.28) are linear mappings from $\mathbf{R}^{2^j}$ onto $\mathbf{R}^{2^{j-1}}$ the PWT can be represented as

$$\begin{aligned} \boldsymbol{c}_{j-1} &= \boldsymbol{A}_j \boldsymbol{c}_j \\ \boldsymbol{d}_{j-1} &= \boldsymbol{B}_j \boldsymbol{c}_j \end{aligned}$$

where $\boldsymbol{A}_j$ and $\boldsymbol{B}_j$ are 2^{j-1} by 2^j matrices containing the filter coefficients. For $D = 6$ and $j = 4$ we have

$$\boldsymbol{A}_4 = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & & & & & & & & & & \\ & & a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & & & & & & & & \\ & & & & a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & & & & & & \\ & & & & & & a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & & & & \\ & & & & & & & & a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & & \\ & & & & & & & & & & a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ a_4 & a_5 & & & & & & & & & & & a_0 & a_1 & a_2 & a_3 \\ a_2 & a_3 & a_4 & a_5 & & & & & & & & & & & a_0 & a_1 \end{pmatrix}$$

and

$$\boldsymbol{B}_4 = \begin{pmatrix} b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & & & & & & & & & & \\ & & b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & & & & & & & & \\ & & & & b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & & & & & & \\ & & & & & & b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & & & & \\ & & & & & & & & b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & & \\ & & & & & & & & & & b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \\ b_4 & b_5 & & & & & & & & & & & b_0 & b_1 & b_2 & b_3 \\ b_2 & b_3 & b_4 & b_5 & & & & & & & & & & & b_0 & b_1 \end{pmatrix}$$

These are shift-circulant matrices (see Definition 8.1). Moreover, the rows are orthonormal by equations (2.22) and (2.23): $\sum_k a_k a_{k+2n} = \delta_{0,n}$ and $\sum_k b_k b_{k+2n} = \delta_{0,n}$. $\boldsymbol{B}_j$ is similar, but with b_k's in place of a_k's; hence the rows of $\boldsymbol{B}_j$ are also orthonormal. Moreover, the inner product of any row in $\boldsymbol{A}_j$ with any row in $\boldsymbol{B}_j$ is zero by Theorem 2.1, i.e. $\boldsymbol{A}_j \boldsymbol{B}_j^T = 0$.

We now combine $\boldsymbol{A}_j$ and $\boldsymbol{B}_j$ to obtain a $2^j \times 2^j$ matrix

$$\boldsymbol{Q}_j = \begin{bmatrix} \boldsymbol{A}_j \\ \boldsymbol{B}_j \end{bmatrix}$$

which represents the combined mapping

$$\begin{bmatrix} \boldsymbol{c}_{j-1} \\ \boldsymbol{d}_{j-1} \end{bmatrix} = \boldsymbol{Q}_j \boldsymbol{c}_j \tag{3.30}$$

Equation (3.30) is the matrix representation of a PWT step as defined by (3.27) and (3.28). The matrix $\boldsymbol{Q}_j$ is orthogonal since

$$\boldsymbol{Q}_j \boldsymbol{Q}_j^T = \begin{bmatrix} \boldsymbol{A}_j \\ \boldsymbol{B}_j \end{bmatrix} [\boldsymbol{A}_j^T \boldsymbol{B}_j^T] = \begin{bmatrix} \boldsymbol{A}_j \boldsymbol{A}_j^T & \boldsymbol{A}_j \boldsymbol{B}_j^T \\ \boldsymbol{B}_j \boldsymbol{A}_j^T & \boldsymbol{B}_j \boldsymbol{B}_j^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_j & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_j \end{bmatrix}$$

Hence the mapping (3.30) is inverted by

$$\boldsymbol{c}_j = \boldsymbol{Q}_j^T \begin{bmatrix} \boldsymbol{c}_{j-1} \\ \boldsymbol{d}_{j-1} \end{bmatrix} \tag{3.31}$$

Equation (3.31) is the matrix representation of an IPWT step as defined in (3.29).

The FWT from level J to level $J_0 = J - \lambda$ can now be expressed as the matrix-vector product

$$\boxed{\boldsymbol{d} = \boldsymbol{W}^\lambda \boldsymbol{c}} \tag{3.32}$$

where $\boldsymbol{c} = \boldsymbol{c}_J$,

$$\boldsymbol{d} = \begin{pmatrix} \boldsymbol{c}_{J_0} \\ \boldsymbol{d}_{J_0} \\ \boldsymbol{d}_{J_0+1} \\ \vdots \\ \boldsymbol{d}_{J-1} \end{pmatrix}$$

and

$$\boldsymbol{W}^\lambda = \hat{\boldsymbol{Q}}_{J_0} \hat{\boldsymbol{Q}}_{J_0+1} \cdots \hat{\boldsymbol{Q}}_{J-1} \boldsymbol{Q}_J.$$

The matrix $\hat{\boldsymbol{Q}}_j$, of order 2^J, is given by

$$\hat{\boldsymbol{Q}}_j = \left[\begin{array}{c|c} \boldsymbol{Q}_j & \\ \hline & \boldsymbol{I}_{k(j,J)} \end{array}\right]$$

where $\boldsymbol{I}_{k(j,J)}$ denotes the $k \times k$ identity matrix with $k = k(j,J) = 2^J - 2^j$. It follows that $\boldsymbol{W}^\lambda$ is orthogonal[2] and hence the IFWT is given by

$$\boldsymbol{c} = \left(\boldsymbol{W}^\lambda\right)^T \boldsymbol{d}$$

3.3.3 A more general FWT

Suppose that elements to be transformed may contain other prime factors than 2, i.e. $N = K2^J$, with $J, K \in \mathbf{Z}$ and $\langle K \rangle_2 = 1$. In this case, the FWT can still be defined by mappings of the form (3.27) and (3.28) with a maximal depth $\lambda = J$, i.e. $J_0 = 0$. It will be convenient to use a slightly different notation for the algorithms in Chapters 5, 6, and 8. Hence, let $S_i = N/2^i$ and

$$c_k^i \equiv c_{J-i,k}$$
$$d_k^i \equiv d_{J-i,k}$$

for $i = 0, 1, \ldots, \lambda - 1$ and $k = 0, 1, \ldots, S_i - 1$. We can now define the FWT as follows:

Definition 3.1 *Fast wavelet transform (FWT)*
Let $J, K \in \mathbf{N}$ with $\langle K \rangle_2 = 1$ and $N = K2^J$. Let $\mathbf{c}^0 = \{c_k^0\}_{k=0}^{N-1}$ be given. The FWT is then computed by the recurrence formulas

$$\begin{aligned} c_n^{i+1} &= \sum_{l=0}^{D-1} a_l c^i_{\langle l+2n \rangle_{S_i}} \\ d_n^{i+1} &= \sum_{l=0}^{D-1} b_l c^i_{\langle l+2n \rangle_{S_i}} \end{aligned} \tag{3.33}$$

where $i = 0, 1, \ldots, \lambda - 1$, $S_i = N/2^i$, and $n = 0, 1, \ldots, S_{i+1} - 1$.

[2]The orthonormality of $\boldsymbol{W}^\lambda$ is one motivation for the choice of dilation equations mentioned in Remark 2.4.

The relation between the scaling function coefficients and the underlying function is unimportant for the FWT algorithm per se: In numerical applications we often need to apply the FWT to vectors of function values or even arbitrary vectors. Hence let $\boldsymbol{x} \in \mathbf{R}^N$. Then the expression

$$\boxed{\breve{\boldsymbol{x}} = \boldsymbol{W}^{\lambda}\boldsymbol{x}} \tag{3.34}$$

is defined by putting $\boldsymbol{c}^0 = \boldsymbol{x}$, performing λ steps of of (3.33), and letting

$$\breve{\boldsymbol{x}} = \left\{ \{c_k^{\lambda}\}_{k=0}^{S_\lambda - 1}, \quad \{\{d_k^i\}_{k=0}^{S_i - 1}\}_{i=\lambda,\lambda-1,\dots,1} \right\}$$

If λ is omitted in (3.34) it will assume its maximal value J. Note that $\boldsymbol{W}^{\lambda} = \boldsymbol{I}$ for $\lambda = 0$. The IFWT is written similarly as

$$\boxed{\boldsymbol{x} = (\boldsymbol{W}^{\lambda})^T \breve{\boldsymbol{x}}} \tag{3.35}$$

corresponding to the recurrence formula

$$c_l^i = \sum_{n=n_1(l)}^{n_2(l)} c_{\langle n \rangle_{S_{i+1}}}^{i+1} a_{l-2n} + d_{\langle n \rangle_{S_{i+1}}}^{i+1} b_{l-2n}$$

where $i = \lambda - 1, \lambda - 2, \dots, 1, 0$, $S_i = N/2^i$, $l = 0, 1, \dots, S_i - 1$, and $n_1(l)$ and $n_2(l)$ are defined as in (3.25).

The Matlab function `fwt`($\boldsymbol{x}$, D, λ) computes (3.34) and `ifwt`($\breve{\boldsymbol{x}}$, D, λ) computes (3.35).

3.3.4 Complexity of the FWT algorithm

One step of (3.33), the PWT, involves DS_i additions and DS_i multiplications. The number of floating point operations is therefore

$$F_{\text{PWT}}(S_i) = 2DS_i \tag{3.36}$$

Let N be the total number of elements to be transformed and assume that the FWT is carried out to depth λ. The FWT consists of λ applications of the PWT to successively shorter vectors so the total work is

$$
\begin{aligned}
F_{\text{FWT}}(N) &= \sum_{i=0}^{\lambda-1} F_{\text{PWT}}\left(\frac{N}{2^i}\right) \\
&= \sum_{i=0}^{\lambda-1} 2D\frac{N}{2^i} \\
&= 2DN\frac{1-\frac{1}{2^\lambda}}{1-\frac{1}{2}} \\
&= 4DN\left(1-\frac{1}{2^\lambda}\right) \;<\; 4DN
\end{aligned} \tag{3.37}
$$

D is normally constant throughout wavelet analysis, so the complexity is $\mathcal{O}(N)$. The IFWT has the same complexity as the FWT. For comparison we mention that the complexity of the fast Fourier transform (FFT) is $\mathcal{O}(N \log_2 N)$.

3.3.5 2D FWT

Using the definition of $\boldsymbol{W}^\lambda$ from the previous section, we define the 2D FWT as

$$\breve{\boldsymbol{X}} = \boldsymbol{W}^{\lambda_M} \boldsymbol{X} (\boldsymbol{W}^{\lambda_N})^T \tag{3.38}$$

where $\boldsymbol{X}, \breve{\boldsymbol{X}} \in \mathbf{R}^{M,N}$. The parameters λ_M and λ_N are the transform depths in the first and second dimensions, respectively. Equation (3.38) can be expressed as M 1D wavelet transforms of the rows of $\boldsymbol{X}$ followed by N 1D wavelet transforms of the columns of $\boldsymbol{X}(\boldsymbol{W}^{\lambda_N})^T$, i.e.

$$\breve{\boldsymbol{X}} = \boldsymbol{W}^{\lambda_M} \left(\boldsymbol{W}^{\lambda_N} \boldsymbol{X}^T\right)^T$$

Therefore it is straightforward to compute the 2D FWT given the 1D FWT from Definition 3.1. It follows that the inverse 2D FWT is defined as

$$\boldsymbol{X} = \left(\boldsymbol{W}^{\lambda_M}\right)^T \breve{\boldsymbol{X}} \boldsymbol{W}^{\lambda_N} \tag{3.39}$$

The Matlab function `fwt2`($\boldsymbol{X}$, D, λ_M, λ_N) computes (3.38) and `ifwt2`($\breve{\boldsymbol{X}}$, D, λ_M, λ_N) computes (3.39).

Using (3.37) we find the computational work of the 2D FWT as follows

$$
\begin{aligned}
F_{\text{FWT2}}(M,N) &= M\; F_{\text{FWT}}(N) + N\; F_{\text{FWT}}(M) \\
&= M4DN\left(1-\frac{1}{2^{\lambda_N}}\right) + N4DM\left(1-\frac{1}{2^{\lambda_M}}\right) \\
&= 4DMN\left(2-\frac{1}{2^{\lambda_M}}-\frac{1}{2^{\lambda_N}}\right) \;<\; 8DMN
\end{aligned} \tag{3.40}
$$

The computational work of the inverse 2D FWT is the same.

Chapter 4

Approximation properties

4.1 Accuracy of the multiresolution spaces

4.1.1 Approximation properties of V_J

We will now discuss the pointwise approximation error introduced when f is approximated by an expansion in scaling functions at level J. Let $J \in \mathbf{Z}$, $f \in L^2(\mathbf{R})$ and assume that f is P times differentiable everywhere.

For an arbitrary, but fixed x, we define the pointwise error as

$$e_J(x) = f(x) - (P_{V_J} f)(x), \quad x \in \mathbf{R}$$

where $(P_{V_J} f)(x)$ is the orthogonal projection of f onto the approximation space V_J as defined in Section 2.1.

Recall that $P_{V_J} f$ has expansions in terms of scaling functions as well as in terms of wavelets. The wavelet expansion for $P_{V_J} f$ is

$$\left(P_{V_J} f\right)(x) = \sum_{k=-\infty}^{\infty} c_{J_0,k}\phi_{J_0,k}(x) + \sum_{j=J_0}^{J-1} \sum_{k=-\infty}^{\infty} d_{j,k}\psi_{j,k}(x) \tag{4.1}$$

and by letting $J \to \infty$ temporarily we get a wavelet expansion for f itself:

$$f(x) = \sum_{k=-\infty}^{\infty} c_{J_0,k}\phi_{J_0,k}(x) + \sum_{j=J_0}^{\infty} \sum_{k=-\infty}^{\infty} d_{j,k}\psi_{j,k}(x) \tag{4.2}$$

Then, subtracting (4.2) from (4.1) we obtain an expression for the error e_J in terms of the wavelets at scales $j \geq J$:

$$e_J(x) = \sum_{j=J}^{\infty} \sum_{k=-\infty}^{\infty} d_{j,k}\psi_{j,k}(x) \tag{4.3}$$

Define

$$C_\psi = \max_{x \in I_{j,k}} \left| \psi(2^j x - k) \right| = \max_{y \in [0, D-1]} |\psi(y)|$$

Hence $\max_{x \in I_{j,k}} |\psi_{j,k}(x)| = 2^{j/2} C_\psi$ and using Theorem 2.5, we find that

$$|d_{j,k}\psi_{j,k}(x)| \leq C_P 2^{-jP} \max_{\xi \in I_{j,k}} \left| f^{(P)}(\xi) \right| C_\psi$$

Recall that

$$\text{supp}(\psi_{j,k}) = I_{j,k} = \left[\frac{k}{2^j}, \frac{k + D - 1}{2^j} \right]$$

Hence, there are at most $D - 1$ intervals $I_{j,k}$ containing a given value of x. Thus for any x only $D - 1$ terms in the inner summation in (4.3) are nonzero. Let I_j be the union of all these intervals, i.e.

$$I_j(x) = \bigcup_{\{l : x \in I_{j,l}\}} I_{j,l}$$

and let

$$\mu_j^P(x) = \max_{\xi \in I_j(x)} \left| f^{(P)}(\xi) \right|$$

Then one finds a common bound for all terms in the inner sum:

$$\sum_{k=-\infty}^{\infty} |d_{j,k}\psi_{j,k}(x)| \leq C_\psi C_P 2^{-jP}(D-1)\mu_j^P(x)$$

The outer sum can now be evaluated using the fact that

$$\mu_J^P(x) \geq \mu_{J+1}^P(x) \geq \mu_{J+2}^P(x) \geq \cdots$$

and we establish the bound

$$\begin{aligned} |e_J(x)| &\leq C_\psi C_P (D-1) \mu_J^P(x) \sum_{j=J}^{\infty} 2^{-jP} \\ &= C_\psi C_P (D-1) \mu_J^P(x) \frac{2^{-JP}}{1 - 2^{-P}} \end{aligned}$$

Thus we see that that for an arbitrary but fixed x the approximation error will be bounded as

$$\boxed{|e_J(x)| = \mathcal{O}(2^{-JP})}$$

This is exponential decay with respect to the resolution J. Furthermore, the greater the number of vanishing moments P, the faster the decay.

Finally, note that each error term $d_{j,k}\psi_{j,k}(x)$ is zero for $x \notin I_{j,k}$. This means that $e_J(x)$ depends only on $f(y), y \in [x, x + (D-1)/2^J)]$. This is what was also observed in Chapter 1.

4.1.2 Approximation properties of $\tilde{V}_J$

We now consider the approximation error in the periodic case. Let $f \in L^2([0,1])$ and assume that its periodic extension (2.55) is P times differentiable everywhere. Furthermore, let $J \geq J_0$ where J_0 is given as in (2.47) and define the approximation error as

$$\tilde{e}_J(x) = f(x) - (P_{\tilde{V}_J} f)(x), \quad x \in [0,1]$$

where $(P_{\tilde{V}_J} f)(x)$ is the orthogonal projection of f onto the approximation space $\tilde{V}_J$ as defined in Definition 2.4.

Using the periodic wavelet expansion (2.53) and proceeding as in the non-periodic case we find that

$$\tilde{e}_J(x) = \sum_{j=J}^{\infty} \sum_{k=0}^{2^j-1} d_{j,k} \tilde{\psi}_{j,k}(x) \tag{4.4}$$

Since the coefficients $d_{j,k}$ are the same as in the non-periodic case by (2.58), Theorem (2.5) applies and we can repeat the analysis from Section 4.1.1 and, indeed, we obtain that

$$\boxed{|\tilde{e}_J(x)| = \mathcal{O}(2^{-JP}), \quad x \in [0,1]}$$

We will now consider the infinity norm of $\tilde{e}_J$ defined by

$$\|\tilde{e}_J\|_\infty = \max_{x \in [0,1]} |\tilde{e}_J(x)|$$

A similar analysis as before yields

$$\begin{aligned}
\|\tilde{e}_J\|_\infty &\leq \sum_{j=J}^{\infty} \sum_{k=0}^{2^j-1} |d_{j,k}| \max_{x \in I_{j,k}} \left|\tilde{\psi}_{j,k}(x)\right| \\
&\leq C_\psi C_P \sum_{j=J}^{\infty} \sum_{k=0}^{2^j-1} 2^{j/2} 2^{-j(P+\frac{1}{2})} \max_{\xi \in I_{j,k}} \left|f^{(P)}(\xi)\right| \\
&= C_\psi C_P \max_{\xi \in [0,1]} \left|f^{(P)}(\xi)\right| \sum_{j=J}^{\infty} 2^{-jP} \\
&= C_\psi C_P \max_{\xi \in [0,1]} \left|f^{(P)}(\xi)\right| \frac{2^{-JP}}{1-2^{-P}}
\end{aligned}$$

Hence

$$\boxed{\|\tilde{e}_J\|_\infty = \mathcal{O}(2^{-JP})}$$

Finally, consider the 2-norm of $\tilde{e}_J$:

$$\|\tilde{e}_J\|_2^2 = \int_0^1 \tilde{e}_J^2(x)\,dx \leq \int_0^1 \|\tilde{e}_J\|_\infty^2 \;dx = \|\tilde{e}_J\|_\infty^2 \int_0^1 \;dx = \|\tilde{e}_J\|_\infty^2$$

hence we have also in this case

$$\boxed{\|\tilde{e}_J\|_2 = \mathcal{O}(2^{-JP})} \tag{4.5}$$

4.2 Wavelet compression errors

In this Section we will see how well a function $f \in \tilde{V}_J$ is approximated by a wavelet expansion where small wavelet coefficients have been discarded. We will restrict attention to the case of the interval $[0, 1]$.

It was shown in Section 2.1.6 that the wavelet coefficients corresponding to a region where f is smooth decay rapidly and are not affected by regions where f is not so smooth. We now investigate the error committed when small wavelet coefficients are discarded. We will refer to those as the *insignificant* wavelet coefficients while those remaining will be referred to as the *significant* wavelet coefficients. These definitions are rendered quantitative in the following.

Let ε be a given threshold for separating the insignificant wavelet coefficients from the significant ones. We define an index set identifying the indices of the significant wavelet coefficients at level j:

$$T_j^\varepsilon = \left\{k : 0 \leq k \leq 2^j - 1 \quad \wedge \quad |d_{j,k}| > \varepsilon\right\}$$

Hence the indices of the insignificant coefficients are given by

$$R_j^\varepsilon = T_j^0 \setminus T_j^\varepsilon$$

With this notation we can write an ε-truncated wavelet expansion for f as follows

$$\left(P_{\tilde{V}_J} f\right)^\varepsilon (x) = \sum_{k=0}^{2^{J_0}-1} c_{J_0,k}\tilde{\phi}_{J_0,k}(x) + \sum_{j=J_0}^{J-1} \sum_{k \in T_j^\varepsilon} d_{j,k}\tilde{\psi}_{j,k}(x) \tag{4.6}$$

Let $N_s(\varepsilon)$ be the number of all significant wavelet coefficients, i.e.

$$N_s(\varepsilon) = \sum_{j=J_0}^{J-1} \# T_j^\varepsilon + 2^{J_0}$$

where $\#$ denotes the cardinality of T_j^ε. The last term corresponds to the scaling function coefficients that must be present in the expansion because they provide the coarse approximation on which the wavelets build the fine structures. If we let $N = 2^J$ be the dimension of the finest space $\tilde{V}_J$ then we can define the symbol N_r to be the number of insignificant coefficients in the wavelet expansion, i.e.

$$N_r(\varepsilon) = N - N_s(\varepsilon)$$

We then define the error introduced by this truncation as

$$\begin{aligned} \tilde{e}_J^\varepsilon(x) &= P_{\tilde{V}_J} f - \left(P_{\tilde{V}_J} f\right)^\varepsilon \\ &= \sum_{j=J_0}^{J-1} \sum_{k \in R_j^\varepsilon} d_{j,k} \psi_{j,k}(x) \end{aligned}$$

with the number of terms being equal to $N_r(\varepsilon)$. The 2-norm of $\tilde{e}_J^\varepsilon(x)$ is then found as

$$\begin{aligned} \|\tilde{e}_J^\varepsilon(x)\|_2^2 &= \left\| \sum_{j=J_0}^{J-1} \sum_{k \in R_j^\varepsilon} d_{j,k} \psi_{j,k}(x) \right\|_2^2 \\ &= \sum_{j=J_0}^{J-1} \sum_{k \in R_j^\varepsilon} |d_{j,k}|^2 \\ &\leq \varepsilon^2 N_r(\varepsilon) \end{aligned}$$

so whenever the threshold is ε then

$$\boxed{\|\tilde{e}_J^\varepsilon(x)\|_2 \leq \varepsilon \sqrt{N_r(\varepsilon)}} \tag{4.7}$$

For the infinity norm the situation is slightly different. For reasons that will soon become clear we now redefine the index set as

$$T_j^\varepsilon = \left\{ k : 0 \leq k \leq 2^j - 1 \;\; \wedge \;\; |d_{j,k}| > \varepsilon / 2^{j/2} \right\}$$

and $R_j^\varepsilon = T_j^0 \setminus T_j^\varepsilon$. Note that we now modify the threshold according to the scale at which the wavelet coefficient belongs. Applying the infinity norm to $\tilde{e}_J^\varepsilon$ with the new definition of T_j yields

$$\begin{aligned}
\|\tilde{e}^{\varepsilon}_J(x)\|_\infty &= \sum_{j=J_0}^{J-1} \sum_{k\in R^{\varepsilon}_j} \max_x(|d_{j,k}\psi_{j,k}(x)|) \\
&= C_\psi \sum_{j=J_0}^{J-1} \sum_{k\in R^{\varepsilon}_j} |d_{j,k}|\, 2^{j/2} \\
&\leq C_\psi \sum_{j=J_0}^{J-1} \sum_{k\in R^{\varepsilon}_j} \varepsilon \\
&= C_\psi \varepsilon N_r(\varepsilon)
\end{aligned}$$

so whenever the threshold is ε then

$$\boxed{\|\tilde{e}^{\varepsilon}_J(x)\|_\infty \leq C_\psi \varepsilon N_r(\varepsilon)} \tag{4.8}$$

The difference between (4.7) and (4.8) lies first and foremost in the fact that the threshold is scaled in the latter case. This means that the threshold essentially decreases with increasing scale such that more wavelet coefficients will be included at the finer scales. Hence $N_r(\varepsilon)$ will tend to be smaller for the ∞-norm than for the 2-norm.

Remark 4.1 *A heuristic way of estimating N_s is given as follows: Let $f(x)$ be a smooth function with one singularity located at $x = x_s$. Then $d_{j,k} > \varepsilon$ for those j and k where $x_s \in I_{j,k}$. Hence we have $D - 1$ significant coefficients at each level yielding a total of $\lambda(D-1)$. Hence if f has q singularities we can estimate N_s to be roughly proportional to*

$$q\lambda(D-1)$$

4.3 Scaling function coefficients or function values ?

It was mentioned in Section 3.3.3 that the fast wavelet transform can be applied directly to sequences of function values instead of scaling function coefficients. Since this is often done in practice (see, for example, [SN96, p. 232]), yet rarely justified, we consider it here.

Suppose that $f \in L^2(\mathbf{R})$ is approximately constant on the interval $I_{J,l}$ given by (2.21), a condition that is satisfied if J is sufficiently large since the length of

$I_{J,l}$ is $(D-1)/2^J$. Then $f(x) \approx f(l/2^J)$ for $x \in I_{J,l}$ and we find that

$$\begin{aligned} c_{J,l} &= 2^{J/2} \int_{I_{J,l}} f(x)\phi(2^J x - l)\, dx \\ &\approx 2^{J/2} f(l/2^J) \int_{I_{J,l}} \phi(2^J x - l)\, dx \\ &= 2^{-J/2} f(l/2^J) \int_0^{D-1} \phi(y)\, dy \end{aligned}$$

or

$$c_{J,l} \approx 2^{-J/2} f(l/2^J)$$

In vector notation this becomes

$$\boldsymbol{c} \approx 2^{-J/2} \boldsymbol{f}$$

and from (3.32) follows

$$\boldsymbol{d} \approx 2^{-J/2} \boldsymbol{W} \boldsymbol{f} \tag{4.9}$$

Hence, the elements in $\breve{\boldsymbol{f}} = \boldsymbol{W}\boldsymbol{f}$ behave approximately like those in $\boldsymbol{d}$ (except for the constant factor) when J is sufficiently large.

4.4 A compression example

One of the most successful applications of the wavelet transform is image compression. Much can be said about this important field, but an exhaustive account would be far beyond the scope of this thesis. However, with the wavelet theory developed thus far we can give a simple but instructive example of the principles behind wavelet image compression. For details, we refer to [Chu97, SN96, H$^+$94, VK95] among others.

Let $\boldsymbol{X}$ be an $M \times N$ matrix representation of a digital image. Each element of $\boldsymbol{X}$ corresponds to one pixel with its value corresponding to the light intensity. Figure 4.1 shows an example of a digital image encoded in $\boldsymbol{X}$.

Applying the 2D FWT from (3.38) to $\boldsymbol{X}$ yields

$$\breve{\boldsymbol{X}} = \boldsymbol{W}^{\lambda_M} \boldsymbol{X} \left(\boldsymbol{W}^{\lambda_N} \right)^T$$

which is shown in Figure 4.2 with $D = 6$. The upper left block is a smoothed approximation of the original image. The other blocks represent details at various

Figure 4.1: A digital image $\boldsymbol{X}$. Here $M = 256, N = 512$.

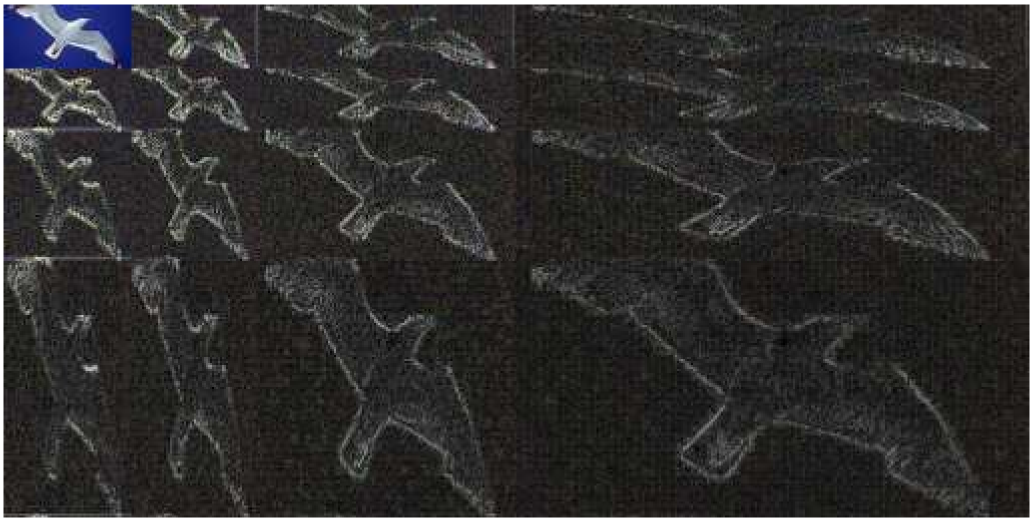


Figure 4.2: The wavelet spectrum $\check{\boldsymbol{X}}$. Here $\lambda_M = \lambda_N = 3$ and $D = 6$.

Figure 4.3: Reconstructed image $\boldsymbol{Y}$. Compression ratio $1 : 100$.

scales and it can be seen that the significant elements are located where the original image has edges. The large smooth areas are well represented by the coarse approximation so no further details are needed there.

Assuming that the image is obtained as function values of some underlying function which has some smooth regions we expect many of the elements in $\breve{\boldsymbol{X}}$ to be small. The simplest compression strategy is to discard small elements in $\breve{\boldsymbol{X}}$, and for this purpose we define

$$\breve{\boldsymbol{X}}^{\varepsilon} \leftarrow \text{trunc}(\breve{\boldsymbol{X}}, \varepsilon) = \{[\breve{\boldsymbol{X}}]_{m,n}, \left|[\breve{\boldsymbol{X}}]_{m,n}\right| > \varepsilon\}$$

By inverse FWT one obtains the approximation

$$\boldsymbol{Y} = \left(\boldsymbol{W}^{\lambda_M}\right)^T \breve{\boldsymbol{X}}^{\varepsilon} \boldsymbol{W}^{\lambda_N}$$

which is shown in Figure 4.3. The threshold ε has been chosen such that only 1% of the wavelet coefficients are retained in $\breve{\boldsymbol{X}}^{\varepsilon}$. This gives the error

$$\boxed{E_{\text{FWT}} = \frac{\|\boldsymbol{Y} - \boldsymbol{X}\|_2}{\|\boldsymbol{X}\|_2} = 6.15 \times 10^{-2}}$$

We point out that there is no norm that accurately reflects the way people perceive the error introduced when compressing an image. The only reliable measure is the subjective impression of the image quality [Str92, p. 364]. Attempts at "objective"

Figure 4.4: Reconstructed image $\boldsymbol{F}^T \hat{\boldsymbol{X}}^{\varepsilon} \boldsymbol{F}$.

error measurement customarily make use of the peak signal-to-noise ratio (PSNR) which is essentially based on the 2-norm [Chu97, p. 180], [JS94, p. 404].

In order to assess the merits of the procedure described above, we now repeat it with the discrete Fourier transform (DFT): Using the Fourier matrix $\boldsymbol{F}_N$ defined in (C.3) in place of $\boldsymbol{W}$ we get

$$\hat{\boldsymbol{X}} = \boldsymbol{F}_M \boldsymbol{X} \boldsymbol{F}_N^T$$

Retaining 1% of the elements in $\hat{\boldsymbol{X}}$ as before and transforming back yields the image shown in Figure 4.4. The discarded elements in this case correspond to high frequencies so the compressed image becomes blurred. The corresponding relative error is now

$$\boxed{E_{\text{DFT}} = 1.03 \times 10^{-1}}$$

In practical applications one does not use the DFT on the entire image. Rather, the DFT is applied to smaller *blocks* which are then compressed separately. However, this approach can lead to artefacts at the block boundaries which degrade image quality. In any case, this example serves to illustrate why the wavelet transform is an attractive alternative to the Fourier transform; edges and other localized small scale features are added where needed, and omitted where they are not. This is all a direct consequence of Theorem 2.5.

Finally, we mention that a practical image compression algorithm does much more than merely throw away small coefficients. Typically, more elaborate truncation (quantization) schemes are combined with entropy coding to minimize the amount of data needed to represent the signal [Chu97, Str92, H$^+$94].

Part II

Fast Wavelet Transforms on Supercomputers

Chapter 5

Vectorization of the Fast Wavelet Transform

Problems involving the FWT are typically large and wavelet transforms can be time-consuming even though the algorithmic complexity is proportional to the problem size. The use of high performance computers is one way of speeding up the FWT.

In this chapter and the next we will describe our efforts to implement the 1D and the 2D FWT on a selection of high-performance computers, especially the Fujitsu VPP300. For simplicity we let $K = 1$ (see Definition 3.1) throughout these chapters.

The Fujitsu VPP300 is a vector-parallel computer. This means that it consists of a number of vector processors connected in an efficient network. Good vector performance on the individual processors is therefore crucial to good parallel performance. In this chapter we discuss the implementation and performance of the FWT on *one* node of the VPP300. In Chapter 6 we discuss parallelization of the FWT and report results of the parallel implementation on *several* nodes on the VPP300. Some of the material presented here has also appeared in [NH95].

5.1 The Fujitsu VPP300

The vector units on the VPP300 run with a clock frequency of 142 MHz. They can execute 8 multiplications and 8 additions per clock cycle, leading to a peak performance of 2.272 Gflop/s per processor. The actual performance, however, depends on many factors such as

1. vector length

2. memory stride

3. arithmetic density

4. ratio of arithmetic operations to load/store operations

5. type of operations

A vector processor is designed to perform arithmetic operations on *vectors* of numbers through hardware pipelining. There is an overhead involved with each vector instruction, so good performance requires long vector lengths.

As with all modern computers, the memory speed of the VPP falls short of the processor speed. To overcome this problem, memory is arranged in banks with consecutive elements spread across the banks. Stride-one access then means that the memory banks have time to recover between consecutive memory accesses so that they are always ready to deliver a piece of data at the rate at which it is requested. Furthermore, the VPP300 has a special instruction for this which is faster than any non-uniform memory access. Finally, on computer architectures using cache, stride-one means that all elements in a cache line will be used before it is flushed. In either case, a stride different from one can lead to poor performance because the processor has to wait until the data are ready.

Optimal performance on the VPP300 requires that 8 multiplications and 8 additions occur every clock cycle. Therefore, any loop containing only addition or multiplication can never run faster than half the peak performance. Also, the use of load and store pipes is crucial. The VPP300 has one load and one store pipe, so addition of two vectors, say, can at best run at 1/4 of the peak performance because two loads are needed at each iteration. Finally, the type of arithmetic operations is crucial to the performance. For example, a division takes seven cycles on the VPP. Taking all these issues into account, we see that good vector performance requires operations of the form

for $n = 0 : N - 1$
$\quad [\boldsymbol{y}]_n \leftarrow a * [\boldsymbol{x}]_n + b$

where a and b are scalars and $\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of length N, with N being large. For details see [Uni96, HJ88].

5.2 1D FWT

The basic operation in the 1D FWT can be written in the form

for $n = 0 : S/2 - 1$
$\quad [\boldsymbol{w}]_n \leftarrow [\boldsymbol{w}]_n + a_l * [\boldsymbol{x}]_{\langle l+2n \rangle_S}$

N	F	CPU time (μs)	Mflop/s
1024	81840	676	121
2048	163760	844	194
4096	327600	1118	293
8192	655280	1790	366
16384	1310640	3260	402
32768	2621360	5650	464
65536	5242800	10180	515
131072	10485680	19311	543
262144	20971440	38130	550
524288	41942960	68646	611

Table 5.1: Timings of the FWT. $D = 20$, $N = 2^J$, $J = 10, 11, \ldots, 19$, and $\lambda = J$.

as defined by the recurrence formulas (3.33). The arithmetic density as well as the ratio of arithmetic operations to load/store operations are good. However, memory is accessed with stride-two because of the inherent double shift in the wavelet transform, and indices must be wrapped because of periodicity. Therefore, optimal performance is not expected for the 1D FWT.

Our implementation of the FWT on one node of the VPP300 yields the performance shown in Table 5.1. We make the following observations from Table 5.1: Firstly, the performance is far from optimal even for the largest value of N. Secondly, the performance improves only slowly as N increases. To understand the latter property we conduct a performance analysis of one step (the PWT) of the recurrence formulas (3.33). Since this is a simple operation on one vector, we assume that the computation time in the vector processor follows the model

$$T = t_s + t_v F \tag{5.1}$$

where F is the number of floating point operations, T is the total execution time, t_v is the computation time for one floating point operation in the pipeline, and t_s is the startup time. The performance expressed in floating point operations per second is then

$$R = \frac{F}{T} \tag{5.2}$$

Letting F go to infinity results in the theoretically optimal performance

$$R_\infty \equiv \lim_{F\to\infty} \frac{F}{T} = \lim_{F\to\infty} \frac{F}{t_s + t_v F} = \frac{1}{t_v} \tag{5.3}$$

Let $\alpha \in [0,1]$ be the fraction of R_∞ which is achieved for a given problem size F_α. Then F_α is found from (5.2) with $R = \alpha R_\infty$:

$$\frac{\alpha}{t_v} = \frac{F_\alpha}{t_s + t_v F_\alpha}$$

which has the solution

$$F_\alpha = \frac{\alpha}{1-\alpha}\,\frac{t_s}{t_v}$$

In particular, for $\alpha = 1/2$ we find

$$F_{1/2} = \frac{t_s}{t_v}$$

which is another characteristic performance parameter for the algorithm in question. F_α can now be expressed in terms of $F_{1/2}$ as $F_\alpha = F_{1/2}\,\alpha/(1-\alpha)$. For example, to reach 80 % of the maximum performance, a problem size of $F = 4F_{1/2}$ is required, and $F = 9F_{1/2}$ is needed to reach 90 %. The parameter $F_{1/2}$ can therefore be seen as a measure of how quickly the performance approaches R_∞. A large value of $F_{1/2}$ means that the problem must be very large in order to get good performance. Hence we wish $F_{1/2}$ to be as small as possible.

In order to estimate the characteristic parameters $F_{1/2}$ and R_∞ we use measurements of CPU time in the VPP300. Table 5.2 shows the timings of the sequence of steps needed to compute the full FWT with $\lambda = J = 19$, $N = 2^J$, and $D = 20$ (the last result in Table 5.1), i.e. the PWT applied succesively to vectors of length $S = N, N/2, \ldots, 2$. Using these measurements we estimate the parameters t_s and t_v (in the least squares sense) to be

$$t_s = 76\ \mu\text{s} \quad \text{and} \quad t_v = 0.00157\ \mu\text{s}$$

Consequently

$$\begin{aligned} (R_\infty)_{\text{PWT}} &= 637 \ \text{Mflop/s and} \\ (F_{1/2})_{\text{PWT}} &= 48222 \ \text{Operations} \end{aligned} \qquad (5.4)$$

It can be verified that the values predicted by the linear model correspond well with those observed in Table 5.2. The execution time of the PWT thus follows the model

$$T_{\text{PWT}}(S) = t_s + t_v F_{\text{PWT}}(S)$$

S	F	$T(\mu\text{s})$	R(Mflop/s)
524288	20971520	32974	636
262144	10485760	16539	634
131072	5242880	8283	633
65536	2621440	4188	626
32768	1310720	2142	612
16384	655360	1120	585
8192	327680	598	548
4096	163840	344	476
2048	81920	180	455
1024	40960	130	315
512	20480	97	211
256	10240	85	120
128	5120	80	64
64	2560	50	32
32	1280	38	16
16	640	36	8
8	320	32	4
4	160	28	2
2	80	29	1

Table 5.2: Timings of the PWT. $D = 20$, $N = 2^J$, $J = 19$, and $\lambda = J$.

which can be used to predict the execution time of the FWT to depth λ_N as follows

$$\begin{aligned} T_{\text{FWT}}(N) &= \sum_{i=0}^{\lambda_N - 1} T_{\text{PWT}}\left(\frac{N}{2^i}\right) \\ &= \sum_{i=0}^{\lambda_N - 1} t_s + t_v F_{\text{PWT}}\left(\frac{N}{2^i}\right) \\ &= t_s \lambda_N + t_v \sum_{i=0}^{\lambda_N - 1} F_{\text{PWT}}\left(\frac{N}{2^i}\right) \end{aligned}$$

or, if λ_N assumes its maximal value

$$\boxed{T_{\text{FWT}}(N) = t_s \log_2 N + t_v F_{\text{FWT}}(N)} \tag{5.5}$$

Figure 5.1 shows a plot of (5.5) together with the measurements from Table 5.1.

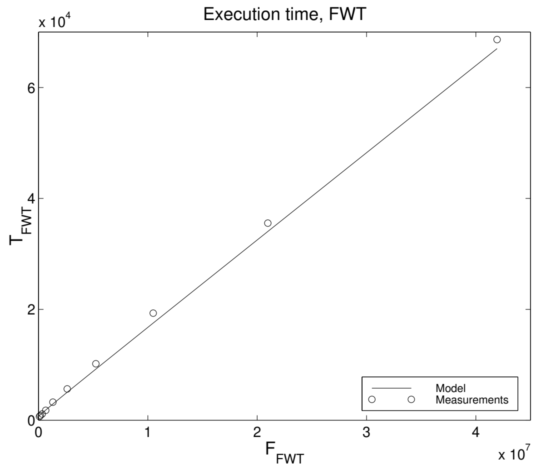


Figure 5.1: The execution time for the FWT plotted as a function of floating point operations. The circles are the measured times while the line is the result of the model.

We know from (3.37) that $F_{\text{FWT}}(N) < 4DN$. Hence $(\log_2 N)/F_{\text{FWT}}(N) \to 0$ for $N \to \infty$, and we see that the asymptotic performance of the FWT is the same as that of the PWT:

$$(R_\infty)_{\text{FWT}} = (R_\infty)_{\text{PWT}} = \frac{1}{t_v}$$

However, the equation to be solved for $(F_{1/2})_{\text{FWT}}$ is not linear. Let $N_{1/2}$ be the vector length corresponding to $(F_{1/2})_{\text{FWT}}$. Then $(F_{1/2})_{\text{FWT}} = F_{\text{FWT}}(N_{1/2}) = 4D(N_{1/2} - 1)$ by (3.37) and putting $R = R_\infty/2$ (corresponding to $\alpha = 0.5$) in (5.2) we get

$$\begin{aligned} \frac{(R_\infty)_{\text{FWT}}}{2} &= \frac{(F_{1/2})_{\text{FWT}}(N_{1/2})}{T_{\text{FWT}}(N_{1/2})} \\ \frac{1}{2t_v} &= \frac{(F_{1/2})_{\text{FWT}}}{t_s \log_2 N_{1/2} + t_v (F_{1/2})_{\text{FWT}}} \\ t_v (F_{1/2})_{\text{FWT}} &= t_s \log_2 N_{1/2} \\ t_v 4D(N_{1/2} - 1) &= t_s \log_2 N_{1/2} \end{aligned}$$

For the values t_s and t_v obtained from timings of the PWT we find the estimate $N_{1/2} = 7794$ which yields

$$(F_{1/2})_{\text{FWT}} = (F_{1/2})_{\text{PWT}} \ \log_2 N_{1/2} = 623420 \ \text{ operations}$$

This agrees with the observed values in Table 5.1. It is seen, both from the observations and the model, that the value of $(F_{1/2})_{\text{FWT}}$ is very high compared to that of the PWT (5.4). The explanation for this is revealed by the analysis just carried out: A new vector operation is started up for each PWT so the startup time t_s counts $\log_2 N$ times regardless of the vector length; *no matter how long the initial vector length N, the PWT will eventually be applied to short vectors*.

Like the stride-two memory access, this property is inherent in the full 1D FWT and we conclude that this computation is not particularly suitable for vectorization. However, if the depth λ_N is taken to be smaller than J then the inefficiency due to short vector lengths becomes less severe.

5.3 Multiple 1D FWT

Consider a matrix $\boldsymbol{X} \in \mathbf{R}^{M,N}$. We assume that $\boldsymbol{X}$ is stored by columns so that consecutive elements in each column are located in consecutive positions in memory. Applying the 1D FWT to every column of $\boldsymbol{X}$ leads to inefficient data access and large $F_{1/2}$ as described in the previous section. By applying the FWT to the *rows* of $\boldsymbol{X}$ instead, one can vectorize over the columns such that all elements will be accessed with stride-one in vectors of length M. We will refer to this procedure as the **multiple 1D FWT** (MFWT). Applying the MFWT to a matrix $\boldsymbol{X}$ corresponds to computing the expression

$$\boldsymbol{X}(\boldsymbol{W}^{\lambda_N})^T \tag{5.6}$$

where $\boldsymbol{W}^{\lambda_N}$ is defined as in (3.34). Since there are M rows, the number of floating point operations needed are

$$F_{\text{MFWT}}(M,N) = 4DMN\left(1 - 1/2^{\lambda_N}\right) \tag{5.7}$$

The recurrence formulas now take the form

$$\begin{aligned} c^{i+1}_{m,n} &= \sum_{l=0}^{D-1} a_l c^{i}_{m,\langle l+2n\rangle_{S_i}} \\ d^{i+1}_{m,n} &= \sum_{l=0}^{D-1} b_l c^{i}_{m,\langle l+2n\rangle_{S_i}} \end{aligned} \tag{5.8}$$

where $i = 0, 1, \dots, \lambda_N - 1$, $m = 0, 1, \dots, M-1$, and $n = 0, 1, \dots, S_{i+1} - 1$. Timings for the MFWT with $\lambda_N = J = 10$, $N = 2^J$, and $D = 20$, where only the vectorized dimension M is varied, are shown in Table 5.3.

