Lecture Notes for

AMSC/CMSC 660, Scientific Computing I

By

Jian-Guo Liu and Simon P. Schurr

jliu@math.umd.edu, spschurr@math.umd.edu

Applied Mathematics and Scientific Computation Program

University of Maryland at College Park

December 2001

Chapter Three - Fast Fourier Transforms, Wavelets, and their Applications

1 Introduction

In this chapter we will provide the basic theory and algorithms for (Fast) Fourier Transforms and Wavelet Transforms, which are capable of being applied to many fundamental problems in science and engineering. Such applications include the solving of Differential Equations, image compression, and the use of filtering to recover images.

2 Fourier Transforms

We introduce some basic types of Fourier Transforms (FTs). A Fourier Transform takes an input, which could be a function $f(x)$ or a set of discrete data values $\{f_j\}$ sampled from a function, and produces an output. The output can also be a function $\hat{f}(\xi)$ or a set of discrete data values $\{\hat{f}_k\}$. In general, the inputs and outputs are complex. In the case of one-dimensional inputs and outputs, some FTs and their corresponding inverse FTs are as follows:

Continuous input on $\mathbb{R}$: Let $f : \mathbb{R} \to \mathbb{C}$. Then

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \xi x}\, dx \ ,$$

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x}\, d\xi \ .$$

Continuous periodic input: Let $f : \mathbb{R} \to \mathbb{C}$ be *1-periodic*, viz., $f(x+1) = f(x)\ \forall x \in \mathbb{R}$. Then

$$\hat{f}_k = \int_0^1 f(x) e^{-2\pi i k x}\, dx,\ k \in \mathbb{Z} \ ,$$

$$f(x) = \sum_{k\in\mathbb{Z}} \hat{f}_k e^{2\pi i k x} \ .$$

Discrete input on $\mathbb{R}$: Let $\{f_j\}$ - where $f_j := f(\frac{j}{N})$ - be a set of values sampled from a function $f : \mathbb{R} \to \mathbb{C}$ at equally spaced points (spaced a distance $\frac{1}{N}$ apart). Then

$$\hat{f}(\xi) = \frac{1}{N} \sum_{j\in\mathbb{Z}} f_j e^{-2\pi i j \xi/N} \ ,$$

$$f_j = \int_{-N/2}^{N/2} \hat{f}(\xi) e^{2\pi i j \xi/N} \, d\xi, \ j \in \mathbb{Z} \ .$$

We have restricted the output $\hat{f}(\xi)$ to a finite domain $\xi \in [-\frac{N}{2}, \frac{N}{2})$. ($\frac{N}{2}$ is known as the *Nyquist frequency*.) Therefore if we have a function f whose transform $\hat{f}$ vanishes outside the interval $[-\frac{N_0}{2}, \frac{N_0}{2})$, then we can obtain full information about f from the samples $f(\frac{j}{N})$ if the spacing between the sample points is sufficiently small: $\frac{1}{N} \leq \frac{1}{N_0}$. This result is known as the 'sampling theorem'.

Discrete input on $[-M/2, M/2)$: Let $\{f_j\}$ - where $f_j := f(\frac{j}{N})$ - be a set of values sampled from a function $f : [-M/2, M/2) \to \mathbb{C}$ at equally spaced points (spaced a distance $\frac{1}{N}$ apart). Then

$$\hat{f}_k = \frac{1}{N} \sum_{j=-NM/2}^{NM/2-1} f_j e^{-2\pi i j k/(NM)}, \quad k = -NM/2, \cdots, MN/2 \ ,$$

$$f_j = \frac{1}{M} \sum_{k=-NM/2}^{NM/2} \hat{f}_k e^{2\pi i j k/(NM)}, \quad j = -NM/2, \cdots, MN/2 - 1 \ .$$

Discrete periodic input: Let $\{f_j\}_{j=0}^{N-1}$ - where $f_j := f(\frac{j}{N})$ - be a set of values sampled from a 1-periodic function $f : \mathbb{R} \to \mathbb{C}$. Then

$$\hat{f}_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j e^{-2\pi i j k/N}, \quad k = 0, \cdots, N-1 \ , \tag{1}$$

$$f_j = \sum_{k=0}^{N-1} \hat{f}_k e^{2\pi i j k/N}, \quad j = 0, \cdots, N-1 \ .$$

We also mention three important special cases of the FT with discrete periodic data:

Discrete Sine Transform (DST): Let $\{f_j\}_{j=0}^{N-1}$ - where $f_j := f(\frac{j}{N})$ - be a set of values sampled from an *odd 2-periodic* function $f : \mathbb{R} \to \mathbb{C}$. Then

$$\begin{aligned} \hat{f}_k &= \frac{1}{N} \sum_{j=1}^{N-1} f_j \sin\left(\frac{jk\pi}{N}\right), \quad k = 1, \cdots, N-1 \ , \\ f_j &= \sum_{k=1}^{N-1} \hat{f}_k \sin\left(\frac{jk\pi}{N}\right), \quad j = 1, \cdots, N-1 \ . \end{aligned} \tag{2}$$

We now introduce a variant of the Σ notation that has been used previously:

$$\sum_{j=a}^{b}{}' \, g_j \equiv \left(\sum_{j=a}^{b} \, g_j\right) - \tfrac{1}{2}(g_a + g_b).$$

Discrete Cosine Transform (DCT): Let $\{f_j\}_{j=0}^{N-1}$ - where $f_j := f(\frac{j}{N})$ - be a set of values sampled from an *even 2-periodic* function $f : \mathbb{R} \to \mathbb{C}$. Then

$$\hat{f}_k = \frac{1}{N} \sum_{j=0}^{N}{}' \, f_j \cos\left(\frac{jk\pi}{N}\right), \quad k = 0, \cdots, N \ ,$$

$$f_j = \sum_{k=0}^{N}{}' \, \hat{f}_k \cos\left(\frac{jk\pi}{N}\right), \quad j = 0, \cdots, N \ .$$

Real Discrete Fourier Transform: Let $\{f_j\}_{j=0}^{N-1}$ - where $f_j := f(\frac{j}{N})$ and N is even - be a set of values sampled from a *real 1-periodic* function $f : \mathbb{R} \to \mathbb{R}$. Then

$$a_k = \frac{2}{N} \sum_{j=0}^{N-1} f_j \cos\left(\frac{2jk\pi}{N}\right), \ k = 0, \cdots, \frac{N}{2}, \quad b_k = \frac{2}{N} \sum_{j=1}^{N-1} f_j \sin\left(\frac{2jk\pi}{N}\right), \ k = 1, \cdots, \frac{N}{2}-1 \ ,$$

$$f_j = \sum_{k=0}^{N/2}{}' \left(a_k \cos\left(\frac{2jk\pi}{N}\right) + b_k \sin\left(\frac{2jk\pi}{N}\right)\right), \ j = 0, \cdots, N-1 \ .$$

The factors $\frac{1}{N}$ and $\frac{2}{N}$ which appear in the above Transforms, are normalizing constants:

$$\frac{1}{N} \sum_{j=0}^{N-1} e^{-2\pi ijk/N} e^{2\pi ij\ell/N} = \begin{cases} 1 : & k \equiv \ell \pmod N \\ 0 : & k \not\equiv \ell \pmod N \end{cases} .$$

(Note