We will now derive a performance model for this case. Each step of the MFWT applies a PWT of length S_i to the M rows of $\boldsymbol{X}$. Hence, by (3.36) the number of flops are $2DS_iM$. Vectorization is achieved by putting m into the innermost loop so computations on each column can be assumed to follow the linear model for

M	F	$T(\mu s)$	R(Mflop/s)
16	1309440	11240	120
32	2618880	10889	240
64	5237760	11133	474
128	10475520	11474	912
256	20951040	14941	1402
512	41902080	23594	1777
1024	83804160	43711	1919
2048	167608320	84994	1974

Table 5.3: Timings of the MFWT. $\lambda_N = J = 10$, $N = 2^J$, $D = 20$, and $M = 16, 32, \ldots, 2048$.

vectorization (5.1). Hence the execution time for one step of the MFWT (5.8) is $S_i(t_s + 2DMt_v)$ and the execution time for the entire MFWT is

$$\begin{aligned} T_{\text{MFWT}} &= \sum_{i=0}^{\lambda_N - 1} \frac{N}{2^i}(t_s + 2DMt_v) \\ &= 2N\left(1 - \frac{1}{2^{\lambda_N}}\right)(t_s + 2DMt_v) \end{aligned} \tag{5.9}$$

The performance is then given by

$$R_{\text{MFWT}} = \frac{F_{\text{MFWT}}}{T_{\text{MFWT}}} = \frac{2DM}{t_s + 2DMt_v}$$

and $(R_\infty)_{\text{MFWT}} = 1/t_v$ as usual. However, we observe that the performance measure is independent of the depth λ_N. The parameter $M_{1/2}$ is found by solving

$$\frac{1}{2t_v} = \frac{2DM_{1/2}}{t_s + 2DM_{1/2}t_v}$$

which has the solution

$$M_{1/2} = \frac{t_s}{2Dt_v}$$

Hence

$$\begin{aligned} (F_{1/2})_{\text{MFWT}} &= 4DN\left(1 - \frac{1}{2^{\lambda_N}}\right)M_{1/2} \\ &= 2N\left(1 - \frac{1}{2^{\lambda_N}}\right)t_s/t_v \end{aligned}$$

Using (5.9) and the measurements in Table 5.3 we get the new estimates

$$t_s = 3.73\ \mu\text{s} \quad \text{and} \quad t_v = 0.000450\ \mu\text{s}$$

These estimates are different from those of the 1D case and reflect the fact that the MFWT algorithm performs better on the VPP300. Consequently we now have

$$\begin{aligned} (R_\infty)_{\text{MFWT}} &= 2.222 \ \text{ Gflop/s and} \\ (F_{1/2})_{\text{MFWT}} &= 16959000 \ \text{ Operations} \end{aligned}$$

the latter corresponding to a vector length $M_{1/2} = 208$. These values are close to being optimal on the VPP300 (recall that the peak performance per processor is 2.272 Gflop/s). Finally, since $(F_{1/2})_{\text{MFWT}}$ grows with N we note that the MFWT is best for matrices with $M \geq N$.

5.4 2D FWT

Recall from Section 6.3 that the 2D wavelet transform is defined by the matrix product

$$\breve{\boldsymbol{X}} = \boldsymbol{W}^{\lambda_M} \boldsymbol{X} (\boldsymbol{W}^{\lambda_N})^T \tag{5.10}$$

The expression $\boldsymbol{X}(\boldsymbol{W}^{\lambda_N})^T$ leads to vector operations on vectors of length M and stride-one data access as described in Section 5.3. This is not the case for the expression $\boldsymbol{W}^{\lambda_M}\boldsymbol{X}$, because it consists of a collection of columnwise 1D transforms which do not access the memory efficiently as described in Section 5.2. However (5.10) can be rewritten as

$$\breve{\boldsymbol{X}}^T = \left(\boldsymbol{X}(\boldsymbol{W}^{\lambda_N})^T\right)^T (\boldsymbol{W}^{\lambda_M})^T$$

yielding the efficiency of the multiple 1D FWT at the cost of one transpose step. We call this the **split-transpose algorithm**. It consists of the following three stages:

Algorithm 5.1: split-transpose

1. $\boldsymbol{Z} = \boldsymbol{X}(\boldsymbol{W}^{\lambda_N})^T$
2. $\boldsymbol{U} = \boldsymbol{Z}^T$
3. $\breve{\boldsymbol{X}}^T = \boldsymbol{U}(\boldsymbol{W}^{\lambda_M})^T$

M	F	$T(\mu s)$	R(Mflop/s)
16	2538240	11903	213
32	5158400	12748	404
64	10398720	14305	726
128	20879360	17686	1180
256	41840640	27167	1540
512	83763200	47686	1756
1024	167608320	91751	1826

Table 5.4: Timings of 2D FWT (FWT2). $\lambda_N = J = 10$, $N = 2^J$, $D = 20$, and $M = 16, 32, \ldots, 1024$.

Transposition can be implemented efficiently on a vector processor by accessing the matrix elements along the diagonals [Heg95], so the 2D FWT retains the good vector performance of the MFWT. This is verified by the timings in Table 5.4.

Disregarding the time for the transposition step, a simple model for the 2D FWT execution time is

$$\boxed{T_{\text{FWT2}}(M, N) = T_{\text{MFWT}}(M, N) + T_{\text{MFWT}}(N, M)} \tag{5.11}$$

where T_{MFWT} is given by (5.9). Figure 5.2 shows predicted execution times versus measured execution times, and it is seen that (5.11) predicts the performance of the 2D FWT well.

The asymptotic performance $(R_\infty)_{\text{FWT2}}$ can now be determined. Using (3.40), (5.2), and (5.11) we get

$$R_{\text{FWT2}} = \frac{4DMN(2 - 1/2^{\lambda_M} - 1/2^{\lambda_N})}{2N(1 - 1/2^{\lambda_N})(t_s + 2DMt_v) + 2M(1 - 1/2^{\lambda_M})(t_s + 2DNt_v)}$$

Assuming that $\lambda_M = \log_2(M)$ and $\lambda_N = \log_2(N)$ (the maximal depths) and that M and N are large, this yields the estimate

$$R_{\text{FWT2}} \approx \frac{1}{\frac{M+N}{4DMN} t_s + t_v} \tag{5.12}$$

and letting $M \to \infty$ we obtain

$$(R_\infty)_{\text{FWT2}}(N) = \frac{1}{\frac{t_s}{4DN} + t_v} \tag{5.13}$$

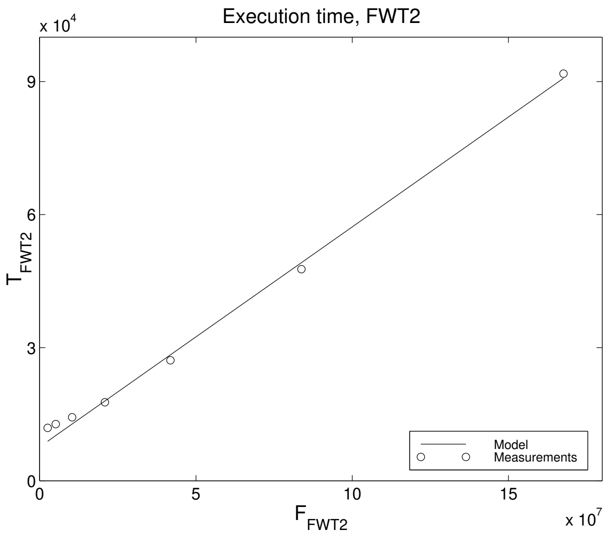


Figure 5.2: The measured times for the 2D FWT compared to the performance model (5.11).

It is seen that if N goes to infinity as well, we obtain the usual expression $(1/t_v)$, but for the present case, with $N = 1024$, we have from (5.12)

$$(R_\infty)_{\text{FWT2}} = 2.018 \quad \text{Gflop/s}$$

which is only slightly less than $(R_\infty)_{\text{MFWT}}$. Similarly, to find $(F_{1/2})_{\text{FWT2}}$ for the problem at hand we use (5.12) and (5.13) to get the equation

$$\frac{2DN}{t_s + 4DNt_v} = \frac{1}{\frac{M_{1/2}+N}{4DM_{1/2}N}t_s + t_v}$$

From this we find

$$M_{1/2} = \frac{t_s}{4Dt_v + t_s/N} \approx 94$$

corresponding to $(F_{1/2})_{\text{FWT2}} = 1.5326 \times 10^7$. It is seen that this value is very close to that of the MFWT despite the fact that the transposition step was included in the measured execution times. However, the corresponding vector length $M_{1/2}$ is much smaller as can also be seen in Table 5.4. The reason for this is the fact that where the MFWT suffers from short vector lengths when M is small, the 2D FWT suffers only in stage 1 of the split-transpose algorithm (5.1). Stage 3 will vectorize over N and yield good performance in this case. We see that the 2D FWT performs very well indeed on the VPP300, even for rectangular matrices.

5.5 Summary

The FWT has been implemented on the VPP 300. The one-dimensional version has a relatively high value of $N_{1/2}$ and stride-two memory access so the performance is not very good. The 2D FWT can be arranged so that these problems are avoided, and a performance of more than 80% of the theoretical peak performance is achieved even for relatively small problems. This is fortunate as the 2D FWT is computationally more intensive than the 1D FWT and consequently, it justifies better the use of a supercomputer.

Chapter 6

Parallelization of the Fast Wavelet Transform

With a parallel architecture, the aim is to distribute the work among several processors in order to compute the result faster or to be able to solve larger problems than what is possible with just one processor. Let $T^0(N)$ be the time it takes to compute the FWT with a sequential algorithm on one processor. Ideally, the time needed to compute the same task on P processors[1] is then $T^0(N)/P$. However, there are a number of reasons why this ideal is rarely possible to meet:

1. There will normally be some computational overhead in the form of bookkeeping involved in the parallel algorithm. This adds to the execution time.

2. If r is the fraction of the program statements which is parallelizable then the execution time on P processors is bounded from below by $(1-r)T^0(N) + rT^0(N)/P$ which is also larger than the ideal. This is known as Amdahl's law .

3. The processors might not be assigned the same amount of work. This means that some processors will be idle while others are doing more than their fair share of the work. In that case, the parallel execution time will be determined by the processor which is the last to finish. This is known as the problem of good load balancing.

4. Processors must communicate information and synchronize in order for the arithmetic to be performed on the correct data and in the correct sequence. This communication and synchronization will delay the computation depending on the amount which is communicated and the frequency by which it occurs.

[1] In this chapter P stands for the number of processors and should not be confused with the number of vanishing moments.

$p=0$	$p=1$
$c_0^0c_1^0c_2^0c_3^0c_4^0c_5^0c_6^0c_7^0$	$c_8^0c_9^0c_{10}^0c_{11}^0c_{12}^0c_{13}^0c_{14}^0c_{15}^0$
$\downarrow$	$\downarrow$
$c_0^1c_1^1c_2^1c_3^1c_4^1c_5^1c_6^1c_7^1$	$d_0^1d_1^1d_2^1d_3^1d_4^1d_5^1d_6^1d_7^1$
$\downarrow$	
$c_0^2c_1^2c_2^2c_3^2\ d_0^2d_1^2d_2^2d_3^2$	$d_0^1d_1^1d_2^1d_3^1d_4^1d_5^1d_6^1d_7^1$
$\downarrow$	
$c_0^3c_1^3\ d_0^3d_1^3\quad d_0^2d_1^2d_2^2d_3^2$	$d_0^1d_1^1d_2^1d_3^1d_4^1d_5^1d_6^1d_7^1$

Table 6.1: Standard data layout results in poor load balancing. The shaded sub-vectors are those parts which do not require further processing. Here $P = 2$, $N = 16$, and $\lambda = 3$.

In this chapter we will discuss different parallelization strategies for the FWTs with special regard to the effects of load balancing, communication, and synchronization. We will disregard the influence of the first two points since we assume that the parallel overhead is small and that the FWT per se has no significant unparallelizable part. However, in applications using the FWT this problem may become significant. Most of the material covered in this chapter has also appeared in [NH97].

6.1 1D FWT

We will now address the problem of distributing the work needed to compute the FWT ($\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$) as defined in Definition 3.1 on P processors denoted by $p = 0, 1, \ldots, P-1$. We assume that the processors are organized in a ring topology such that $\langle p-1\rangle_P$ and $\langle p+1\rangle_P$ are the left and right neighbors of processor p, respectively. Assume also, for simplicity, that N is a multiple of P and that the initial vector $\boldsymbol{x}$ is distributed such that each processor receives the same number of consecutive elements. This means that processor p holds the elements

$$\{c_n^0\}_n,\ n = p\frac{N}{P},\ p\frac{N}{P}+1,\ \ldots,\ (p+1)\frac{N}{P}-1$$

A question that is crucial to the performance of a parallel FWT is how to chose the optimal distribution of $\boldsymbol{y}$ and the intermediate vectors.

We consider first the data layout suggested by the sequential algorithm in Definition 3.1. This is shown in Table 6.1. It is seen that distributing the results of each transform step evenly across the processors results in a poor load balancing

$p = 0$	$p = 1$
$c_0^0 c_1^0 c_2^0 c_3^0 c_4^0 c_5^0 c_6^0 c_7^0$	$c_8^0 c_9^0 c_{10}^0 c_{11}^0 c_{12}^0 c_{13}^0 c_{14}^0 c_{15}^0$
$\downarrow$	$\downarrow$
$c_0^1 c_1^1 c_2^1 c_3^1$ $d_0^1 d_1^1 d_2^1 d_3^1$	$c_4^1 c_5^1 c_6^1 c_7^1$ $d_4^1 d_5^1 d_6^1 d_7^1$
$\downarrow$	$\downarrow$
$c_0^2 c_1^2$ $d_0^2 d_1^2$ $d_0^1 d_1^1 d_2^1 d_3^1$	$c_2^2 c_3^2$ $d_2^2 d_3^2$ $d_4^1 d_5^1 d_6^1 d_7^1$
$\downarrow$	$\downarrow$
c_0^3 d_0^3 $d_0^2 d_1^2$ $d_0^1 d_1^1 d_2^1 d_3^1$	c_1^3 d_1^3 $d_2^2 d_3^2$ $d_4^1 d_5^1 d_6^1 d_7^1$

Table 6.2: Good load balancing is obtained by using a different data layout. The shaded sub-vectors are those parts which do not require further processing. Again, we have $P = 2$, $N = 16$, and $\lambda = 3$.

because each step works with the lower half of the previous vector only. The processors containing parts that are finished early are idle in the subsequent steps. In addition, global communication is required in the first step because every processor must know the values on every other processor in order to compute its own part of the wavelet transform. In subsequent steps this communication will take place among the active processors only. This kind of layout was used in [DMC95] where it was observed that optimal load balancing could not be achieved, and also in [Lu93] where the global communication was treated by organizing the processors of a connection machine (CM-2) in a pyramid structure.

However, we can obtain perfect load balancing and avoid global communication by introducing another ordering of the intermediate and resulting vectors. This is shown in Table 6.2. Processor p will now compute and store the elements $\{c_n^{i+1}\}_n$ and $\{d_n^{i+1}\}_n$ where

$$\begin{aligned} n &= p\frac{N}{P2^{i+1}},\ p\frac{N}{P2^{i+1}}+1,\ \dots,\ (p+1)\frac{N}{P2^{i+1}}-1 \\ i &= 0,1,\dots,\lambda-1 \end{aligned} \tag{6.1}$$

Let now $S_i^P = S_i/P = N/(P2^i)$. Then the recurrence formulas are almost the same as (3.33):

$$\begin{aligned} c_n^{i+1} &= \sum_{l=0}^{D-1} a_l c^i_{\langle l+2n \rangle_{S_i}} \\ d_n^{i+1} &= \sum_{l=0}^{D-1} b_l c^i_{\langle l+2n \rangle_{S_i}} \end{aligned} \tag{6.2}$$

where $i = 0, 1, \dots, \lambda - 1$ and $n = pS_{i+1}^P, pS_{i+1}^P + 1, \dots, (p+1)S_{i+1}^P - 1$. The difference lies in the periodic wrapping which is still global, i.e. elements from processor 0 must be copied to processor $P-1$. However, it turns out that this is just a special case of the general communication pattern for the algorithms, as described in Section 6.1.1.

Note that the layout shown in Table 6.2 is a permutation of the layout shown in Table 6.1 because each processor essentially performs a *local* wavelet transform of its data. However, the ordering suggested by Table 6.1 and also by Figure 3.3 is by no means intrinsic to the FWT so this permutation is not a disadvantage at all. Rather, one might argue as follows:

> Local transforms reflect better the essence of the wavelet philosophy because all scale information concerning a particular position remains on the same processor.

This layout is even likely to increase performance for further processing steps (such as compression) because it preserves locality of the data.

Note also that the local transforms in this example have reached their ultimate form on each processor after only three steps and that it would not be feasible to continue the recursion further (i.e. by letting $\lambda = 4$ and splitting $\{c_0^3, c_1^3\}$ into $\{c_0^4, d_0^4\}$) because then $N/(P2^\lambda) < 1$, (6.1) no longer holds, and the resulting data distribution would lead to load imbalance as with the algorithm mentioned above. Thus, to maintain good load balancing we must have an upper bound on λ:

$$\lambda \leq \log_2 \left(\frac{N}{P} \right)$$

In fact, this bound has to be even more restrictive in order to avoid excessive communication. We will return to this in Section 6.1.1.

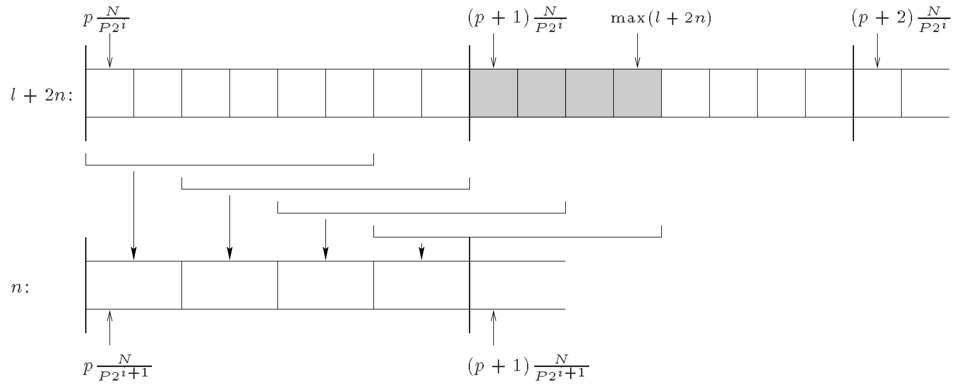


Figure 6.1: Computations on processor p involve $D-2$ elements from processor $p+1$. Here $D = 6$ and $N/(P2^i) = 8$. The lines of width D indicate the filters as they are applied for different values of n.

6.1.1 Communication

We will now consider the amount of communication required for the parallel 1D FWT. Consider the computations done by processor p on a row vector as indicated in Figure 6.1. The quantities in (6.2) can be computed without any communication provided that the index $l+2n$ does not refer to elements on other processors, i.e.

$$\begin{aligned} l+2n &\leq (p+1)\frac{N}{P2^i}-1 \\ n &\leq (p+1)\frac{N}{P2^{i+1}}-\frac{l+1}{2} \end{aligned} \tag{6.3}$$

A sufficient condition (independent of l) for this is

$$\boxed{n \leq (p+1)\frac{N}{P2^{i+1}}-\frac{D}{2}} \tag{6.4}$$

since $l \in [0, D-1]$. We use this criterion to separate the local computations from those that may require communication.

For a fixed $n > (p+1)N/(P2^{i+1}) - D/2$ computations are still local as long as (6.3) is fulfilled, i.e. when

$$\boxed{l \leq (p+1)\frac{N}{P2^i}-2n-1} \tag{6.5}$$

However, when l becomes larger than this, the index $l+2n$ will point to elements residing on a processor located to the right of processor p. The largest value of

$l + 2n$ (found from (6.2) and (6.1)) is

$$\max(l + 2n) = (p + 1)\frac{N}{P2^i} + D - 3 \tag{6.6}$$

The largest value of $l + 2n$ for which communication is not necessary is

$$(p + 1)\frac{N}{P2^i} - 1$$

Subtracting this quantity from (6.6) we find that exactly $D - 2$ elements must be communicated to processor p at each step of the FWT as indicated in Figure 6.1.

A tighter bound on λ

It is a condition for good performance that the communication pattern described above takes place between nearest neighbors only. Therefore, we want to avoid situations where processor p needs data from processors other than its right neighbor $\langle p + 1\rangle_P$ so we impose the additional restriction

$$\begin{aligned} \max(l + 2n) &\le (p + 2)\frac{N}{P2^i} - 1 \\ (p + 1)\frac{N}{P2^i} + D - 3 &\le (p + 2)\frac{N}{P2^i} - 1 \\ D - 2 &\le \frac{N}{P2^i} \end{aligned} \tag{6.7}$$

Since we want (6.7) to hold for all $i = 0, 1, \ldots, \lambda - 1$ we get

$$D - 2 \le \frac{N}{P2^{\lambda-1}}$$

from which we obtain the final bound on λ:

$$\boxed{\lambda \le \log_2\left(\frac{2N}{(D - 2)P}\right)} \tag{6.8}$$

For $N = 256$, $D = 8$, $P = 16$, for example, we find

$$\lambda \le 5$$

The bound given in (6.8) is not as restrictive as it may seem: Firstly, for the applications where a parallel code is called for, one normally has $N \gg \max(P, D)$, secondly, in most practical wavelet applications one takes λ to be a fixed small number, say 4–5 [Str96], and thirdly, should the need arise for a large value of λ, one could use a sequential code for the last steps of the FWT as these will not involve large amounts of data.

6.2 Multiple 1D FWT

The considerations from the previous section are still valid if we replace single elements with columns. This is a parallel version of the MFWT described in Section 5.3. Figure 6.2 shows the data layout of the parallel MFWT algorithm.

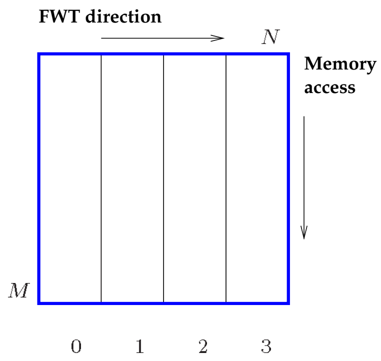


Figure 6.2: Multiple FWT. Data are distributed columnwise on the processors. The FWT is organized rowwise in order to access data with stride one.

The amount of necessary communication is now $M(D-2)$ elements instead of $D-2$, the columns of $\boldsymbol{X}$ are distributed blockwise on the processors, and the transformation of the rows of $\boldsymbol{X}$ involves the recursion formulas corresponding to $\boldsymbol{X}\boldsymbol{W}_N^T$. The recursion formulas take the same form as in Section 5.3. The only difference from the sequential case is that n is now given as in (6.1).

We are now ready to give the algorithm for computing one step of the multiple 1D FWT. The full transform is obtained by repeating this step for $i = 0, 1, \ldots, \lambda - 1$. The algorithm falls naturally into the following three phases:

1. **Communication phase:** $D-2$ columns are copied from the right neighbor as these are sufficient to complete all subsequent computations locally. We denote these columns by the block $\bar{c}^i_{:,0:D-3}$.

2. **Fully local phase:** The interior of each block is transformed, possibly overlapping the communication process.

3. **Partially remote phase:** When the communication has completed, the remaining elements are computed using $\bar{c}^i_{:,l+2n-N/(P2^i)}$ whenever $l + 2n \geq N/(P2^i)$.

Algorithm 6.1: MFWT, level $i \to i+1$

$S_i^P = \frac{N}{P2^i}$
$p =$"my processor id" $\in [0 : P-1]$

!
! Communication phase
!

send $c^i_{:,0:D-3}$ **to** processor $\langle p-1 \rangle_P$
receive $\bar{c}^i_{:,0:D-3}$ **from** processor $\langle p+1 \rangle_P$

!
! Fully local phase, cf. (6.4)
!

for $n = 0 : S_i^P/2 - D/2$

$c^{i+1}_{:,n} = \sum_{l=0}^{D-1} a_l c^i_{:,l+2n}$! $\min(l+2n) = 0$

$d^{i+1}_{:,n} = \sum_{l=0}^{D-1} b_l c^i_{:,l+2n}$! $\max(l+2n) = S_i^P - 1$

end

!
! Partially remote phase
! communication must be finished at this point
!

for $n = S_i^P/2 - D/2 + 1 : S_i^P/2 - 1$

!
! Local part, cf (6.5)
!

$c^{i+1}_{:,n} = \sum_{l=0}^{S_i^P-2n-1} a_l c^i_{:,l+2n}$! $\min(l+2n) = S_i^P - D + 2$

$d^{i+1}_{:,n} = \sum_{l=0}^{S_i^P-2n-1} b_l c^i_{:,l+2n}$! $\max(l+2n) = S_i^P - 1$

!
! Remote part, use $\bar{c}^j_{:,0:D-3}$
!

$c^{i+1}_{:,n} = \sum_{l=S_i^P-2n}^{D-1} a_l \bar{c}^i_{:,l+2n-S_i^P}$! $\min(l+2n) = S_i^P$

$d^{i+1}_{:,n} = \sum_{l=S_i^P-2n}^{D-1} b_l \bar{c}^i_{:,l+2n-S_i^P}$! $\max(l+2n) = S_i^P + D - 3$

end

6.2.1 Performance model for the multiple 1D FWT

The purpose of this section is to focus on the impact of the proposed communication scheme on performance with particular regard to speedup and efficiency. We will consider the theoretically best achievable performance of the multiple 1D FWT algorithm. Recall that (5.6) can be computed using

$$F_{\text{MFWT}}(N) = 4DMN\left(1 - \frac{1}{2^{\lambda_N}}\right) \tag{6.9}$$

floating point operations. We emphasize the dependency on N because it denotes the dimension over which the problem is parallelized.

Let t_f be the average time it takes to compute one floating point operation on a given computer[2]. Hence, the time needed to compute (5.6) sequentially is

$$T^0_{\text{MFWT}}(N) = F_{\text{MFWT}}(N)t_f \tag{6.10}$$

and the theoretical sequential performance becomes

$$R^0_{\text{MFWT}}(N) = \frac{F_{\text{MFWT}}(N)}{T^0_{\text{MFWT}}(N)} \tag{6.11}$$

In our proposed algorithm for computing (5.6) the amount of double precision numbers that must be communicated between adjacent neighbors at each step of the wavelet transform is $M(D-2)$ as described in Section 6.2. Let t_l be the time it takes to initiate the communication (latency) and t_d the time it takes to send one double precision number. Since there are λ steps in the wavelet transform a simple model for the total communication time is

$$C_{\text{MFWT}} = \lambda(t_l + M(D-2)t_d) \tag{6.12}$$

Note that C_{MFWT} grows linearly with M but that it is independent of the number of processors P as well as the size of the second dimension N!

Combining the expression for computation time and communication time we obtain a model describing the total execution time on P processors ($P > 1$) as

$$\boxed{T^P_{\text{MFWT}}(N) = \frac{T^0_{\text{MFWT}}(N)}{P} + C_{\text{MFWT}}} \tag{6.13}$$

[2]This model for sequential performance is simplified by disregarding effects arising from the use of cache memory, pipelining or super scalar processors. Adverse effects resulting from suboptimal use of these features are assumed to be included in t_f to give an average estimate of the actual execution time. Thus, if we estimate t_f from the sequential model (6.10), it will normally be somewhat larger than the nominal value specified for a given computer. In case of the linear model for vector performance (5.1) we get for example $t_f = t_s/F + t_v$.

and the performance of the parallel algorithm is

$$R^P_{\text{MFWT}}(N) = \frac{F_{\text{MFWT}}(N)}{T^P_{\text{MFWT}}(N)} \tag{6.14}$$

The expressions for performance in (6.11), (6.14), and (6.13) lead to a formula for the speedup of the MFWT algorithm:

$$S^P_{\text{MFWT}}(N) = \frac{T^0_{\text{MFWT}}(N)}{T^P_{\text{MFWT}}(N)} = \frac{P}{1 + PC_{\text{MFWT}}/T^0_{\text{MFWT}}(N)}$$

The **efficiency** of the parallel implementation is defined as the speedup per processor and we have

$$E^P_{\text{MFWT}}(N) = \frac{S^P_{\text{MFWT}}(N)}{P} = \frac{1}{1 + PC_{\text{MFWT}}/T^0_{\text{MFWT}}(N)} \tag{6.15}$$

It can be seen from (6.15) that for constant N, the efficiency will decrease when the number of processors P is increased.

We will now investigate how the above algorithm scales with respect to the number of processors when the amount of work per processor is held constant. Thus let N_1 be the constant size of a problem on one processor. Then the total problem size becomes $N = PN_1$ and we find from (6.9) and (6.10) that $T^0_{\text{MFWT}}(PN_1) = PT^0_{\text{MFWT}}(N_1)$ because the computational work of the FWT is linear in N. This means in turn that the efficiency for the scaled problem takes the form

$$E^P_{\text{MFWT}}(PN_1) = \frac{1}{1 + \frac{PC_{\text{MFWT}}}{PT^0_{\text{MFWT}}(N_1)}} = \frac{1}{1 + C_{\text{MFWT}}/T^0_{\text{MFWT}}(N_1)}$$

Since $E^P_{\text{MFWT}}(PN_1)$ is independent of P the scaled efficiency is constant. Hence the multiple 1D FWT algorithm is fully scalable.

6.3 2D FWT

In this section we will consider two approaches to parallelize the split-transpose algorithm for the 2D FWT as described in Section 5.4.

The first approach is similar to the way 2D FFTs can be parallelized [Heg96] in that it uses the sequential multiple 1D FWT and a parallel transpose algorithm; we denote it the **replicated FWT**. The second approach makes use of the parallel multiple 1D FWT described in Section 6.2 to avoid the parallel transposition. We denote this approach the **communication-efficient FWT**.

In both cases we assume that the transform depth is the same in each dimension, i.e. $\lambda = \lambda_M = \lambda_N$. Then we get from (3.40) and (5.7) that the sequential execution time for the 2D FWT is

$$T^0_{\text{FWT2}}(N) = 2T^0_{\text{MFWT}}(N). \tag{6.16}$$

6.3.1 Replicated FWT

The most straightforward way of dividing the work involved in the 2D FWT algorithm among a number of processors is to parallelize along the first dimension in $\boldsymbol{X}$, such that a sequence of 1D row transforms are executed independently on each processor. This is illustrated in Figure 6.3. Since we replicate independent row transforms on the processors we denote this approach the replicated FWT (RFWT) algorithm. Here it is assumed that the matrix $\mathbf{X}$ is distributed such that

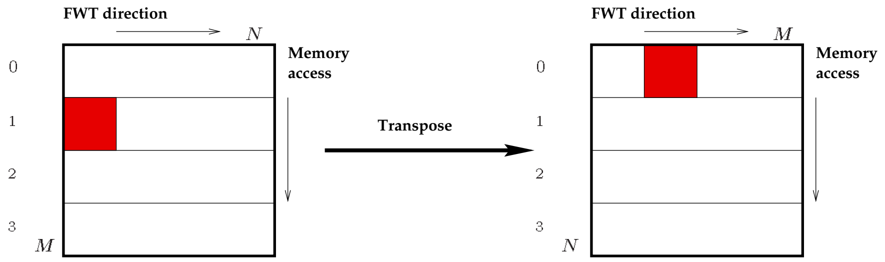


Figure 6.3: Replicated FWT. The shaded block moves from processor 1 to 0.

each processor receives the same number of consecutive rows of $\mathbf{X}$. The first and the last stages of Algorithm 5.1 are thus done without any communication. However, the intermediate stage, the transposition, causes a substantial communication overhead. A further disadvantage of this approach is the fact that it reduces the maximal vector length available for vectorization from M to M/P (and from N to N/P). This is a problem for vector architectures such as the Fujitsu VPP300 as decribed in section 5.3.

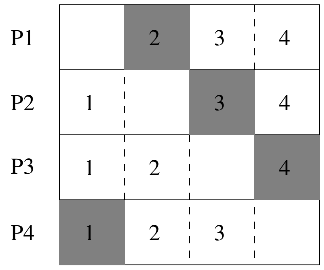


Figure 6.4: Communication of blocks, first block-diagonal shaded.

A similar approach was adopted in [LS95] where a 2D FWT was implemented on the MasPar - a data parallel computer with 2048 processors. It was noted that "the transpose operations dominate the computation time" and a speedup of no more than 6 times relative to the best sequential program was achieved.

A suitable parallel transpose algorithm needed for the replicated FWT is one that moves data in wrapped block diagonals as outlined in the next section.

Parallel transposition and data distribution

Assume that the rows of the matrix $\boldsymbol{X}$ are distributed over the processors, such that each processor gets M/P consecutive rows, and that the transpose $\boldsymbol{X}^T$ is distributed such that each processor gets N/P rows. Imagine that the part of matrix $\boldsymbol{X}$ that resides on each processor is split columnwise into P blocks, as suggested in Figure 6.4, then the blocks denoted by i are moved to processor i during the transpose. In total each processor must send $P-1$ blocks and each block contains M/P times N/P elements of $\boldsymbol{X}$. Hence, following the notation in Section 6.2.1, we get the model for communication time of a parallel transposition

$$C_{\text{RFWT}} = (P-1)\left(t_l + \frac{MN}{P^2}t_d\right) \tag{6.17}$$

Note that C_{RFWT} grows linearly with M, N and P (for P large).

Performance model for the replicated FWT

We are now ready to derive a performance model for the replicated FWT algorithm. Using (6.16) and (6.17) we obtain the parallel execution time as

$$T^P_{\text{RFWT}}(N) = \frac{T^0_{\text{FWT2}}(N)}{P} + C_{\text{RFWT}}$$

and the theoretical speedup for the scaled problem $N = PN_1$ is

$$S^P_{\text{RFWT}}(PN_1) = \frac{P}{1 + C_{\text{RFWT}}/T^0_{\text{FWT2}}(N_1)} \tag{6.18}$$

We will return to this expression in Section 6.3.3.

6.3.2 Communication-efficient FWT

In this section we combine the multiple 1D FWT described in Section 6.2 and the replicated FWT idea described in Section 6.3.1 to get a 2D FWT that combines the best of both worlds. The first stage of Algorithm 5.1 is computed using the parallel multiple 1D FWT as given in Algorithm 6.1, so consecutive *columns* of $\mathbf{X}$ must be distributed to the processors. However, the last stage uses the layout from the replicated FWT, i.e. consecutive *rows* are distributed to the processors. This is illustrated in Figure 6.5.

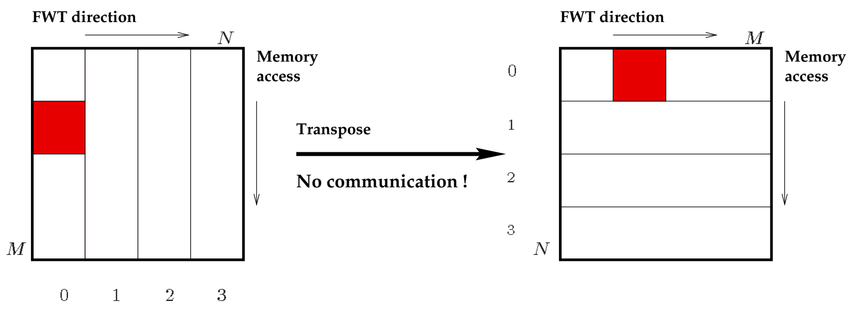


Figure 6.5: Communication-efficient FWT. Data in shaded block stay on processor 0.

The main benefit using this approach is that the transpose step is done without any communication whatsoever. The only communication required is that of the multiple 1D FWT, namely the transmission of $M(D-2)$ elements between nearest neighbors, so most of the data stay on the same processor throughout the computations. The result will therefore be permuted in the N-dimension as described in Section 6.1 and ordered normally in the other dimension. We call this algorithm the communication-efficient FWT (CFWT) .

The performance model for the communication-efficient FWT is a straightforward extension of the MFWT because the communication part is the same, so we get the theoretical speedup

$$S^P_{\text{CFWT}}(PN_1) = \frac{P}{1 + C_{\text{MFWT}}/T^0_{\text{FWT2}}(N_1)} \tag{6.19}$$

where C_{MFWT} and $T^0_{\text{FWT2}}(N_1)$ are as given in (6.12) and (6.16) respectively.

6.3.3 Comparison of the 2D FWT algorithms

We can now compare the theoretical performance of the RFWT (6.18) and the CFWT (6.19) with regard to their respective dependencies on P and N_1.

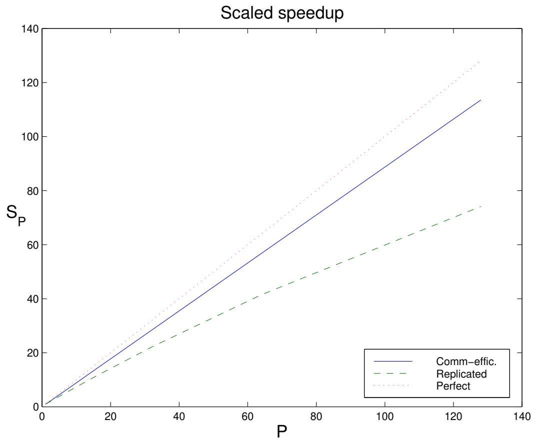


Figure 6.6: The theoretical scaled speedup of the replicated FWT algorithm and the communication-efficient FWT shown together with the line of perfect speedup. The predicted performances correspond to a problem with $M = 512,\ N = 128,\ D = 12$. The characteristic parameters were measured on an IBM SP2 to be $t_d = 0.2\ \mu$s, $t_l = 200\ \mu$s, $t_f = 6\ n$s. The performance of the communication-efficient FWT is much closer to the line of perfect speedup than the performance of the replicated FWT, and the slope of the curve remains constant.

In case of the CFWT the ratio $C_{\text{MFWT}}/T^0_{\text{FWT2}}(N_1)$ is constant with respect to P whereas the corresponding ratio for the RFWT in (6.18) goes as $\mathcal{O}(P)$:

$$\frac{C_{\text{RFWT}}}{T^0_{\text{FWT2}}(N_1)} \approx \frac{(P-1)t_l + \frac{P-1}{P^2} M N_1 t_d}{8DMN_1} = \mathcal{O}(P)$$

This means that the efficiency of the RFWT will deteriorate as P grows while it will stay constant for the CFWT. The corresponding speedups are shown in Figure 6.6.

When P is fixed and the problem size N_1 grows, then $C_{\text{MFWT}}/T^0_{\text{FWT2}}(N_1)$ goes to zero, which means that the scaled efficiency of the CFWT will approach the ideal value 1. For the RFWT the corresponding ratio approaches a positive constant as N_1 grows:

$$\frac{C_{\text{RFWT}}}{T^0_{\text{FWT2}}(N_1)} \quad \rightarrow \quad \frac{(P-1)t_d}{8DP^2} \text{ for } N_1 \rightarrow \infty$$

P	N	Mflop/s	Efficiency (%)	Estim. eff.
0	128	183		
1	128	176	96.18	97.61
2	256	301	82.24	82.12
4	512	601	82.10	82.12
8	1024	1199	81.90	82.12
16	2048	2400	81.97	82.12
32	4196	4796	81.90	82.12

Table 6.3: Communication-efficient FWT on the SP2. $N = PN_1$, $N_1 = 128$, $M = 128$, $D = 10$. $P = 0$ signifies sequential performance. Estimated efficience is given as $S^P_{\text{CFWT}}(PN_1)/P$ where $S^P_{\text{CFWT}}(PN_1)$ is given as in (6.19).

This means that the scaled efficiency of the RFWT is bounded by a constant less than one – no matter how large the problem size. The asymptotic scaled efficiencies of the two algorithms are summarized below

	$P \to \infty$	$N_1 \to \infty$
Replicated FWT:	$\dfrac{1}{1+\mathcal{O}(P)}$	$\dfrac{1}{1+\frac{(P-1)t_d}{8DP^2}}$
Communication-efficient FWT:	$\dfrac{1}{1+\frac{C_{\text{MFWT}}}{T^0_{\text{FWT2}}(N_1)}}$	1

6.3.4 Numerical experiments

We have implemented the communication-efficient FWT on two different MIMD computer architectures, namely the IBM SP2 and the Fujitsu VPP300. On the SP2 we used MPI for the parallelism whereas the proprietary VPP Fortran was used on the VPP300.

The IBM SP2 is a parallel computer which is different from the VPP300. Each node on the SP2 is essentially a workstation which does not achieve the performance of a vector processor such as the VPP300. High performance on the SP2 must therefore be achieved through a higher degree of parallelism than on the VPP300 and scalability to a high number of processors is more urgent in this case. The measured performances on the IBM SP2 are shown in Table 6.3.

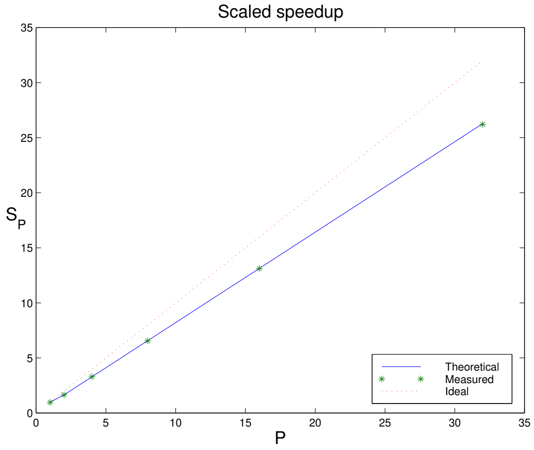


Figure 6.7: Scaled speedup, communication-efficient FWT (IBM SP2). These graphs depict how the theoretical performance model does, in fact, give a realistic prediction of the actual performance.

P	N	Mflop/s	Efficiency (%)
0	512	1300	
1	512	1278	98.31
2	1024	2551	98.12
4	2048	5058	97.27
8	4096	10186	97.94

Table 6.4: Communication-efficient FWT on the VPP300. $N \propto P$, $M = 512$, $D = 10$. $P = 0$ signifies sequential performance.

It is seen that the performance scales well with the number of processors and, furthermore, that it agrees with the predicted speedup as shown in Figure 6.7. The parallel performance on the Fujitsu VP300 is shown in Table 6.4. We have not estimated the characteristic numbers t_l, t_d, t_f for this machine, but it is nevertheless clear that the performance scales almost perfectly with the number of processors also in this case.

6.4 Summary

We have developed a new parallel algorithm for computing the 2D wavelet transform, the communication-efficient FWT.

Our new approach avoids the use of a distributed matrix transpose and performs significantly better than algorithms that require such a transpose. This is due to the fact that the communication volume of a parallel transpose is larger than necessary for computing the 2D FWT.

> The communication-efficient FWT is optimal in the sense that the scaled efficiency is *independent* of the number of processors and that it approaches 1 as the problem size is increased.

Implementations on the Fujitsu VPP300 and the IBM SP2 confirms the scalability of the CFWT algorithm.

Part III

Wavelets and Partial Differential Equations

Chapter 7

Wavelets and differentiation matrices

7.1 Previous wavelet applications to PDEs

Even though the field of wavelet theory has had a great impact on other fields, such as signal processing, it is not yet clear whether it will have a similar impact on numerical methods for solving partial differential equations (PDEs).

In the early nineties people were very optimistic because it seemed that the nice properties of wavelets would automatically lead to efficient solution methods for PDEs. The reason for this optimism was the fact that many nonlinear PDEs have solutions containing local phenomena (e.g. formation of shocks) and interactions between several scales (e.g. turbulence). Such solutions can often be well represented in wavelet bases, as explained in the previous chapters. It was therefore believed that efficient wavelet-based numerical schemes for solving these PDEs would follow from wavelet compression properties [BMP90, LT90b, LPT92, HPW94, GL94, CP96, PW96, F^+96, DKU96].

However, this early optimism remains to be honored. Wavelets have not had the expected impact on differential equations, partly because the computational work is not necessarily reduced by applying wavelet compression - even though the solution is sparsely represented in a wavelet basis. In the following chapters we will discuss some of the most promising approaches and shed some light on the obstacles that must be overcome in order to obtain successful wavelet-based PDE solvers.

Schematically, wavelet-based methods for PDEs can be separated into the following classes:

Class 1: Methods based on scaling function expansions

The unknown solution is expanded in scaling functions at some chosen level J and is solved using a Galerkin approach. Because of their compact support, the scaling functions can be regarded as alternatives to splines or the piecewise polynomials used in finite element schemes. While this approach is important in its own right, it cannot exploit wavelet compression. Hence methods in this category are not adaptive [LT90a, QW93, WA94, LR94, Jam93, RE97]. However, this approach has many points of interest. Leland Jameson [Jam93] has shown that one obtains a method which exhibits *super convergence* at the grid points, the order of approximation being twice as large as that of the projection of the solution onto the space spanned by scaling functions. Johan Waldén [Wal96] has shown that the size of the differentiation filters grows faster than the optimal centered finite difference method of the same order. In the limit, as the order goes to infinity, it is shown that $\int \phi'(x-k)\phi(x)\,dx \to (-1)^k/k$ for $D \to \infty$. Finally, we mention that Teresa Regińska and others [RE97, EBR97] use scaling functions to regularize the solution of the sideways heat equation. This is an ill-posed problem in the sense that the solution does not depend continuously on the initial condition. By expanding the solution in scaling functions, high frequency components can be filtered away and continuous dependence of the initial condition is restored.

Class 2: Methods based on wavelet expansions

The PDE is solved using a Galerkin approach as in the first class. In this case, however, the unknown solution is expressed in terms of *wavelets* instead of scaling functions so wavelet compression can be applied; either to the solution [LT90b], the differential operator [BCR91, Bey92, EOZ94, XS92], or both [CP96, BK97, PW96]. Several different approaches have been considered for exploiting the sparsity of a wavelet representation. One is to perform all operations in the wavelet domain [BN96, BMP90, Wal96]. The operators are sparse but the number of significant coefficients in the solutions $N_s(\varepsilon)$ has to be very small compared to the dimension of the problem N for the methods to be efficient. This is especially true for non-linear operations. However, recent work by Beylkin and Keiser [BK97] suggests that some nonlinearities may be efficiently treated in the wavelet domain.

In another approach linear operations such as differentiation are done in the wavelet domain and nonlinear operations such as squaring in the physical domain. This is by far the most common approach and it is used by [Kei95, CP96, FS97, PW96, EOZ94, VP96] and the author. This approach involves a number of transformations between the physical domain and the wavelet domain in each time step, and this can introduce considerable overhead. Hence, the wavelet compression potential of the solution must be very large for this approach to be feasible.

An interesting aspect of the wavelet approach is that certain operators represented with respect to a wavelet basis become sparser when raised to higher powers [EOZ94]. From this property one can obtain an efficient time-stepping scheme for certain evolution equations. This method has been employed in [CP96, Dor95] to solve the heat equation.

Finally, we mention that several papers have exploited the so-called *Non-standard* or *BCR* form of the differential operator [BCR91, Bey92, Dor95, BK97]. This form is sparser than the standard form, but to apply the non-standard form of the differential operator to a function one needs to know, at all scales, the scaling function coefficients of the function as well as its wavelet coefficients. This representation is redundant and, even though the function may be sparse in its wavelet representation, the scaling function representation may not be sparse.

We will refer to methods from Class 1 and 2 as **projection methods**.

Class 3: Wavelets and finite differences

In the third approach wavelets are used to derive adaptive finite difference methods. Instead of expanding the solution in terms of wavelets, the wavelet transform is used to determine where the finite difference grid must be refined or coarsened to optimally represent the solution. The computational overhead is low because one works with point values in the physical representation. One approach was developed by Leland Jameson, [Jam94, Jam96] under the name Wavelet Optimized Finite Difference Method (WOFD). Waldén describes a filter bank method in [Wal96], Matts Holmström has introduced the Sparse Point Representation (SPR) in [Hol97], and also [PK91] have used wavelets to localize where to apply grid refinement.

Class 4: Other methods

There are a few approaches that use wavelets in ways that do not fit into any of the previous classes. Examples are operator wavelets [JS93, EL], anti-derivatives of wavelets [XS92], the method of travelling wavelets [PB91, WZ94], and wavelet-preconditioning [Bey94].

Operator wavelets are wavelets that are (bi-)orthogonal with respect to an inner product designed for the particular differential operator in question. This is not a general method since it works only for certain operators. The use of anti-derivatives of wavelets is similar to that of operator wavelets.

In the method of travelling wavelets, an initial condition is expanded using only a few wavelets which are then propagated in time. A disadvantage of this method is that these few wavelets may be unable to express the solution after it has been propagated for a time.

Finally, in [Bey94] it is shown that any finite difference matrix representation of periodized differential operators can be preconditioned so that the condition number is $\mathcal{O}(1)$. Furthermore, if the difference matrix is represented in a wavelet basis (the standard form) then the preconditioner is a diagonal matrix. Thus, wavelets play an auxiliary role in that they provide a means to reduce the condition number of the operator.

In this chapter and Chapter 8 we describe the necessary background for methods belonging to Class 1 and 2. In Chapter 9 we give some applications of these methods to PDEs, and we also illustrate the WOFD method from Class 3. We will not discuss further methods belonging to class 4.

7.2 Connection coefficients

A natural starting point for the projection methods is the topic of two-term **connection coefficients**. The description adopted here is similar to the one described in [LRT91] and [Bar96]. An alternative method is described in [Kun94]. We define the connection coefficients as

$$\Gamma^{d_1,d_2}_{j,l,m} = \int_{-\infty}^{\infty} \phi^{(d_1)}_{j,l}(x)\phi^{(d_2)}_{j,m}(x)\,dx,\ j,l,m \in \mathbf{Z}$$

where d_1 and d_2 are orders of differentiation. We will assume for now that these derivatives are well-defined.

Using the change of variable $x \leftarrow (2^j x - l)$ one obtains

$$\Gamma^{d_1,d_2}_{j,l,m} = 2^{jd}\int_{-\infty}^{\infty} \phi^{(d_1)}(x)\phi^{(d_2)}(x-m+l)\,dx = 2^{jd}\Gamma^{d_1,d_2}_{0,0,m-l}$$

where $d = d_1 + d_2$. Repeated integration by parts yields the identity $\Gamma^{d_1,d_2}_{0,0,n} = (-1)^{d_1}\Gamma^{0,d}_{0,0,n}$ because the scaling functions have compact support. Hence

$$\Gamma^{d_1,d_2}_{j,l,m} = (-1)^{d_1}2^{jd}\Gamma^{0,d}_{0,0,m-l}$$

Therefore it is sufficient to consider only one order of differentiation (d) and one shift parameter $(m-l)$ and we define

$$\boxed{\Gamma^d_l = \int_{-\infty}^{\infty} \phi(x)\phi^{(d)}_l(x)\,dx, \quad l \in \mathbf{Z}} \tag{7.1}$$

Consequently

$$\Gamma^{d_1,d_2}_{j,l,m} = (-1)^{d_1} 2^{jd} \Gamma^d_{m-l} \tag{7.2}$$

and we note the property

$$\Gamma^d_n = (-1)^d \Gamma^d_{-n}, \quad n \in [2-D, D-2] \tag{7.3}$$

which is obtained using the change of variable $x \leftarrow x - l$ in (7.1) followed by repeated integration by parts.

In the following we will restrict our attention to the problem of computing (7.1). The supports of ϕ and $\phi_l^{(d)}$ overlap only for $-(D-2) \le l \le D-2$ so there are $2D-3$ nonzero connection coefficients to be determined. Let

$$\mathbf{\Gamma}^d = \{\Gamma^d_l\}_{l=2-D}^{D-2}$$

and assume that $\phi \in C^d(\mathbf{R})$. Then taking the identity (3.21) with $j = 1$ and differentiating it d times leads to

$$\phi_l^{(d)}(x) = \sum_{k=0}^{D-1} a_k \phi_{1,2l+k}^{(d)}(x) = 2^d \sqrt{2} \sum_{k=0}^{D-1} a_k \phi_{2l+k}^{(d)}(2x) \tag{7.4}$$

Substituting the dilation equation (2.17) and (7.4) into (7.1) yields

$$\begin{aligned}
\Gamma^d_l &= \int_{-\infty}^{\infty} \left[\sqrt{2} \sum_{r=0}^{D-1} a_r \phi_r(2x) \right] \left[2^d \sqrt{2} \sum_{s=0}^{D-1} a_s \phi_{2l+s}^{(d)}(2x) \right] dx \\
&= 2^{d+1} \sum_{r=0}^{D-1} \sum_{s=0}^{D-1} a_r a_s \int_{-\infty}^{\infty} \phi_r(2x) \phi_{2l+s}^{(d)}(2x)\, dx, \qquad x \leftarrow 2x \\
&= 2^{d} \sum_{r=0}^{D-1} \sum_{s=0}^{D-1} a_r a_s \int_{-\infty}^{\infty} \phi_r(x) \phi_{2l+s}^{(d)}(x)\, dx, \qquad x \leftarrow x - r \\
&= 2^{d} \sum_{r=0}^{D-1} \sum_{s=0}^{D-1} a_r a_s \int_{-\infty}^{\infty} \phi(x) \phi_{2l+s-r}^{(d)}(x)\, dx
\end{aligned}$$

or

$$\sum_{r=0}^{D-1} \sum_{s=0}^{D-1} a_r a_s \Gamma^d_{2l+s-r} = \frac{1}{2^d} \Gamma^d_l, \quad l \in [2-D, D-2] \tag{7.5}$$

Let $n = 2l + s - r$. We know that Γ_n^d is nonzero only for $n \in [2-D, D-2]$ and that $s = r + n - 2l$ as well as r must be restricted to $[0, D-1]$. This is fulfilled for $\max(0, 2l-n) \leq r \leq \min(D-2, D-2+2l-n)$. Let $p = 2l - n$ and define

$$\overline{a}_p = \sum_{r=r_1(p)}^{r_2(p)} a_r a_{r-p}$$

where $r_1(p) = \max(0, p)$ and $r_2(p) = \min(D-1, D-1+p)$. Hence (7.5) becomes

$$\sum_{n=2-D}^{D-2} \overline{a}_{2l-n} \Gamma_n^d = \frac{1}{2^d} \Gamma_l^d, \quad l \in [2-D, D-2]$$

The matrix-vector form of this relation is

$$\boxed{\left(\boldsymbol{A} - 2^{-d} \boldsymbol{I}\right) \boldsymbol{\Gamma}^d = \boldsymbol{0}} \tag{7.6}$$

where $\boldsymbol{A}$ is a $(2D-3) \times (2D-3)$ matrix with the elements

$$[\boldsymbol{A}]_{l,n} = \overline{a}_{2l-n}, \quad l, n \in [2-D, D-2]$$

We note the following properties of $\overline{a}$:

- Because of the orthogonality property (2.22) we have

$$\overline{a}_p = \begin{cases} 1 & \text{for } p = 0 \\ 0 & \text{for } p = \pm 2, \pm 4, \pm 6 \ldots \end{cases}$$

- Also, $\overline{a}_p = \overline{a}_{-p}$ so we need only compute $\overline{a}_p$ for $p > 0$.

- Finally, a consequence of Theorem 2.3 is that

$$\sum_{p \text{ odd}} \overline{a}_p = 1$$

  Hence all columns add to one, which means that $\boldsymbol{A}$ has the left eigenvector $[1, 1, \ldots, 1]$ corresponding to the eigenvalue 1, $d = 0$ in (7.6).

Consequently, $\boldsymbol{A}$ has the structure shown here for $D = 6$:

$$\begin{bmatrix}
0 & \overline{a}_5 & & & & & & & \\
0 & \overline{a}_3 & 0 & \overline{a}_5 & & & & & \\
1 & \overline{a}_1 & 0 & \overline{a}_3 & 0 & \overline{a}_5 & & & \\
0 & \overline{a}_1 & 1 & \overline{a}_1 & 0 & \overline{a}_3 & 0 & \overline{a}_5 & \\
0 & \overline{a}_3 & 0 & \overline{a}_1 & 1 & \overline{a}_1 & 0 & \overline{a}_3 & 0 \\
 & \overline{a}_5 & 0 & \overline{a}_3 & 0 & \overline{a}_1 & 1 & \overline{a}_1 & 0 \\
 & & & \overline{a}_5 & 0 & \overline{a}_3 & 0 & \overline{a}_1 & 1 \\
 & & & & & \overline{a}_5 & 0 & \overline{a}_3 & 0 \\
 & & & & & & & \overline{a}_5 & 0
\end{bmatrix}$$

Equation (7.6) has a non-trivial solution if 2^{-d} is an eigenvalue of $\boldsymbol{A}$. Numerical calculations for $D = 4, 6, \ldots, 30$ indicate that 2^{-d} is an eigenvalue for $d = 0, 1, \ldots, D-1$ and that the dimension of each corresponding eigenspace is 1. Hence one additional equation is needed to normalize the solution.

To this end we use the property of vanishing moments. Recall that $P = D/2$. Assuming that $d < P$ we have from (2.25) that

$$x^d = \sum_{l=-\infty}^{\infty} M_l^d \phi(x-l)$$

Differentiating both sides of this relation d times yields

$$d! = \sum_{l=-\infty}^{\infty} M_l^d \phi^{(d)}(x-l)$$

Multiplying by $\phi(x)$ and integrating we then obtain

$$\begin{aligned} d! \int_{-\infty}^{\infty} \phi(x)\, dx &= \sum_{l=-\infty}^{\infty} M_l^d \int_{-\infty}^{\infty} \phi(x)\phi^{(d)}(x-l)\, dx \\ &= \sum_{l=2-D}^{D-2} M_l^d \int_{-\infty}^{\infty} \phi(x)\phi^{(d)}(x-l)\, dx \end{aligned}$$

Hence we get

$$\boxed{\sum_{l=2-D}^{D-2} M_l^d \Gamma_l^d = d!} \tag{7.7}$$

which closes the the system (7.6). The computation of the moments needed for this equation is described in Appendix (A). $\boldsymbol{\Gamma}^d$ is then found as follows: Let $\boldsymbol{v}^d$ be an eigenvector corresponding to the eigenvalue 2^{-d} in (7.6). Then $\boldsymbol{\Gamma}^d = k\boldsymbol{v}^d$ for some constant k, which is fixed according to (7.7).

The Matlab function `conn(`d`,` D`)` computes $\boldsymbol{\Gamma}^d$ for a given wavelet genus D.

Remark 7.1 *There is one exception to the statement that 2^{-d} is an eigenvalue of $\boldsymbol{A}$ for $d = 0, 1, \ldots, D-1$: Let $D = 4$. Then the eigenvalues of $\boldsymbol{A}$ are*

$$\frac{1}{8}, \quad \frac{1}{4} + 6.4765 \times 10^{-9} i, \quad \frac{1}{4} - 6.4765 \times 10^{-9} i, \quad \frac{1}{2}, \quad 1$$

Consequently $\frac{1}{4}$ is not an eigenvalue of $\boldsymbol{A}$ and connection coefficients for the combination $D = 4, d = 2$ are not well defined.

7.2.1 Differentiability

The question of differentiability of ϕ (and hence ψ) is nontrivial and not fully understood [SN96, p. 199–200], see e.g. [Dau92, p. 215–249] and [SN96, p. 242–245] for a discussion. However, some basic results are given in [Eir92] and stated in Table 7.1. The space $C^\alpha(\mathbf{R})$ denotes the space of functions having continuous

D	2	4	6	8	10	12	14	16	18	20
α	–	0	1	1	1	1	2	2	2	2
β	0	0	1	1	2	2	2	2	3	3

Table 7.1: Regularity of scaling functions and wavelets, where $\phi, \psi \in C^\alpha(\mathbf{R})$ and $\phi, \psi \in H^\beta(\mathbf{R})$.

derivatives of order $\leq \alpha$. The space $H^\beta(\mathbf{R})$ is a Sobolev space defined as

$$H^\beta(\mathbf{R}) = \{f \in L^2(\mathbf{R}) : f^{(d)} \in L^2(\mathbf{R}), \quad |d| \leq \beta\}$$

see, e.g. [Fol95, p. 190]. This latter concept is a generalization of ordinary differentiability, hence $\alpha \leq \beta$.

As we shall see numerical experiments reveal that the connection coefficients work for higher orders of differentiation than those specified in Table 7.1. See Section 7.4.

7.3 Differentiation matrix with respect to scaling functions

Let f be a function in $V_J \bigcap C^d(\mathbf{R})$, $J \in \mathbf{N}_0$. The connection coefficients described in the previous section can be used to evaluate the dth order derivative of f in terms of its scaling function coefficients. Differentiating both sides of (2.11) d times we obtain

$$f^{(d)}(x) = \sum_{l=-\infty}^{\infty} c_{J,l}\phi_{J,l}^{(d)}(x), \quad x \in \mathbf{R} \tag{7.8}$$

$f^{(d)}$ will in general not belong to V_J so we project $f^{(d)}$ back onto V_J

$$(P_{V_J} f^{(d)})(x) = \sum_{k=-\infty}^{\infty} c_{J,k}^{(d)}\phi_{J,k}(x), \quad x \in \mathbf{R} \tag{7.9}$$

where, according to (2.12),

$$c_{J,k}^{(d)} = \int_{-\infty}^{\infty} f^{(d)}(x)\phi_{J,k}(x)\,dx \tag{7.10}$$

Substituting (7.8) into (7.10) we find

$$\begin{aligned}
c_{J,k}^{(d)} &= \sum_{l=-\infty}^{\infty} c_{J,l} \int_{-\infty}^{\infty} \phi_{J,k}(x)\phi_{J,l}^{(d)}(x)\,dx \\
&= \sum_{l=-\infty}^{\infty} c_{J,l}\Gamma_{J,k,l}^{0,d} \\
&= \sum_{l=-\infty}^{\infty} c_{J,l} 2^{Jd}\Gamma_{l-k}^{d} \\
&= \sum_{n=-\infty}^{\infty} c_{J,n+k}\, 2^{Jd}\Gamma_{n}^{d}, \quad -\infty < k < \infty
\end{aligned}$$

We used (7.2) for the second last equality. Since Γ_n^d is only nonzero for $n \in [2-D, D-2]$ we find that

$$c_{J,k}^{(d)} = \sum_{n=2-D}^{D-2} c_{J,n+k} 2^{Jd}\Gamma_n^d, \quad J,k \in \mathbf{Z} \tag{7.11}$$

Recall from (2.60) that if f is 1-periodic then

$$c_{J,l} = c_{J,l+p2^J}, \quad l,p \in \mathbf{Z}$$

and

$$c_{J,k}^{(d)} = c_{J,k+p2^J}^{(d)}, \quad k,p \in \mathbf{Z}$$

Hence, it is sufficient to consider 2^J coefficients of either type and (7.11) becomes

$$c_{J,k}^{(d)} = \sum_{n=2-D}^{D-2} c_{J,\langle n+k\rangle_{2^J}} 2^{Jd}\Gamma_n^d, \quad k = 0,1,\ldots,2^J-1 \tag{7.12}$$

This system of equations can be represented in matrix-vector form

$$\boxed{\boldsymbol{c}^{(d)} = \boldsymbol{D}^{(d)}\boldsymbol{c}} \tag{7.13}$$

where

$$[\boldsymbol{D}^{(d)}]_{k,\langle n+k\rangle_{2^J}} = 2^{Jd}\Gamma_n^d, \qquad k = 0,1,\ldots,2^J-1, \\ n = 2-D, 3-D, \ldots, D-2$$

and $\boldsymbol{c}^{(d)} = [c_{J,0}^{(d)}, c_{J,1}^{(d)}, \ldots, c_{J,2^J-1}^{(d)}]$.

We will refer to the matrix $\boldsymbol{D}^{(d)}$ as the **differentiation matrix** of order d. It can be seen from (7.3) that $\boldsymbol{D}^{(d)}$ is symmetric for d even and skew-symmetric for d odd. Furthermore, it follows that $\boldsymbol{D}^{(d)}$ is circulant as defined in Definition C.1 and that it has bandwidth $2D-3$. The differentiation matrix has the following structure (shown for $D=4$ and $J=3$):

$$\boldsymbol{D}^{(d)} = 2^3 \begin{bmatrix} \Gamma_0^d & \Gamma_1^d & \Gamma_2^d & 0 & 0 & 0 & (-1)^d\Gamma_2^d & (-1)^d\Gamma_1^d \\ (-1)^d\Gamma_1^d & \Gamma_0^d & \Gamma_1^d & \Gamma_2^d & 0 & 0 & 0 & (-1)^d\Gamma_2^d \\ (-1)^d\Gamma_2^d & (-1)^d\Gamma_1^d & \Gamma_0^d & \Gamma_1^d & \Gamma_2^d & 0 & 0 & 0 \\ 0 & (-1)^d\Gamma_2^d & (-1)^d\Gamma_1^d & \Gamma_0^d & \Gamma_1^d & \Gamma_2^d & 0 & 0 \\ 0 & 0 & (-1)^d\Gamma_2^d & (-1)^d\Gamma_1^d & \Gamma_0^d & \Gamma_1^d & \Gamma_2^d & 0 \\ 0 & 0 & 0 & (-1)^d\Gamma_2^d & (-1)^d\Gamma_1^d & \Gamma_0^d & \Gamma_1^d & \Gamma_2^d \\ \Gamma_2^d & 0 & 0 & 0 & (-1)^d\Gamma_2^d & (-1)^d\Gamma_1^d & \Gamma_0^d & \Gamma_1^d \\ \Gamma_1^d & \Gamma_2^d & 0 & 0 & 0 & (-1)^d\Gamma_2^d & (-1)^d\Gamma_1^d & \Gamma_0^d \end{bmatrix}$$

An important special case is $d=1$, and we define

$$\boxed{\boldsymbol{D} = \boldsymbol{D}^{(1)}} \tag{7.14}$$

7.4 Differentiation matrix with respect to physical space

We will restrict our attention to the periodic case. Recall from Section 3.2.3 that

$$\boldsymbol{f} = \boldsymbol{T}\boldsymbol{c}, \quad \boldsymbol{c} = \boldsymbol{T}^{-1}\boldsymbol{f}$$

where $\boldsymbol{f}$ are the grid values of a function $f \in \tilde{V}_J$ defined on the unit interval. $\boldsymbol{c}$ is the vector of scaling function coefficients corresponding to f, and $\boldsymbol{T}$ is the matrix defined in (3.16) which maps from the scaling function coefficients onto the grid values. Similarly, the projection of $f^{(d)}$ onto $\tilde{V}_J$ satisfies

$$\boldsymbol{f}^{(d)} = \boldsymbol{T}\boldsymbol{c}^{(d)}$$

Then it follows from (7.13) that

$$\boldsymbol{f}^{(d)} = \boldsymbol{T}\boldsymbol{D}^{(d)}\boldsymbol{T}^{-1}\boldsymbol{f}$$

We call $\boldsymbol{D}^{(d)}$ the differentiation matrix with respect to the coefficient space, by virtue of (7.13), and $\boldsymbol{T}\boldsymbol{D}^{(d)}\boldsymbol{T}^{-1}$ the differentiation matrix with respect to physical space. The matrices $\boldsymbol{T}$ and $\boldsymbol{D}^{(d)}$ are both circulant with the same dimensions $(2D-3)\times(2D-3)$, so they are diagonalized by the *same* matrix, namely the Fourier matrix $\boldsymbol{F}_{2D-3}$ defined in (C.3). Therefore, they commute, according to Theorem C.5, and we find that

$$\boldsymbol{T}\boldsymbol{D}^{(d)}\boldsymbol{T}^{-1} = \boldsymbol{D}^{(d)}\boldsymbol{T}\boldsymbol{T}^{-1} = \boldsymbol{D}^{(d)}$$

and

$$\boxed{\boldsymbol{f}^{(d)} = \boldsymbol{D}^{(d)}\boldsymbol{f}} \tag{7.15}$$

Hence $\boldsymbol{D}^{(d)}$ is the differentiation matrix with respect to *both* coefficient space and physical space.

If f is an arbitrary 1-periodic function then $\boldsymbol{f}^{(d)}$ will generally not be exact due to approximation errors and we define

$$E^{(d)}(f,J) = \max_{k=0,1,\ldots,2^J-1} \left| \left[\boldsymbol{f}^{(d)}\right]_k - f^{(d)}\left(\frac{k}{2^J}\right) \right|$$

Jameson [Jam93] establishes the following convergence result for the differentiation matrix $\boldsymbol{D}^{(1)}$:

$$\boxed{f \in C^D(\mathbf{R}) \quad \Rightarrow \quad E^{(1)}(f,J) \le C2^{-JD}} \tag{7.16}$$

where C is a constant. Assume that similar results hold for higher orders of differentiation, i.e.

$$E^{(d)}(f,J) = C2^{-JR}, \quad R \in \mathbf{R}, \quad d \ge 1 \tag{7.17}$$

where R possibly depends on d and D. Taking the logarithm on both sides of (7.17) yields

$$\log_2(E^{(d)}(f,J)) = \log_2(C2^{-JR}) = \log_2(C) - JR$$

and we can find the convergence rate R from the differences

$$\log_2(E^{(d)}(f,J)) - \log_2(E^{(d)}(f,J+1)) = R$$

$D\backslash d$	1	2	3	4	5	6	7	8	9	10	11	12
4	4	–	2	–	–	–	–	–	–	–	–	–
6	6	4	4	2	2	–	–	–	–	–	–	–
8	8	6	6	4	4	2	2	–	–	–	–	–
10	10	8	8	6	6	4	4	2	2	–	–	–
12	12	10	10	8	8	6	6	4	4	2	2	–

Table 7.2: Convergence rates R (rounded) for the differentiation matrix shown for different values of D and orders of differentiation d. Here, $f(x) = \sin(4\pi x)$, and $J = 3, 4, \ldots, 8$. Note that the case $D = 4, d = 2$ is not valid as mentioned in Remark 7.1.

Numerical results are summarized in Table 7.2 and it is verified that (7.16) applies. For higher orders of differentiation, there is still convergence, but with lower convergence rates. We observe that R appears to be given by

$$R = D - 2\lfloor d/2 \rfloor, \quad d = 0, 1, \ldots, D-1$$

Moreover, we observe that numerical differentiation is successfully carried out to much higher orders than those suggested in Table 7.1.

The convergence rate $R = D$ for $d = 1$ can be achieved also for higher orders by redefining the differentiation process for $d > 1$. Let

$$\overline{\boldsymbol{f}}^{(d)} = \boldsymbol{D}^d \boldsymbol{f}$$

i.e. the dth derivative of f is approximated by repeated application of the first order differentiation matrix. Define

$$\overline{E}^{(d)}(f, J) = \max_{k=0,1,\ldots,2^J-1} \left| \left[\overline{\boldsymbol{f}}^{(d)} \right]_k - f^{(d)}\left(\frac{k}{2^J}\right) \right|$$

Then

$$\overline{E}^{(d)}(f, J) \leq C 2^{-J\overline{R}}, \quad \overline{R} \in \mathbf{R}$$

with $\overline{R} = D$. This is confirmed in Table 7.3.

Let $B(\boldsymbol{D}^d)$ denote the bandwidth of $\boldsymbol{D}^d$. We know from (7.13) that $B(\boldsymbol{D}) = 2D - 3$, and the general formula turns out to be:

$$B(\boldsymbol{D}^d) = \min(dB(\boldsymbol{D}) - d + 1, \ 2^J)$$

$D \backslash d$	1	2	3	4	5	6	7	8	9	10	11	12
4	4	4	4	4	4	4	4	4	4	4	4	4
6	6	6	6	6	6	6	6	6	6	6	6	6
8	8	8	8	8	8	8	8	8	8	8	8	8
10	10	10	10	10	10	10	10	10	10	10	10	10
12	12	12	12	12	12	12	12	12	12	12	12	12

Table 7.3: Convergence rates (rounded) for $\boldsymbol{D}^d$ shown for different values of D and orders of differentiation d. Again, $f(x) = \sin(4\pi x)$, and $J = 3, 4, \ldots, 8$.

$D \backslash d$	1	2	3	4	5	6	7	8	9	10	11	12
4	5	9	13	17	21	25	29	33	37	41	45	49
6	9	17	25	33	41	49	57	65	73	81	89	97
8	13	25	37	49	61	73	85	97	109	121	133	145
10	17	33	49	65	81	97	113	129	145	161	177	193
12	21	41	61	81	101	121	141	161	181	201	221	241

Table 7.4: Bandwidths of $\boldsymbol{D}^d$ shown for the values of d and D in Figure 7.3.

Hence the improved convergence for $d > 1$ comes at the cost of increased bandwidth. Table 7.4 gives the bandwidths corresponding to the convergence rates given in Table 7.3 assuming that J is sufficiently large.

The Matlab function `diftest(`D`)` computes the convergence rates for $\boldsymbol{D}^{(d)}$ and $\boldsymbol{D}^d$.

7.4.1 Differentiation matrix for functions with period L

If the function to be differentiated is periodic with period L,

$$f(x) = f(x + L), \quad x \in R$$

then we can map one period to the unit interval and apply the various transform matrices there as described in Section 3.2.4. Thus, let $y = x/L$ and define

$$g(y) \equiv f(Ly) = f(x)$$

Then g is 1-periodic and we define the vector $\boldsymbol{g} = [g_0, g_1, \ldots, g_{2^J-1}]$ by

$$g_k = (P_{\tilde{V}_J} g)(k/2^J), \quad k = 0, 1, \ldots, 2^J - 1$$

Let $\boldsymbol{f} = \boldsymbol{g}$. Then $\boldsymbol{f}$ approximates $f(x)$ at $x = k\,L/2^J$. From (7.15) we have

$$\boldsymbol{g}^{(d)} = \boldsymbol{D}^{(d)} \boldsymbol{g}$$

Hence, by the chain rule

$$f^{(d)}(x) = \frac{1}{L^d} g^{(d)}(y)$$

we have

$$\boldsymbol{f}^{(d)} = \frac{1}{L^d} \boldsymbol{g}^{(d)} = \frac{1}{L^d} \boldsymbol{D}^{(d)} \boldsymbol{g} = \frac{1}{L^d} \boldsymbol{D}^{(d)} \boldsymbol{f} \tag{7.18}$$

The Matlab function `difmatrix(`d`,`N`,`L`,`D`)` computes the differentiation matrix $\boldsymbol{D}^{(d)}/L^d$ for wavelet genus D and size N and period L.

7.5 Differentiation matrix with respect to wavelets

We will restrict our attention to the periodic case. Let f be a function in $\tilde{V}_J$. Proceeding as in the previous section, we differentiate both sides of (2.53) d times and obtain

$$f^{(d)}(x) = \sum_{l=0}^{2^{J_0}-1} c_{J_0,l} \tilde{\phi}^{(d)}_{J_0,l}(x) + \sum_{j=J_0}^{J-1} \sum_{l=0}^{2^j-1} d_{j,l} \tilde{\psi}^{(d)}_{j,l}(x) \tag{7.19}$$

Projecting $f^{(d)}$ onto $\tilde{V}_J$ yields

$$(P_{V_J} f^{(d)})(x) = \sum_{l=0}^{2^{J_0}-1} c^{(d)}_{J_0,l} \tilde{\phi}_{J_0,l}(x) + \sum_{j=J_0}^{J-1} \sum_{l=0}^{2^j-1} d^{(d)}_{j,l} \tilde{\psi}_{j,l}(x)$$

where

$$\begin{aligned} c^{(d)}_{J_0,l} &= \int_{-\infty}^{\infty} f^{(d)}(x) \tilde{\phi}_{J_0,l}(x)\, dx, & & l = 0, 1, \ldots, 2^{J_0} - 1 \\ d^{(d)}_{j,l} &= \int_{-\infty}^{\infty} f^{(d)}(x) \tilde{\psi}_{j,l}(x)\, dx, & & \begin{aligned} j &= J_0, J_0 + 1, \ldots, J - 1 \\ l &= 0, 1, \ldots, 2^j - 1 \end{aligned} \end{aligned} \tag{7.20}$$

Recall that given the scaling function coefficients of $f^{(d)}$ on the finest level, the FWT (3.33) can be used to obtain the wavelet coefficients above. Hence

$$\boldsymbol{d}^{(d)} = \boldsymbol{W}\boldsymbol{c}^{(d)}$$

where $\boldsymbol{c}^{(d)}$ is defined as in Section 7.3 and $\boldsymbol{d}^{(d)}$ contains the coefficients in (7.20). Using (7.13) and (3.35) we then obtain

$$\boldsymbol{d}^{(d)} = \boldsymbol{W}\boldsymbol{D}^{(d)}\boldsymbol{c} = \boldsymbol{W}\boldsymbol{D}^{(d)}\boldsymbol{W}^T\boldsymbol{d}$$

or

$$\boxed{\boldsymbol{d}^{(d)} = \breve{\boldsymbol{D}}^{(d)}\boldsymbol{d}} \tag{7.21}$$

where we have defined

$$\breve{\boldsymbol{D}}^{(d)} = \boldsymbol{W}\boldsymbol{D}^{(d)}\boldsymbol{W}^T \tag{7.22}$$

$\breve{\boldsymbol{D}}^{(d)}$ is the differentiation matrix with respect to the wavelet coefficients. We observe that $\breve{\boldsymbol{D}}^{(d)}$ is obtained as a 2D FWT of the differentiation matrix $\boldsymbol{D}^{(d)}$.

In contrast to $\boldsymbol{D}^{(d)}$, the matrix $\breve{\boldsymbol{D}}^{(d)}$ is not circulant. Instead, it has a characteristic finger band pattern as illustrated in Figure (7.1). As will be described in Chapter 8, one can take advantage of this structure to compute $\breve{\boldsymbol{D}}^{(d)}$ efficiently.

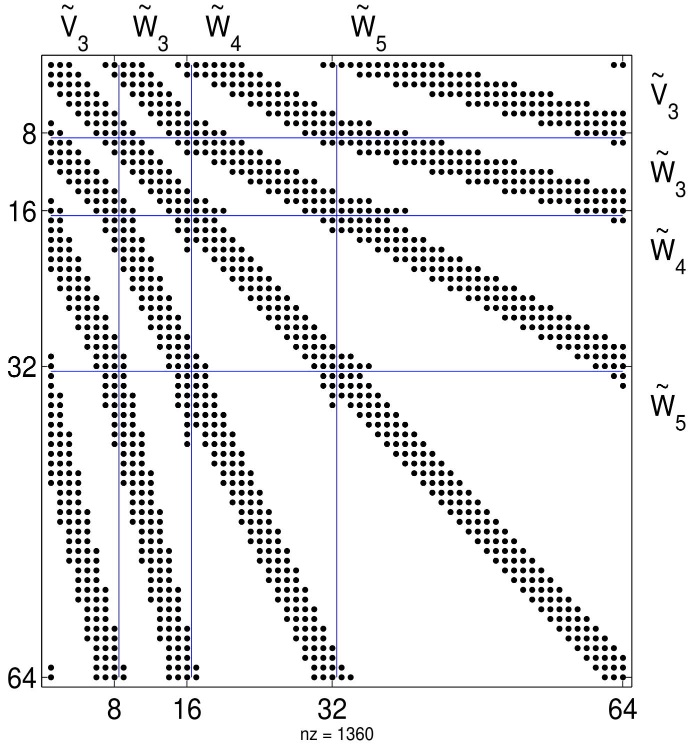


Figure 7.1: Sparsity pattern of $\breve{\boldsymbol{D}}^{(d)}$ when $D = 4$, $L = 1$, $J = 6$, $\lambda = 3$, $d = 1$. The pattern is identical for $d > 1$.

Chapter 8

2D Fast Wavelet Transform of a circulant matrix

In this chapter we will describe an algorithm for computing the 2D fast wavelet transform (2D FWT) of a circulant $N \times N$ matrix $\boldsymbol{A}$. The 2D FWT is defined in (3.38) and circulant matrices are discussed in Appendix C.

Recall that the 2D FWT is a mapping $\boldsymbol{A} \to \boldsymbol{H}$ given as

$$\boldsymbol{H} = \boldsymbol{W}\boldsymbol{A}\boldsymbol{W}^T$$

where $\boldsymbol{W}$ is defined as in (3.34). We will show that this can be done in $\mathcal{O}(N)$ steps and that $\boldsymbol{H}$ can be represented using $\mathcal{O}(N)$ elements in a suitable data structure. Using this structure we describe an efficient algorithm for computing the matrix-vector product

$$\boldsymbol{H}\boldsymbol{x}$$

where $\boldsymbol{x}$ is an arbitrary vector of length N. This algorithm also has complexity $\mathcal{O}(N)$ and will be used in Chapter 9 in a wavelet method for solving PDEs developed in 9.3.1.

This chapter is based on [Nie97] and the approach follows ideas proposed by Philippe Charton [Cha96]. However, many of the details, the data structure and the complexity analysis are, to our knowledge, presented here for the first time.

8.1 The wavelet transform revisited

Let N and λ be integers of the form given in Definition 3.1 and let $\boldsymbol{c}^i$ be a given vector with elements $[c^i_0, c^i_1, \dots, c^i_{N-1}]$ and let $\boldsymbol{d}^i$ be defined similarly.

Recall that the 1D FWT is defined (and computed) by the recurrence formulas from Definition 3.1:

$$
\begin{aligned}
c_m^{i+1} &= \sum_{k=0}^{D-1} a_k c^i_{\langle k+2m\rangle_{S_i}} \\
d_m^{i+1} &= \sum_{k=0}^{D-1} b_k c^i_{\langle k+2m\rangle_{S_i}}
\end{aligned}
$$

for $i = 0, 1, \dots, \lambda - 1$, $m = 0, 1, \dots, S_{i+1} - 1$, and $S_i = N/2^i$. Figure 8.1 illustrates how a vector is transformed.

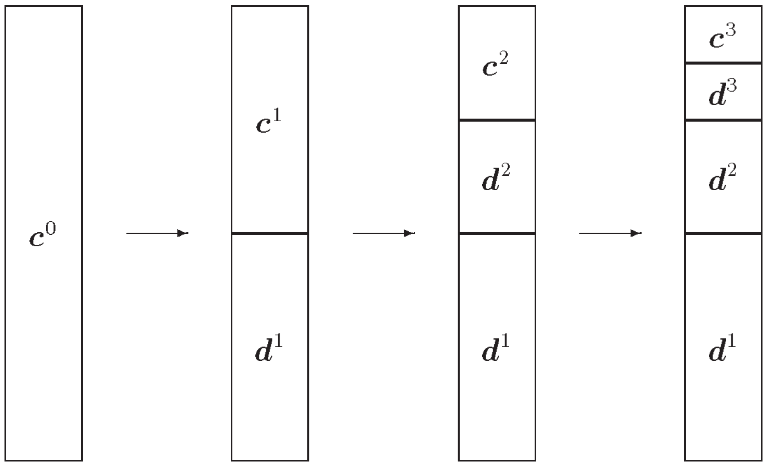


Figure 8.1: The steps of a 1D wavelet transform for $\lambda = 3$ and $J = 4$.

The 2D FWT defined in (3.38) has also a recursive formulation. The 2D recurrence formulas are straightforward generalizations of the 1D formulas, and they decompose a matrix analogously to the way the 1D recurrence formulas decompose a vector.

The recursion is initialized by assigning a matrix (assumed to be square for simplicity) $\boldsymbol{A} \in \mathbf{R}^{N,N}$ to the initial block[1] which we denote $\boldsymbol{CC}^{0,0}$. This block is then successively split into smaller blocks denoted $\boldsymbol{CC}^{i,j}$, $\boldsymbol{DC}^{i,j}$, $\boldsymbol{CD}^{i,j}$, and $\boldsymbol{DD}^{i,j}$ for $i, j = 1, 2, \dots, \lambda$. The block dimensions are determined by the superscripts i, j: A block with indices i, j has $S_i = N/2^i$ rows and $S_j = N/2^j$ columns.

[1]We point out that, throughout this chapter, we use two-character symbols for certain matrices and block-matrices.

The steps of the 2D wavelet transform for $\lambda = 3$ are shown in Figure 8.2 with $\boldsymbol{H}$ being the aggregation of all blocks after the final step shown in the upper right corner in the figure. (Comparison with the 1D wavelet transform shown in Figure 8.1 can be helpful for understanding this scheme). Note that each step of the transform produces blocks that have attained their final values, namely those of the type $\boldsymbol{DD}^{i,j}$, and subsequent steps work on blocks of the type $\boldsymbol{CC}^{j,j}$, $\boldsymbol{CD}^{i,j}$, and $\boldsymbol{DC}^{i,j}$. The formulas for these three types of blocks are given below.

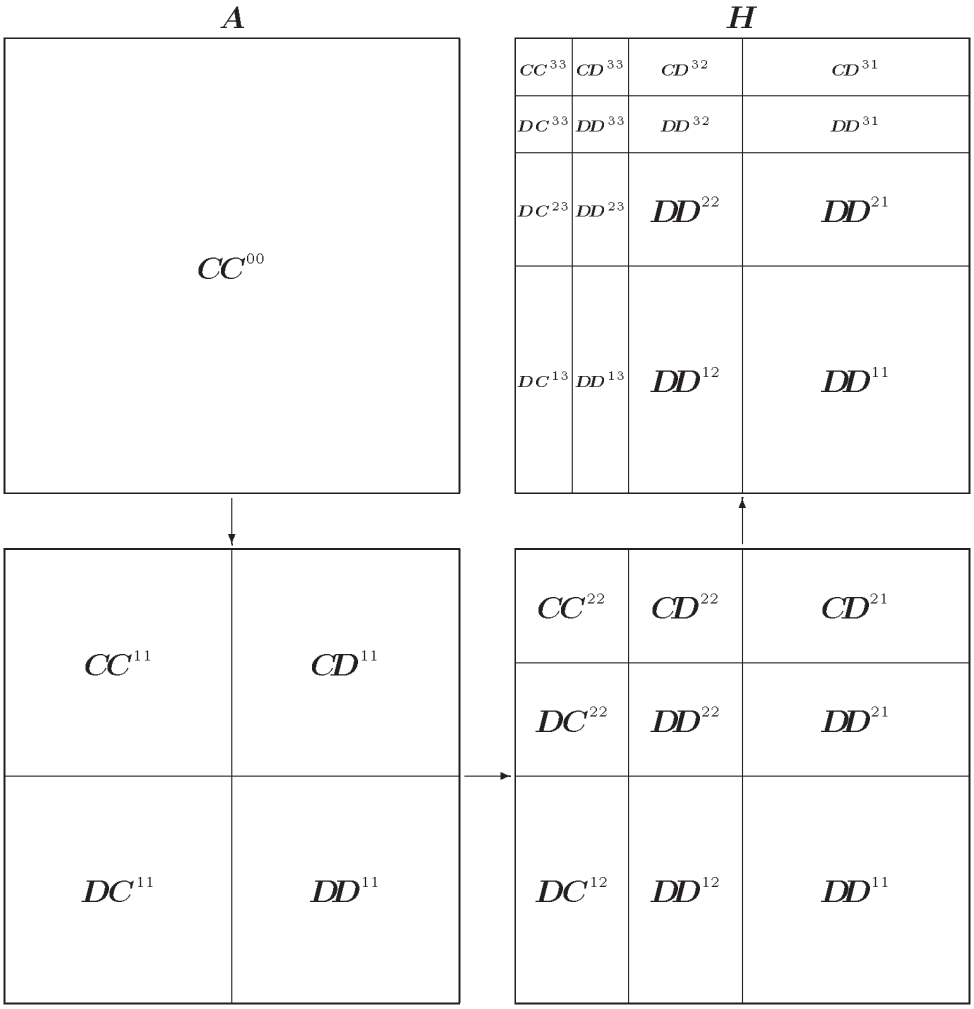


Figure 8.2: The steps of a 2D wavelet transform for $\lambda = 3$ and $J = 4$. The resulting matrix $\boldsymbol{H}$ has the characteristic "arrow-shaped" block structure.

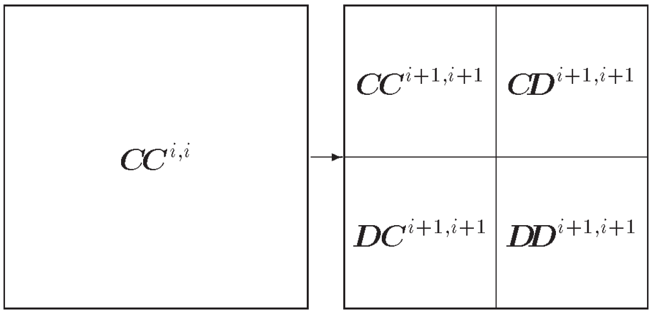


Figure 8.3: The transform of a square block on the diagonal yields four new square blocks. This operation corresponds to the decompositions given in (8.1) to (8.4).

Blocks on the diagonal

Consider the square blocks $\boldsymbol{CC}^{i,i}$ in Figure 8.2. Each step of the 2D wavelet transform splits such a block into four new blocks denoted $\boldsymbol{CC}^{i+1,i+1}$, $\boldsymbol{CD}^{i+1,i+1}$, $\boldsymbol{DC}^{i+1,i+1}$, and $\boldsymbol{DD}^{i+1,i+1}$. Figure 8.3 illustrates the decomposition of this type. Let $CC^{i,j}_{m,n} = [\boldsymbol{CC}^{i,j}]_{m,n}$ and similarly for $CD^{i+1}_{m,n}$, $DC^{i+1}_{m,n}$, and $DD^{i+1}_{m,n}$. The recurrence formulas for this decomposition are then given as follows:

$$CC^{i+1,i+1}_{m,n} = \sum_{k=0}^{D-1}\sum_{l=0}^{D-1} a_k a_l \, CC^{i,i}_{\langle k+2m\rangle_{S_i},\langle l+2n\rangle_{S_i}} \tag{8.1}$$

$$CD^{i+1,i+1}_{m,n} = \sum_{k=0}^{D-1}\sum_{l=0}^{D-1} a_k b_l \, CC^{i,i}_{\langle k+2m\rangle_{S_i},\langle l+2n\rangle_{S_i}} \tag{8.2}$$

$$DC^{i+1,i+1}_{m,n} = \sum_{k=0}^{D-1}\sum_{l=0}^{D-1} b_k a_l \, CC^{i,i}_{\langle k+2m\rangle_{S_i},\langle l+2n\rangle_{S_i}} \tag{8.3}$$

$$DD^{i+1,i+1}_{m,n} = \sum_{k=0}^{D-1}\sum_{l=0}^{D-1} b_k b_l \, CC^{i,i}_{\langle k+2m\rangle_{S_i},\langle l+2n\rangle_{S_i}} \tag{8.4}$$

for $i = 0, 1, \ldots, \lambda - 1$ and $m, n = 0, 1, \ldots, S_{i+1} - 1$.

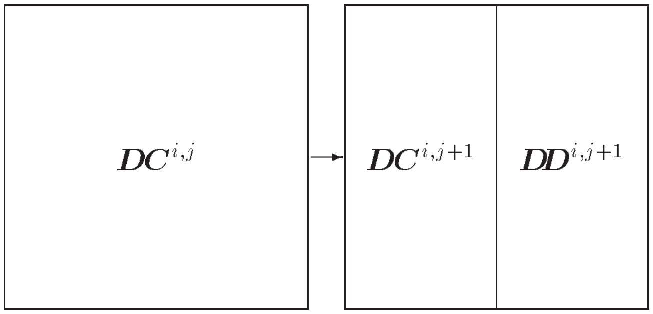


Figure 8.4: Transform of a block below the diagonal yields two new rectangular blocks. This operation corresponds to the row transforms given in (8.5) and (8.6).

Blocks below the diagonal

Blocks of the type $\boldsymbol{DC}^{i,j}$ ($j \geq i$) are split in *one* direction only as indicated in Figure 8.4. The recurrence formulas are the 1D formulas applied to each row of the block $\boldsymbol{DC}^{i,j}$:

$$DC^{i,j+1}_{m,n} = \sum_{l=0}^{D-1} a_l \, DC^{i,j}_{m,\langle l+2n \rangle_{S_j}} \tag{8.5}$$

$$DD^{i,j+1}_{m,n} = \sum_{l=0}^{D-1} b_l \, DC^{i,j}_{m,\langle l+2n \rangle_{S_j}} \tag{8.6}$$

for $j = i, i+1, \ldots, \lambda - 1$, $m = 0, 1, \ldots, S_i - 1$, and $n = 0, 1, \ldots, S_{j+1} - 1$.

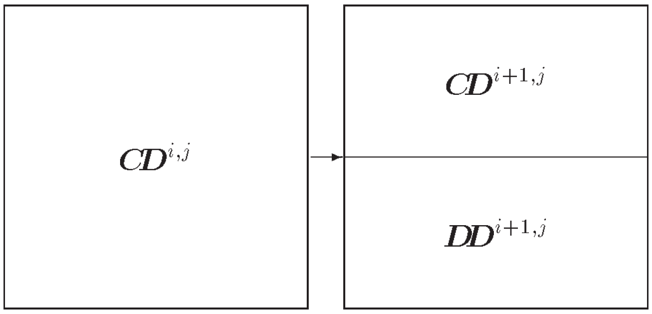


Figure 8.5: Transform of a block above the diagonal yields two new rectangular blocks. This operation corresponds to the column transforms given in (8.7) and (8.8).

Blocks above the diagonal

For blocks $\boldsymbol{CD}^{i,j}$ with $i \geq j$ we have a splitting as shown in Figure 8.5. The recurrence formulas are the 1D formulas applied to each column of $\boldsymbol{CD}^{i,j}$:

$$CD^{i+1,j}_{m,n} = \sum_{k=0}^{D-1} a_k \, CD^{i,j}_{\langle k+2m \rangle_{S_i},n} \tag{8.7}$$

$$DD^{i+1,j}_{m,n} = \sum_{k=0}^{D-1} b_k \, CD^{i,j}_{\langle k+2m \rangle_{S_i},n} \tag{8.8}$$

for $i = j, j+1, \ldots, \lambda - 1$, $m = 0, 1, \ldots, S_{i+1} - 1$, and $n = 0, 1, \ldots, S_j - 1$.

8.2 2D wavelet transform of a circulant matrix

The 2D FWT of circulant matrices give rise to structured matrix blocks which we will call **shift-circulant matrices**. We begin by giving the definition.

Let A be an $M \times M$ circulant matrix as defined in Definition C.1 and let $\{a_m\}_{m=0,1,\dots,M-1}$ be the first column of A. Then

$$[\boldsymbol{A}]_{m,n} = a_{\langle m-n \rangle_M}, \quad m, n = 0, 1, \dots, M-1$$

A shift-circulant matrix is a generalization of a circulant matrix which we define as follows

Definition 8.1 (Shift-circulant matrix)

1. *Let* $\boldsymbol{A}$ *be an* $M \times N$ *matrix where* $M \geq N$ *with* M *divisible by* N, *and let* $\{a_m\}_{m=0,1,\dots,M-1}$ *be the first* column *of* $\boldsymbol{A}$. *Then* $\boldsymbol{A}$ *is* **column-shift-circulant** *if*

$$[\boldsymbol{A}]_{m,n} = a_{\langle m-\sigma n \rangle_M}, \quad m = 0, 1, \dots, M-1, \; n = 0, 1, \dots, N-1$$

   where $\sigma = M/N$.

2. *Let* $\boldsymbol{A}$ *be an* $M \times N$ *matrix where* $N \geq M$ *with* N *divisible by* M, *and let* $\{a_n\}_{n=0,1,\dots,N-1}$ *be the first* row *of* $\boldsymbol{A}$. *Then* $\boldsymbol{A}$ *is* **row-shift-circulant** *if*

$$[\boldsymbol{A}]_{m,n} = a_{\langle n-\sigma m \rangle_N}, \quad m = 0, 1, \dots, M-1, \; n = 0, 1, \dots, N-1$$

   where $\sigma = N/M$.

The number σ *is a positive integer that denotes the amount by which columns or rows are shifted.*

A column-shift-circulant 4×2 matrix ($\sigma = 2$) has the form

$$\begin{pmatrix} a_0 & a_2 \\ a_1 & a_3 \\ a_2 & a_0 \\ a_3 & a_1 \end{pmatrix}$$

A row-shift-circulant 2×4 matrix ($\sigma = 2$) has the form

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_2 & a_3 & a_0 & a_1 \end{pmatrix}$$

Note that a circulant matrix is both column-shift-circulant and row-shift-circulant with $\sigma = 1$.

Let $\{a_n\}_{n=0,1,\ldots,N-1}$ be the first column of a circulant matrix $\boldsymbol{A}$. Using this column vector as a point of departure we will now show how to compute a representation of $\boldsymbol{H}$ using only one vector per block. Note that according to the recurrence equations (8.1) to (8.8), the operations can be divided into 2D transforms of blocks on the diagonal and 1D row or column transforms of off-diagonal blocks. We will treat these cases separately.

8.2.1 Blocks on the diagonal

Lemma 8.1 *Let $\boldsymbol{CC}^{i,i}$ be a $S_i \times S_i$ circulant matrix. Then $\boldsymbol{CC}^{i+1,i+1}$, $\boldsymbol{CD}^{i+1,i+1}$, $\boldsymbol{DC}^{i+1,i+1}$, and $\boldsymbol{DD}^{i+1,j+1}$ defined by (8.1) to (8.4), respectively, are circulant matrices.*

Proof: We will prove the lemma for $\boldsymbol{CD}^{i+1,i+1}$ only since the other cases are completely analogous.

By assumption $\boldsymbol{CC}^{i,i}$ is circulant, i.e.

$$CC^{i,i}_{m,n} = cc^{i,i}_{\langle m-n\rangle_{S_i}}$$

where $\boldsymbol{cc}^{i,i}$ is the first column of $\boldsymbol{CC}^{i,i}$. Equation (8.2) then becomes

$$CD^{i+1,i+1}_{m,n} = \sum_{k=0}^{D-1}\sum_{l=0}^{D-1} a_k b_l \, cc^{i,i}_{\langle\langle k+2m\rangle_{S_i} - \langle l+2n\rangle_{S_i}\rangle_{S_i}} \tag{8.9}$$

Considering now the index of the typical term of (8.9) and using Lemma B.1 and B.2 derived in Appendix B we find that

$$\begin{aligned}
\langle\langle k+2m\rangle_{S_i} - \langle l+2n\rangle_{S_i}\rangle_{S_i} &= \langle\langle 2(m-n)\rangle_{S_i} + k - l\rangle_{S_i} \\
&= \langle 2\langle m-n\rangle_{S_i/2} + k - l\rangle_{S_i} \\
&= \langle 2\langle m-n\rangle_{S_{i+1}} + k - l\rangle_{S_i}
\end{aligned}$$

This expression does not depend on the individual values of m and n but only on their *difference*. Therefore, $CD^{i+1,i+1}_{m,n}$ depends only on $\langle m-n\rangle_{S_{i+1}}$ which proves that $\boldsymbol{CD}^{i+1,i+1}$ is circulant. Hence $CD^{i+1,i+1}_{m,n} = cd^{i+1,i+1}_{\langle m-n\rangle_{S_{i+1}}}$ for some column vector $\boldsymbol{cd}^{i+1,i+1}$. □

Having established that $\boldsymbol{CD}^{i+1,i+1}$ is circulant, we can now give a formula for the vector $\boldsymbol{cd}^{i+1,i+1}$. Putting $n = 0$ in equation (8.9) and using Lemma B.1 we obtain the first column of $\mathbf{CD}^{i+1,i+1}$:

$$
\begin{aligned}
cd_m^{i+1,i+1} = CD_{m,0}^{i+1,i+1} &= \sum_{k=0}^{D-1}\sum_{l=0}^{D-1} a_k b_l \, cc^{i,i}_{\langle\langle 2m+k\rangle_{S_i} - \langle l\rangle_{S_i}\rangle_{S_i}} \\
&= \sum_{k=0}^{D-1}\sum_{l=0}^{D-1} a_k b_l \, cc^{i,i}_{\langle 2m+k-l\rangle_{S_i}}
\end{aligned}
\tag{8.10}
$$

where $m = 0, 1, \ldots, S_{i+1} - 1$. A computationally more efficient expression for $cd_m^{i+1,i+1}$ can be derived by rearranging the terms in (8.10) so that we sum over the *differences* in the indices. More precisely, we split the double sum into terms where $k - l \geq 0$ and terms where $k - l < 0$ and rearrange these separately. Thus let $p = k - l$. The terms in (8.10) with $k - l \geq 0$ are

$$
\sum_{k=0}^{D-1}\sum_{l=0}^{k} a_k b_l \, cc^{i,i}_{\langle 2m+k-l\rangle_{S-i}} = \sum_{k=0}^{D-1}\sum_{l=0}^{k} a_{l+p} b_l \, cc^{i,i}_{\langle 2m+p\rangle_{S_i}} \tag{8.11}
$$

The following table shows p as a function of k and l (for $D = 6$):

$k\backslash l$	0	1	2	3	4	5
0	0					
1	1	0				
2	2	1	0			
3	3	2	1	0		
4	4	3	2	1	0	
5	5	4	3	2	1	0

Summing over the diagonals yields an alternative expression for (8.11):

$$
\sum_{k=0}^{D-1}\sum_{l=0}^{k} a_{l+p} b_l \, cc^{i,i}_{\langle 2m+p\rangle_{S_i}} = \sum_{p=0}^{D-1}\left[\sum_{l=0}^{D-1-p} a_{l+p} b_l\right] cc^{i,i}_{\langle 2m+p\rangle_{S_i}} \tag{8.12}
$$

Similarly, we take the terms from (8.10) with $k - l < 0$ and set $p = l - k > 0$:

$$
\sum_{k=0}^{D-1}\sum_{l=k+1}^{D-1} a_k b_{k+p} \, cc^{i,i}_{\langle 2m-p\rangle_{S_i}} = \sum_{p=1}^{D-1}\left[\sum_{k=0}^{D-1-p} a_k b_{k+p}\right] cc^{i,i}_{\langle 2m-p\rangle_{S_i}} \tag{8.13}
$$

Now let

$$
q_{ab}^{p} = \sum_{k=0}^{D-1-p} a_k b_{k+p}, \quad p = 0, 1, \ldots, D-1
$$

Combining (8.12) and (8.13) we can rewrite (8.10) as

$$\begin{aligned} cd_m^{i+1,i+1} &= \\ & q_{ab}^0 cc_{2m}^{i,i} + \sum_{p=1}^{D-1} \left[q_{ab}^p \, cc_{\langle 2m-p \rangle_{S_i}}^{i,i} + q_{ba}^p \, cc_{\langle 2m+p \rangle_{S_i}}^{i,i} \right] \end{aligned} \tag{8.14}$$

for $m = 0, 1, \ldots, S_{i+1} - 1$.

The vectors $\boldsymbol{cc}^{i+1,i+1}$, $\boldsymbol{dc}^{i+1,i+1}$, and $\boldsymbol{dd}^{i+1,i+1}$ are computed similarly but with the filters q_{aa}^p, q_{ba}^p, q_{bb}^p, respectively. The formulas are

$$\begin{aligned} cc_m^{i+1,i+1} &= \\ & q_{aa}^0 cc_{2m}^{i,i} + \sum_{p=1}^{D-1} \left[q_{aa}^p \, cc_{\langle 2m-p \rangle_{S_i}}^{i,i} + q_{aa}^p \, cc_{\langle 2m+p \rangle_{S_i}}^{i,i} \right] \end{aligned} \tag{8.15}$$

$$\begin{aligned} dc_m^{i+1,i+1} &= \\ & q_{ba}^0 cc_{2m}^{i,i} + \sum_{p=1}^{D-1} \left[q_{ba}^p \, cc_{\langle 2m-p \rangle_{S_i}}^{i,i} + q_{ab}^p \, cc_{\langle 2m+p \rangle_{S_i}}^{i,i} \right] \end{aligned} \tag{8.16}$$

$$\begin{aligned} dd_m^{i+1,i+1} &= \\ & q_{bb}^0 cc_{2m}^{i,i} + \sum_{p=1}^{D-1} \left[q_{bb}^p \, cc_{\langle 2m-p \rangle_{S_i}}^{i,i} + q_{bb}^p \, cc_{\langle 2m+p \rangle_{S_i}}^{i,i} \right] \end{aligned} \tag{8.17}$$

It follows from (2.22) that

$$\begin{aligned} q_{aa}^0 = q_{bb}^0 &= 1 \\ q_{ab}^p = q_{ba}^p &= 0, \quad p \text{ even} \\ q_{aa}^p = q_{bb}^p &= 0, \quad p \text{ even}, \quad p > 0 \end{aligned}$$

so the computational work in computing (8.14)–(8.17) is reduced accordingly.

8.2.2 Blocks below the diagonal

We now turn to the task of computing the *lower* off-diagonal blocks $\boldsymbol{DC}^{i,j+1}$ and $\boldsymbol{DD}^{i,j+1}$, $j \geq i$ defined by (8.5) and (8.6), respectively. The operations differ from those of the diagonal cases by being applied in one dimension only. Therefore blocks tend to become more rectangular which means that they are not necessarily circulant in the ordinary sense. However, as we shall see, the typical block is still represented by a single vector, one which is shifted to match the rectangular shape. The block is then a *column-shift-circulant* matrix in the sense of Definition 8.1.

Lemma 8.2 *Let $\boldsymbol{DC}^{i,j}$, $j \geq i$, be a $S_i \times S_j$ column-shift-circulant matrix. Then $\boldsymbol{DC}^{i,j+1}$ and $\boldsymbol{DD}^{i,j+1}$ defined by (8.5) and (8.6), respectively, are column-shift-circulant matrices.*

Proof: We will give the proof for the case of $\boldsymbol{DC}^{i,j}$ only. By assumption $\boldsymbol{DC}^{i,j}$ is column-shift-circulant, i.e.

$$DC^{i,j}_{m,n} = dc^{i,j}_{\langle m-\sigma n\rangle_{S_i}}$$

where $\boldsymbol{dc}^{i,j}$ is the first column of $\boldsymbol{DC}^{i,j}$ and $\sigma = S_i/S_j = 2^{j-i}$. Equation (8.5) then becomes

$$\begin{aligned} DC^{i,j+1}_{m,n} &= \sum_{l=0}^{D-1} a_l \, DC^{i,j}_{m,\langle l+2n\rangle_{S_j}} \\ &= \sum_{l=0}^{D-1} a_l \, dc^{i,j}_{\langle m-\sigma\langle l+2n\rangle_{S_j}\rangle_{S_i}} \end{aligned} \tag{8.18}$$

We consider the index of the typical term of (8.18) and use Lemmas B.2 and B.1 in Appendix B to obtain the following:

$$\begin{aligned} \langle m-\sigma\langle l+2n\rangle_{S_j}\rangle_{S_i} &= \langle m-\langle \sigma(l+2n)\rangle_{\sigma S_j}\rangle_{S_i} \\ &= \langle m-\langle \sigma l+2\sigma n\rangle_{S_i}\rangle_{S_i} \\ &= \langle m-\sigma l-2\sigma n\rangle_{S_i} \end{aligned}$$

Therefore

$$DC^{i,j+1}_{m,n} = \sum_{l=0}^{D-1} a_l \, dc^{i,j}_{\langle m-\sigma l-2\sigma n\rangle_{S_i}} \tag{8.19}$$

Equation (8.19) establishes the existence of a vector, $\boldsymbol{dc}^{i,j+1}$ say, such that $\boldsymbol{DC}^{i,j+1}$ has the desired column-shift-circulant form

$$DC^{i,j+1}_{m,n} = dc^{i,j+1}_{\langle m-2\sigma n\rangle_{S_i}}$$

for $m = 0, 1, \ldots, S_i - 1$ and $n = 0, 1, \ldots, S_{j+1} - 1$. □

We can now look for an explicit formula for the vector $\boldsymbol{dc}^{i,j+1}$. Taking the first column ($n = 0$) of $DC^{i,j+1}_{m,n}$ in (8.19) gives the result:

$$dc^{i,j+1}_m = DC^{i,j+1}_{m,0} = \sum_{l=0}^{D-1} a_l \, dc^{i,j}_{\langle m-\sigma l\rangle_{S_i}}, \quad m = 0, 1, \ldots, S_i - 1 \tag{8.20}$$

An analysis similar to the above establishes that $\boldsymbol{DD}^{i,j+1}$ is column-shift-circulant with respect to the vector $\boldsymbol{dd}^{i,j+1}$ given by

$$dd_m^{i,j+1} = DD_{m,0}^{i,j+1} = \sum_{l=0}^{D-1} b_l \, dc_{\langle m-\sigma l\rangle_{S_i}}^{i,j}, \; m = 0, 1, \ldots, S_i - 1 \tag{8.21}$$

Since the initial block $\boldsymbol{DC}^{i,i}$ is circulant according to Lemma 8.1, it is also column-shift-circulant with $\sigma = S_i/S_i = 1$ and we can use the column vector $\boldsymbol{dc}^{i,i}$ as computed from (8.16) directly in (8.20) and (8.21) for the case $i = j$.

8.2.3 Blocks above the diagonal

The *upper* off-diagonal blocks $\boldsymbol{CD}^{i+1,j}$ and $\boldsymbol{DD}^{i+1,j}$, $i \geq j$, are computed according to (8.7) and (8.8). The situation is completely analogous to (8.20) and (8.21) but the blocks are now *row-shift-circulant* matrices represented by row vectors $\boldsymbol{cd}^{i+1,j}$ and $\boldsymbol{dd}^{i+1,j}$, respectively, as stated by Lemma 8.3.

Lemma 8.3 *Let $\boldsymbol{CD}^{i,j}$, $i \geq j$, be a $S_i \times S_j$ row-shift-circulant matrix. Then $\boldsymbol{CD}^{i+1,j}$ and $\boldsymbol{DD}^{i+1,j}$ defined by (8.7) and (8.8), respectively, are row-shift-circulant matrices.*

Proof: We will give the proof for the case of $\boldsymbol{CD}^{i,j}$ only. The proof is completely similar to that of Lemma 8.2, but the assumption is now that $\boldsymbol{CD}^{i,j}$ is row-shift-circulant, i.e.

$$CD_{m,n}^{i,j} = cd_{\langle n-\sigma m\rangle_{S_j}}^{i,j}$$

where $\boldsymbol{cd}^{i,j}$ is the first row of $\boldsymbol{CD}^{i,j}$ and $\sigma = S_j/S_i = 2^{i-j}$. Equation (8.7) then becomes

$$\begin{aligned} CD_{m,n}^{i+1,j} &= \sum_{k=0}^{D-1} a_k \, CD_{\langle k+2m\rangle_{S_i},n}^{i,j} \\ &= \sum_{k=0}^{D-1} a_k \, cd_{\langle n-\sigma\langle k+2m\rangle_{S_i}\rangle_{S_j}}^{i,j} \\ &= \sum_{k=0}^{D-1} a_k \, cd_{\langle n-\sigma k-2\sigma m\rangle_{S_j}}^{i,j} \end{aligned}$$

Therefore $\boldsymbol{CD}^{i+1,j}$ has the desired row-shift-circulant form

$$CD_{m,n}^{i+1,j} = cd_{\langle n-2\sigma m\rangle_{S_j}}^{i+1,j}$$

for $m = 0, 1, \ldots, S_{i+1} - 1$ and $n = 0, 1, \ldots, S_j - 1$. $\square$

The formulas for the row vectors $\boldsymbol{cd}^{i+1,j}$ and $\boldsymbol{dd}^{i+1,j}$ follow by taking the first *rows* ($m = 0$) of $\boldsymbol{CD}^{i+1,j}_{m,n}$ and $\boldsymbol{DD}^{i+1,j}_{m,n}$, respectively:

$$cd^{i+1,j}_n = CC^{i+1,j}_{0,n} = \sum_{k=0}^{D-1} a_k \, cd^{i,j}_{\langle n-\sigma k\rangle_{S_j}}, \; n = 0, 1, \dots, S_j - 1 \tag{8.22}$$

$$dd^{i+1,j}_n = DD^{i+1,j}_{0,n} = \sum_{k=0}^{D-1} b_k \, cd^{i,j}_{\langle n-\sigma k\rangle_{S_j}} \; n = 0, 1, \dots, S_j - 1 \tag{8.23}$$

However, one minor issue remains to be dealt with before a viable algorithm can be established: While the initial blocks $\boldsymbol{CD}^{i,i}$ defined according to (8.2) are circulant and therefore also row-shift-circulant (with $\sigma = 1$), they are represented by column vectors when computed according to equation (8.14). However, (8.22) and (8.23) work with a row vector $\boldsymbol{cd}^{i,j}$. Therefore we must modify each $\boldsymbol{cd}^{i,i}$ so that it represents the first *row* of $\boldsymbol{CD}^{i,i}$ instead of the first *column*.
From Definition C.1 of a circulant matrix we have that

$$CD^{i,i}_{m,n} = cd^{i,i}_{\langle m-n\rangle_{S_i}}, \;\; n = 0, 1, \dots, S_i - 1$$

where $\boldsymbol{cd}^{i,i}$ is the first column of $\boldsymbol{CD}^{i,i}$. Putting $m = 0$ then yields the first row:

$$CD^{i,i}_{0,n} = cd^{i,i}_{\langle -n\rangle_{S_i}}, \;\; n = 0, 1, \dots, S_i - 1$$

To obtain a row representation for $\boldsymbol{CD}^{i,i}$ we can therefore take the result from equation (8.14) and convert it as follows

$$cd^{i,i}_n \leftarrow cd^{i,i}_{\langle -n\rangle_{S_i}}$$

Alternatively, we can modify equation (8.14) to produce the row vector directly:

$$cd^{i+1,i+1}_n = q^0_{ab} cc^{i,i}_{\langle -2n\rangle_{S_i}} + \sum_{p=1}^{D-1} \left[q^p_{ab} \, cc^{i,i}_{\langle -2n-p\rangle_{S_i}} + q^p_{ba} \, cc^{i,i}_{\langle -2n+p\rangle_{S_i}} \right] \tag{8.24}$$

for $n = 0, 1, \dots, S_{i+1} - 1$.

The equations for blocks above the diagonal ((8.22) and (8.23)) and equations for blocks below the diagonal ((8.20) and (8.21)) can now be computed with the same algorithm.

8.2.4 Algorithm

We will now state an algorithm for the 2D wavelet transform of a circulant matrix $\boldsymbol{A}$. Let CIRPWT1 be a function that implements 2D decompositions of blocks on the diagonal according to (8.15), (8.16), (8.17), and (8.24):

$$[\boldsymbol{cc}^{i+1,i+1}, \boldsymbol{cd}^{i+1,i+1}, \boldsymbol{dc}^{i+1,i+1}, \boldsymbol{dd}^{i+1,i+1}] = \text{CIRPWT1}(\boldsymbol{cc}^{i,i})$$

Moreover, let CIRPWT2 be a function that implements 1D decompositions of the form described in equations (8.20) and (8.21). This function can also be used for computations of (8.22) and (8.23) as mentioned above.

$$\begin{aligned}
[\boldsymbol{dc}^{i,j+1}, \boldsymbol{dd}^{i,j+1}] &= \text{CIRPWT2}(\boldsymbol{dc}^{i,j}),\ j \geq i \\
[\boldsymbol{cd}^{i+1,j}, \boldsymbol{dd}^{i+1,j}] &= \text{CIRPWT2}(\boldsymbol{cd}^{i,j}),\ i \geq j
\end{aligned}$$

With these functions the example shown in Figure 8.2 can be computed as follows: Let $\boldsymbol{cc}^{0,0}$ be the column vector representing the initial circulant matrix $\boldsymbol{CC}^{0,0}$. Then

$$\begin{aligned}
[\boldsymbol{cc}^{1,1}, \boldsymbol{cd}^{1,1}, \boldsymbol{dc}^{1,1}, \boldsymbol{dd}^{1,1}] &= \text{CIRPWT1}(\boldsymbol{cc}^{0,0}) \\
\\
[\boldsymbol{cc}^{2,2}, \boldsymbol{cd}^{2,2}, \boldsymbol{dc}^{2,2}, \boldsymbol{dd}^{2,2}] &= \text{CIRPWT1}(\boldsymbol{cc}^{1,1}) \\
[\boldsymbol{dc}^{1,2}, \boldsymbol{dd}^{1,2}] &= \text{CIRPWT2}(\boldsymbol{dc}^{1,1}) \\
[\boldsymbol{cd}^{2,1}, \boldsymbol{dd}^{2,1}] &= \text{CIRPWT2}(\boldsymbol{cd}^{1,1}) \\
\\
[\boldsymbol{cc}^{3,3}, \boldsymbol{cd}^{3,3}, \boldsymbol{dc}^{3,3}, \boldsymbol{dd}^{3,3}] &= \text{CIRPWT1}(\boldsymbol{cc}^{2,2}) \\
[\boldsymbol{dc}^{2,3}, \boldsymbol{dd}^{2,3}] &= \text{CIRPWT2}(\boldsymbol{dc}^{2,2}) \\
[\boldsymbol{cd}^{3,2}, \boldsymbol{dd}^{3,2}] &= \text{CIRPWT2}(\boldsymbol{cd}^{2,2}) \\
[\boldsymbol{dc}^{1,3}, \boldsymbol{dd}^{1,3}] &= \text{CIRPWT2}(\boldsymbol{dc}^{1,2}) \\
[\boldsymbol{cd}^{3,1}, \boldsymbol{dd}^{3,1}] &= \text{CIRPWT2}(\boldsymbol{cd}^{2,1})
\end{aligned}$$

In general, the algorithm is

For $i = 0, 1, \ldots, \lambda - 1$
    $[\boldsymbol{cc}^{i+1,i+1}, \boldsymbol{cd}^{i+1,i+1}, \boldsymbol{dc}^{i+1,i+1}, \boldsymbol{dd}^{i+1,i+1}] = \text{CIRPWT1}(\boldsymbol{cc}^{i,i})$
    For $j = i, i-1, \ldots, 1$
        $[\boldsymbol{dc}^{j,i+1}, \boldsymbol{dd}^{j,i+1}] = \text{CIRPWT2}(\boldsymbol{dc}^{j,i})$
        $[\boldsymbol{cd}^{i+1,j}, \boldsymbol{dd}^{i+1,j}] = \text{CIRPWT2}(\boldsymbol{cd}^{i,j})$
    end
end

This algorithm describes the process of computing the 2D wavelet transform as described in the previous section. However, it does not distinguish among vectors that should be kept in the final result and vectors that are merely intermediate stages of the transform. The vectors $\boldsymbol{dc}^{j,i}$ and $\boldsymbol{cd}^{i,j}$, for example, are part of the final result for $i = \lambda$ only, so all other vectors can be discarded at some point.

In practice we prefer an algorithm that makes explicit use of a *fixed* storage area and that does not store unnecessary information. Therefore, we will introduce a modified notation that is suitable for such an algorithm and also convenient for the matrix-vector multiplication which is described in Section 8.4.

8.2.5 A data structure for the 2D wavelet transform

As described in Section 8.1 the result $\boldsymbol{H}$ of a 2D wavelet transform is a block matrix with a characteristic block structure (see Figure 8.2). We now introduce a new notation for these blocks as follows

$$\boldsymbol{H}^{i,j} = \begin{cases} \boldsymbol{CC}^{\lambda,\lambda} & \text{for } i,j=0 \\ \boldsymbol{CD}^{\lambda,\lambda-j+1} & \text{for } i=0, 1 \le j \le \lambda \\ \boldsymbol{DC}^{\lambda-i+1,\lambda} & \text{for } 1 \le i \le \lambda, j=0 \\ \boldsymbol{DD}^{\lambda-i+1,\lambda-j+1} & \text{for } 1 \le i,j \le \lambda \end{cases} \tag{8.25}$$

This is illustrated in Figure 8.6 for the case $\lambda = 3$.

$\boldsymbol{H}^{00}$	$\boldsymbol{H}^{01}$	$\boldsymbol{H}^{02}$	$\boldsymbol{H}^{03}$
$\boldsymbol{H}^{10}$	$\boldsymbol{H}^{11}$	$\boldsymbol{H}^{12}$	$\boldsymbol{H}^{13}$
$\boldsymbol{H}^{20}$	$\boldsymbol{H}^{21}$	$\boldsymbol{H}^{22}$	$\boldsymbol{H}^{23}$
$\boldsymbol{H}^{30}$	$\boldsymbol{H}^{31}$	$\boldsymbol{H}^{32}$	$\boldsymbol{H}^{33}$

Figure 8.6: The new notation for the block structure of $\boldsymbol{H}$ (for $\lambda = 3$) as defined by (8.25). Compare with $\boldsymbol{H}$ in Figure 8.2.

We can now use indices $i, j = 0, 1, \ldots, \lambda$ to label the blocks of $\boldsymbol{H}$ in a straightforward manner. It follows from this definition that each block $\boldsymbol{H}^{i,j}$ is an N^i by N^j matrix with

$$N^k = \begin{cases} N/2^{\lambda} & = & S_{\lambda} & \text{for } k=0 \\ N/2^{\lambda-k+1} & = & S_{\lambda-k+1} & \text{for } 1 \le k \le \lambda \end{cases} \tag{8.26}$$

Note that $N^0 = N^1$. Because all the blocks $\boldsymbol{H}^{i,j}$ are either circulant or shift-circulant, we can represent them by vectors $\boldsymbol{h}^{i,j}$ with

$$h^{i,j}_m = \begin{cases} H^{i,j}_{m,0} & \text{for } i \geq j \\ H^{i,j}_{0,m} & \text{for } i < j \end{cases} \tag{8.27}$$

where $m = 0, 1, \ldots, \max(N^i, N^j) - 1$. It follows from (8.26) that the length of $\boldsymbol{h}^{i,j}$ is

$$\begin{cases} N^i & \text{for } i \geq j \\ N^j & \text{for } i < j \end{cases} \tag{8.28}$$

and the shift parameter σ is now given according to Definition 8.1 as

$$\sigma = \begin{cases} N^i/N^j & \text{for } i \geq j \\ N^j/N^i & \text{for } i < j \end{cases}$$

Equation (8.27) suggests a data structure consisting of the vector variables $\boldsymbol{h}^{i,j}$ which refer to the actual arrays representing the final stage of the wavelet transform (e.g. $\boldsymbol{dd}^{\lambda-i+1,\lambda-j+1}$). This structure can also be used to store the intermediate vectors from the recursion if we allow the variables $\boldsymbol{h}^{i,j}$ to assume different values (and different lengths) during the computation – for example by using pointer variables. This is demonstrated in Figure 8.7.

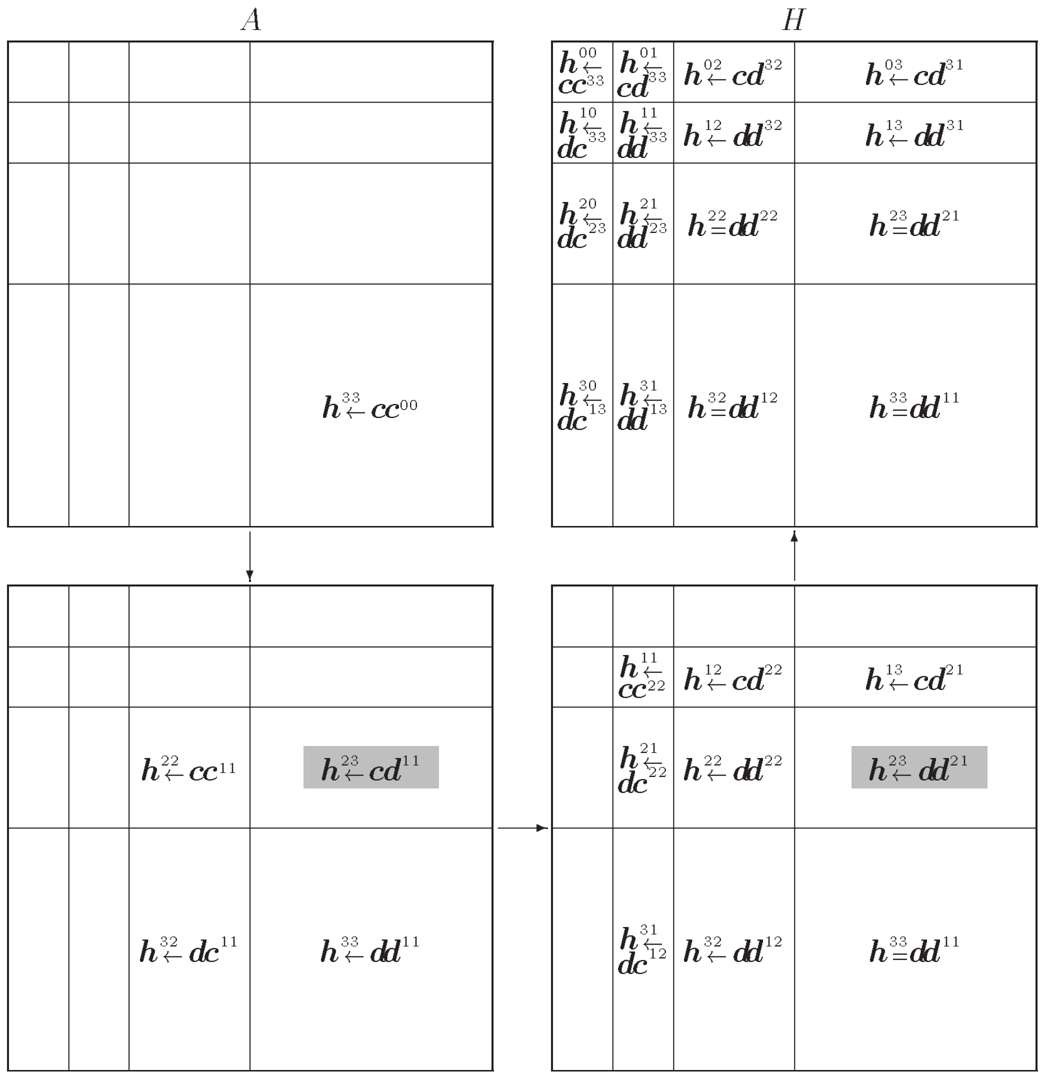


Figure 8.7: The use of (pointer) variables $\boldsymbol{h}^{i,j}$ to implement the 2D wavelet transform of a circulant matrix. Intermediate results are referenced by variables $\boldsymbol{h}^{i,j}$ before they attain their final values. For example, consider the shaded parts: $\boldsymbol{h}^{2,3}$ stores first $\boldsymbol{cd}^{1,1}$ which is then split further into $\boldsymbol{cd}^{2,1}$ and $\boldsymbol{dd}^{2,1}$. Then $\boldsymbol{dd}^{2,1}$ is stored in $\boldsymbol{h}^{2,3}$ overwriting $\boldsymbol{cd}^{1,1}$.

From Figure 8.7 we arrive at the final formulation of the algorithm in Section 8.2.4.

Algorithm 8.1: Circulant 2D wavelet transform (CIRFWT)

$\boldsymbol{h}^{\lambda,\lambda} \leftarrow \boldsymbol{cc}^{0,0}$
For $j = \lambda, \lambda-1, \ldots, 1$
  $[\boldsymbol{h}^{j-1,j-1}, \boldsymbol{h}^{j-1,j}, \boldsymbol{h}^{j,j-1}, \boldsymbol{h}^{j,j}] \leftarrow \text{CIRPWT1}(\boldsymbol{h}^{j,j})$
  For $i = j+1, j+2, \ldots \lambda$
    $[\boldsymbol{h}^{i,j-1}, \boldsymbol{h}^{i,j}] \leftarrow \text{CIRPWT2}(\boldsymbol{h}^{i,j})$
    $[\boldsymbol{h}^{j-1,i}, \boldsymbol{h}^{j,i}] \leftarrow \text{CIRPWT2}(\boldsymbol{h}^{j,i})$
  end
end

where CIRPWT1 is derived from (8.15), (8.16), (8.17), and (8.24):

$$h_m^{j-1,j-1} \leftarrow q_{aa}^0 h_{2m}^{j,j} + \sum_{p=1}^{D-1} \left[q_{aa}^p \, h_{\langle 2m-p \rangle_{Nj}}^{j,j} + q_{aa}^p \, h_{\langle 2m+p \rangle_{Nj}}^{j,j} \right] \tag{8.29}$$

$$h_m^{j-1,j} \leftarrow q_{ab}^0 h_{\langle -2m \rangle_{Nj}}^{j,j} + \sum_{p=1}^{D-1} \left[q_{ab}^p \, h_{\langle -2m-p \rangle_{Nj}}^{j,j} + q_{ba}^p \, h_{\langle -2m+p \rangle_{Nj}}^{j,j} \right] \tag{8.30}$$

$$h_m^{j,j-1} \leftarrow q_{ba}^0 h_{2m}^{j,j} + \sum_{p=1}^{D-1} \left[q_{ba}^p \, h_{\langle 2m-p \rangle_{Nj}}^{j,j} + q_{ab}^p \, h_{\langle 2m+p \rangle_{Nj}}^{j,j} \right] \tag{8.31}$$

$$h_m^{j,j} \leftarrow q_{bb}^0 h_{2m}^{j,j} + \sum_{p=1}^{D-1} \left[q_{bb}^p \, h_{\langle 2m-p \rangle_{Nj}}^{j,j} + q_{bb}^p \, h_{\langle 2m+p \rangle_{Nj}}^{j,j} \right] \tag{8.32}$$

where $m = 0, 1, \ldots, N^{j-1} - 1$. For $i \geq j$ CIRPWT2 is derived from (8.20) and (8.21):

$$h_m^{i,j-1} \leftarrow \sum_{l=0}^{D-1} a_l \, h_{\langle m-\sigma l \rangle_{N^i}}^{i,j} \tag{8.33}$$

$$h_m^{i,j} \leftarrow \sum_{l=0}^{D-1} b_l \, h_{\langle m-\sigma l \rangle_{N^i}}^{i,j} \tag{8.34}$$

where $m = 0, 1, \ldots, N^{j-1} - 1$. For $i < j$ CIRPWT2 is

$$h_m^{i-1,j} \leftarrow \sum_{l=0}^{D-1} a_l \, h_{\langle m-\sigma l \rangle_{Nj}}^{i,j} \tag{8.35}$$

$$h_m^{i,j} \leftarrow \sum_{l=0}^{D-1} b_l \, h_{\langle m-\sigma l \rangle_{Nj}}^{i,j} \tag{8.36}$$

where $m = 0, 1, \dots, N^{i-1} - 1$. Note that we use exactly the same code for CIRPWT2 by exchanging the indices i and j in Algorithm 8.1. Our example now takes the form

$$
\begin{aligned}
[\boldsymbol{h}^{2,2}, \boldsymbol{h}^{2,3}, \boldsymbol{h}^{3,2}, \boldsymbol{h}^{3,3}] &\leftarrow \text{CIRPWT1}(\boldsymbol{h}^{3,3}) \\
\\
[\boldsymbol{h}^{1,1}, \boldsymbol{h}^{1,2}, \boldsymbol{h}^{2,1}, \boldsymbol{h}^{2,2}] &\leftarrow \text{CIRPWT1}(\boldsymbol{h}^{2,2}) \\
[\boldsymbol{h}^{3,1}, \boldsymbol{h}^{3,2}] &\leftarrow \text{CIRPWT2}(\boldsymbol{h}^{3,2}) \\
[\boldsymbol{h}^{1,3}, \boldsymbol{h}^{2,3}] &\leftarrow \text{CIRPWT2}(\boldsymbol{h}^{2,3}) \\
\\
[\boldsymbol{h}^{0,0}, \boldsymbol{h}^{0,1}, \boldsymbol{h}^{1,0}, \boldsymbol{h}^{1,1}] &\leftarrow \text{CIRPWT1}(\boldsymbol{h}^{1,1}) \\
[\boldsymbol{h}^{2,0}, \boldsymbol{h}^{2,1}] &\leftarrow \text{CIRPWT2}(\boldsymbol{h}^{2,1}) \\
[\boldsymbol{h}^{0,2}, \boldsymbol{h}^{1,2}] &\leftarrow \text{CIRPWT2}(\boldsymbol{h}^{1,2}) \\
[\boldsymbol{h}^{3,0}, \boldsymbol{h}^{3,1}] &\leftarrow \text{CIRPWT2}(\boldsymbol{h}^{3,1}) \\
[\boldsymbol{h}^{0,3}, \boldsymbol{h}^{1,3}] &\leftarrow \text{CIRPWT2}(\boldsymbol{h}^{1,3})
\end{aligned}
$$

8.2.6 Computational work

We will now derive an estimate of the complexity of Algorithm 8.1, but first we need to establish the following lemma:

Lemma 8.4

$$\boxed{\sum_{k=0}^{\lambda-1} N^k = \frac{N}{2}}$$

Proof: Using (8.26), we write

$$
\begin{aligned}
\sum_{k=0}^{\lambda-1} N^k &= \frac{N}{2^\lambda} + \sum_{k=1}^{\lambda-1} \frac{N}{2^{\lambda-k+1}} \\
&= N\left(\frac{1}{2^\lambda} + \sum_{k=2}^{\lambda} \frac{1}{2^k}\right) \\
&= N\left(\frac{1}{2^\lambda} + \frac{1-\frac{1}{2^{\lambda+1}}}{1-\frac{1}{2}} - \frac{1}{2} - 1\right) \\
&= N\left(\frac{1}{2^\lambda} + 2 - \frac{1}{2^\lambda} - \frac{1}{2} - 1\right) \\
&= \frac{N}{2}
\end{aligned}
$$

□

Equations (8.29)–(8.32) each require $(4(D-1)+2)N^{j-1}$ flops so the complexity of CIRPWT1 is

$$F_{\text{CIRPWT1}}(N) = (16(D-1)+8)N$$

with $N = N^{j-1}$.

Equations (8.33)–(8.36) each require $2DN$ flops so the complexity of CIRPWT2 is

$$F_{\text{CIRPWT2}}(N) = 4DN$$

with $N = N^{j-1}$ or $N = N^{i-1}$.

The total complexity of Algorithm 8.1 is thus

$$F_{\text{CIRFWT}}(N) = \sum_{j=1}^{\lambda} \left(F_{\text{CIRPWT1}}(N^{j-1}) + \sum_{i=j+1}^{\lambda} F_{\text{CIRPWT2}}(N^{j-1}) + F_{\text{CIRPWT2}}(N^{i-1}) \right)$$

Hence

$$\begin{aligned}
F_{\text{CIRFWT}}(N) &= \sum_{j=1}^{\lambda} \left((16(D-1)+8)N^{j-1} + \sum_{i=j+1}^{\lambda} 4DN^{j-1} + 4DN^{i-1} \right) \\
&< 4D \left(\sum_{j=1}^{\lambda} 4N^{j-1} + \sum_{i=2}^{\lambda} N^{j-1} + N^{i-1} \right) \\
&< 4D \left((4+\lambda-1) \sum_{j=1}^{\lambda} N^{j-1} + \sum_{i=1}^{\lambda} N^{i-1} \right) \\
&= 4D \left((4+\lambda-1)\frac{N}{2} + \frac{N}{2} \right)
\end{aligned}$$

where we have used Lemma 8.4 twice. Consequently, we obtain the following bound for the complexity of Algorithm 8.1:

$$\boxed{F_{\text{CIRFWT}}(N) < 2DN(4+\lambda)} \tag{8.37}$$

8.2.7 Storage

In this section we will investigate the storage requirement for $\boldsymbol{H}$ as a function of the transform depth λ when the data structure proposed in Section 8.2.5 is used.

Lemma 8.5 *The number of elements needed to represent the result of λ steps of the wavelet transform of a circulant $N \times N$ matrix as computed by Algorithm 8.1 is*

$$S_N(\lambda) = N\left(1 + \sum_{k=1}^{\lambda} \frac{k}{2^{\lambda-k}}\right), \ \lambda = 0, 1, \ldots, \lambda_{max} \tag{8.38}$$

Proof: We will prove (8.38) by induction:
Induction start: For $\lambda = 0$ there is only one block, namely $\boldsymbol{H}^{0,0}$ which is represented by the vector $\boldsymbol{h}^{0,0}$ of length N. Thus, $S_N(0) = N$ and (8.38) therefore holds for $\lambda = 0$.

Induction step: The induction hypothesis is

$$S_N(\lambda - 1) = N\left(1 + \sum_{k=1}^{\lambda-1} \frac{k}{2^{\lambda-1-k}}\right) \tag{8.39}$$

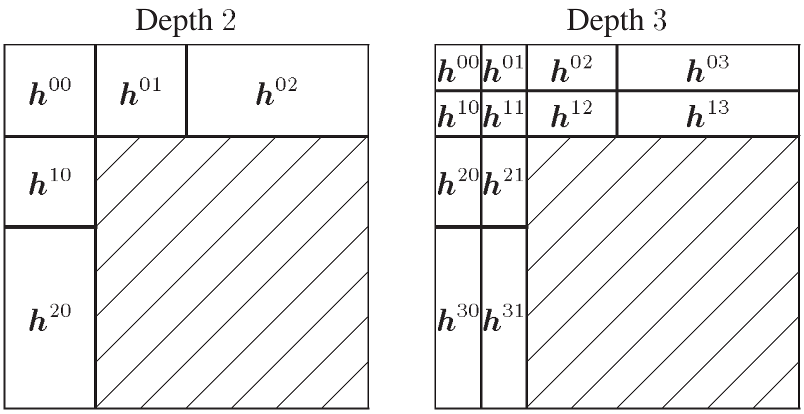


Figure 8.8: The differences between a wavelet transform of depth $\lambda - 1$ and λ (shown for $\lambda = 3$).

The storage differences between a wavelet transform of depth $\lambda - 1$ and λ lie in the uppermost and leftmost blocks of the structure as indicated in Figure 8.8. The rest are the same.

Depth 2		Depth 3	
Vector	Length	Vector	Length
$\boldsymbol{h}^{00}$	$N/4$	$\boldsymbol{h}^{00}$	$N/8$
		$\boldsymbol{h}^{01}, \boldsymbol{h}^{10}, \boldsymbol{h}^{11}$	$N/8$
$\boldsymbol{h}^{01}, \boldsymbol{h}^{10}$	$N/4$	$\boldsymbol{h}^{02}, \boldsymbol{h}^{12}, \boldsymbol{h}^{20}, \boldsymbol{h}^{21}$	$N/4$
$\boldsymbol{h}^{02}, \boldsymbol{h}^{20}$	$N/2$	$\boldsymbol{h}^{03}, \boldsymbol{h}^{13}, \boldsymbol{h}^{30}, \boldsymbol{h}^{31}$	$N/2$

Table 8.1: The number of elements in the vectors shown in Figure 8.8 ($\lambda = 3$).

Recall that each block is represented by a vector. We will subtract the number of elements in the affected vectors at depth $\lambda - 1$ from (8.39) and then add the elements from vectors resulting from a transform to depth λ. The vector lengths for the example $\lambda = 3$ are shown in Table 8.1
According to (8.26) – (8.28) the vector $\boldsymbol{h}^{00}$ at depth $\lambda - 1$ has the length

$$N^0 = N/2^{\lambda-1} \tag{8.40}$$

This is replaced by 4 new vectors ($\boldsymbol{h}^{00}, \boldsymbol{h}^{01}, \boldsymbol{h}^{10}, \boldsymbol{h}^{11}$) at depth λ each having the length $N/2^\lambda$. We write this number as

$$4N\frac{1}{2^\lambda} = N\left(\frac{1}{2^{\lambda-1}} + \frac{1}{2^{\lambda-1}}\right) \tag{8.41}$$

The blocks above the diagonal that are replaced are those in the upper row. They are represented by a vector of length $N/2$, a vector of length $N/4$ down to a vector of length $N/2^{\lambda-1}$. The number of elements in vectors corresponding to *column* blocks is the same. Hence the number of elements we must subtract from (8.39) is

$$2\sum_{k=1}^{\lambda-1}\frac{N}{2^k} = 2\sum_{k=1}^{\lambda-1}\frac{N}{2^{\lambda-k}} = \sum_{k=1}^{\lambda-1}\frac{N}{2^{\lambda-1-k}} \tag{8.42}$$

The splitting of these off-diagonal blocks will create twice as many blocks so the number of elements introduced will be

$$2\sum_{k=1}^{\lambda-1}\frac{N}{2^{\lambda-1-k}} = 2\sum_{k=2}^{\lambda}\frac{N}{2^{\lambda-k}} \tag{8.43}$$

Now we subtract from (8.39) the total number of elements in the vectors that are replaced ((8.40) and (8.42)) and add the number of elements introduced ((8.41) and (8.43)), i.e.

$$
\begin{aligned}
S_N(\lambda) &= S_N(\lambda-1) - \frac{N}{2^{\lambda-1}} - \sum_{k=1}^{\lambda-1} \frac{N}{2^{\lambda-1-k}} + N\left(\frac{1}{2^{\lambda-1}} + \frac{1}{2^{\lambda-1}}\right) + 2\sum_{k=2}^{\lambda} \frac{N}{2^{\lambda-k}} \\
&= N\left(1 + \sum_{k=1}^{\lambda-1} \frac{k}{2^{\lambda-1-k}} - \frac{1}{2^{\lambda-1}} - \sum_{k=1}^{\lambda-1} \frac{1}{2^{\lambda-1-k}} + \frac{1}{2^{\lambda-1}} + \frac{1}{2^{\lambda-1}} + 2\sum_{k=2}^{\lambda} \frac{1}{2^{\lambda-k}}\right) \\
&= N\left(1 + \sum_{k=1}^{\lambda-1} \frac{k-1}{2^{\lambda-1-k}} + \frac{1}{2^{\lambda-1}} + 2\sum_{k=2}^{\lambda} \frac{1}{2^{\lambda-k}}\right) \\
&= N\left(1 + \sum_{k=2}^{\lambda} \frac{k-2}{2^{\lambda-k}} + \frac{1}{2^{\lambda-1}} + 2\sum_{k=2}^{\lambda} \frac{1}{2^{\lambda-k}}\right) \\
&= N\left(1 + \sum_{k=2}^{\lambda} \frac{k}{2^{\lambda-k}} + \frac{1}{2^{\lambda-1}}\right) \\
&= N\left(1 + \sum_{k=1}^{\lambda} \frac{k}{2^{\lambda-k}}\right)
\end{aligned}
$$

which is the desired formula. □

We consider now the sum in (8.38) and define

$$
f(\lambda) = \sum_{k=1}^{\lambda} \frac{k}{2^{\lambda-k}}, \quad \lambda = 0, 1, 2, \ldots
$$

With a change of variables we can rewrite f as

$$
f(\lambda) = \sum_{k=0}^{\lambda-1} \frac{\lambda-k}{2^k}
$$

This is an **arithmetic-geometric series** which has the closed form [Spi93, p. 107]

$$
\begin{aligned}
f(\lambda) &= \frac{\lambda(1-(1/2)^\lambda)}{1-1/2} - \frac{1-\lambda(1/2)^{\lambda-1} + (\lambda-1)(1/2)^\lambda}{2(1-1/2)^2} \\
&= 2\lambda(1-(1/2)^\lambda) - 2(1-\lambda(1/2)^{\lambda-1} + (\lambda-1)(1/2)^\lambda) \\
&= 2\lambda - 2 - 4\lambda(1/2)^\lambda + 2\lambda(1/2)^{\lambda-1} + 2(1/2)^\lambda \\
&= 2\lambda - 2 + 2(1/2)^\lambda
\end{aligned}
$$

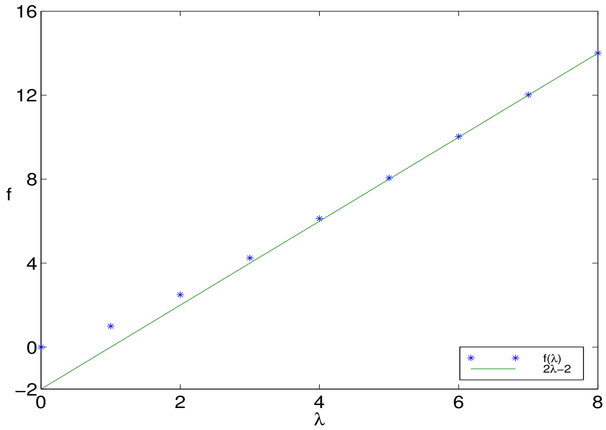


Figure 8.9: The function $f(\lambda)$ behaves asymptotically like $2\lambda - 2$.

We observe that $f(\lambda)$ behaves asymptotically as $2\lambda - 2$. Figure 8.9 shows a plot of $f(\lambda)$ together with its asymptote.

The storage requirement can now be expressed as

$$\begin{aligned} S_N(\lambda) &= N(1 + f(\lambda)) \\ &= N(2\lambda - 1 + 2(1/2)^\lambda), \quad \lambda = 0, 1, \ldots, \lambda_{max} \end{aligned}$$

and we have the bound

$$\boxed{S_N(\lambda) \leq 2\lambda N, \quad \lambda \geq 1} \tag{8.44}$$

8.3 2D wavelet transform of a circulant, banded matrix

An important special case of a circulant matrix is when $\boldsymbol{A}$ is a *banded* circulant matrix such as the differentiation matrix $\boldsymbol{D}$ given in (7.14). In this case each column of $\boldsymbol{A}$ consists of a piece which is zero and a piece which is regarded as non-zero. In certain columns the non-zero part is wrapped around. Consequently, it is sufficient to store only the non-zero part of the first column along with an index δ determining how it must be aligned in the first column relative to the full-length vector. The length of the non-zero part is the **bandwidth** of $\boldsymbol{A}$, and we will denote it by L in this chapter.

It turns out that each block of the 2D wavelet transform retains a banded structure, so the vector representing it need only include the non-zero part. Therefore the storage requirements can be considerably less than that given by (8.38). An example of this structure is given in Figure 8.10.

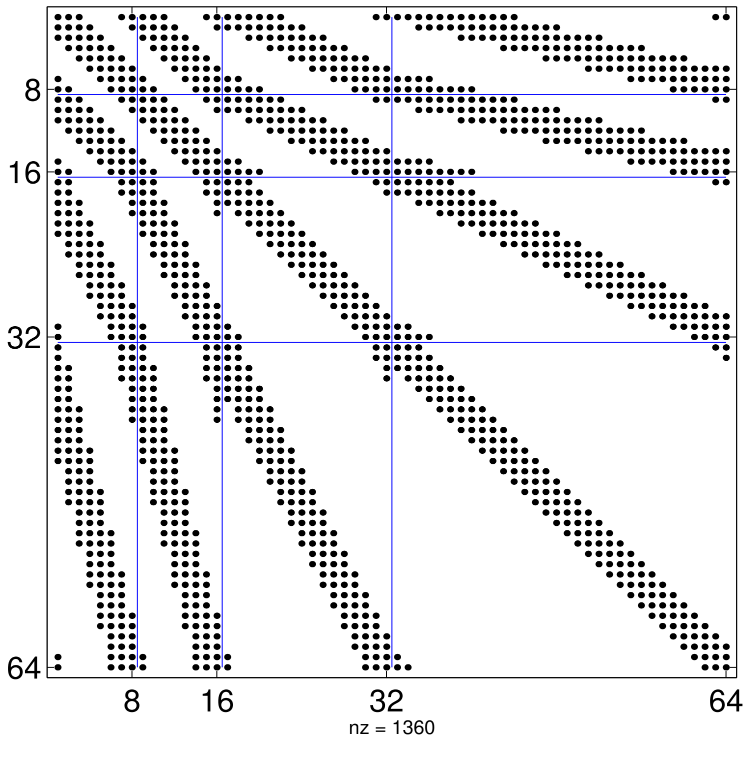


Figure 8.10: The structure of a wavelet transform of a 64×64 circulant banded matrix. Here $L = 3$, $D = 4$ and $\lambda = 3$. This is the same block structure as in Figure 8.6 and in Figure 7.1.

For each block, we know the non-zero values of the first column (or row in the case of blocks above the diagonal) represented by the vector $\boldsymbol{v}^{i,j} = [v_0^{i,j}, v_1^{i,j}, \ldots, v_{L-1}^{i,j}]^T$, the amount by which it is shifted (σ) and how it is aligned with respect to the first element of the block (δ). The block $\boldsymbol{H}^{3,1}$, say, has the general structure

$$
\boldsymbol{H}^{3,1} = \begin{bmatrix}
v_2^{3,1} & & & \\
v_3^{3,1} & & & \\
v_4^{3,1} & v_0^{3,1} & & \\
 & v_1^{3,1} & & \\
 & v_2^{3,1} & & \\
 & v_3^{3,1} & & \\
 & v_4^{3,1} & v_0^{3,1} & \\
 & & v_1^{3,1} & \\
 & & v_2^{3,1} & \\
 & & v_3^{3,1} & \\
 & & v_4^{3,1} & v_0^{3,1} \\
 & & & v_1^{3,1} \\
 & & & v_2^{3,1} \\
 & & & v_3^{3,1} \\
v_0^{3,1} & & & v_4^{3,1} \\
v_1^{3,1} & & &
\end{bmatrix} \tag{8.45}
$$

In this example we have $\delta = 3$, $N^3 = 16$, $N^1 = 4$, $\sigma = 4$ (from Definition 8.1), bandwidth $L^{3,1} = 5$, and

$$
h_m^{3,1} = \begin{cases} v_{\langle m+2 \rangle_{16}}^{3,1} & \text{for } \langle m+2 \rangle_{16} \in [0, 4] \\ 0 & \text{otherwise} \end{cases}
$$

with

$$
H_{m,n}^{3,1} = h_{\langle m-4n \rangle_{16}}^{3,1}
$$

If the matrix is not banded we have the special case $\boldsymbol{v}^{i,j} = \boldsymbol{h}^{i,j}$ and $\delta = 1$ so (8.38) applies exactly.

8.3.1 Calculation of bandwidths

Given the bandwidth L of the original matrix $\boldsymbol{A}$ it is possible to derive a formula for the bandwidths of each block of the wavelet transform.

Let $L^{i,j}$ be the bandwidth of block $\boldsymbol{H}^{i,j}$ shown in Figure 8.6. We can then use the recurrence formulas for the 2D wavelet transform to obtain the desired formulas.

Blocks on the diagonal

We start with the blocks on the diagonal given by equation (8.1) to (8.4) and consider again only the ***cd*** block as the typical case. Let us recall (8.29):

$$h_m^{j-1,j-1} \leftarrow q_{aa}^0 h_{2m}^{j,j} + \sum_{p=1}^{D-1} \left[q_{aa}^p \, h_{\langle 2m-p \rangle_{Nj}}^{j,j} + q_{aa}^p \, h_{\langle 2m+p \rangle_{Nj}}^{j,j} \right]$$

for $m = 0, 1, \ldots, N^{j-1} - 1$. Assume that $\boldsymbol{h}^{j,j}$ is banded, i.e. zero outside a band of length $L^{j,j}$ as shown in Figure 8.11. Then we can use the recurrence formula to compute the bandwidth of $\boldsymbol{h}^{j-1,j-1}$. Without loss of generality we may assume that the nonzero part is wholly contained in the vector; i.e. there is no wrap-around.

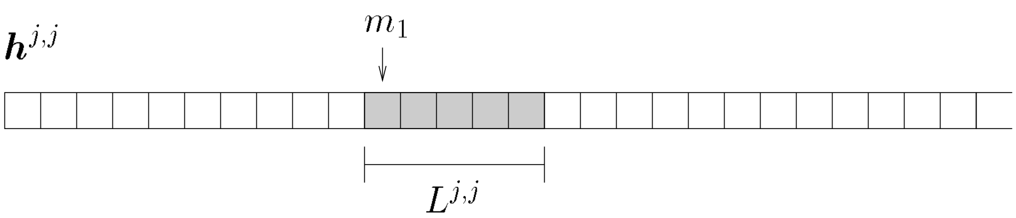


Figure 8.11: The band of $\boldsymbol{h}^{j,j}$ has length $L^{j,j}$ and starts at index m_1.

Since $\boldsymbol{h}^{j,j}$ is zero outside the interval starting at m_1 of length $L^{j,j}$, we see that $h_m^{j-1,j-1}$ will be zero only if $2m - p \geq m_1 + L^{j,j}$ or $2m + p < m_1$ for all $p \in [0, D-1]$. This leads immediately to the inequalities

$$\begin{aligned} 2m + (D-1) &< m_1 \\ 2m - (D-1) &\geq m_1 + L^{j,j} \end{aligned}$$

or

$$\begin{aligned} m &\geq \frac{m_1 + L^{j,j} + D - 1}{2} \quad \text{or} \\ m &< \frac{m_1 - D + 1}{2} \end{aligned}$$

The length of this interval is then the bandwidth of $\boldsymbol{h}^{j-1,j-1}$:

$$\begin{aligned} L^{j-1,j-1} &= \frac{m_1 + L^{j,j} + D - 1}{2} - \frac{m_1 - D + 1}{2} \\ &= \frac{L^{j,j}}{2} + D - 1 \end{aligned} \tag{8.46}$$

However, since the bandwidth is an integer the fraction must be rounded either up or down if $L^{j,j}$ is odd. Which of these operations to choose depends on m_1 as illustrated in Figure 8.12.

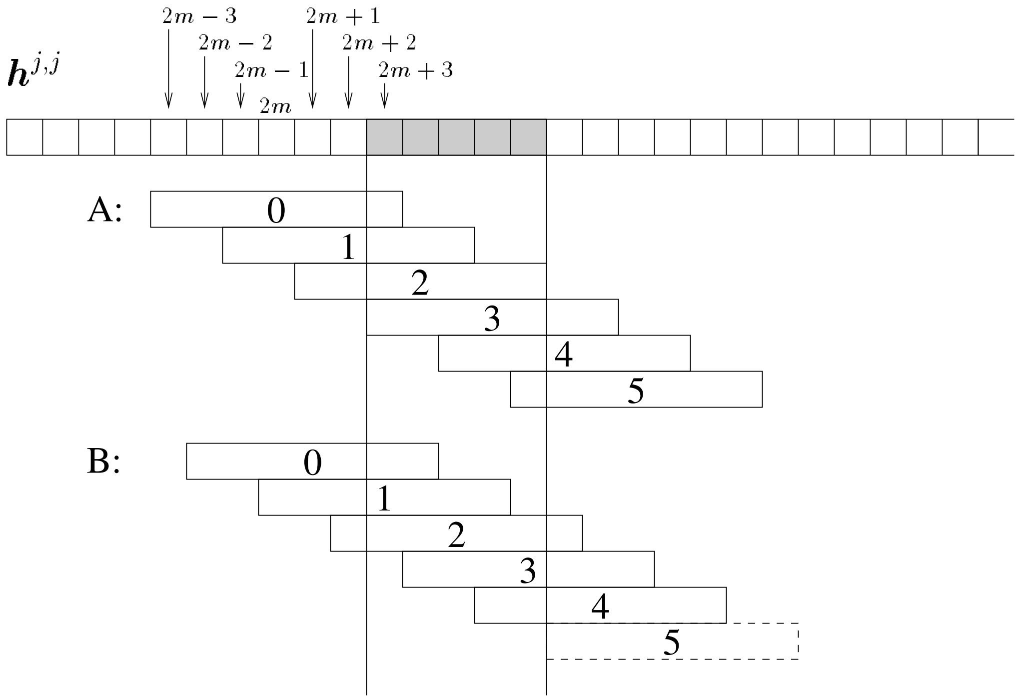


Figure 8.12: The computation in equation (8.14) can be viewed as a sliding filter of length $2D-1$ applied to the band of $\boldsymbol{h}^{j,j}$. The numbers indicate offset with respect to $2m = (m_1 - D + 1)/2$. The resulting bandwidth depends on how the initial bandwidth is aligned with this sliding filter. In this example $L^{j,j} = 5$, $D = 4$ so $L^{j-1,j-1}$ is either 5 (case B) or 6 (case A) depending on how the convolutions happen to align with the non-zero block. Thus case A corresponds to rounding up and case B corresponds to rounding down in (8.46).

We have chosen always to round upwards because it yields an upper bound for the bandwidth. Thus the formula becomes

$$L^{j-1,j-1} = \left\lceil \frac{L^{j,j}}{2} \right\rceil + D - 1 \tag{8.47}$$

We observe that equation (8.47) has two fixed points, namely

$$L^{j-1,j-1} = L^{j,j} = \begin{cases} 2D-2 \\ 2D-1 \end{cases}$$

and it turns out that there is convergence to one of these values depending on whether the initial $L^{j,j}$ is smaller than $2D-2$ or larger than $2D-1$. However, the important fact is that these fixed points are *more related to the wavelet genus* D than to the original bandwidth L.

Blocks below and above the diagonal

The bandwidths of the blocks below the diagonal are found from recurrence formulas of the form

$$h_m^{i,j-1} = \sum_{l=0}^{D-1} a_l h^{i,j}_{\langle m-\sigma l\rangle_{N^i}}$$

Again, we disregard wrapping and, proceeding as in the case of $\boldsymbol{h}^{j-1,j-1}$ above, we find that $h_m^{i,j-1}$ is zero only if $m-\sigma l \geq m_1 + L^{i,j}$ or $m-\sigma l < m_1$ for all $l \in [0, D-1]$. This leads to the inequalities

$$\begin{aligned} m &< m_1 \\ m-\sigma(D-1) &\geq m_1 + L^{i,j} \end{aligned}$$

Consequently, the interval length for which $h_m^{i,j-1} \neq 0$ is

$$\begin{aligned} L^{i,j-1} &= m_1 + L^{i,j} + \sigma(D-1) - m_1 \\ &= L^{i,j} + \sigma(D-1) \end{aligned}$$

Performing similar computations for blocks above the diagonal yields the result

$$L^{i,j-1} = L^{j-1,i} = L^{i,j} + \sigma(D-1) \tag{8.48}$$

Since σ is the ratio between N^i and N^j, equation (8.48) shows that *the bandwidth grows exponentially* as the difference between i and j increases. Figures 8.13 and 8.14 show examples of the bandwidths for different transform levels.

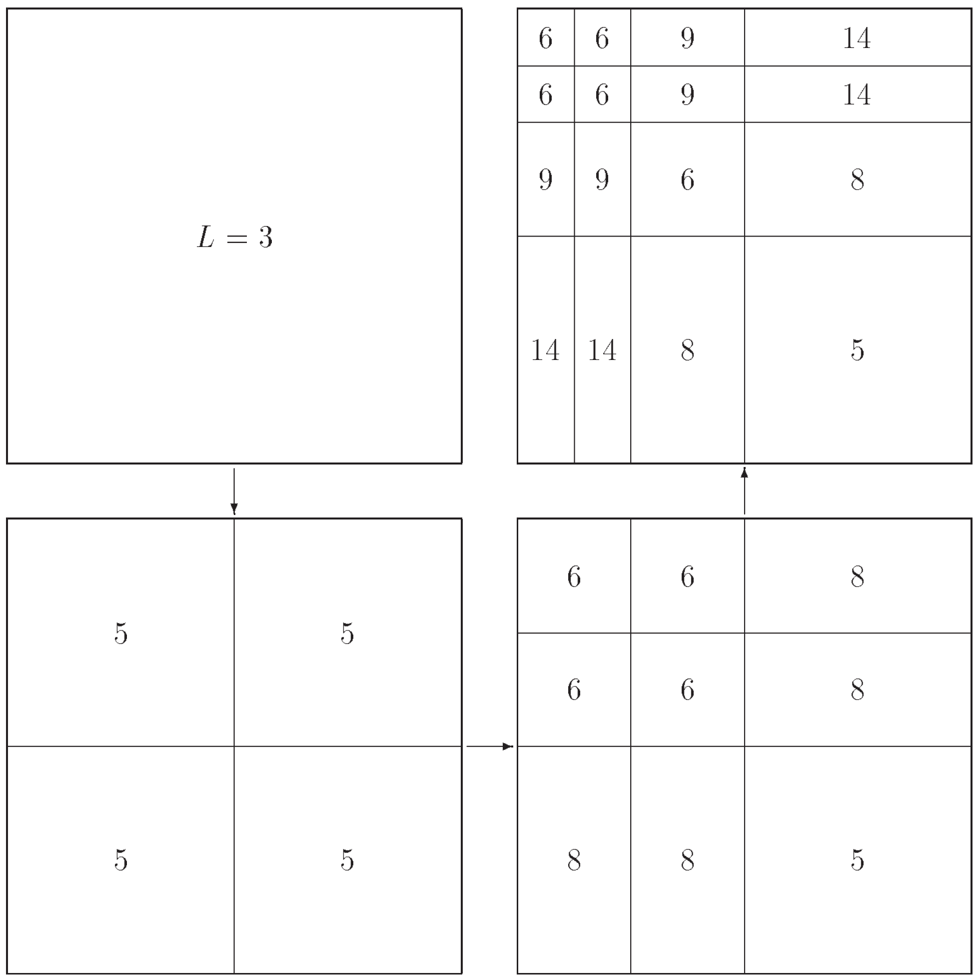


Figure 8.13: The bandwidths for transform depths $\lambda = 0, 1, 2, 3$. The initial bandwidth is 3 and $D = 4$. These bandwidths are upper bounds since actual bandwidths may be one less than the predicted values as can be seen in Figure 8.10.

6	6	9	15	26
6	6	9	15	26
9	9	6	9	14
15	15	9	6	8
26	26	14	8	5

Figure 8.14: The bandwidths for transform depth $\lambda = 4$.

8.4 Matrix-vector multiplication in a wavelet basis

We now turn to the problem of computing the matrix-vector product $\boldsymbol{y} = \boldsymbol{Hx}$ where $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^N$ and $\boldsymbol{H}$ is given as in (8.25). This system has the form shown in Figure 8.15.

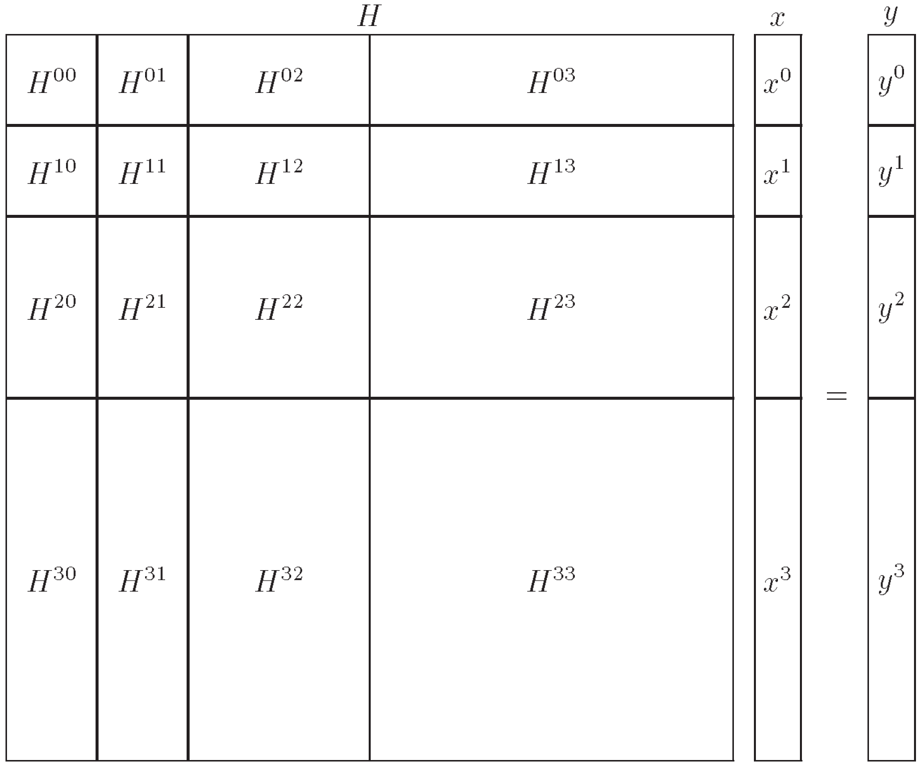


Figure 8.15: The structure of $\boldsymbol{y} = \boldsymbol{Hx}$ for $\lambda = 3$.

The vector y may be computed block-wise as follows

$$\boldsymbol{y}^i = \sum_{j=0}^{\lambda} \boldsymbol{H}^{i,j}\boldsymbol{x}^j, \quad i = 0, 1, \ldots, \lambda \tag{8.49}$$

where λ is the depth of the wavelet transform. The symbols $\boldsymbol{x}^j$, $\boldsymbol{y}^i$, and $\boldsymbol{H}^{i,j}$ denote the different *blocks* of the $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{H}$ as indicated in Figure 8.6. The computation in (8.49) is thus broken down into the tasks of computing the products which we will denote by

$$\boldsymbol{y}^{i,j} = \boldsymbol{H}^{i,j}\boldsymbol{x}^j, \quad i, j = 0, 1, \ldots, \lambda - 1 \tag{8.50}$$

In the following we distinguish between blocks on or below the diagonal ($i \geq j$) and blocks above the diagonal ($i < j$).

8.4.1 Blocks on or below the diagonal

Let $\boldsymbol{v}^{i,j}$, $i \geq j$, be the vector of length $L^{i,j}$ representing the nonzero part of the first column of $\boldsymbol{H}^{i,j}$, i.e.

$$h_m^{i,j} = \begin{cases} v_{\langle m+\delta-1\rangle_{N^i}}^{i,j} & \text{for } \langle m+\delta-1\rangle_{N^i} \in [0, L^{i,j}-1] \\ 0 & \text{otherwise} \end{cases}$$

and

$$H_{m,n}^{i,j} = h_{\langle m-\sigma n\rangle_{N^i}}^{i,j}$$

where $m = 0, 1, \ldots, N^i$, $\sigma = N^i/N^j$, and δ is the offset relative to the upper left element (see (8.45)). From equation (8.50) we see that the typical element of $y^{i,j}$ can be computed column wise as

$$\begin{aligned} y_m^{i,j} &= \sum_{n=0}^{N^j-1} H_{m,n}^{i,j} x_n^j \\ &= \sum_{n=0}^{N^j-1} h_{\langle m-\sigma n\rangle_{N^i}}^{i,j} x_n^j \\ &= \sum_{n=0}^{N^j-1} v_{\langle m-\sigma n+\delta-1\rangle_{N^i}}^{i,j} x_n^j \end{aligned} \tag{8.51}$$

For each n this computation is only valid for those $m \in [0, N^i - 1]$ where $v_{\langle m-\sigma n+\delta-1\rangle_{N^i}}^{i,j}$ is defined, namely where

$$0 \leq \langle m - \sigma n + \delta - 1\rangle_{N^i} \leq L^{i,j} - 1 \tag{8.52}$$

Let k and l be defined such that $k \le m \le l$ whenever (8.52) is satisfied. Then we can find k from the requirement

$$\begin{aligned} \langle k - \sigma n + \delta - 1 \rangle_{N^i} &= 0 \Leftrightarrow \\ k &= \langle \sigma n - \delta + 1 \rangle_{N^i} \end{aligned}$$

and the last row as $l = \langle k + L^{i,j} - 1 \rangle_{N^i}$. Letting

$$y^{i,j}_{k:l} = [y^{i,j}_k, y^{i,j}_{k+1}, \dots, y^{i,j}_l]$$

then we can write the computation (8.51) compactly as

$$y^{i,j}_{k:l} = y^{i,j}_{k:l} + x^j_n v^{i,j}_{0:L^{i,j}-1}, \quad n = 0, 1, \dots, N^j - 1 \tag{8.53}$$

When $k > l$ the band is wrapped and (8.53) must be modified accordingly.

If the vector x is a wavelet spectrum then many of its elements are normally close to zero as described in Chapter 4. Therefore we will design the algorithm to disregard computations involving elements in x^j where

$$\left| x^j_n \right| < \varepsilon$$

The algorithm is given below

Algorithm 8.2: $\boldsymbol{y}^{i,j} = \boldsymbol{H}^{i,j}\boldsymbol{x}^j$, $i \ge j$

```
For n = 0 to N^j − 1
    if |x^j_n| > ε then
        k = ⟨σn − δ + 1⟩_{N^i}
        l = ⟨k + L^{i,j} − 1⟩_{N^i}
        if k < l then
            y^{i,j}_{k:l} = y^{i,j}_{k:l} + x^j_n v^{i,j}
        else                                        (wrap)
            y^{i,j}_{0:l} = y^{i,j}_{0:l} + x^j_n v^{i,j}_{L^{i,j}−l:L−1}
            y^{i,j}_{k:N^i−1} = y^{i,j}_{k:N^i−1} + x_n v^{i,j}_{0:L^{i,j}−l−1}
        end
    end
end
```

8.4.2 Blocks above the diagonal

Let $v^{i,j}$, $i < j$, be the vector of length $L^{i,j}$ representing the nonzero part of the first *row* of $\boldsymbol{H}^{i,j}$, i.e.

$$h^{i,j}_n = \begin{cases} v^{i,j}_{\langle n+\delta-1 \rangle_{N^j}} & \text{for } \langle n + \delta - 1 \rangle_{N^j} \in [0, L^{i,j} - 1] \\ 0 & \text{otherwise} \end{cases} \tag{8.54}$$

where

$$H^{i,j}_{m,n} = h^{i,j}_{\langle n-\sigma m\rangle_{N^j}} \tag{8.55}$$

with $\sigma = N^j/N^i$. An example (shown for $\boldsymbol{H}^{1,3}$) is

$$\begin{bmatrix}
v_2 v_3 & v_4 v_5 v_6 v_7 & \boxed{v_8 v_9 v_{10} 0} & & & & & & v_0 v_1 \\
& v_0 v_1 v_2 v_3 & \boxed{v_4 v_5 v_6 v_7} & v_8 v_9 v_{10} 0 & & & & & \\
& & \boxed{v_0 v_1 v_2 v_3} & v_4 v_5 v_6 v_7 & v_8 v_9 v_{10} 0 & & & & \\
& & & v_0 v_1 v_2 v_3 & v_4 v_5 v_6 v_7 & v_8 v_9 v_{10} 0 & & & \\
& & & & v_0 v_1 v_2 v_3 & v_4 v_5 v_6 v_7 & v_8 v_9 v_{10} 0 & & \\
& & & & & v_0 v_1 v_2 v_3 & v_4 v_5 v_6 v_7 & v_8 v_9 v_{10} 0 & \\
v_{10} 0 & & & & & & v_0 v_1 v_2 v_3 & v_4 v_5 v_6 v_7 & v_8 v_9 \\
v_6 v_7 & v_8 v_9 v_{10} 0 & & & & & & v_0 v_1 v_2 v_3 & v_4 v_5
\end{bmatrix}$$

Here $\sigma = 4, \delta = 3$, $N^1 = 8$, $N^3 = 32$, $L^{1,3} = 12$ (padded with one zero). The superscripts i, j has been dropped in this example.

In order to be able to disregard small elements in x as in the previous section, we would like to convert the row representation to a column oriented format. As indicated in the example above, the block $\boldsymbol{H}^{i,j}$ can be characterized completely by σ column vectors each with length $L^{i,j}/\sigma$ together with some book-keeping information. We will assume that $L^{i,j}$ is a multiple of σ, possibly obtained through padding with zeros as suggested in the example above.

Now we choose these vectors from the columns

$$L^{i,j} - \delta, L^{i,j} - \delta - 1, \dots, L^{i,j} - \delta - \sigma + 1$$

which are the σ last columns where the top row is non-zero: the shaded area in the example above. Let these column vectors be denoted $Z^{i,j}_{:,d}$ for $d = 0, 1, \dots, \sigma - 1$. From (8.54) and (8.55) we get

$$\begin{aligned}
Z^{i,j}_{m,d} &= H^{i,j}_{m,L^{i,j}-\delta-d} \\
&= h^{i,j}_{\langle L^{i,j}-\delta-d-\sigma m\rangle_{N^j}} \\
&= v^{i,j}_{\langle L^{i,j}-\delta-d-\sigma m+\delta-1\rangle_{N^j}} \\
&= v^{i,j}_{\langle L^{i,j}-\sigma m-d-1\rangle_{N^j}}
\end{aligned} \tag{8.56}$$

for $m = 0, 1, \dots, L^{i,j}/\sigma - 1$ and $d = 0, 1, \dots, \sigma - 1$. In the example above we have

$$Z^{i,j}_{:,0} = \begin{pmatrix} 0 \\ v_7 \\ v_3 \end{pmatrix}, \; Z^{i,j}_{:,1} = \begin{pmatrix} v_{10} \\ v_6 \\ v_2 \end{pmatrix}, \; Z^{i,j}_{:,2} = \begin{pmatrix} v_9 \\ v_5 \\ v_1 \end{pmatrix}, \; Z^{i,j}_{:,3} = \begin{pmatrix} v_8 \\ v_4 \\ v_0 \end{pmatrix}$$

Equation (8.50) can then be computed columnwise using $Z^{i,j}_{:,d}$ instead of $\boldsymbol{H}^{i,j}$. The typical element of $y^{i,j}$ is

$$\begin{aligned} y^{i,j}_m &= \sum_{n=0}^{N^j-1} H^{i,j}_{m,n} x^j_n \\ &= \sum_{n=0}^{N^j-1} Z^{i,j}_{s,d} x^j_n \end{aligned}$$

Let k and l be defined such that $k \leq m \leq l$ whenever $0 \leq s \leq L^{i,j}/\sigma - 1$ and let n be fixed. Our task is now to determine d, together with k and l, such that $Z^{i,j}_{s,d} = H^{i,j}_{m,n}$. Therefore we put $s = 0$ and look for k, d such that

$$Z^{i,j}_{0,d} = H^{i,j}_{k,n} \tag{8.57}$$

In other words: For n given, we want to know which vector to use and at which row it must be aligned.

Next we insert the definitions (8.55) and (8.56) in (8.57) and use (8.54) to obtain the equation

$$v^{i,j}_{\langle L^{i,j}-d-1\rangle_{N^j}} = h^{i,j}_{\langle n-\sigma k\rangle_{N^j}} = v^{i,j}_{\langle n-\sigma k+\delta-1\rangle_{N^j}}$$

which is fulfilled whenever

$$\begin{aligned} \langle L^{i,j} - d - 1\rangle_{N^j} &= \langle n - \sigma k + \delta - 1\rangle_{N^j} \Leftrightarrow \\ \langle L^{i,j} - d - n + \sigma k - \delta\rangle_{N^j} &= 0 \end{aligned}$$

Let $L = L^{i,j} - n - \delta$. Then we can write the requirement as

$$\langle L - d + \sigma k\rangle_{N^j} = 0 \tag{8.58}$$

Since we need k in the interval $[0, N^i - 1]$ we rewrite (8.58) using Lemma B.2:

$$\begin{aligned} 0 &= \langle L - d + \sigma k\rangle_{N^j} \\ &= \langle (L - d + \sigma k)/\sigma\rangle_{N^j/\sigma} \\ &= \langle k + (L - d)/\sigma\rangle_{N^i} \end{aligned}$$

from which we get

$$k = \langle (d - L)/\sigma\rangle_{N^i}$$

For this to be well-defined σ must be a divisor in $(d - L)$, i.e. $\langle d - L\rangle_\sigma = 0$ so we must choose

$$d = \langle L\rangle_\sigma$$

Expanding the expression for L we obtain the desired expressions

$$d = \langle L^{i,j} - n - \delta \rangle_{\sigma} \tag{8.59}$$
$$k = \langle (d - L^{i,j} + n + \delta)/\sigma \rangle_{N^i} \tag{8.60}$$

Finally, l is obtained as

$$l = \langle k + L^{i,j}/\sigma - 1 \rangle_{N^i} \tag{8.61}$$

Let $Z^{i,j}_{s,d}$ be defined by (8.56). Using (8.59), (8.60) and (8.61) we can formulate the algorithm as

Algorithm 8.3: $\boldsymbol{y}^{i,j} = \boldsymbol{H}^{i,j}\boldsymbol{x}^{j}$, $i < j$

```
For n = 0 to N^j - 1
    if |x_n| > ε then
        d = ⟨L^{i,j} - n - δ⟩_σ
        k = ⟨(d - L^{i,j} + n + δ)/σ⟩_{N^i}
        l = ⟨k + L^{i,j}/σ - 1⟩_{N^i}
        if k < l then
            y^{i,j}_{k:l} = y^{i,j}_{k:l} + x^j_n Z^{i,j}_{:,d}
        else                                        (wrap)
            y^{i,j}_{0:l} = y^{i,j}_{0:l} + x^j_n Z^{i,j}_{L^{i,j}/σ-l:L^{i,j}/σ-1,d}
            y^{i,j}_{k:N^i-1} = y^{i,j}_{k:N^i-1} + x^j_n Z^{i,j}_{0:L^{i,j}/σ-l-1,d}
        end
    end
end
```

8.4.3 Algorithm

The full algorithm for computing (8.49) is

Algorithm 8.4: Matrix-vector multiplication (CIRMUL)

```
y = 0
For j = 0 to λ
    For i = 0 to λ
        If i ≥ j then
            y^i = y^i + y^{i,j} computed with Algorithm 8.2
        else
            y^i = y^i + y^{i,j} computed with Algorithm 8.3
```

8.4.4 Computational work

We are now ready to look at the computational work required for the matrix-vector multiplication $\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}$. We take (8.49) as the point of departure and start by considering the typical block

$$\boldsymbol{H}^{i,j}\boldsymbol{x}^j$$

The length of $\boldsymbol{x}^j$ is N^j so for blocks on and below the diagonal of $\boldsymbol{H}$ $(i \geq j)$ in Algorithm 8.2 there are $L^{i,j}N^j$ multiplications and the same number of additions. Hence the work is

$$2L^{i,j}N^j$$

floating point operations. For blocks above the diagonal $(j > i)$ in Algorithm 8.3 there are $N^j L^{i,j}/\sigma = L^{i,j}N^i$ multiplications and additions, so the work here is

$$2L^{i,j}N^i$$

The total work can therefore be written as follows

$$\sum_{j=0}^{\lambda} 2L^{j,j}N^j + \sum_{i=1}^{\lambda}\sum_{j=0}^{i-1} 2L^{i,j}N^j + \sum_{j=1}^{\lambda}\sum_{i=0}^{j-1} 2L^{i,j}N^i$$

where the first sum corresponds to blocks on the diagonal, the second to blocks below the diagonal, and the third to blocks above the diagonal. Since $L^{i,j} = L^{j,i}$ we can swap the indices of the last double sum to find the identity

$$\sum_{j=1}^{\lambda}\sum_{i=0}^{j-1} 2L^{i,j}N^i = \sum_{i=1}^{\lambda}\sum_{j=0}^{i-1} 2L^{i,j}N^j$$

Hence we may write the total work of the matrix-vector multiplication with the wavelet transform of a circulant matrix as

$$\boxed{F_{\text{CIRMUL}} = 2\sum_{j=0}^{\lambda} L^{j,j}N^j + 4\sum_{i=1}^{\lambda}\sum_{j=0}^{i-1} L^{i,j}N^j} \tag{8.62}$$

We recall that

$$N^j = \begin{cases} N/2^{\lambda} & \text{for } j = 0 \\ N/2^{\lambda-j+1} & \text{for } 1 \leq j \leq \lambda \end{cases}$$

and that $L^{i,j}$ is given by the recurrence formulas

$$\begin{aligned} L^{j-1,j-1} &= \lceil L^{j,j}/2 \rceil + D - 1 \\ L^{i,j-1} &= L^{i,j} + 2^{i-j}(D-1) \\ L^{\lambda+1,\lambda+1} &= L \text{ (the bandwidth of the original matrix } A) \end{aligned}$$

Tables 8.2, 8.3, and 8.4 show F_{CIRMUL} evaluated for various values of L, D, λ, and N.

	$\lambda = 3, L = 5$			
N	$D = 2$	$D = 4$	$D = 6$	$D = 8$
32	784	1472	1808	2064
64	1568	3072	4576	5792
128	3136	6144	9280	12288
256	6272	12288	18560	24576
512	12544	24576	37120	49152
1024	25088	49152	74240	98304
2048	50176	98304	148480	196608

Table 8.2: The number of floating point operations F_{CIRMUL} as a function of N for different values of D.

	$N = 256, D = 4$			
λ	$L = 3$	$L = 4$	$L = 5$	$L = 6$
0	1536	2048	2560	3072
1	5120	5120	6144	6144
2	8448	8448	9216	9216
3	11520	11520	12288	12288
4	14592	14592	15360	15360
5	17664	17664	18432	18432

Table 8.3: The number of floating point operations F_{CIRMUL} shown for different values of λ and L. Note that $L = 2k$ and $L = 2k + 1$ ($k \in \mathbf{N}$) yield the same values for $\lambda > 0$. This is a direct consequence of the rounding done in (8.62).

	$N = 256, L = 5$			
λ	$D = 2$	$D = 4$	$D = 6$	$D = 8$
0	2560	2560	2560	2560
1	4096	6144	8192	10240
2	5120	9216	13312	17408
3	6272	12288	18560	24576
4	7360	15360	23680	31808
5	8416	18432	28736	38336

Table 8.4: The number of floating point operations F_{CIRMUL} shown for different values of λ and D.

Table 8.2 shows that F_{CIRMUL} depends linearly on N. Moreover, Tables 8.3 and 8.4 show that the computational work grows with the bandwidth L, the wavelet genus D, and the transform depth λ. Suppose that L is given. Then we see that the matrix-vector multiplication is most efficient if we take no steps of the wavelet transform ($\lambda = 0$). Consequently, any work reduction must be sought in truncation of the vector $\boldsymbol{x}$. This can be justified because $\boldsymbol{x}$ will often be a 1D wavelet transform of the same depth (λ) as the matrix $\boldsymbol{H}$. Therefore, depending on λ and the underlying application, we expect many elements in $\boldsymbol{x}$ to be close to zero so we may be able to discard them in order to reduce the computational work. Consider Table 8.3. If we take $N = 256$, $L = 3$, $\lambda = 4$ as an example, the question boils down to whether such a truncation of $\boldsymbol{x}$ can reduce 14592 operations to less than the 1536 operations required for $\lambda = 0$ (no transform). Assuming that the work depends linearly on the number of non-zero elements in $\boldsymbol{x}$, this means that $\boldsymbol{x}$ must be reduced by a factor of 10 (at least) before any work reduction is obtained.

8.5 Summary

We have derived an algorithm for computing the 2D wavelet transform of a circulant $N \times N$ matrix $\boldsymbol{A}$ in $\mathcal{O}(N)$ steps. More specifically we derived the bound

$$\boxed{F_{\text{CIRFWT}}(N) < 2DN(4 + \lambda)}$$

The algorithm works with a data structure that requires no more than $2\lambda N$ elements instead of N^2. If A is also banded the storage requirements are reduced further. Then we have shown how a banded matrix-vector multiplication using the proposed storage scheme can be implemented and the complexity of this algorithm was analyzed. It was found that the work needed to compute the matrix-vector multiplication is linear in N and grows with the bandwidth (L), the wavelet genus (D), and the depth of the wavelet transform λ. On the other hand, the work is reduced when small elements in $\boldsymbol{x}$ are discarded.

Chapter 9

Examples of wavelet-based PDE solvers

We consider here the task of solving partial differential equations in one space dimension using wavelets. We assume that the boundary conditions are periodic and make use of the periodized wavelets described in Section 2.3 and the material developed in Chapters 7 and 8. Sections 9.1 and 9.2 treat linear problems and serve to set the stage for Sections 9.3 and 9.4 which deal with nonlinear problems.

We begin by considering a **periodic boundary value problem** because it is a simple and illustrative example.

9.1 A periodic boundary value problem

Consider the 1D Helmholtz equation

$$\left.\begin{array}{rcl} -u'' + \alpha u &=& f(x) \\ u(x) &=& u(x+1) \end{array}\right\} \quad x \in \mathbf{R} \tag{9.1}$$

where $\alpha \in \mathbf{R}$ and $f(x) = f(x+1)$.

We look for 1-periodic solutions, so it suffices to consider $u(x)$ on the interval $0 \leq x < 1$.

9.1.1 Representation with respect to scaling functions

We begin the discretization of (9.1) by replacing $u(x)$ with the approximation

$$u_J(x) = \sum_{k=0}^{2^J-1} (c_u)_{J,k} \tilde{\phi}_{J,k}(x), \quad J \in \mathbf{N}_0 \tag{9.2}$$

Following the approach that lead to (7.9) we find that

$$u''_J(x) = \sum_{k=0}^{2^J-1} (c_u^{(2)})_{J,k} \tilde{\phi}_{J,k}(x) \tag{9.3}$$

where $(c_u^{(2)})_{J,k}$ is given as in (7.12), i.e.

$$(c_u^{(2)})_{J,k} = [\boldsymbol{D}^{(2)} \boldsymbol{c}_u]_k = \sum_{n=2-D}^{D-2} (c_u)_{J,\langle n+k \rangle_{2^J}} 2^{Jd} \Gamma_n^d, \quad k = 0, 1, \dots, 2^J - 1$$

with Γ_n^d defined in (7.1).

We can use the Galerkin method to determine the coefficients $(c_u)_{J,k}$. Multiplying (9.1) by $\tilde{\phi}_{J,l}(x)$ and integrating over the unit interval yields the relation

$$-\int_0^1 u''_J(x) \tilde{\phi}_{J,l}(x)\, dx + \alpha \int_0^1 u_J(x) \tilde{\phi}_{J,l}(x)\, dx = \int_0^1 f(x) \tilde{\phi}_{J,l}(x)\, dx$$

Using (9.2), (9.3), and the orthonormality of the periodized scaling functions (2.48) we get

$$-(c_u^{(2)})_{J,l} + \alpha (c_u)_{J,l} = (c_f)_{J,l}, \quad l = 0, 1, \dots, 2^J - 1$$

where

$$(c_f)_{J,l} = \int_0^1 f(x) \tilde{\phi}_{J,l}(x)\, dx \tag{9.4}$$

In vector notation this becomes

$$-\boldsymbol{c}_u^{(2)} + \alpha \boldsymbol{c}_u = \boldsymbol{c}_f \tag{9.5}$$

and using (7.13) we arrive at the linear system of equations

$$\boxed{\boldsymbol{A} \boldsymbol{c}_u = \boldsymbol{c}_f} \tag{9.6}$$

where

$$\boldsymbol{A} = -\boldsymbol{D}^{(2)} + \alpha \boldsymbol{I} \tag{9.7}$$

Alternatively, we can replace $\boldsymbol{D}^{(2)}$ by $\boldsymbol{D}^2$, where $\boldsymbol{D}$ is given in (7.14), and obtain

$$\boxed{\boldsymbol{A} \boldsymbol{c}_u = \boldsymbol{c}_f} \tag{9.8}$$

where

$$\boldsymbol{A} = -\boldsymbol{D}^2 + \alpha \boldsymbol{I} \tag{9.9}$$

Equations (9.6) and (9.8) represent the scaling function discretizations of (9.1). Hence this method belongs to Class 1 described in Section 7.1. We observe that (9.6) has a unique solution if α does not belong to the set of eigenvalues of $\boldsymbol{D}^{(2)}$. Similarly (9.8) has a unique solution if α does not belong to the set of eigenvalues of $\boldsymbol{D}^2$.

	$-\log_2(\|u-u_J\|_{J,\infty})$				
	$\boldsymbol{A}=-\boldsymbol{D}^{(2)}+\alpha\boldsymbol{I}$		$\boldsymbol{A}=-\boldsymbol{D}^2+\alpha\boldsymbol{I}$		
J	$D=6$	$D=8$	$D=4$	$D=6$	$D=8$
2	1.99	3.40	1.80	3.05	4.18
3	4.81	8.15	5.71	8.60	11.42
4	8.57	13.87	9.65	14.46	19.21
5	12.51	19.81	13.63	20.43	27.16
6	16.50	25.79	17.63	26.42	35.15
7	20.50	31.79	21.63	32.42	42.48
8	24.50	40.03	25.63	38.61	42.01
9	28.50	35.48	29.63	39.27	40.85

Table 9.1: The error $-\log_2(\|u-u_J\|_{J,\infty})$ shown for different values of J, D and the choice of differentiation matrix. Convergence rates are seen from the differences between successive measurements. The value $D=4$ is omitted in the case of (9.7) because the combination $D=4$, $d=2$ is invalid as described in Remark 7.1.

Accuracy

Let $f(x)=(4\pi^2+\alpha)\sin(2\pi x)$. The solution of (9.1) is then

$$u(x)=\sin(2\pi x)$$

Define

$$\|u\|_\infty = \max_{0\le x<1} |u(x)| \tag{9.10}$$

$$\|u\|_{J,\infty} = \max_{k=0,1,\ldots,2^J-1} \left|u(k/2^J)\right| \tag{9.11}$$

Table 9.1 shows how the error $\|u-u_J\|_{J,\infty}$ depends on J, D, and the choice of $\boldsymbol{D}^{(2)}$ or $\boldsymbol{D}^2$. Until the onset of rounding errors, the convergence rates obtained in Chapter 7 are recovered. More precisely,

$$\|u-u_J\|_{J,\infty} = \mathcal{O}\left(2^{-J(D-2)}\right) \quad \text{in the case of (9.7)}$$
$$\|u-u_J\|_{J,\infty} = \mathcal{O}\left(2^{-JD}\right) \quad \text{in the case of (9.9)}$$

Table 9.2 shows how the error $\|u-u_J\|_\infty$ depends on J, D, and the choice of $\boldsymbol{D}^{(2)}$ or $\boldsymbol{D}^2$. In this case, we find

$$\|u-u_J\|_\infty = \mathcal{O}\left(2^{-JD/2}\right) \quad \text{in the case of (9.7)}$$
$$\|u-u_J\|_\infty = \mathcal{O}\left(2^{-JD/2}\right) \quad \text{in the case of (9.9)}$$

	$-\log_2(\|u-u_J\|_\infty)$				
	$\boldsymbol{A}=-\boldsymbol{D}^{(2)}+\alpha\boldsymbol{I}$		$\boldsymbol{A}=-\boldsymbol{D}^2+\alpha\boldsymbol{I}$		
J	$D=6$	$D=8$	$D=4$	$D=6$	$D=8$
2	1.15	2.18	0.99	1.68	2.10
3	3.36	5.84	2.41	3.78	5.69
4	6.45	9.61	4.16	6.61	9.54
5	9.53	13.51	6.08	9.58	13.49
6	12.56	17.49	8.06	12.57	17.48
7	15.56	21.48	10.06	15.57	21.48
8	18.57	25.48	12.06	18.57	25.48
9	21.57	29.45	14.06	21.57	29.48

Table 9.2: The error $-\log_2(\|u-u_J\|_\infty)$ shown for different values of J, D and the differentiation matrix. Convergence rates are seen from the differences between successive measurements. The value $D=4$ is omitted in case of (9.7) because the combination $D=4, d=2$ is invalid as described in Remark 7.1.

which agrees with the approximation properties described in (4.5).

The high convergence observed in Table 9.1 means that the solution $u(x)$ is approximated to Dth order in the norm $\|\cdot\|_{J,\infty}$ even though the the subspace $\tilde{V}_J$ can only represent exactly polynomials up to degree $D/2-1$, as explained in Section 4.1. This phenomenon, known as *super convergence*, is also encountered in the finite element method, [WM85, p. 106], [AB84, p. 231].

9.1.2 Representation with respect to wavelets

Taking (9.6) as point of departure and using the relations

$$\begin{aligned} \boldsymbol{d}_u &= \boldsymbol{W}\boldsymbol{c}_u \\ \boldsymbol{d}_f &= \boldsymbol{W}\boldsymbol{c}_f \end{aligned}$$

yields

$$\boldsymbol{A}\boldsymbol{W}^T\boldsymbol{d}_u = \boldsymbol{W}^T\boldsymbol{d}_f$$

Let

$$\breve{\boldsymbol{A}} = \boldsymbol{W}\boldsymbol{A}\boldsymbol{W}^T = \boldsymbol{W}\left(-\boldsymbol{D}^{(2)}+\alpha\boldsymbol{I}\right)\boldsymbol{W}^T = -\breve{\boldsymbol{D}}^{(2)}+\alpha\boldsymbol{I}$$

where $\breve{\boldsymbol{D}}^{(2)}$ is defined according to (7.22). Then

$$\boxed{\breve{\boldsymbol{A}}\boldsymbol{d}_u = \boldsymbol{d}_f} \tag{9.12}$$

which is the wavelet discretization of (9.1). This method belongs to Class 2 described in Section 7.1. Hence, there is a potential for using wavelet compression to reduce the computational complexity of solving (9.12).

9.1.3 Representation with respect to physical space

Multiplying (9.5) by $\boldsymbol{T}$ and using (3.17) yields

$$-\boldsymbol{u}^{(2)} + \alpha \boldsymbol{u} = \boldsymbol{f}$$

where

$$\boldsymbol{f} = \boldsymbol{T}\boldsymbol{c}_f = \left\{ \left(P_{\tilde{V}_J} f \right)(x_l) \right\}_{l=0}^{2^J-1} \tag{9.13}$$

From the relation $\boldsymbol{u}^{(2)} = \boldsymbol{D}^{(2)}\boldsymbol{u}$ (7.15) we obtain the equation

$$\boxed{\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}} \tag{9.14}$$

where $\boldsymbol{A}$ is given by (9.7).

An important variant of (9.14) is obtained by redefining $\boldsymbol{f}$ as the vector

$$\boldsymbol{f} = \{ f(x_l) \}_{l=0}^{2^J-1}$$

This produces a collocation scheme for the solution of (9.1).

Because this method is essentially based on scaling functions and does not use wavelet compression it belongs to Class 1 described in Section 7.1.

9.1.4 Hybrid representation

We mention a second possibility for discretizing (9.1) using wavelets. It is simple and several authors follow this approach, e.g. [EOZ94, CP96, PW96]. This is essentially a combination of the approaches that lead to (9.14) and (9.12), and we proceed as follows: Multiplying (9.14) from the left by $\boldsymbol{W}$ and using the identity $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}$ one obtains

$$\boldsymbol{W}\boldsymbol{A}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{u} = \boldsymbol{W}\boldsymbol{f}$$

Defining the wavelet transformed vectors as

$$\begin{aligned} \breve{\boldsymbol{f}} &= \boldsymbol{W}\boldsymbol{f} \quad \text{and} \\ \breve{\boldsymbol{u}} &= \boldsymbol{W}\boldsymbol{u} \end{aligned}$$

then yields

$$\boxed{\breve{\boldsymbol{A}}\breve{\boldsymbol{u}} = \breve{\boldsymbol{f}}} \tag{9.15}$$

where $\breve{\boldsymbol{A}}$ is the same as in (9.12).

This approach bypasses the scaling coefficient representation and relies on the fact that the FWT can be applied directly to function values of f. Indeed, using this approach, the differentiation matrix $\boldsymbol{A}$ might as well be derived from finite differences, finite elements or spectral methods.

From Section 4.3 we know that the elements in $\breve{\boldsymbol{u}}$ will behave similarly as the true wavelet coefficients $\boldsymbol{d}_u$. Therefore, wavelet compression is as likely in this case as with the pure wavelet representation (9.12). Hence this method belongs to Class 2 described in Section 7.1.

9.2 The heat equation

We consider now the periodic initial-value problem for the heat equation

$$\left.\begin{aligned} u_t &= \nu u_{xx} + f(x), \;\; t > 0 \\ u(x,0) &= h(x) \\ u(x,t) &= u(x+1,t), \;\; t \geq 0 \end{aligned}\right\} \quad x \in \mathbf{R} \tag{9.16}$$

where ν is a positive constant, $f(x) = f(x+1)$ and $h(x) = h(x+1)$. The discretization strategies are time-dependent analogues of those in Section 9.1.

9.2.1 Representation with respect to scaling functions

We consider first the Galerkin method for (9.16) and proceed as in Section 9.1.1. The time-dependent version of (9.2) is

$$u_J(x,t) = \sum_{k=0}^{2^J-1} (c_u)_{J,k}(t)\tilde{\phi}_{J,k}(x) \tag{9.17}$$

After an analysis similar to that which lead to (9.5), we arrive at the Galerkin discretization of (9.16):

$$\begin{aligned} \frac{d}{dt}\boldsymbol{c}_u(t) &= \nu \boldsymbol{D}^{(2)}\boldsymbol{c}_u(t) + \boldsymbol{c}_f, \quad t > 0 \\ \boldsymbol{c}_u(0) &= \boldsymbol{c}_h \end{aligned} \tag{9.18}$$

where $\boldsymbol{c}_f$ is given by (9.4) and

$$(c_h)_{J,k} = \int_0^1 h(x)\tilde{\phi}_{J,k}(x)\,dx, \quad k = 0, 1, \ldots, 2^J - 1 \tag{9.19}$$

9.2.2 Representation with respect to wavelets

Multiplying (9.18) from the left by $\boldsymbol{W}$ and inserting the identity $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}$ yields

$$\frac{d}{dt}\boldsymbol{W}\boldsymbol{c}_u(t) = \nu\boldsymbol{W}\boldsymbol{D}^{(2)}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{c}_u(t) + \boldsymbol{W}\boldsymbol{c}_f, \quad t > 0$$

From the identities

$$\begin{aligned}
\boldsymbol{d}_u(t) &= \boldsymbol{W}\boldsymbol{c}_u(t)\\
\boldsymbol{d}_f(t) &= \boldsymbol{W}\boldsymbol{c}_f(t)\\
\breve{\boldsymbol{D}}^{(2)} &= \boldsymbol{W}\boldsymbol{D}^{(2)}\boldsymbol{W}^T
\end{aligned}$$

we then obtain

$$\frac{d}{dt}\boldsymbol{d}_u(t) = \nu\breve{\boldsymbol{D}}^{(2)}\boldsymbol{d}_u(t) + \boldsymbol{d}_f, \quad t > 0 \tag{9.20}$$

with the initial condition

$$\boldsymbol{d}_u(0) = \boldsymbol{W}\boldsymbol{c}_h$$

9.2.3 Representation with respect to physical space

Proceeding as above but this time multiplying (9.18) from the left by $\boldsymbol{T}$ yields

$$\frac{d}{dt}\boldsymbol{T}\boldsymbol{c}_u(t) = \nu\boldsymbol{T}\boldsymbol{D}^{(2)}\boldsymbol{T}^{-1}\boldsymbol{T}\boldsymbol{c}_u(t) + \boldsymbol{T}\boldsymbol{c}_f, \quad t > 0$$

Using the relations

$$\begin{aligned}
\boldsymbol{u}(t) &= \boldsymbol{T}\boldsymbol{c}_u(t)\\
\boldsymbol{f} &= \boldsymbol{T}\boldsymbol{c}_f \;=\; \left\{\left(P_{\tilde{V}_J}f\right)(x_l)\right\}_{l=0}^{2^J-1}\\
\boldsymbol{D}^{(2)} &= \boldsymbol{T}\boldsymbol{D}^{(2)}\boldsymbol{T}^{-1}
\end{aligned}$$

we find

$$\frac{d}{dt}\boldsymbol{u}(t) = \nu\boldsymbol{D}^{(2)}\boldsymbol{u}(t) + \boldsymbol{f}, \quad t > 0 \tag{9.21}$$

with the initial condition

$$\boldsymbol{u}(0) = \boldsymbol{h} = \left\{ \left(P_{\tilde{V}_J} h \right) (x_l) \right\}_{l=0}^{2^J-1} \tag{9.22}$$

Also in this case we can produce a collocation scheme by redefining

$$\begin{aligned} \boldsymbol{f} &= \{f(x_l)\}_{l=0}^{2^J-1} \\ \boldsymbol{h} &= \{h(x_l)\}_{l=0}^{2^J-1} \end{aligned}$$

9.2.4 Hybrid representation

Multiplying (9.21) from the left with $\boldsymbol{W}$ and proceeding as in Section 9.1.4 yields

$$\frac{d}{dt}\breve{\boldsymbol{u}}(t) = \nu \breve{\boldsymbol{D}}^{(2)} \breve{\boldsymbol{u}}(t) + \breve{\boldsymbol{f}}, \quad t > 0 \tag{9.23}$$

with the initial condition

$$\breve{\boldsymbol{u}}(0) = \boldsymbol{W}\boldsymbol{h}$$

9.2.5 Time stepping in the wavelet domain

A number of time-stepping schemes are available for the system (9.20). We consider here the backward Euler method defined by

$$\frac{(\boldsymbol{d}_u)_{n+1} - (\boldsymbol{d}_u)_n}{\Delta t} = \nu \breve{\boldsymbol{D}}^{(2)} (\boldsymbol{d}_u)_{n+1} + \boldsymbol{d}_f \tag{9.24}$$

where $(\boldsymbol{d}_u)_n = \boldsymbol{d}_u(n\Delta t)$. This leads to the recursion

$$\begin{aligned} (\boldsymbol{d}_u)_{n+1} &= \breve{\boldsymbol{A}}^{-1} \left((\boldsymbol{d}_u)_n + \Delta t \boldsymbol{d}_f \right), \quad n = 0, 1, \ldots, n_1 - 1 \\ (\boldsymbol{d}_u)_0 &= \boldsymbol{d}_u(0) \end{aligned} \tag{9.25}$$

where $n_1 \in \mathbf{N}$ and

$$\breve{\boldsymbol{A}} = \boldsymbol{I} - \nu \Delta t \breve{\boldsymbol{D}}^{(2)}$$

The backward Euler time-stepping schemes for (9.18), (9.21), and (9.23) are completely analogous.

The matrix $\breve{\boldsymbol{A}}$ has the characteristic "finger-band" pattern shown in Figure 7.1, and it turns out that so does $\breve{\boldsymbol{A}}^{-1}$ for small Δt when small entries are removed [CP96]. Figure 9.1 shows an example of $\breve{\boldsymbol{A}}^{-1}$ where elements smaller than 10^{-11} have been removed.

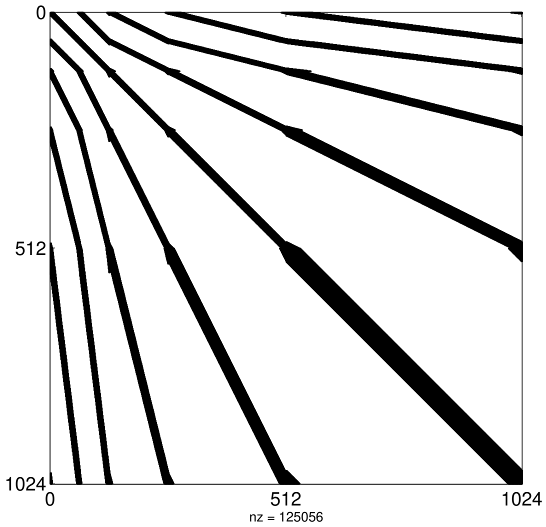


Figure 9.1: $\breve{\boldsymbol{A}}^{-1}$ where elements smaller than 10^{-11} have been removed. $D = 8$, $\lambda = 3$, $J = 10$, $\Delta t = 2^{-11}$, and $\nu = 0.05/\pi$.

Moreover, we can expect many elements in $\boldsymbol{d}_u$ and $\boldsymbol{d}_f$ to be small by virtue of Theorem 2.5. Consequently, there is a potential for compression of both the coefficient matrix as well as the solution vector. This is true for both (9.20) and (9.23) but we give the algorithm for (9.20) only since the case (9.23) is completely analogous.

Let trunc$(\boldsymbol{d}, \varepsilon)$ be a function that returns only the significant elements in a given vector $\boldsymbol{d}$, i.e. let

$$\text{trunc}(\boldsymbol{d}, \varepsilon) = \{d_{j,k}, \;\; |d_{j,k}| > \varepsilon\}$$

Similarly, let

$$\text{trunc}(\boldsymbol{A}, \varepsilon) = \{[\boldsymbol{A}]_{m,n}, \;\; |[\boldsymbol{A}]_{m,n}| > \varepsilon\}$$

Let ε_V be the compression threshold for the vector and ε_M be the compression threshold for the matrix and define $\boldsymbol{d}^{\varepsilon_V} = \text{trunc}(\boldsymbol{d}, \varepsilon_V)$ and $\breve{\boldsymbol{A}}^{\varepsilon_M} = \text{trunc}(\breve{\boldsymbol{A}}, \varepsilon_M)$.

Following ideas given in [CP96] we can now give a computational procedure for computing (9.25) using wavelet compression.

Algorithm 9.1

$$\begin{aligned}
1: \quad (\breve{\boldsymbol{A}}^{-1})^{\varepsilon_M} &\leftarrow \mathsf{trunc}\,(\boldsymbol{W}\boldsymbol{A}^{-1}\boldsymbol{W}^T, \varepsilon_M) \\
2: \quad \boldsymbol{d}_f^{\varepsilon_V} &\leftarrow \mathsf{trunc}\,(\boldsymbol{W}\boldsymbol{c}_f, \varepsilon_V) \\
3: \quad (\boldsymbol{d}_u)_0^{\varepsilon_V} &\leftarrow \mathsf{trunc}\,(\boldsymbol{W}\boldsymbol{c}_h, \varepsilon_V)
\end{aligned}$$

for $n = 0, 1, \ldots, n_1 - 1$

$$4: \quad (\boldsymbol{d}_u)_{n+1} \;\leftarrow\; (\breve{\boldsymbol{A}}^{-1})^{\varepsilon_M} \left((\boldsymbol{d}_u)_n^{\varepsilon_V} + \Delta t \boldsymbol{d}_f^{\varepsilon_V}\right)$$

$$5: \quad (\boldsymbol{d}_u)_{n+1}^{\varepsilon_V} \;\leftarrow\; \mathsf{trunc}\left((\boldsymbol{d}_u)_{n+1}, \varepsilon_V\right)$$

end

$$6: \quad (\boldsymbol{c}_u)_{n_1} \;\leftarrow\; \boldsymbol{W}^T (\boldsymbol{d}_u)_{n_1}^{\varepsilon_V}$$

$$7: \quad (\boldsymbol{u})_{n_1}^{\varepsilon_M, \varepsilon_V} \;\leftarrow\; \boldsymbol{T}(\boldsymbol{c}_u)_{n_1}$$

A few comments are appropriate:

- **Step 1:** We have

$$\breve{\boldsymbol{A}}^{-1} = \left(\boldsymbol{W}\boldsymbol{A}\boldsymbol{W}^T\right)^{-1} = \left(\boldsymbol{W}^T\right)^{-1} \boldsymbol{A}^{-1}\boldsymbol{W}^{-1} = \boldsymbol{W}\boldsymbol{A}^{-1}\boldsymbol{W}^T$$

  Hence, step 1 can be done using the 2D FWT (3.38). However, since $\boldsymbol{A}$ is circulant then $\boldsymbol{A}^{-1}$ is also circulant by Theorem C.6 and it can be computed efficiently using the FFT as explained in Appendix C. Consequently, $\breve{\boldsymbol{A}}^{-1}$ can be computed and stored efficiently using Algorithm 8.1.

- **Steps $2, 3$:** $\boldsymbol{c}_h, \boldsymbol{c}_f$ are both computed according to (3.17), i.e. by the quadrature formulas

$$\begin{aligned} \boldsymbol{c}_f &= \boldsymbol{T}^{-1}\boldsymbol{f} \\ \boldsymbol{c}_u &= \boldsymbol{T}^{-1}\boldsymbol{h} \end{aligned}$$

  where

$$\begin{aligned} \boldsymbol{f} &= \{f(x_k)\}_{k=0}^{2^J-1} \\ \boldsymbol{h} &= \{h(x_k)\}_{k=0}^{2^J-1} \end{aligned}$$

- **Step 4:** It is essential for the success of this algorithm that the computation of the matrix-vector product fully exploits the compressed form of both matrix and vectors. This can be done, for example, using Algorithm 8.4. In [CP96, p. 7] fast multiplication is based on a general sparse format for both matrix and vector.

- **Steps $6, 7$:** Finally, the computed vector of wavelet coefficients is transformed back into the physical domain. We denote the computed solution $\boldsymbol{u}_{n_1}^{\varepsilon_M, \varepsilon_V}$, because it depends on both thresholds.

To test Algorithm 9.1 we have applied it to the problem

$$\left.\begin{array}{rcl} u_t & = & \nu u_{xx} + \sin(2\pi x), \quad t > 0 \\ u(x,0) & = & 0 \end{array}\right\} \quad x \in \mathbf{R} \tag{9.26}$$

The numerical parameters are $J = 10$, $\Delta t = 1/2^{10}$, $n_1 = 2^{10}$ (making $0 \le t \le 1$), $\nu = 0.01/\pi$, $\lambda = 3$, and $D = 8$. The vector $\boldsymbol{u}_{n_1}^{\varepsilon_M,\varepsilon_V}$ is the result of Algorithm 9.1 given the thresholds ε_M and ε_V. Hence, we define the relative compression error as

$$E^{\varepsilon_M,\varepsilon_V} = \frac{\left\|\boldsymbol{u}_{n_1}^{\varepsilon_M,\varepsilon_V} - \boldsymbol{u}_{n_1}^{0,0}\right\|_\infty}{\left\|\boldsymbol{u}_{n_1}^{0,0}\right\|_\infty}$$

where

$$\|\boldsymbol{u}\|_\infty = \max_{k=0,1,\ldots,2^J-1} |u_l|$$

Table 9.3 shows the percentage of significant elements in $(\breve{\boldsymbol{A}}^{-1})^{\varepsilon_M}$ and $(\boldsymbol{d}_u)_{n_1}^{\varepsilon_V}$ for various values of ε_V and ε_M. Moreover, the relative error introduced by compression $E^{\varepsilon_M,\varepsilon_V}$ is given for each case. It is seen that significant compression can be achieved in both the matrix and the solution vector.

$\varepsilon_V = 0$	% elem		$\varepsilon_M = 0$	% elem	
ε_M	$(\breve{\boldsymbol{A}}^{-1})^{\varepsilon_M}$	$E^{\varepsilon_M,\varepsilon_V}$	ε_V	$(\boldsymbol{d}_u)_{n_1}^{\varepsilon_V}$	$E^{\varepsilon_M,\varepsilon_V}$
10^{-11}	19.06	10^{-11}	10^{-11}	95.02	10^{-11}
10^{-10}	16.56	10^{-12}	10^{-10}	49.12	10^{-10}
10^{-9}	14.08	$10^{-10.5}$	10^{-9}	37.79	10^{-9}
10^{-8}	11.59	10^{-9}	10^{-8}	22.17	10^{-8}
10^{-7}	9.28	$10^{-7.5}$	10^{-7}	11.82	10^{-7}
10^{-6}	7.20	10^{-6}	10^{-6}	5.96	10^{-6}
10^{-5}	5.18	$10^{-4.5}$	10^{-5}	4.20	10^{-5}

Table 9.3: Percentage of elements retained in $(\breve{\boldsymbol{A}}^{-1})^{\varepsilon_M}$ and $(\boldsymbol{d}_u)_{n_1}^{\varepsilon_V}$ together with the resulting compression error.

9.3 The nonlinear Schrödinger equation

We consider now the periodic initial-value problem for the **nonlinear Schrödinger equation**:

$$\left.\begin{array}{rcll} u_t & = & \mathcal{L}u + i\gamma\,|u|^2\,u, & t > 0 \\ u(x,0) & = & h(x) & \\ u(x,t) & = & u(x+L,t), & t \geq 0 \end{array}\right\} x \in \mathbf{R} \tag{9.27}$$

where $h(x) = h(x+L)$ and

$$\mathcal{L} = -\frac{i}{2}\beta_2\frac{\partial^2}{\partial x^2} - \frac{\alpha}{2}$$

This equation describes the propagation of a pulse envelope in an optical fiber. β_2 is a dispersion parameter, α is a measure of intensity loss, and γ is the magnitude of the nonlinear term which counteracts dispersion for certain waveforms [Agr89, FJAF78, LBA81, PAP86]. Sometimes the term $\frac{1}{6}\beta_3\frac{\partial^3}{\partial x^3}$, which represents third order dispersion, is added to $\mathcal{L}$.

Because of periodicity we can restrict the spatial domain of (9.27) to the interval $[-L/2, L/2[$. Let $N = 2^J$ and define a grid consisting of the points

$$x_l = \left(\frac{l}{N} - \frac{1}{2}\right)L, \quad l = 0, 1, \ldots, N-1$$

Define the vector $\boldsymbol{u}(t)$ such that

$$u_l(t) = u_J(x_l, t), \quad l = 0, 1, \ldots, N-1$$

where $u_J(x,t)$ is an approximate solution of (9.27) of the form (9.17). Define similarly the vectors $\boldsymbol{u}^{(2)}(t)$ and $\boldsymbol{f}(t)$ as

$$\begin{array}{rcl} u_l^{(2)}(t) & = & u_J''(x_l, t), \quad l = 0, 1, \ldots, N-1 \\ f_l(t) & = & f(x_l, t), \quad l = 0, 1, \ldots, N-1 \end{array}$$

Hence the elements in $\boldsymbol{u}$ approximate function values of u in the interval $[-L/2, L/2[$. With regard to our using the interval $[L/2, L/2[$ instead of $[0, 1[$ we mention that the mappings $\boldsymbol{Tc}$ and $\boldsymbol{Wc}$ are unchanged as described in Section 3.2.4 but the differentiation matrix must be scaled according to (7.18). Hence, we arrive at the following initial problem for $\boldsymbol{u}$ formulated with respect to physical space:

$$\begin{array}{rcl} \dfrac{d}{dt}\boldsymbol{u}(t) & = & \boldsymbol{L}\boldsymbol{u}(t) + \boldsymbol{N}(\boldsymbol{u}(t))\,\boldsymbol{u}(t), \quad t \geq 0 \\ \boldsymbol{u}(0) & = & \boldsymbol{h} \equiv [h(x_0), h(x_1), \ldots, h(x_{N-1})]^T \end{array} \tag{9.28}$$

where

$$\begin{aligned} \boldsymbol{L} &= -\frac{i}{2}\beta_2\frac{\boldsymbol{D}^{(2)}}{L^2} - \frac{\alpha}{2}\boldsymbol{I} \\ \boldsymbol{N}(\boldsymbol{u}(t)) &= i\gamma \mathrm{diag}(|u_l(t)|^2, \quad l = 0, 1, \ldots, N-1) \end{aligned}$$

9.3.1 Wavelet split-step method

A well established time-stepping scheme for solving (9.27) in optics applications is the **split-step method** which is also known as the beam propagation method. Traditionally it is applied in conjunction with a spatial discretization scheme based on a Fourier spectral method in which case it is known as the Fourier split-step method, or split-step FFT, (See, for example, [Agr89, p. 44–48] or [New92, p. 413–423]). However, we will use split-stepping with the spatial discretization described above and call it the **wavelet split-step method**. A similar approach has been investigated in [GL94, PW96].

The split-step method derives from the fact that the solution of problem (9.28) satisfies the identity

$$\boldsymbol{u}(t+\Delta t) = \exp\left(\Delta t \boldsymbol{L} + \int_t^{t+\Delta t} \boldsymbol{N}(\boldsymbol{u}(\tau))\, d\tau\right) \boldsymbol{u}(t)$$

We now introduce the approximations

$$\int_t^{t+\Delta t} \boldsymbol{N}(\boldsymbol{u}(\tau))\, d\tau \approx \frac{\Delta t}{2}[\boldsymbol{N}(\boldsymbol{u}(t)) + \boldsymbol{N}(\boldsymbol{u}(t+\Delta t))]$$

and

$$\begin{aligned} &\exp\left(\Delta t \boldsymbol{L} + \frac{\Delta t}{2}[\boldsymbol{N}(\boldsymbol{u}(t)) + \boldsymbol{N}(\boldsymbol{u}(t+\Delta t))]\right) \approx \\ &\quad \exp\left(\frac{\Delta t}{2}\boldsymbol{L}\right) \exp\left(\frac{\Delta t}{2}[\boldsymbol{N}(\boldsymbol{u}(t)) + \boldsymbol{N}(\boldsymbol{u}(t+\Delta t))]\right) \exp\left(\frac{\Delta t}{2}\boldsymbol{L}\right) \end{aligned}$$

Using these we obtain the time-stepping procedure

$$\begin{aligned} \boldsymbol{u}_{n+1} &= \boldsymbol{E} \exp\left(\frac{\Delta t}{2}[\boldsymbol{N}(\boldsymbol{u}_n) + \boldsymbol{N}(\boldsymbol{u}_{n+1})]\right) \boldsymbol{E}\boldsymbol{u}_n, \quad n = 0, 1, \ldots, n_1 - 1 \\ \boldsymbol{u}_0 &= \boldsymbol{h} \end{aligned} \tag{9.29}$$

where $\boldsymbol{u}_n = \boldsymbol{u}(n\Delta t)$, $\boldsymbol{h}$ is the vector defined in (9.28), and where

$$\boldsymbol{E} = \exp\left(\frac{\Delta t}{2}\boldsymbol{L}\right)$$

The vector $\boldsymbol{u}_{n+1}$ appears also on the right-hand side of (9.29) so each time step requires an iterative procedure. Ours is the following: For each n let

$$\boldsymbol{u}_{n+1}^{(0)} = \boldsymbol{u}_n$$

and iterate until convergence,

$$\boldsymbol{u}_{n+1}^{(q+1)} = \boldsymbol{E} \exp\left(\frac{\Delta t}{2}[\boldsymbol{N}(\boldsymbol{u}_n) + \boldsymbol{N}(\boldsymbol{u}_{n+1}^{(q)})]\right) \boldsymbol{E}\boldsymbol{u}_n, \quad q = 0, 1, \ldots$$

Defining the function

$$\boxed{\text{linstep}(\boldsymbol{v}) = \boldsymbol{E}\boldsymbol{v}}$$

we have the following algorithm for (9.29):

Algorithm 9.2: Split-step method

1: $\boldsymbol{u}_0 \leftarrow \boldsymbol{h} = \{h(x_k)\}_{k=0}^{2^J-1}$
2: $\boldsymbol{E} \leftarrow \exp\left(\frac{\Delta t}{2}\boldsymbol{L}\right)$
3: $\boldsymbol{N}_1 \leftarrow \boldsymbol{N}(\boldsymbol{u}_0)$
    for $n = 0, 1, \ldots, n_1 - 1$
4:     $\boldsymbol{v}_n \leftarrow \text{linstep}(\boldsymbol{u}_n)$
5:     $\boldsymbol{N}_0 \leftarrow \boldsymbol{N}_1$
        Iterate until convergence
6:         $\boldsymbol{u}_{n+1} \leftarrow \text{linstep}\left(\exp\left(\frac{\Delta t}{2}[\boldsymbol{N}_0 + \boldsymbol{N}_1]\right)\boldsymbol{v}_n\right)$
7:         $\boldsymbol{N}_1 \leftarrow \boldsymbol{N}(\boldsymbol{u}_{n+1})$

9.3.2 Matrix exponential of a circulant matrix

$\boldsymbol{N}(\boldsymbol{u}_n)$ is a diagonal matrix for every value of n, so the exponentiation required in step 6 of Algorithm 9.2 can be done cheaply. The matrix $\boldsymbol{L}$, on the other hand, is not diagonal but it is circulant. Hence, with regard to step 2, we need a procedure for computing the matrix exponential

$$\boldsymbol{E} = e^{\boldsymbol{A}} \tag{9.30}$$

where $\boldsymbol{A}$ is the circulant matrix

$$\boldsymbol{A} = \frac{\Delta t}{2}\boldsymbol{L}$$

Standard methods for computing (9.30) are mostly based on matrix multiplications [GL89] which have complexity $\mathcal{O}(N^3)$. Hence the computation of (9.30) can be very expensive.

However, by using the fact that $\boldsymbol{A}$ is circulant one can compute (9.30) in $\mathcal{O}(N \log_2 N)$ floating point operations. From Theorem C.4 we have the factorization

$$\boldsymbol{A} = \boldsymbol{F}_N^{-1} \boldsymbol{\Lambda}_a \boldsymbol{F}_N$$

where $\boldsymbol{F}_N$ is the Fourier matrix defined in (C.3), $\boldsymbol{\Lambda}_a = \text{diag}(\hat{\boldsymbol{a}})$, $\hat{\boldsymbol{a}} = \boldsymbol{F}_N \boldsymbol{a}$, and $\boldsymbol{a}$ is the first column in $\boldsymbol{A}$. By a standard matrix identity

$$\boldsymbol{E} = \exp(\boldsymbol{A}) = \boldsymbol{F}_N^{-1} \exp(\boldsymbol{F}_N \boldsymbol{A} \boldsymbol{F}_N^{-1}) \boldsymbol{F}_N = \boldsymbol{F}_N^{-1} \exp(\boldsymbol{\Lambda}_a) \boldsymbol{F}_N$$

The matrix $\exp(\boldsymbol{\Lambda}_a)$ is diagonal so we know from Theorem C.4 that $\boldsymbol{E}$ is circulant. Hence we can write

$$\boldsymbol{E} = \boldsymbol{F}_N^{-1} \boldsymbol{\Lambda}_e \boldsymbol{F}_N$$

where $\boldsymbol{\Lambda}_e = \text{diag}(\hat{\boldsymbol{e}})$, $\hat{\boldsymbol{e}} = \boldsymbol{F}_N \boldsymbol{e}$, and $\boldsymbol{e}$ is the first column in $\boldsymbol{E}$. Equating $\boldsymbol{\Lambda}_e$ and $\exp(\boldsymbol{\Lambda}_a)$ yields

$$\hat{\boldsymbol{e}} = [e^{\hat{a}_0}, e^{\hat{a}_1}, \ldots, e^{\hat{a}_{N-1}}]^T$$

Hence we have the following fast algorithm for computing (9.30):

Algorithm 9.3: Exponential of circulant matrix A

$\boldsymbol{a}$	$\leftarrow$	$[\boldsymbol{A}]_{:,1}$	! Extract first column of $\boldsymbol{A}$
$\hat{\boldsymbol{a}}$	$\leftarrow$	$\boldsymbol{F}_N \boldsymbol{a}$	! FFT
$\hat{\boldsymbol{e}}$	$\leftarrow$	$\{\exp(\hat{a}_k)\}_{k=0}^{N-1}$	! Pointwise exponentiation
$\boldsymbol{e}$	$\leftarrow$	$\boldsymbol{F}_N^{-1} \hat{\boldsymbol{e}}$	! IFFT
$[\boldsymbol{E}]_{m,n}$	$\leftarrow$	$e_{\langle m-n \rangle_N}, \quad m, n = 0, 1, \ldots, N-1$	! Assemble $\boldsymbol{E}$

The matrices $\boldsymbol{A}$ and $\boldsymbol{E}$ need not be stored explicitly. Hence the first and the last steps here can be omitted in practice. In step 2 of Algorithm 9.2, where we use Algorithm 9.3, only $\boldsymbol{e}$ is computed. The computational complexity follows from that of the FFT and its inverse.

In the problem at hand $\boldsymbol{A} = \frac{\Delta t}{2} \boldsymbol{L}$, where $\boldsymbol{L}$ is circulant and banded. It follows from the definition of the matrix exponential

$$\boldsymbol{E} = \exp(\boldsymbol{A}) = \sum_{k=0}^{\infty} \boldsymbol{A}^k / k! \tag{9.31}$$

that $\boldsymbol{E}$ is dense. However, it turns out that many entries in $\boldsymbol{E}$ are very small and that $\boldsymbol{E}$ assumes circulant band form if truncated with a small threshold. In [GL94, PW96] $\boldsymbol{E}$ is approximated by truncating the series (9.31) to a few terms. This also yields a circulant band form, but the process is less exact and not significantly faster than the one presented here.

9.3.3 Compression

We can exploit the compression potential of wavelets by an appropriate implementation of linstep. Let

$$\breve{\boldsymbol{E}} = \boldsymbol{W}\boldsymbol{E}\boldsymbol{W}^T \tag{9.32}$$

and

$$(\breve{\boldsymbol{E}})^{\varepsilon_M} = \text{trunc}(\breve{\boldsymbol{E}}, \varepsilon_M)$$

Then the product $\boldsymbol{E}\boldsymbol{v}$ that is computed by linstep can be formulated as

$$\boldsymbol{E}\boldsymbol{v} = \boldsymbol{W}^T\breve{\boldsymbol{E}}\boldsymbol{W}\boldsymbol{v} = \boldsymbol{W}^T\breve{\boldsymbol{E}}\breve{\boldsymbol{v}} \approx \boldsymbol{W}^T(\breve{\boldsymbol{E}})^{\varepsilon_M}\breve{\boldsymbol{v}}^{\varepsilon_V}$$

where $\breve{\boldsymbol{v}}^{\varepsilon_V} = \text{trunc}(\breve{\boldsymbol{v}}, \varepsilon_V)$. Hence we obtain the following version of linstep which can be used in step 4 and 6 of Algorithm 9.2.

Algorithm 9.4: $\boldsymbol{u} \leftarrow \text{linstep}(\boldsymbol{v})$

$$\begin{aligned} \breve{\boldsymbol{v}} &\leftarrow \boldsymbol{W}\boldsymbol{v} \\ (\breve{\boldsymbol{v}})^{\varepsilon_V} &\leftarrow \text{trunc}(\breve{\boldsymbol{v}}, \varepsilon_V) \\ \breve{\boldsymbol{u}} &\leftarrow (\breve{\boldsymbol{E}})^{\varepsilon_M}(\breve{\boldsymbol{v}})^{\varepsilon_V} \\ \boldsymbol{u} &\leftarrow \boldsymbol{W}^T\breve{\boldsymbol{u}} \end{aligned}$$

Since $\boldsymbol{E}$ is circulant, $\breve{\boldsymbol{E}}$ can be computed efficiently using Algorithm 8.1. Since $\boldsymbol{E}$ is dense $\breve{\boldsymbol{E}}$ is dense too, but under truncation, even with very small thresholds, it assumes the finger-band pattern associated with differential operators in wavelet bases. Figure 9.2 shows $\breve{\boldsymbol{E}}$ truncated to machine precision for $\lambda = 0, 1, 2, 3, 4, 5$, the case $\lambda = 0$ corresponding to $\boldsymbol{E}$.

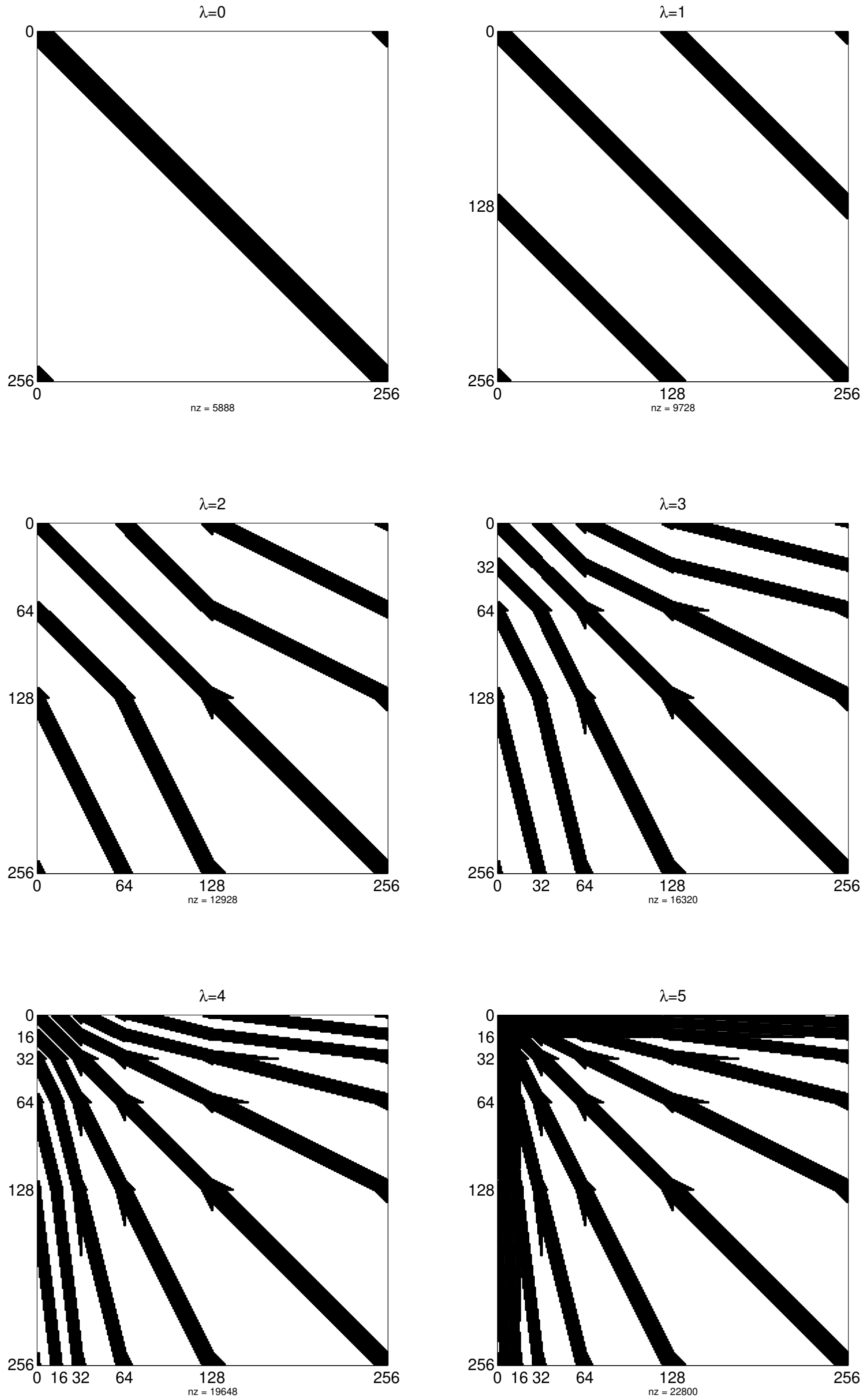


Figure 9.2: $\breve{E}^{\varepsilon_M}$ for $\lambda = 0, 1, 2, 3, 4, 5$ truncated to the precision $\varepsilon_M = 2.2204 \times 10^{-16}$. The remaining parameters are $D = 8$, $J = 8$, $\Delta t = 1/2^{10}$, $L = 60$, $\beta_2 = -2$, and $\alpha = 0$.

λ	% elem. retained $\breve{\boldsymbol{E}}^{\varepsilon_M}$	$\left(\breve{\boldsymbol{E}}^*\right)^{\varepsilon_M}$
0	8.98	12.11
1	14.84	17.97
2	19.73	22.46
3	24.90	27.34
4	29.98	32.42
5	34.79	37.13

Table 9.4: The percentage of elements retained in $\breve{\boldsymbol{E}}^{\varepsilon_M}$ and $(\breve{\boldsymbol{E}}^*)^{\varepsilon_M}$ for $\varepsilon_M = 2.2204 \times 10^{-16}$, $D = 8$, $J = 8$, $\Delta t = 1/2^{10}$, $L = 60$, $\beta_2 = -2$, $\alpha = 0$. It is seen that the compression potential of $\breve{\boldsymbol{E}}^*$ is slightly lower than that of $\breve{\boldsymbol{E}}$.

Recall from Chapter 7 that the differentiation process has higher convergence if $\boldsymbol{D}^{(2)}$ is replaced by $\boldsymbol{D}^2$, but that this comes at the cost of a larger bandwidth. We want to compare these two choices with respect to compression potential. Let

$$\begin{aligned}
\boldsymbol{L}^* &= -\frac{i}{2}\beta_2\frac{\boldsymbol{D}^2}{L^2} - \frac{\alpha}{2}\boldsymbol{I} \\
\boldsymbol{E}^* &= \exp\left(\frac{\Delta t}{2}\boldsymbol{L}^*\right) \\
\breve{\boldsymbol{E}}^* &= \boldsymbol{W}\boldsymbol{E}^*\boldsymbol{W}^T
\end{aligned}$$

The percentages of elements retained in $\breve{\boldsymbol{E}}$ and $\breve{\boldsymbol{E}}^*$ with truncation to machine precision are shown in Table 9.4 for $\lambda = 0, 1, 2, 3, 4, 5$.

Figure 9.3 shows $\breve{\boldsymbol{E}}^{\varepsilon_M}$ for different choices of ε_M and Table 9.5 shows the corresponding percentages of elements retained in $\breve{\boldsymbol{E}}$ and $\breve{\boldsymbol{E}}^*$. It is observed that the compression potential of $\boldsymbol{E}^*$ is slightly lower than that of $\boldsymbol{E}$, so we will use only $\boldsymbol{E}$ in the following.

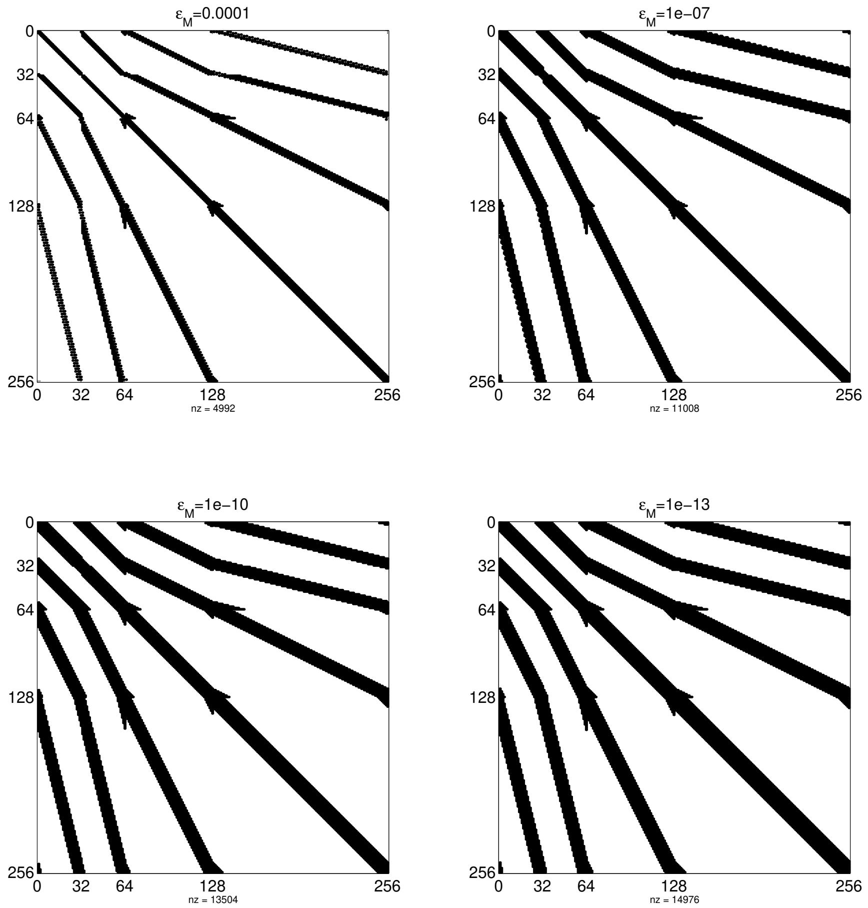


Figure 9.3: $\breve{\boldsymbol{E}}^{\varepsilon_M}$ for $\varepsilon_M = 10^{-4}, 10^{-7}, 10^{-10}, 10^{-13}$. The remaining parameters are $D = 8$, $J = 8$, $\lambda = 3$, $\Delta t = 1/2^{10}$, $L = 60$, $\beta_2 = -2$, and $\alpha = 0$.

	% elem. retained	
ε_M	$\breve{\boldsymbol{E}}^{\varepsilon_M}$	$\left(\breve{\boldsymbol{E}}^*\right)^{\varepsilon_M}$
10^{-4}	7.62	6.25
10^{-7}	16.80	17.38
10^{-10}	20.61	21.39
10^{-13}	22.85	24.61

Table 9.5: The percentage of elements retained in $\breve{\boldsymbol{E}}^{\varepsilon_M}$ and $(\breve{\boldsymbol{E}}^*)^{\varepsilon_M}$ for $\varepsilon_M = 10^{-4}, 10^{-7}, 10^{-10}, 10^{-13}$. The remaining parameters are $D = 8$, $J = 8$, $\lambda = 3$, $\Delta t = 1/2^{10}$, $L = 60$, $\beta_2 = -2$, and $\alpha = 0$. Hence, the first column corresponds to the matrices shown in Figure 9.3.

Thresholds			% elem. retained		
ε_M	ε_V	$E^{\varepsilon_M,\varepsilon_V}$	$\breve{\boldsymbol{E}}^{\varepsilon_M}$	$\breve{\boldsymbol{u}}_{n_1}^{\varepsilon_V}$	CPU time (s)
0	0	2.23×10^{-6}	100	100	963
10^{-15}	0	2.23×10^{-6}	38	100	541
10^{-11}	10^{-15}	2.23×10^{-6}	4	100	333
10^{-11}	10^{-12}	2.23×10^{-6}	4	89	257
10^{-11}	10^{-9}	2.24×10^{-6}	4	30	173
10^{-11}	10^{-7}	6.62×10^{-6}	4	16	141
10^{-11}	10^{-6}	6.88×10^{-5}	4	10	126

Table 9.6: Error, compression and execution time data for a problem with known solution (compression of $\breve{\boldsymbol{u}}_{n_1}$ is given for $n_1 = 2000$).

The wavelet split-step method as defined by Algorithm 9.2 and the implementation of linstep given in Algorithm 9.4 has been implemented in Fortran 90 on a CRAY C92A. Algorithm 9.3 was used for computing the matrix exponential $\boldsymbol{E} = \exp\left(\frac{\Delta t}{2}\boldsymbol{L}\right)$, Algorithm 8.1 was used for (9.32) using the data structure described in Section 8.2.5, and Algorithm 8.4 was used for computing the product

$$(\breve{\boldsymbol{E}})^{\varepsilon_M}(\breve{\boldsymbol{v}})^{\varepsilon_V}$$

Finally, the transforms $\boldsymbol{W}\boldsymbol{v}$ and $\boldsymbol{W}^T\breve{\boldsymbol{v}}$ were implemented using the FWT and the IFWT defined in Section 3.3. Hence, the algorithm has the complexity $\mathcal{O}(N)$ regarding both storage and floating point operations.

To test the performance of this algorithm, we have applied it to a case of problem (9.27) with a known solution. The data are $\beta_2 = -2$, $\alpha = 0$, $\gamma = 2$ and $h(x) = e^{ix}\text{sech}(x)$, which yield the solution $u(x,t) = e^{ix}\text{sech}(x - 2t)$. The numerical data are $L = 60$, $J = 11$ (making $N = 2048$), $\lambda = 3$, $n_1 = 2000$, and $\Delta t = 0.001$. The wavelets used were those of genus $D = 8$. Table 9.6 shows the relative error, percentage of compression achieved in $\breve{\boldsymbol{E}}$ and $\breve{\boldsymbol{u}}_{n_1}$, and measured CPU time of n_1 time steps as functions of the threshold parameters ε_M and ε_V. The relative error is defined here as

$$E^{\varepsilon_M,\varepsilon_V} = \frac{\max_{k=0,1,\ldots,2^J-1} |[\boldsymbol{u}_{n_1}]_k - u(x_k,1)|}{\max_{k=0,1,\ldots,2^J-1} |[\boldsymbol{u}_{n_1}]_k|}$$

It is seen that wavelet compression in this case can yield a speedup of about seven with a relative error of about 7×10^{-5}.

	% elem. retained		
λ	$\breve{\boldsymbol{E}}^{\varepsilon_M}$	$\breve{\boldsymbol{u}}_{n_1}^{\varepsilon_V}$	CPU time (s)
0	3.08	44.97	49
1	3.13	25.20	76
2	3.47	18.02	107
3	4.02	15.83	141
4	4.60	14.36	180
5	5.24	13.43	228
6	5.90	13.18	287
7	7.22	13.77	346

Table 9.7: Compression ratios of $\breve{\boldsymbol{E}}$ and $\breve{\boldsymbol{u}}_{n_1}$ ($n_1 = 2000$) and execution time in (CPU seconds) of the wavelet split-step algorithm for different values of λ. Here $\varepsilon_M = 10^{-15}$ and $\varepsilon_V = 10^{-7}$.

Table 9.7 shows how performance depends on the transform depth λ. It is seen that even though the compression potential of $\breve{\boldsymbol{u}}_{n_1}$ increases with λ, it is not enough to balance the increased work inherent in the matrix vector multiplications. This should be compared to Table 8.3 which indicates that the work grows with λ unless $\breve{\boldsymbol{u}}_{n_1}$ is compressed excessively. The gain from compression is outweighed by the cost of the wavelet transform in Algorithm 9.4. and that of computing Algorithm 8.4.

To illustrate a realistic application, we show in Figure 9.4 a computed "breather solution" obtained from the data $\beta_2 = -2$, $\alpha = 0$, $\gamma = 2$, $h(x) = 2\text{sech}(x)$, $L = 60$, $N = 2048$, $\lambda = 3$, $\Delta t = 0.001$ and $D = 8$. The figure gives only the middle third of the x-interval.

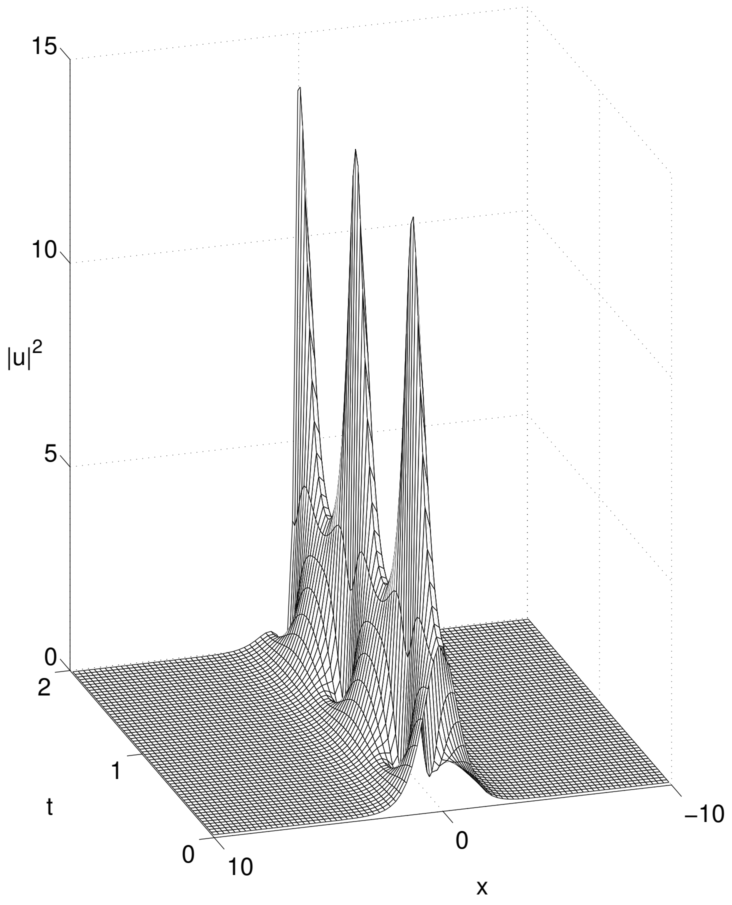


Figure 9.4: A “breather solution” of problem (9.27).

9.3.4 FFT split-step method

To assess the performance of the wavelet split-step method we compare it to an implementation of the FFT split-step method which is the de facto standard for solving the nonlinear Schrödinger equation [New92].

The FFT split-step method exploits the fact that the Fourier differentiation matrix $\boldsymbol{D}_F^{(d)}$, given in Appendix D, is diagonal. Hence,

$$\boldsymbol{L}_F = -\frac{i}{2}\beta_2 \frac{\boldsymbol{D}_F^{(2)}}{L^2} - \frac{\alpha}{2}\boldsymbol{I}$$

is diagonal and

$$\hat{\boldsymbol{E}} = \exp(\frac{\Delta t}{2}\boldsymbol{L}_F) \tag{9.33}$$

is also diagonal. The FFT split-step algorithm is therefore given by Algorithm 9.2 but with step 2 replaced by (9.33) and linstep$(\boldsymbol{v})$ (used in steps 4 and 6) redefined as

Algorithm 9.5: $\boldsymbol{u} \leftarrow \text{linstep}(\boldsymbol{v})$

$$\begin{aligned} \hat{\boldsymbol{v}} &\leftarrow \boldsymbol{F}_N \boldsymbol{v} \\ \hat{\boldsymbol{u}} &\leftarrow \hat{\boldsymbol{E}}\hat{\boldsymbol{v}} \\ \boldsymbol{u} &\leftarrow \boldsymbol{F}_N^{-1}\hat{\boldsymbol{u}} \end{aligned}$$

The product $\hat{\boldsymbol{E}}\hat{\boldsymbol{v}}$ is computed in N operations so the complexity for large N is dominated by that of the FFT and hence is $\mathcal{O}(N \log_2 N)$. This should be compared to the wavelet split-step method where the FWT as well as the matrix-vector product has linear complexity, the latter, however, involving a large constant.

Replacing Algorithm 9.4 by Algorithm 9.5 renders Algorithm 9.2 the FFT split-step method. The performance of the FFT split-step method with the same parameters as in Table 9.6 yields an execution time of 12s with a relative error of 2.23×10^{-6}. The FFT routine used for this purpose was a library routine optimized for vectorization on the CRAY C92A. In contrast, the implementation of Algorithm 8.4 did not vectorize well due to short vector lengths and inefficient memory access. Table 9.8 shows how the execution time grows with N for the FFT split-step method and the wavelet split-step method. In the wavelet case the parameters are $D = 8$, $\lambda = 3$, $\varepsilon_M = 10^{-11}$, and ε_V as given in the table.

9.4 Burgers' equation

We consider now Burgers' equation which is frequently used to study adaptive methods because it models formation of shocks. The periodic initial-value prob-

		CPU time (s)	
N	FFT	$\varepsilon_V = 10^{-9}$	$\varepsilon_V = 10^{-7}$
128	1	35	32
256	2	47	39
512	4	64	55
1024	6	101	83
2048	11	173	141
4096	21	295	236
8192	42	534	436
16384	85	1043	865

Table 9.8: Execution time (CPU seconds) for the FFT split-step method and the wavelet split-step method both applied to (9.27). The latter method is shown for two different compression thresholds.

lem for a particular form of Burgers' equation is

$$\left.\begin{aligned} u_t &= \nu u_{xx} - (u+\rho)u_x, & t > 0 \\ u(x,0) &= h(x) & \\ u(x,t) &= u(x+1,t), & t \geq 0 \end{aligned}\right\} \quad x \in \mathbf{R} \tag{9.34}$$

where ν is a positive constant, $\rho \in \mathbf{R}$ and $h(x) = h(x+1)$. Burgers' equation with $\rho = 0$ describes the evolution of u under nonlinear advection and linear dissipation. A nonzero value of ρ adds linear advection to the system.

This problem is discretized in the same manner as (9.42), and we obtain the system

$$\begin{aligned} \frac{d}{dt}\boldsymbol{u}(t) &= \boldsymbol{L}\boldsymbol{u}(t) + \boldsymbol{N}(\boldsymbol{u}(t))\,\boldsymbol{u}(t), \quad t \geq 0 \\ \boldsymbol{u}(0) &= \boldsymbol{h} \equiv [h(x_0), h(x_1), \ldots, h(x_{N-1})]^T \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{L} &= \nu \boldsymbol{D}^{(2)} - \rho \boldsymbol{I} \\ \boldsymbol{N}(\boldsymbol{u}(t)) &= -\mathrm{diag}(\boldsymbol{D}\boldsymbol{u}(t)) \end{aligned}$$

with $\boldsymbol{D} = \boldsymbol{D}^{(1)}$. This has the same form as (9.28) and an appropriate modification of Algorithm 9.2 can be used to solve it.

The matrix-vector product $\boldsymbol{u}^{(1)} = \boldsymbol{D}\boldsymbol{u}$ can be computed analogously to the computation of the matrix-vector multiplications in Algorithm 9.4. Hence we have

Algorithm 9.6: $\boldsymbol{u}^{(1)} \leftarrow \boldsymbol{D}\boldsymbol{u}$

$$\begin{aligned}
\breve{\boldsymbol{u}} &\leftarrow \boldsymbol{W}\boldsymbol{u} \\
(\breve{\boldsymbol{u}})^{\varepsilon_V} &\leftarrow \text{trunc}(\breve{\boldsymbol{u}}, \varepsilon_V) \\
\breve{\boldsymbol{u}}^{(1)} &\leftarrow (\breve{\boldsymbol{D}})^{\varepsilon_D}(\breve{\boldsymbol{u}})^{\varepsilon_V} \\
\boldsymbol{u}^{(1)} &\leftarrow \boldsymbol{W}^T\breve{\boldsymbol{u}}^{(1)}
\end{aligned}$$

where $\breve{\boldsymbol{D}} = \boldsymbol{W}\boldsymbol{D}\boldsymbol{W}^T$ and

$$(\breve{\boldsymbol{D}})^{\varepsilon_D} = \text{trunc}(\breve{\boldsymbol{D}}, \varepsilon_D)$$

Unlike $\boldsymbol{E}$, which is dense, the matrix $\boldsymbol{D}$ is a banded matrix (see Section 7.3) so $\breve{\boldsymbol{D}}$ will have the "finger band" structure without applying any compression. However, the wavelet transform introduces rounding errors which we remove by choosing ε_D to be a fixed small precision, namely $\varepsilon_D = 10^{-13}$, as this is sufficient to retain the sparsity of $\breve{\boldsymbol{D}}$. Hence we use Algorithms 8.1 and 8.4 for computing $\breve{\boldsymbol{D}}$ and the product $(\breve{\boldsymbol{D}})^{\varepsilon_D}(\breve{\boldsymbol{u}})^{\varepsilon_V}$, respectively.

To test this procedure on (9.34) we have chosen the values $\nu = 0.005$, $\rho = 0$, and $h(x) = \sin(2\pi x)$. The numerical data are $J = 10$, $\lambda = 3$, $\varepsilon_M = 10^{-11}$, $\varepsilon_V = 10^{-10}$, $\Delta t = 0.005$, $n_1 = 100$, and $D = 8$. The implementation was done in Fortran 90 on a 200 MHz PC using the Linux operating system.

Figure 9.5 shows the evolution of $u(x,t)$ for $0 \le t < 0.5$. It is seen that a large gradient forms at $x = 0.5$ whereas u is very smooth elsewhere. Hence we expect a large wavelet compression potential.

Table 9.9 shows the influence on performance of the parameter ε_V with ε_M (and ε_D) fixed. It is seen that the execution time drops as fewer elements are retained in $\breve{\boldsymbol{u}}_{n_1}$. However, we observe that the split-step method ceases to converge when the compression error becomes too large.

Table 9.10 shows the influence on performance of the transform depth λ for fixed values of ε_V and ε_M. It is seen that the execution time is reduced by compression as λ grows from 0 to 4. For $\lambda > 4$ the execution time increases because the compression is not enough to balance the increased work inherent in the matrix vector multiplications.

However, as the problem size grows, so does the compression potential. Table 9.11 shows the best execution times and compression rations obtained for different values of N. It is seen that by adjusting the parameters λ, ε_V and ε_M one can obtain very large compression ratios and a CPU time which grows even slower than N. Consequently, we see that for this particular problem, wavelet compression leads to a feasible solution method provided that the parameters N, D, λ, ε_V, and ε_M are chosen appropriately.

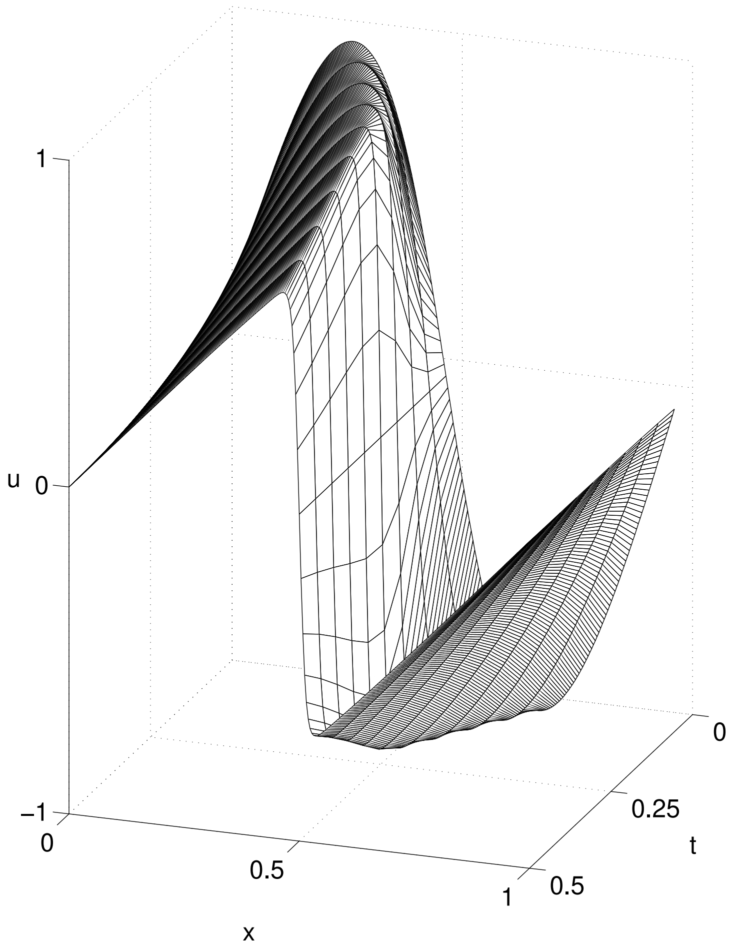


Figure 9.5: Solution of (9.34) shown for $t = 0, 0.05, 0.1, \ldots, 0.5$. $\nu = 0.002$, $\rho = 0$, $u(x, 0) = \sin(2\pi x)$.

9.5 Wavelet Optimized Finite Difference method

In this section we will outline a completely different approach to solving partial differential equations with the aid of wavelets. This method is due to Leland Jameson [Jam94, Jam96] and is called the Wavelet Optimized Finite Difference Method (WOFD). It works by using wavelets to generate an irregular grid which is then exploited for the finite difference method. Hence this method belongs to Class 3 as described in Section 7.1.

		% elem. retained			CPU time (s)		
ε_V	Rel err	$\breve{\boldsymbol{E}}^{\varepsilon_M}$	$\breve{\boldsymbol{D}}^{\varepsilon}_{D}$	$\breve{\boldsymbol{u}}^{\varepsilon_V}_{n_1}$	Total	WT	Mult
10^{-13}	3.8×10^{-7}	15.63	2.60	99.22	115	18	91
10^{-12}	4.8×10^{-7}	15.63	2.60	92.68	101	18	77
10^{-11}	2.7×10^{-7}	15.63	2.60	73.63	89	18	66
10^{-10}	4.2×10^{-6}	15.63	2.60	71.83	85	18	57
10^{-9}	4.6×10^{-5}	15.63	2.60	65.82	71	18	50
10^{-8}	No convergence						

Table 9.9: Effect of the parameter ε_V on performance. The remaining parameters are $\lambda = 3$, $\varepsilon_M = 10^{-12}$, $N = 2048$, $D = 8$, $\Delta t = 0.005$, and $n_1 = 100$. Second column shows the relative error at $n = n_1$ in the infinity norm compared to a reference solution computed using the FFT split-step method. Columns 3–5 show the percentage of elements retained in $\breve{\boldsymbol{E}}^{\varepsilon_M}$, $\breve{\boldsymbol{D}}^{\varepsilon_D}$, and $\breve{\boldsymbol{u}}^{\varepsilon_V}_{n_1}$, respectively. Columns 6–8 show the total execution time, the time spent in the wavelet transforms (FWT and IFWT), and the time spent in computing the products using Algorithm 8.4.

9.5.1 Finite differences on an irregular grid

We begin by defining a finite difference method for an irregular grid. Let u be a twice differentiable 1-periodic function and let there be a set of grid points

$$0 = x_0 < x_1 < x_2 < \cdots < x_{N-1} < 1$$

which are not necessarily equidistant. A way to approximate derivatives of u is to construct a Lagrangian interpolating polynomial through p points and differentiate it. We consider only odd $p \geq 3$ because it makes the algorithm simpler. Let $w = (p-1)/2$ and define

$$\boxed{u_I(x) = \sum_{k=i-w}^{i+w} u(x_k) \frac{P_{w,i,k}(x)}{P_{w,i,k}(x_k)}} \tag{9.35}$$

where,

$$P_{w,i,k}(x) = \prod_{\substack{l=i-w \\ l \neq k}}^{i+w} (x - x_l) \tag{9.36}$$

Because of periodicity, all indices are assumed to be computed modulo N. It follows that $u_I(x_i) = u(x_i)$ for $i = 0, 1, \ldots, N-1$; i.e. u_I interpolates u at the grid points.

		% elem. retained			CPU time (s)		
λ	Rel err	$\breve{\boldsymbol{E}}^{\varepsilon_M}$	$\breve{\boldsymbol{D}}^{\varepsilon_D}$	$\breve{\boldsymbol{u}}_{n_1}^{\varepsilon_V}$	Total	WT	Mult
0	1.8×10^{-7}	19.19	0.63	99.95	89	0	83
1	1.8×10^{-5}	15.53	1.27	94.87	64	10	48
2	1.2×10^{-5}	17.38	1.93	71.92	74	16	53
3	4.6×10^{-5}	15.63	2.60	65.82	74	18	50
4	5.6×10^{-5}	15.23	3.28	62.06	75	18	51
5	5.6×10^{-5}	15.54	3.96	60.40	81	19	55
6	7.4×10^{-5}	16.09	4.64	58.84	85	19	59
7	3.6×10^{-5}	16.71	5.32	53.76	88	19	62
8	2.2×10^{-4}	17.37	6.00	58.89	88	19	75

Table 9.10: $\varepsilon_V = 10^{-9}$, $\varepsilon_M = 10^{-12}$, $N = 2048$, $D = 8$, $\Delta t = 0.005$, $n_1 = 100$. Columns 2–8 have the same meanings as those in Table 9.9.

Differentiation of (9.35) d times yields

$$u_I^{(d)}(x) = \sum_{k=i-w}^{i+w} u(x_k) \frac{P_{w,i,k}^{(d)}(x)}{P_{w,i,k}(x_k)} \tag{9.37}$$

Replacing x by x_i in (9.37) yields a p-point difference approximation for $u^{(d)}(x)$ centered at x_i. Let $\boldsymbol{u} = [u(x_0), u(x_1), \ldots, u(x_{N-1})]$. The derivatives $u^{(d)}(x)$ can then be approximated at all of the grid points by

$$\boxed{\boldsymbol{u}^{(d)} = \boldsymbol{D}_p^{(d)} \boldsymbol{u}} \tag{9.38}$$

where the differentiation matrix $\boldsymbol{D}_p^{(d)}$ is defined by

$$[\boldsymbol{D}_p^{(d)}]_{i,k} = \frac{P_{w,i,k}^{(d)}(x_i)}{P_{w,i,k}(x_k)}, \quad w = (p-1)/2 \tag{9.39}$$

Regarding the first and second order derivatives we find that

$$P_{w,i,k}^{(1)}(x) = \frac{d}{dx} P_{w,i,k}(x) = \sum_{\substack{l=i-w \\ l \neq k}}^{i+w} \prod_{\substack{m=i-w \\ m \neq k,l}}^{i+w} (x - x_m) \tag{9.40}$$

					% elem. retained			CPU time (s)	
N	ε_V	ε_M	λ	Rel err	$\breve{\boldsymbol{E}}^{\varepsilon_M}$	$\breve{\boldsymbol{D}}^{\varepsilon_D}$	$\breve{\boldsymbol{u}}_{n_1}^{\varepsilon_V}$	FWT	FFT
1024	10^{-10}	10^{-11}	3	5×10^{-5}	16	5.20	82	36	37
2048	10^{-9}	10^{-11}	3	5×10^{-5}	13	2.60	65	65	78
4096	10^{-8}	10^{-10}	3	5×10^{-5}	9	1.30	54	127	176
8192	10^{-4}	10^{-8}	5	4×10^{-6}	5	0.99	4	173	404
16384	10^{-4}	10^{-5}	6	8×10^{-5}	3	0.58	2	332	890

Table 9.11: Best execution times for the wavelet split-step method for different problem sizes. The fixed parameters are $D = 8$, $\Delta t = 0.005$, and $n_1 = 100$. Column 1 shows the problem size. Columns 2–4 show the parameters that were chosen to obtain good performance for each N. Columns 5–8 have the same meanings as the corresponding columns in Table 9.9. Column 90 shows the execution time using the wavelet split-step method with the respective parameters and column 10 shows the execution time using the FFT split-step method.

and

$$P^{(2)}_{w,i,k}(x) = \frac{d^2}{dx^2} P_{w,i,k}(x) = \sum_{\substack{l=i-w \\ l \neq k}}^{i+w} \sum_{\substack{m=i-w \\ m \neq k,l}}^{i+w} \prod_{\substack{n=i-w \\ n \neq k,l,m}}^{i+w} (x - x_n) \tag{9.41}$$

It will be recalled that for equidistant grids with step length h the error is described by

$$\left| u_I^{(d)}(x_i) - u^{(d)}(x_i) \right| = \mathcal{O}(h^{p-1}), \quad d = 1, 2$$

provided that u is sufficiently smooth.

9.5.2 The nonlinear Schrödinger equation revisited

Consider again the periodic initial-value problem for the nonlinear Schrödinger equation

$$\left.\begin{aligned} u_t &= \mathcal{L}u + i\gamma \left|u\right|^2 u, t > 0 \\ u(x,0) &= h(x) \\ u(x,t) &= u(x+L,t), \;\; t \geq 0 \end{aligned}\right\} x \in \mathbf{R} \tag{9.42}$$

where

$$\mathcal{L} = -\frac{i}{2}\beta_2 \frac{\partial^2}{\partial x^2} - \frac{\alpha}{2}$$

Proceeding as in Section 9.3, but now using the differentiation matrices defined in (9.39) we obtain

$$\begin{aligned}\frac{d}{dt}\boldsymbol{u}(t) &= \boldsymbol{L}\boldsymbol{u}(t) + \boldsymbol{N}(\boldsymbol{u}(t))\,\boldsymbol{u}(t), \quad t \geq 0 \\ \boldsymbol{u}(0) &= \boldsymbol{h} \equiv [h(x_0), h(x_1), \ldots, h(x_{N-1})]^T\end{aligned} \tag{9.43}$$

where

$$\begin{aligned}\boldsymbol{L} &= -\frac{i}{2}\beta_2\frac{\boldsymbol{D}_p^{(2)}}{L^2} - \frac{\alpha}{2}\boldsymbol{I} \\ \boldsymbol{N}(\boldsymbol{u}(t)) &= i\gamma\,\mathrm{diag}(|u_l(t)|^2, \, l = 0, 1, \ldots, N-1)\end{aligned}$$

The split-step method could now be used for time stepping. However, this involves truncation of matrices and since we no longer represent the solution in the wavelet domain we do not seek adaptivity through these means. Instead we use a standard finite-difference approach method such as the Crank-Nicolson method [Smi85, p. 19]. Then (9.43) is approximated by

$$\begin{aligned}\frac{\boldsymbol{u}(t+\Delta t) - \boldsymbol{u}(t)}{\Delta t} &= \boldsymbol{L}\frac{\boldsymbol{u}(t+\Delta t) + \boldsymbol{u}(t)}{2} \quad + \\ &\quad \boldsymbol{N}\left(\frac{\boldsymbol{u}(t+\Delta t) + \boldsymbol{u}(t)}{2}\right)\frac{\boldsymbol{u}(t+\Delta t) + \boldsymbol{u}(t)}{2}\end{aligned}$$

and we obtain the time-stepping procedure

$$\begin{aligned}\boldsymbol{A}\boldsymbol{u}_{n+1} &= \boldsymbol{B}\boldsymbol{u}_n + \\ &\quad \Delta t\boldsymbol{N}\left(\frac{\boldsymbol{u}_{n+1} + \boldsymbol{u}_n}{2}\right)\frac{\boldsymbol{u}_{n+1} + \boldsymbol{u}_n}{2}, \quad n = 0, 1, \ldots, n_1 - 1 \\ \boldsymbol{u}_0 &= \boldsymbol{h}\end{aligned} \tag{9.44}$$

where $\boldsymbol{A} = \boldsymbol{I} - \frac{\Delta t}{2}\boldsymbol{L}$, $\boldsymbol{B} = \boldsymbol{I} + \frac{\Delta t}{2}\boldsymbol{L}$, and $\boldsymbol{u}_n = \boldsymbol{u}(n\Delta t)$.

As with the split-step method, an iterative procedure is required. The computation of $\boldsymbol{u}_{n+1}$ can be handled by the steps

$$\boldsymbol{u}_{n+1}^{(0)} = \boldsymbol{u}_n$$

and iterate until convergence

$$\begin{aligned}\boldsymbol{A}\boldsymbol{u}_{n+1}^{(q+1)} &= \boldsymbol{B}\boldsymbol{u}_n + \\ &\quad \Delta t\ \boldsymbol{N}\left(\frac{\boldsymbol{u}_{n+1}^{(q)} + \boldsymbol{u}_n}{2}\right)\frac{\boldsymbol{u}_{n+1}^{(q)} + \boldsymbol{u}_n}{2}, \quad q = 0, 1, \ldots\end{aligned} \tag{9.45}$$

We have solved the system (9.45) in Matlab by LU-factorization of $\boldsymbol{A}$ and subsequent forward and backward substitutions in each iteration step.

9.5.3 Grid generation using wavelets

The elements of $\boldsymbol{u}(t)$ approximate the function values $u(x_k, t), k = 0, 1, \ldots, N-1$. The success of an adaptive method relies on a procedure for determining a grid which is dense where u is erratic and sparse where u is smooth.

Recall from Theorem 2.5 that the wavelet coefficients of u satisfy the inequality

$$|d_{j,k}| \leq \frac{1}{L^P} C_P \, 2^{-j(P+\frac{1}{2})} \max_{\xi \in I_{j,k}} \left| u^{(P)}(\xi) \right|$$

where $P = D/2$. Note that we have taken the interval length L into account.

The error at $k/2^j$ depends on the size of the neighboring intervals, and a large value of $|d_{j,k}|$ is an indication that the grid spacing $1/2^j$ is too coarse to resolve u properly in the interval $I_{j,k}$. Hence when a large value of $|d_{j,k}|$ arises, we add points with spacing $1/2^{j+1}$ about position $k/2^j$ to reduce the error locally. In [Jam94] it is suggested that only a few points be added at location $k/2^j$. However, in the light of Theorem 2.5 we have found it reasonable to distribute points evenly over the entire interval $I_{j,k}$ because the large gradient can be located anywhere within the support of the corresponding wavelet.

This can be realized in practice by introducing an equidistant grid at some finest level J. If the solution vector $\boldsymbol{u}$ is defined for a coarser grid, it is then interpolated to values on the fine grid. Then the vector of wavelet coefficients $\boldsymbol{d}$ is computed as

$$\boldsymbol{d} = \boldsymbol{W}\boldsymbol{T}\boldsymbol{u}$$

The WOFD grid is generated by choosing those grid points which correspond to large wavelet coefficients as described above. After the grid is generated, the finite difference equations are constructed according to (9.44) and a number of time steps are taken. Hence, we propose the following algorithm for grid generation:

Algorithm 9.7: Wavelet-based grid generation

Interpolate $\boldsymbol{u}$ from current coarse grid to fine equidistant grid at scale J.

Construct new coarse grid from this expanded $\boldsymbol{u}$:
  Initialize new grid (delete old points)
  Compute wavelet coefficients $d_{j,k}$ of u ($\boldsymbol{u} \to \boldsymbol{c} \to \boldsymbol{d}$).
  Insert grid points where $|d_{j,k}| > \varepsilon$.
  Construct matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ and factorize $\boldsymbol{A}$.

Perform several time steps on new grid.

How many time steps to take between grid evaluations depends on the problem at hand: A rapidly changing solution will require frequent grid evaluations whereas a slowly changing solution can use the same grid for many time steps. We have had good results with the following heuristic: If the number of grid points increases more than a given threshold from one grid to the next, then the previous series of time steps is repeated on the *new* grid.

The initialization of Algorithm 9.7 is

$$u_l = h(x_l), \quad l = 0, 1, \ldots, 2^J - 1$$

9.5.4 Results

Figure 9.6 shows the computed solution of (9.42) at $t = 0.5$ together with the grid generated by the WOFD method. The solution corresponds to the parameters $\beta_2 = -2$, $\gamma = 2$, $\alpha = 0$, $L = 64$, and $h(x) = 2\text{sech}(x)$. The numerical data are $J = 10$ (making $N = 1024$), $\lambda = J$, $\varepsilon = 10^{-4}$, $p = 5$, $\Delta t = 1/2^{10}$, $n_1 = 512$ and the wavelets used are those of genus $D = 8$. It is seen that the WOFD method has

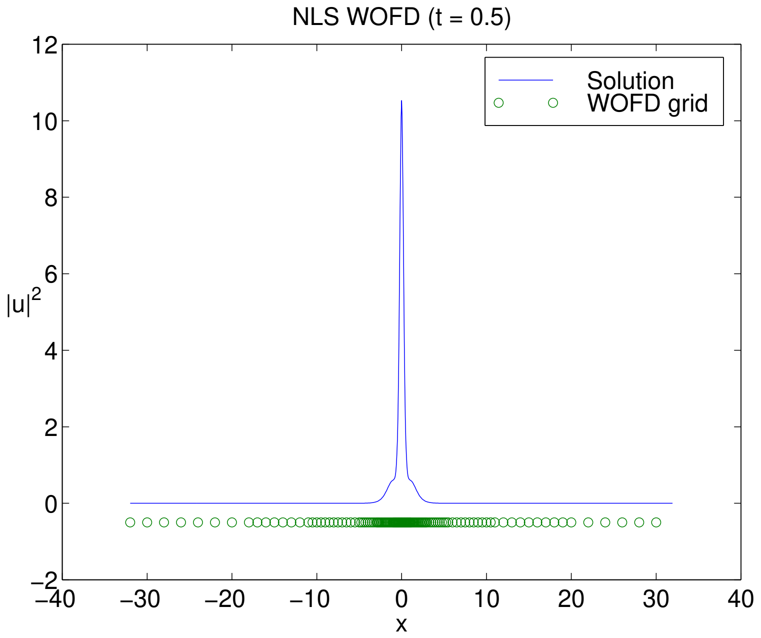


Figure 9.6: A WOFD solution of (9.42).

generated a grid which captures the behavior of the pulse, the grid density being low where the solution is smooth and high where it is not.

	Mflops	CPU (s)	$E^\varepsilon(\boldsymbol{u})$	points
WOFD($\varepsilon = 10^{-4}$)	102	97	$\approx 10^{-4}$	≈ 130
WOFD($\varepsilon = 0$)	700	147	0	1024
FFT split-step	613	96	–	1024

Table 9.12: The performance of the WOFD method compared to that of the FFT split-step. The test problem is (9.42) and the numerical data are $D = 8$, $\lambda = J = 10$, $0 \le t < 0.5$, $p = 5$, $\Delta t = 1/2^{10}$ and $\varepsilon = 10^{-4}$.

Let $\boldsymbol{u}^\varepsilon$ be the result of Algorithm 9.7 using ε as threshold and define the compression error

$$E^\varepsilon = \left\| \boldsymbol{u}^0 - \boldsymbol{u}^\varepsilon \right\|_\infty$$

where the vector $\boldsymbol{u}^0$ ($\varepsilon = 0$) corresponds to the solution obtained using finite differences on the finest grid. For comparison, we have also implemented the FFT [1] split-step method defined in Section 9.3.4 in Matlab. Table 9.12 shows the number of Mflops (measured in Matlab) needed for each method, the corresponding execution time, and the compression error E^ε where applicable. It is seen that the flop count is reduced significantly by the WOFD method but the CPU time is less so. The latter behavior is probably due to inefficient memory references in the updating of the finite difference matrices. However, this experiment suggests that the WOFD method is potentially very efficient for this problem.

The Matlab functions `nlswofd` and `nlsfft` demonstrate the WOFD method and the FFT split-step method, respectively.

9.5.5 Burgers' equation revisited

We now illustrate the adaptivity of WOFD by considering a problem where a gradient can grow and move with time. The problem concerns Burgers' equation as given in (9.34).

To test Algorithm 9.7 on (9.34) we have chosen the values $\nu = 0.002$, $\rho = 0$, $h(x) = \sin(2\pi x)$. The numerical data are $J = 11$, $\lambda = J$, $\varepsilon = 10^{-6}$, $p = 3$, $\Delta t = 1/2^9$, $n_1 = 256$ and $D = 6$. Figure 9.7 show the evolution of u together

[1] The FFT in Matlab does not precompute the exponentials. Hence it is not as efficient as a library FFT.

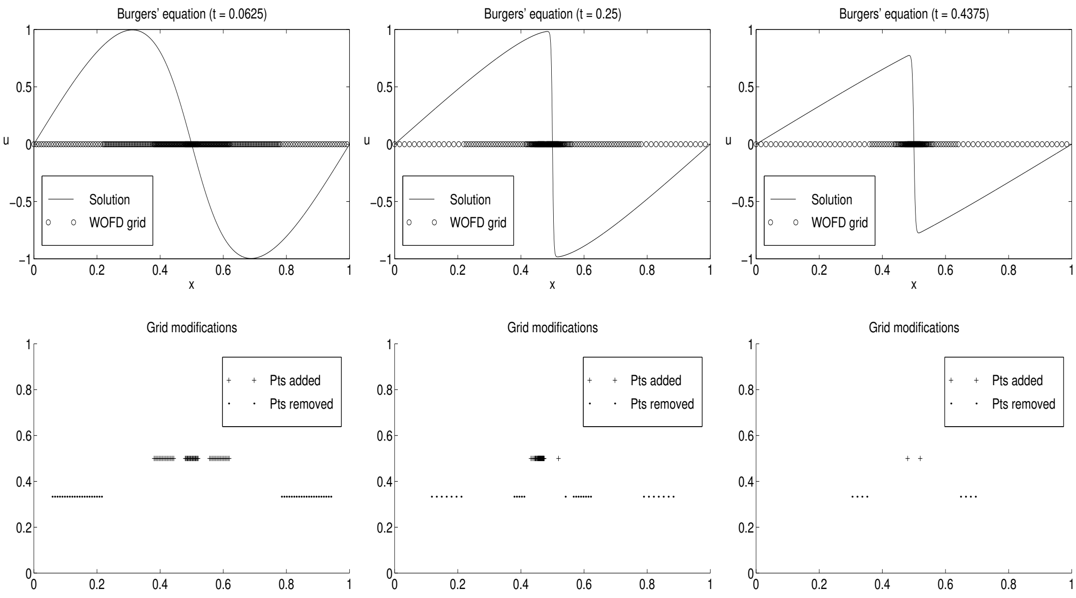


Figure 9.7: Solution of (9.34) with $\nu = 0.002$ and $\rho = 0$ shown together with the WOFD grid at times $t = 0.0625, 0.25, 0.4375$. The numerical data are $J = 11$, $\lambda = J$, $\varepsilon = 10^{-6}$, $p = 3$, $\Delta t = 1/2^9$, and $D = 6$. The lower graphs indicate points that have been added (+) and points that have been removed ($\cdot$) at time t.

with the WOFD grid for different values of t. It is seen that points concentrate around the large gradient as it forms and that the grid becomes very sparse where the solution is smooth. Table 9.13 shows the number of Mflops (measured in Matlab) used for $\varepsilon = 0$ and $\varepsilon = 10^{-6}$.

Finally, Figure 9.9 shows the solution to (9.34) at $t = 0.5$ with the same parameters as before except that we have now introduced linear advection corresponding to the parameter $\rho = 0.02$. It is seen that the WOFD method has captured the shock also in this case.

	Mflops	CPU (s)	$E^{\varepsilon}(\boldsymbol{u})$	points
WOFD($\varepsilon = 10^{-6}$)	12	1.7	$\approx 10^{-4}$	≈ 200
WOFD($\varepsilon = 0$)	46	2.9	0	2048

Table 9.13: The performance of the WOFD method. The test problem is (9.34) with $\nu = 0.002$ and $\rho = 0$ and the numerical data are $D = 6$, $\lambda = J = 11$, $0 \leq t < 0.5$, $p = 3$, $\Delta t = 1/2^9$, and $\varepsilon = 10^{-6}$.

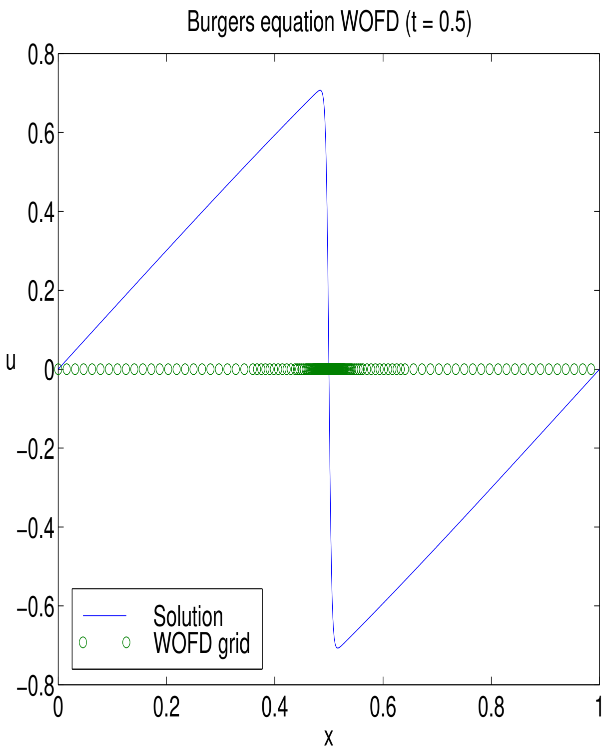
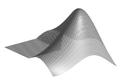


Figure 9.8: Solution of (9.34) with $\nu = 0.002$ and $\rho = 0$ shown together with the WOFD grid at $t = 0.5$. The numerical data are $J = 11$, $\lambda = J$, $\varepsilon = 10^{-6}$, $p = 3$, $\Delta t = 1/2^9$, and $D = 6$.



The Matlab program `burgerwofd` demonstrates the WOFD method for burgers equation.

The wavelet optimized finite difference method (WOFD) is a promising way of solving partial differential equations with the aid of wavelets. However, the method involves many heuristics and much remains to be investigated:

- We have found that the method is sensitive to the the choice of points placed at location $k/2^j$. The optimal strategy still remains to be found.

- The reconstruction on the finest grid is presently done with cubic spline interpolations. This is not necessarily the best approach, and other interpolation schemes should be considered. Alternatively, if the wavelet transform could be computed directly from function values on the irregular grid, the

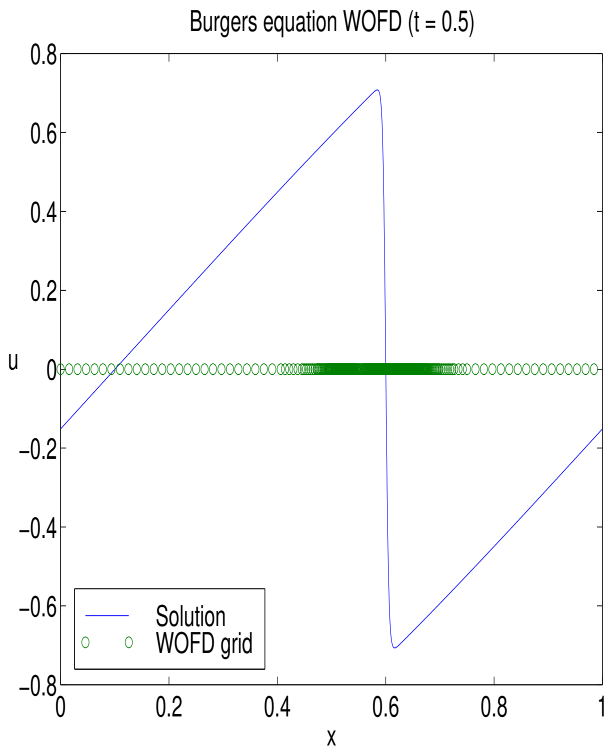


Figure 9.9: Solution of (9.34) with $\nu = 0.002$ and $\rho = 0.2$ shown together with the WOFD grid at $t = 0.5$. The numerical data are $J = 11$, $\lambda = J$, $\varepsilon = 10^{-6}$, $p = 3$, $\Delta t = 1/2^9$, and $D = 6$. The grid points cluster around gradient which has moved from $x = 0.5$ to $x \approx 0.6$.

reconstruction could be avoided altogether. Research towards this end is in progress [Swe96].

- As mentioned earlier, the number of time steps between consecutive grid evaluations should be determined adaptively based on the speed with which the solution evolves. Also this point needs further investigation.

Chapter 10

Conclusion

In part 1 we exposed the theory for compactly supported orthogonal wavelets and their periodized forms and gave estimates of their approximation characteristics. Furthermore, we demonstrated that the fast wavelet transform is a viable alternative to the fast Fourier transform whenever one encounters transient phenomena in functions or, as Gilbert Strang [SN96] puts it, *wavelets are best for piecewise smooth functions*.

In part 2 we showed how one can implement the fast wavelet transform efficiently on a vector computer as well as on parallel architectures. The reason why the wavelet transform lends itself well to parallelization is a consequence of the locality of wavelets. Hence, little communication is required. We observed very good actual performance on the Fujitsu VPP300 as well as the IBM PS2 and it was shown that our parallel algorithm is optimal in the sense that the *scaled efficiency is independent of the number of processors and it approaches one as the problem size is increased*.

In part 3 we developed several wavelet-based algorithms for solving partial differential equations. We derived wavelet differentiation matrices and showed how one can compute the wavelet transform of a circulant $N \times N$ matrix $\boldsymbol{A}$ in $\mathcal{O}(N)$ steps using storage bounded by $2\lambda N$ elements. Further, it was shown that if $\boldsymbol{A}$ is also banded, which is the case for a differentiation matrix, then the work needed for computing a matrix-vector product in the wavelet domain is $\mathcal{O}(N)$. Moreover, it was shown that the complexity grows with the transform depth λ. However, the wavelet compression potential grows with λ too. Consequently, the feasibility of performing such a multiplication depends on whether the gain from wavelet compression outweighs the increased cost of doing the multiplication in the wavelet domain. This is the central question for wavelet-based solvers.

Examples of the numerical solution of partial differential equations were given. In particular, we developed a wavelet split-step method for the nonlinear Schrödinger equation as well as Burgers' equation.

It was observed that the problem size as well as the compression potential must be very large for this method to be competitive with traditional methods. *Whereas it is straightforward to obtain the wavelet expansion of a known function, it can be exceedingly difficult to efficiently obtain the wavelet coefficients of a solution to a partial differential equation.* Similar conclusions were reached by [Wal96, p. III-29, V-1], [FS97, Jam94, VP96, CP96, PW96].

Finally, we outlined a different approach called the wavelet optimized finite difference method. This method uses the wavelet transform as a means for generating an adaptive grid for a traditional finite difference method, and the results for the nonlinear Schrödinger equation and Burgers' equation are promising. The pertinent question is not whether an adaptive grid is better than a fixed grid, but whether it can be generated better and more cheaply through other means than wavelets. However, this example indicates that wavelets can play the role as a powerful tool for dynamic *analysis* of the solution as it evolves.

In both cases we observed that the problem size has to be very large and the solution must be extremely sparse in a wavelet basis before wavelet-based methods for solving partial differential equations have a chance to outperform classical methods – and even then, the advantage relies on appropriate choices of several critical parameters.

We conclude by pointing out what we see as the main obstacles for obtaining truly feasible and competitive wavelet-based solvers for partial differential equations. Part of the problem is the fact that there is always a *finest* level or grid inherent all wavelet approaches. Future research could be directed at methods for avoiding the computations in the space of scaling function or on the finest grid. This would for example require a wavelet transform of data that are not equally spaced.

Another intrinsic problem is that compression errors tend to accumulate when applied to an iterative process. This puts a severe restriction on the amount of admissible compression. An error which is acceptable for a compressed signal may be detrimental for a solution to a partial differential equation when accumulated. Also, when errors are introduced, the solution may become *less smooth* which in turn means that the subsequent wavelet compression potential drops.

Finally, many algebraic operations in the wavelet domain tend to require complicated data structures [BK97, Hol97]. Such data structures may involve pointers and other types of indirect addressing as well as short vectors. For this reason implementations are unlikely to perform well on computer architectures such as vector processors.

Much research in this field is on its way and it is too early to say whether the problems mentioned above will be solved. However, there is no doubt that wavelet analysis has earned its place as an important alternative to Fourier analysis — only the scope of applicability remains to be settled.

Appendix A

Moments of scaling functions

Consider the problem of computing the moments as given in (2.26):

$$M_l^p = \int_{-\infty}^{\infty} x^p \phi(x-l)\,dx, \quad l,p \in \mathbf{Z} \tag{A.1}$$

By the normalization (2.2) we note first that

$$\boxed{M_l^0 = 1, \quad l \in \mathbf{Z}} \tag{A.2}$$

Let $l = 0$. The dilation equation (2.17) then yields

$$\begin{aligned} M_0^p &= \int_{-\infty}^{\infty} x^p \phi(x)\,dx \\ &= \sqrt{2}\sum_{k=0}^{D-1} a_k \int_{-\infty}^{\infty} x^p \phi(2x-k)\,dx \\ &= \frac{\sqrt{2}}{2^{p+1}} \sum_{k=0}^{D-1} a_k \int_{-\infty}^{\infty} y^p \phi(y-k)\,dy, \qquad y = 2x \end{aligned}$$

or

$$M_0^p = \frac{\sqrt{2}}{2^{p+1}} \sum_{k=0}^{D-1} a_k M_k^p \tag{A.3}$$

To reduce the number of unknowns in (A.3) we will eliminate M_k^p for $k \neq 0$. Using the variable transformation $y = x - l$ in (A.1) and following a similar approach as in the derivation on page 20 yields

$$\begin{aligned} M_l^p &= \int_{-\infty}^{\infty} (y+l)^p \phi(y)\,dy \\ &= \sum_{n=0}^{p} \binom{p}{n} l^{p-n} \int_{-\infty}^{\infty} y^n \phi(y)\,dy \end{aligned}$$

or

$$\boxed{M_l^p = \sum_{n=0}^{p} \binom{p}{n} l^{p-n} M_0^n} \tag{A.4}$$

Substituting (A.4) into (A.3) we obtain

$$\begin{aligned} M_0^p &= \frac{\sqrt{2}}{2^{p+1}} \sum_{k=0}^{D-1} a_k \sum_{n=0}^{p} \binom{p}{n} k^{p-n} M_0^n \\ &= \frac{\sqrt{2}}{2^{p+1}} \sum_{n=0}^{p-1} \binom{p}{n} M_0^n \sum_{k=0}^{D-1} a_k k^{p-n} \quad + \quad \frac{\sqrt{2}}{2^{p+1}} M_0^p \underbrace{\sum_{k=0}^{D-1} a_k}_{\sqrt{2}} \end{aligned}$$

Solving for M_0^P yields

$$\boxed{M_0^p = \frac{\sqrt{2}}{2(2^p - 1)} \sum_{n=0}^{p-1} \binom{p}{n} M_0^n \sum_{k=0}^{D-1} a_k k^{p-n}} \tag{A.5}$$

Equation (A.5) can now be used to determine the pth moment of $\phi(x)$, M_0^p for any $p > 0$. For $p = 0$ use (A.2). The translated moments M_l^p are then obtained from (A.4).

Appendix B

The modulus operator

Let $n \in \mathbf{Z}$ then

$$n = pq + r \tag{B.1}$$

where $p, q, r \in \mathbf{Z}$. We denote q the *quotient* of n divided by p and r the *remainder* of that division. The q and r are not uniquely determined from p and n but given p, n we speak of the unique equivalence class consisting of all values of r fulfilling fulfilling (B.1) with $q \in \mathbf{Z}$.

However, one representative of this equivalence stands out. It is called the *principal remainder* and it is defined as

$$r = n \bmod p = n - p \left[\frac{n}{p} \right]$$

where $[\cdot]$ denotes the nearest integer towards zero.

This is the way modulus is implemented in many programming languages such as Matlab. While mathematically correct, it has the inherent inconvenience that a negative r is chosen for $n < 0$. In many applications such as periodic convolution we think of r as being the index of an array or a vector. Therefore, we wish to choose a representative where $r \in [0, p-1]$ for all $n \in \mathbf{Z}$. This can be accomplished by defining

$$\boxed{r = \langle n \rangle_p = n - p \left\lfloor \frac{n}{p} \right\rfloor} \tag{B.2}$$

where $\lfloor n/p \rfloor$ denotes the nearest integer *below* n/p. We have introduced the notation $\langle n \rangle_p$ in order to avoid confusion, and we note that

$$\langle n \rangle_p \quad = \quad n \bmod p \quad \text{for} \quad n > 0, \; p > 1$$

For practical purposes $\langle n \rangle_p$ should not be implemented as in (B.2); rather, it should be written using the built-in modulus function modifying the result whenever needed.

The programming language Fortran 90 includes both definitions, so `MOD(n,p)` is the ordinary modulus operator, where `MODULO(n,p)` implements $\langle n \rangle_p$ (see [MR96, p 177]).

Definition B.1 *Let n be given as in (B.1). If $r = 0$, we say that p is a divisor in n and write*

$$p \mid n$$

It follows that for all $p, q, n, r \in \mathbf{Z}$ we have

$$p \mid (n - r) \quad \text{and} \quad q \mid (n - r)$$

Lemma B.1 *Let $n_1, n_2, q \in \mathbf{Z}$. Then*

$$\begin{aligned} \langle n_1 \pm n_2 \rangle_q &= \langle n_1 \pm \langle n_2 \rangle_q \rangle_q \\ \langle n_1 n_2 \rangle_q &= \langle n_1 \langle n_2 \rangle_q \rangle_q \end{aligned}$$

Proof: We write n_2 as in (B.1) and find

$$\begin{aligned} \langle n_1 \pm n_2 \rangle_q &= \langle n_1 \pm (p_2 q + r_2) \rangle_q \\ &= \langle n_1 \pm r_2 \rangle_q \\ &= \langle n_1 \pm \langle n_2 \rangle_q \rangle_q \end{aligned}$$

and

$$\begin{aligned} \langle n_1 n_2 \rangle_q &= \langle n_1 (p_2 q + r_2) \rangle_q \\ &= \langle n_1 p_2 q + n_1 r_2 \rangle_q \\ &= \langle n_1 r_2 \rangle_q \\ &= \langle n_1 \langle n_2 \rangle_q \rangle_q \end{aligned}$$

□

Lemma B.2 *Let* $k, n, q \in \mathbf{Z}$. *Then*

$$k\langle n\rangle_q = \langle kn\rangle_{kq}$$

Proof: We write n as in (B.1). Then

$$q \mid (n - r)$$

for $q, n, r \in \mathbf{Z}$. When q is a divisor in $n - r$ then also

$$kq \mid k(n - r)$$

from which we get

$$kr = \langle kn\rangle_{kq}$$

$\square$

Appendix C

Circulant matrices and the DFT

We state some properties of circulant matrices and describe their relation to convolution and the discrete Fourier transform (DFT).

Definition C.1 (Circulant matrix) *Let $\boldsymbol{A}$ be an $N \times N$ matrix and let $\boldsymbol{a} = [a_0, a_1, \ldots, a_{N-1}]^T$ be the first column of $\boldsymbol{A}$. Then $\boldsymbol{A}$ is* **circulant** *if*

$$[\boldsymbol{A}]_{m,n} = a_{\langle m-n \rangle_N}, \quad m, n = 0, 1, \ldots, N-1$$

A circulant 4×4 matrix has the form

$$\begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix}$$

Since all columns are shifted versions of $\boldsymbol{a}$, it suffices to store the N elements of $\boldsymbol{a}$ instead of the N^2 elements of $\boldsymbol{A}$. Also, computational work can be saved using the circulant structure.

Matrix vector multiplication with a circulant matrix is closely related to discrete (cyclic) convolution which is defined as follows

Definition C.2 (Discrete convolution) *Let $\boldsymbol{x} = [x_0, x_1, \ldots, x_{N-1}]^T$ and define $\boldsymbol{y}$ and $\boldsymbol{z}$ similarly. Then*

$$\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$$

is the (cyclic) convolution defined by

$$z_m = \sum_{n=0}^{N-1} x_n y_{\langle m-n \rangle_N}, \quad m = 0, 1, \ldots, N-1 \tag{C.1}$$

The discrete Fourier transform (DFT) is closely related to convolution and circulant matrices. We define it as follows:

Definition C.3 (Discrete Fourier transform) *Let $\{x_l\}_{l=0}^{N-1}$ be a sequence of N complex numbers. The sequence $\{\hat{x}_k\}_{k=0}^{N-1}$ defined by*

$$\boxed{\hat{x}_k = \sum_{l=0}^{N-1} x_l \omega_N^{-kl}, \quad k = 0, 1, \ldots, N-1}$$

where $\omega_N = e^{i2\pi/N}$, is the discrete Fourier transform of $\{x_l\}_{l=0}^{N-1}$.

There are a number of variants of the DFT in the literature, and no single definition has a clear advantage over the others. A particular variant should therefore be chosen such that it suits the context best. The choice we have made here is mainly motivated by the fact that it yields a simple form of the convolution theorem (Theorem C.1).

Let $\boldsymbol{x} = [x_0, x_1, \ldots, x_{N-1}]^T$ and $\hat{\boldsymbol{x}} = [\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{N-1}]^T$. Then the DFT can be written in matrix-vector form as

$$\hat{\boldsymbol{x}} = \boldsymbol{F}_N \boldsymbol{x} \tag{C.2}$$

where

$$[\boldsymbol{F}_N]_{k,l} = \omega_N^{-kl} = e^{-i2\pi kl/N} \tag{C.3}$$

is the $N \times N$ Fourier matrix.

The inverse of $\boldsymbol{F}_N$ satisfies the relation

$$\boldsymbol{F}_N^{-1} = \frac{1}{N}\overline{\boldsymbol{F}}_N \tag{C.4}$$

This is seen from the matrix product

$$[\boldsymbol{F}_N]_{m,k}[\overline{\boldsymbol{F}}_N]_{k,n} = \sum_{k=0}^{N-1} \omega_N^{k(n-m)} = \begin{cases} N & m = n \\ 0 & m \neq n \end{cases}$$

Consequently, $\boldsymbol{x} = \boldsymbol{F}_N^{-1}\hat{\boldsymbol{x}}$ and we have

Definition C.4 (Inverse discrete Fourier transform) *Let $\{x_l\}_{l=0}^{N-1}$ and $\{\hat{x}_k\}_{k=0}^{N-1}$ be given as in Definition C.3. Then the inverse discrete Fourier transform (IDFT) is defined as*

$$\boxed{x_l = \frac{1}{N}\sum_{k=0}^{N-1} \hat{x}_k \omega_N^{kl}, \quad l = 0, 1, \ldots, N-1} \tag{C.5}$$

Both the DFT and the IDFT can be computed in $\mathcal{O}(N \log_2 N)$ steps using the fast Fourier transform algorithm (FFT).

The link between DFT and convolution is embodied in the **convolution theorem**, which we state as follows:

Theorem C.1 (Convolution theorem)

$$\boxed{\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y} \quad \Leftrightarrow \quad \hat{\boldsymbol{z}} = \mathrm{diag}(\hat{\boldsymbol{x}})\hat{\boldsymbol{y}}}$$

Proof: We begin by writing the components of $\boldsymbol{x}$ and $\boldsymbol{y}$ in terms of those of $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$, i.e.

$$\begin{aligned} x_n &= \frac{1}{N}\sum_{k=0}^{N-1} \hat{x}_k \omega_N^{kn} \\ y_{\langle m-n \rangle_N} &= \frac{1}{N}\sum_{k=0}^{N-1} \hat{y}_k \omega_N^{k\langle m-n \rangle_N} = \frac{1}{N}\sum_{k=0}^{N-1} (\hat{y}_k \omega_N^{-kn})\omega_N^{km} \end{aligned}$$

since $\exp(i2\pi\langle n \rangle_N/N) = \exp(i2\pi n/N)$. Consequently,

$$\begin{aligned} z_m &= \sum_{n=0}^{N-1}\left(\frac{1}{N}\sum_{k=0}^{N-1} \hat{x}_k \omega_N^{kn} + \frac{1}{N}\sum_{l=0}^{N-1} \hat{y}_l \omega_N^{-ln}\omega_N^{lm}\right) \\ &= \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} \hat{x}_k \hat{y}_l \omega_N^{lm} \sum_{n=0}^{N-1} \omega_N^{n(k-l)} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} \hat{x}_k \hat{y}_k \omega_N^{km} \end{aligned}$$

Hence $\hat{z}_k = \hat{x}_k \hat{y}_k$ for $k = 0, 1, \ldots, N-1$ by the definition of the IDFT (C.5). $\square$

Corollary C.2 *The convolution operator is commutative*

$$\boldsymbol{x} * \boldsymbol{y} = \boldsymbol{y} * \boldsymbol{x}$$

Proof: Using Theorem C.1 we get

$$\boldsymbol{x} * \boldsymbol{y} = \boldsymbol{F}_N^{-1}\mathrm{diag}(\hat{\boldsymbol{x}})\hat{\boldsymbol{y}} = \boldsymbol{F}_N^{-1}\mathrm{diag}(\hat{\boldsymbol{y}})\hat{\boldsymbol{x}} = \boldsymbol{y} * \boldsymbol{x}$$

$\square$

Matrix multiplication with a circulant matrix is equivalent to a convolution:

Lemma C.3 *Let $\boldsymbol{A}$ and $\boldsymbol{a}$ be defined as in Definition C.1 and $\boldsymbol{x} \in \mathbf{R}^N$ then*

$$\boxed{\boldsymbol{A}\boldsymbol{x} = \boldsymbol{a} * \boldsymbol{x}}$$

Proof:

$$\begin{aligned}
[\boldsymbol{A}\boldsymbol{x}]_m &= \sum_{n=0}^{N-1} [\boldsymbol{A}]_{m,n} x_n = \sum_{n=0}^{N-1} a_{\langle m-n \rangle_N} x_n \\
&= [\boldsymbol{x} * \boldsymbol{a}]_m = [\boldsymbol{a} * \boldsymbol{x}]_m, \quad m = 0, 1, \ldots, N-1
\end{aligned}$$

□

We now use the convolution theorem to obtain an alternative and useful characterization of circulant matrices.

Theorem C.4

$$\boldsymbol{A} \text{ is circulant} \quad \Leftrightarrow \quad \boldsymbol{A} = \boldsymbol{F}_N^{-1} \boldsymbol{\Lambda}_a \boldsymbol{F}_N, \quad \boldsymbol{\Lambda}_a = \mathrm{diag}(\hat{\boldsymbol{a}})$$

Proof:

$\Rightarrow$: Let $\boldsymbol{A}$ be an $N \times N$ circulant matrix and let $\boldsymbol{x}$ and $\boldsymbol{y}$ be arbitrary vectors of length N. Then, by Lemma C.3 and Theorem C.1

$$\begin{aligned}
\boldsymbol{y} &= \boldsymbol{A}\boldsymbol{x} \\
&= \boldsymbol{a} * \boldsymbol{x} \\
&= \boldsymbol{F}_N^{-1} \mathrm{diag}(\hat{\boldsymbol{a}}) \hat{\boldsymbol{x}} \\
&= \boldsymbol{F}_N^{-1} \mathrm{diag}(\hat{\boldsymbol{a}}) \boldsymbol{F}_N \boldsymbol{x}
\end{aligned}$$

the factorization follows.

$\Leftarrow$: Let $\boldsymbol{A} = \boldsymbol{F}_N^{-1} \boldsymbol{\Lambda}_a \boldsymbol{F}_N$ and let $\boldsymbol{e}_n = \boldsymbol{I}_{:,n}$ be the n'th unit vector of length N. The element $[\boldsymbol{A}]_{m,n}$ can then be extracted as follows

$$\begin{aligned}
[\boldsymbol{A}]_{m,n} &= [\boldsymbol{F}_N^{-1} \boldsymbol{\Lambda}_a \boldsymbol{F}_N]_{m,n} \\
&= \frac{1}{N} \boldsymbol{e}_m^T \overline{\boldsymbol{F}_N}^T \boldsymbol{\Lambda}_a \boldsymbol{F}_N \boldsymbol{e}_n \\
&= \frac{1}{N} \sum_{k=0}^{N-1} \omega_N^{km} \hat{a}_k \omega_N^{-kn} \\
&= \frac{1}{N} \sum_{k=0}^{N-1} \hat{a}_k \omega_N^{k(m-n)} \\
&= a_{\langle m-n \rangle_N}
\end{aligned}$$

Hence $\boldsymbol{A}$ is circulant according to Definition C.1.

□

Theorem C.5 *Circulant matrices with the same dimensions commute.*

Proof: Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be circulant $N \times N$ matrices according to Definition C.1. Then we have from Theorem C.4

$$\boldsymbol{A}\boldsymbol{B} = \boldsymbol{F}_N^{-1}\boldsymbol{\Lambda}_a\boldsymbol{F}_N\boldsymbol{F}_N^{-1}\boldsymbol{\Lambda}_b\boldsymbol{F}_N = \boldsymbol{F}_N^{-1}\boldsymbol{\Lambda}_a\boldsymbol{\Lambda}_b\boldsymbol{F}_N$$

Since $\boldsymbol{\Lambda}_a$ and $\boldsymbol{\Lambda}_b$ are diagonal, they commute and we find

$$\boldsymbol{A}\boldsymbol{B} = \boldsymbol{F}_N^{-1}\boldsymbol{\Lambda}_b\boldsymbol{\Lambda}_a\boldsymbol{F}_N = \boldsymbol{F}_N^{-1}\boldsymbol{\Lambda}_b\boldsymbol{F}_N\boldsymbol{F}_N^{-1}\boldsymbol{\Lambda}_a\boldsymbol{F}_N = \boldsymbol{B}\boldsymbol{A}$$

$\square$

Theorem C.6

$$\boldsymbol{A} \textit{ is circulant} \quad \Leftrightarrow \quad \boldsymbol{A}^{-1} \textit{ is circulant}$$

with

$$\boldsymbol{A}^{-1} = \boldsymbol{F}_N^{-1}\boldsymbol{\Lambda}_a^{-1}\boldsymbol{F}_N, \quad \boldsymbol{\Lambda}_a = \mathrm{diag}(\hat{\boldsymbol{a}})$$

Proof: Using the factorization given in Theorem C.4 we find

$$\boldsymbol{A}^{-1} = (\boldsymbol{F}_N^{-1}\boldsymbol{\Lambda}_a\boldsymbol{F}_N)^{-1} = \boldsymbol{F}_N^{-1}\boldsymbol{\Lambda}_a^{-1}\boldsymbol{F}_N$$

$\square$

Since $\boldsymbol{\Lambda}_a$ is diagonal, $\boldsymbol{\Lambda}_a^{-1}$ is computed in N operations. Hence the computational complexity of computing $\boldsymbol{A}^{-1}$ is dominated by that of the FFT which is $\mathcal{O}(N \log_2 N)$.

Appendix D

Fourier differentiation matrix

Let f be a 1 periodic function which is d times differentiable and consider its Fourier expansion

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{i2\pi kx}, \quad x \in \mathbf{R} \tag{D.1}$$

where $c_k = \int_0^1 f(x)e^{-i2\pi kx}\,dx$.

We approximate f by the truncated expansion

$$f_N(x) = \sum_{k=-N/2}^{N/2-1} c_k e^{i2\pi kx}, \quad x \in \mathbf{R}$$

Differentiating d times then yields

$$f_N^{(d)}(x) = \sum_{k=-N/2}^{N/2-1} c_k^{(d)} e^{i2\pi kx}, \quad x \in \mathbf{R}$$

where

$$c_k^{(d)} = (i2\pi k)^d c_k, \quad k = -\frac{N}{2}, -\frac{N}{2}+1, \ldots, \frac{N}{2}-1$$

Hence $f_N^{(d)}(x)$ approximates $f^{(d)}(x)$.

Let $N = 2^J$ for $J \in \mathbf{N}$ and $x_l = l/N$, $l = -N/2, -N/2+1, \ldots, N/2-1$. Define the vectors $\boldsymbol{f}$ and $\boldsymbol{f}^{(d)}$ as

$$f_l = f_N(x_l) = \sum_{k=-N/2}^{N/2-1} c_k e^{i2\pi kx_l} = \sum_{k=-N/2}^{N/2-1} c_k \omega_N^{kl} \tag{D.2}$$

$$f_l^{(d)} = f_N^{(d)}(x_l) = \sum_{k=-N/2}^{N/2-1} c_k^{(d)} e^{i2\pi kx_l} = \sum_{k=-N/2}^{N/2-1} c_k^{(d)} \omega_N^{kl} \tag{D.3}$$

for

$$l = -\frac{N}{2}, -\frac{N}{2}+1, \dots, \frac{N}{2}-1$$

Then

$$c_k = \frac{1}{N}\sum_{l=-N/2}^{N/2-1} f_l \omega_N^{-kl}, \quad k = -\frac{N}{2}, -\frac{N}{2}+1, \dots, \frac{N}{2}-1$$

We will now derive a matrix expression for computing $\boldsymbol{f}^{(d)}$ using the Fourier matrix $\boldsymbol{F}_N$ described in (C.3). The problem is that the Fourier matrix is defined for vectors index by $l = 0, 1, \dots, N-1$ whereas our vectors here are shifted by $N/2$. Applying $\boldsymbol{F}_N/N$ directly to the vector $\boldsymbol{f}$ yields

$$\frac{1}{N}\sum_{l=0}^{N-1} f_{l-N/2}\omega_N^{-kl} = \frac{1}{N}\sum_{l'=-N/2}^{N/2-1} f_{l'}\omega_N^{-kl'} \underbrace{\omega_N^{-kN/2}}_{=(-1)^k} = (-1)^k c_k, \quad k = 0, 1, \dots, N-1$$

The coefficients are N-periodic, $c_k = c_{k+N}$, so we can extend the definition of $c_k^{(d)}$ to the interval $[0, N-1]$ as follows:

$$c_k^{(d)} = \begin{cases} 2\pi k c_k & \text{for} \quad k = 0, 1, \dots, N/2-1 \\ 2\pi(k-N)c_k & \text{for} \quad k = N/2, N/2+1, \dots, N-1 \end{cases} \tag{D.4}$$

Let

$$\begin{aligned} \boldsymbol{c} &= \{c_k\}_{k=0}^{N-1} \\ \boldsymbol{c}^{(d)} &= \left\{c_k^{(d)}\right\}_{k=0}^{N-1} \end{aligned}$$

and define the **Fourier differentiation matrix** $\boldsymbol{D}_F^{(d)}$ as the diagonal matrix

$$\left[\boldsymbol{D}_F^{(d)}\right]_{k,k} = \begin{cases} 2\pi k & \text{for} \quad k = 0, 1, \dots, N/2-1 \\ 2\pi(k-N) & \text{for} \quad k = N/2, N/2+1, \dots, N-1 \end{cases} \tag{D.5}$$

then (D.4) has the vector formulation

$$\boxed{\boldsymbol{c}^{(d)} = \boldsymbol{D}_F^{(d)}\boldsymbol{c}}$$

Applying $N\boldsymbol{F}_N^{-1}$ to $\boldsymbol{c}^{(d)}$ then yields

$$\begin{aligned}\sum_{k=0}^{N-1}(-1)^k c_k^{(d)}\omega_N^{kl} &= \sum_{k=0}^{N-1} c_k^{(d)}\omega_N^{kl}\omega_N^{-kN/2}\\ &= \sum_{k=0}^{N-1} c_k^{(d)}\omega_N^{k(l-N/2)}\\ &= f_{l-N/2}^{(d)}, \quad l = 0,1,\dots,N-1\end{aligned}$$

Hence N and $1/N$ cancel out and we arrive at

$$\boxed{\boldsymbol{f}^{(d)} = \boldsymbol{F}_N^{-1}\boldsymbol{D}_F^{(d)}\boldsymbol{F}_N\boldsymbol{f}}$$

The shift in (D.5) can for example be done in Matlab using FFTSHIFT or a similar function. If the period is different from 1, say L, then $\boldsymbol{D}_F^{(d)}$ must be scaled as described in Section 7.4.1.

Appendix E

List of Matlab programs

The following Matlab programs demonstrate selected algorithms and are available on the World Wide Web at

`http://www.imm.dtu.dk/~omni/wapa20.tgz`

More documentation is found in the enclosed README file.

Function	Page	Description
`wavecompare`	10	Compares a wavelet approximation to a Fourier approximation
`basisdemo`	10	Generates and displays periodic wavelets
`daubfilt`	16	Returns a vector containing the filter coefficients $a_0, a_1, \ldots, a_{D-1}$
`low2hi`	17	Computes $\{b_k\}_{k=0}^{D-1}$ from $\{a_k\}_{k=0}^{D-1}$
`filttest`	21	Checks vector of filter coefficients
`cascade`	45	Computes function values for $\phi(x)$ and $\psi(x)$
`dst, idst`	49	Computes the DST and its inverse
`fwt, ifwt`	59	Computes the FWT and its inverse
`fwt2, ifwt2`	60	Computes the 2D FWT and its inverse
`conn`	111	Computes $\mathbf{\Gamma}^d$ for a given wavelet genus D
`diftest`	117	Tests convergence rates for $\boldsymbol{D}^{(d)}$ and $\boldsymbol{D}^d$
`difmatrix`	118	Computes the differentiation matrix $\boldsymbol{D}^{(d)}/L^d$
`nlsfft`	195	Demonstrates the FFT split-step method for the Nonlinear Schrödinger equation
`nlswofd`	195	Demonstrates the WOFD method for the nonlinear Schrödinger equation
`burgerwofd`	197	Demonstrates the WOFD method for Burgers' equation

Bibliography

[AB84] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Value Problems*. Computer Science and Applied Mathematics. Academic Press, 1984.

[Agr89] G. P. Agrawal. *Nonlinear Fiber Optics*. Academic Press, 1989.

[Bar96] V. A. Barker. Computing connection coefficients. Technical Report IMM-REP-1996-5, Technical University of Denmark, 1996.

[BCR91] G. Beylkin, R. Coifman, and V. Rohklin. Fast wavelet transforms and numerical algorithms. *Comm. Pure Appl. Math.*, 44:141–183, 1991.

[Bey92] G. Beylkin. On the representation of operators in bases of compactly supported wavelets. *SIAM J. Num. Anal.*, 29(6):1716–1740, 1992.

[Bey93] G. Beylkin. Wavelets and fast numerical algorithms. In *Different Perspectives on Wavelets*, pages 89–117. AMS, 1993.

[Bey94] G. Beylkin. On wavelet-based algorithms for solving differential equations",. In J. J. Benedetto and M. W. Frazier, editors, *Wavelets - Mathematics and Applications*, chapter 12, pages 449–466. CRC Press, 1994.

[BK97] G. Beylkin and J. M. Keiser. On the adaptive numerical solution of nonlinear partial differential equations in wavelet bases. *J. Comp. Phys.*, 132:233–259, 1997.

[BMP90] E. Bacry, S. Mallat, and G. Papanicolaou. A wavelet based space-time adaptive numerical method for partial differential equations. Technical report, Courant Institute of Mathematical Sciences, New York, 1990.

[BN96] S. Bertoluzza and G. Naldi. A wavelet collocation method for the numerical solution of partial differential equations. *Appl. Comp. Harmonic Anal.*, 3:1–9, 1996.

[Cha96] P. Charton. *Produits de matrices rapides en bases d'ondelettes: application á la résolution numérique d'équations aux dérivés partielles*. PhD thesis, Université de Paris Nord, 1996.

[Chu97] C. K. Chui. *Wavelets: A Mathematical Tool for Signal Analysis*. SIAM, 1997.

[CP96] P. Charton and V. Perrier. A pseudo-wavelet scheme for the two-dimensional Navier-Stokes equations. *Matemática Aplicada e Computational*, 15(2):139–160, 1996.

[Dau88] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Comm. Pure Appl. Math.*, XLI(7):909–996, 1988.

[Dau92] I. Daubechies. *Ten Lectures on Wavelets*. SIAM, 1992.

[DKU96] W. Dahmen, A. Kunoth, and K. Urban. A wavelet Galerkin method for the Stokes equations. *Computing*, 56:259–301, 1996.

[DMC95] B. K. Das, R. N. Mahapatra, and B. N. Chatterli. Modelling of wavelet transform on multistage interconnection network. In *Proceedings of the Australasian Conference on Parallel and Real-Time Systems*, pages 134–142, Freemantle, Western Australia, 28-29 September 1995.

[Dor95] M. Dorobantu. *Wavelet-based Algorithms for Fast PDE Solvers*. PhD thesis, Royal Institue of Technology, Stockholm, 1995.

[EBR97] L. Eldén, F. Berntsson, and T. Regińska. Wavelet and Fourier methods for solving the sideways heat equation. Preprint. Department of Mathematics, Linköbing University, 1997.

[Eir92] T. Eirola. Sobolev characterization of solutions of dilation equations. *SIAM J. Math. Anal.*, 23(4):1015–1030, 1992.

[EL] F. Ekstedt and M. Lindberg. Diagonalization of homogeneous linear operators in biorthogonal wavelet bases.
ftp.math.chalmers.se/pub/users/mlind/diag.a4.ps.gz.

[EOZ94] B. Engquist, S. Osher, and S. Zhong. Fast wavelet based algorithms for linear evolution equations. *SIAM J. Sci. Comp.*, 15(4):755–775, 1994.

[F^+96] M. Farge et al. Wavelets and turbulence. *Proc. IEEE*, 84(4):639–669, 1996.

[FJAF78] M. D. Feit and Jr J. A. Fleck. Light propagation in graded-index optical fibers. *Appl. Optics*, 17(24):3990–3998, 1978.

[Fol95] G. B. Folland. *Introduction to Partial Differential Equations*. Princeton University Press, 1995.

[FS97] J. Frölich and K. Schneider. An adaptive wavelet-vaguelette algorithm for the solution of pdes. *J. Comp. Phys.*, 130(2):174–190, 1997.

[GL89] G. H. Golub and C. Van Loan. *Matrix Computations*. John Hopkins University Press, 2 edition, 1989.

[GL94] L. Gagnon and J. M. Lina. Symmetric daubechies' wavelets and numerical solution of nls equations. *J. Phys. A: Math Gen*, 27:8207–8230, 1994.

[H^{+}94] C. Hsiao et al. Near optimal compression of orthonormal wavelet expansions. In J. J. Benedetto and M. W. Frazier, editors, *Wavelets - Mathematics and Applications*, chapter 11, pages 425–446. CRC Press, 1994.

[Heg95] M. Hegland. An implementation of multiple and multi-variate fast Fourier transforms on vector processors. *SIAM J. Sci. Comp.*, 16(2):271–288, 1995.

[Heg96] M. Hegland. Real and complex fast Fourier transforms on the Fujitsu VPP500. *Parallel Comp.*, 22:539–553, 1996.

[HJ88] R. W. Hockney and C. R. Jesshope. *Parallel Computers 2*. IOP Publishing Ltd, 1988.

[Hol97] M. Holmstrom. *Wavelet Based Methods for Time Dependent PDEs*. PhD thesis, Department of Scientific Computing, Uppsala University, 1997.

[HPW94] F. Heurtaux, F. Planchon, and Mladen Victor Wickerhauser. Scale decomposition in burgers' equation. In J. J. Benedetto and M. W. Frazier, editors, *Wavelets - Mathematics and Applications*, chapter 14, pages 505–523. CRC Press, 1994.

[HW96] E. Hernández and G. Weiss. *A First Course on Wavelets*. CRC Press, 1996.

[Jam93] L. Jameson. On the Daubechies-based wavelet differentiation matrix. *J. Sci. Comp.*, 8(3):267–305, 1993.

[Jam94] L. Jameson. On the wavelet-optimized finite difference method. Technical Report NASA CR-191601, ICASE Report No. 94-9, 1994.

[Jam96] L. Jameson. The wavelet-optimized finite difference method. In G. Erlebacher and M. Y. Hussaini an L. Jameson, editors, *Wavelets. Theory and Applications*, chapter 1.5, pages 16–22. Oxford University Press, 1996.

[JS93] B. Jawerth and W. Sweldens. Wavelet resolution analysis adapted for the fast solution of boundary value ordinary differential equations. In *Proceedings of the sixth Copper Mountain Multigrid Conference*, 1993.

[JS94] B. Jawerth and W. Sweldens. An overview of wavelet based multiresolution analysis. *SIAM Review*, 36(3):377–412, 1994.

[Kai94] G. Kaiser. *A Friendly Guide to Wavelets*. Birkhauser, 94.

[Kei95] J. M. Keiser. *Wavelet Based Approach to Numerical Solution of Nonlinear Partial Differential Equations*. PhD thesis, University of Colorado, 1995.

[Kun94] A. Kunoth. Computing integrals of refinable functions - documentation of the program. Technical report, SINTEF, 1994.

[LBA81] M. Lax, J.H. Batteh, and G.P. Agrawal. Channeling of intense electromagnetic beams. *J. Appl. Phys.*, 52(1):109–125, 1981.

[LPT92] J. Liandrat, V. Perrier, and P. H. Tchamitchan. Numerical resolution of nonlinear partial differential equations using the wavelet approach. In Mary Beth Ruskai et al., editors, *Wavelets and their applications*, pages 227–238. Jones and Bartlett, 1992.

[LR94] G. Leaf and J. M. Restrepo. Wavelet-Galerkin discretization of hyperbolic equations. Technical report, Argonne National Laboratory, USA, 1994.

[LRT91] A. Latto, H. L. Resnikoff, and E. Tenenbaum. The evaluation of connection coefficients of compactly supported wavelets. Technical report, AWARE, Inc, One Memorial Drive, Cambridge, MA 02142-1301, USA, 1991.

[LS95] R. Lang and A. Spray. The 2D wavelet transform on a massively parallel machine. In *Proceedings of the Australasian Conference on Parallel and Real-Time Systems*, pages 325–332, Freemantle, Western Australia, 28-29 September 1995 1995.

[LT90a] A. Latto and E. Tenenbaum. Compactly supported wavelets and the numerical solution of burgers' equation. *C.R. Acad. Sci. Paris, t. 311, Série I*, pages 903–909, 1990.

[LT90b] J. Liandrat and P. Tchamitchian. Resolution of the 1D regularized burgers equation using spatial wavelet approximation. Technical Report 90-83, ICASE, 1990.

[Lu93] J. Lu. Parallelizing mallat algorithm for 2-D wavelet transforms. *Inf. Proc. Letters*, 45:255–259, 1993.

[Mal89] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. PAMI*, 11(7):674–693, 1989.

[Mey93] Y. Meyer. *Wavelets: Algorithms and Applications*. SIAM, 1993.

[MR96] M. Metcalf and J. Reid. *Fortran 90/95 explained*. Oxford Science Publications, 1996.

[New92] A. C. Newell. *Nonlinear Optics*. Addison-Wesley, 1992.

[New93] D. E. Newland. *Random Vibrations, Spectral & Wavelet Analysis*. Longman Scientific & Technical, 1993.

[NH95] O. M. Nielsen and M. Hegland. Two-dimensional fast wavelet transform on the VP2200. Semi-annual report on the Fujitsu-ANU parallel mathematical subroutine library project, Australian National University, 1995.

[NH97] O. M. Nielsen and M. Hegland. A scalable parallel 2D wavelet transform algorithm. ANU Computer Science Technical Reports TR-CS-97-21, Australian National University, 1997. *Submitted to Parallel Computing*.

[Nie97] O. M. Nielsen. Fast 2D wavelet transform of circulant matrices. Technical Report IMM-REP-1997-8, Technical University of Denmark, 1997.

[PAP86] M. J. Potasek, G. P. Agrawal, and S. C. Pinault. Analytic and numerical study of pulse broadening in nonlinear dispersive optical fivbers. *J. Opt. Soc. Am. B*, 3(2):205–211, 1986.

[PB91] V. Perrier and C. Basdevant. Travelling wavelets method. In *Proc. of the US-French 'Wavelets and Turbulence' Workshop*. Princeton, 1991.

[PK91] D. Pathria and G. Karniadakis. Wavelet-aided spectral element flow simulations. Preprint Princeton, 1991.

[PW96] I. Pierce and L. Watkins. Modelling optical pulse propagation in nonlinear media using wavelets. Technical report, School of Electronic Engineering and Computer Systems, University of Wales, Bangor, UK, 1996.

[QW93] S. Quian and J. Weiss. Wavelets and the numerical solution of partial differential equations. *J. comp. phys.*, 106:155–175, 1993.

[RE97] T. Regińska and L. Eldén. Solving the sideways heat equation by a wavelet-Galerkin method. *Inv. Prob.*, 13:1093–1106, 1997.

[Smi85] G.D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford University Press, 1985.

[SN96] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.

[Spi93] M. R. Spiegel. *Mathematical Handbook of Formulas and Tables*. Schaum's Outline Series, 1993.

[Str92] G. Strang. Wavelet transforms versus Fourier transforms. Technical report, MIT, Department of Mathematics, MIT, Cambridge MA 02139, 1992.

[Str94] R. S. Stricharz. Construction of orthonormal wavelets. In John J. Benedetto and Michael W. Frazier, editors, *Wavelets - Mathematics and Applications*, chapter 1, pages 23–51. CRC Press, 1994.

[Str96] G. Strang. Creating and comparing wavelets. Available from the author (`gs@math.mit.edu`), 1996.

[Swe] W. Sweldens. Wavelet resources. Available at `http://www.mathsoft.com/wavelets.html`.

[Swe96] W. Sweldens. Wavelets: What next ? *Proc. IEEE*, 84(4):680–685, 1996.

[Uni96] Australian National University. ANU VPP300. Available at `http://anusf.anu.edu.au/VPP/`, 1996.

[VK95] M. Vetterli and J. Kovačević. *Wavelets and Subband Coding*. Prentice Hall, 1995.

[VP96] O. V. Vasilyev and S. Paolucci. A dynamically adaptive multilevel wavelet collocation method for solving partial differential equation in a finite domain. *J. comp. phys.*, 125:498–512, 1996.

[WA94] J. R. Williams and K. Amaratunga. Wavelet-Galerkin solutions for one-dimensional partial differential equations. *Int J. Num. Meth. Eng.*, 37:2703–2716, 1994.

[Wal96] J. Waldén. *Wavelet Solvers for Hyperbolic PDEs*. PhD thesis, Department of Scientific Computing, Uppsala University, 1996.

[WM85] R. Wait and A. R. Mitchell. *Finite Element Analysis and Applications*. John Wiley & Sons Ltd., 1985.

[WZ94] L. Watkins and Y. R. Zhou. Modelling propagation in optical fibers using wavelets. *J. Lightwave Tech.*, 12(9):1536–1542, 1994.

[XS92] J. C. Xu and W. C. Shann. Galerkin-wavelet methods for two point boundary value problems. In *Num. Math.* [Swe], pages 123–144.

Index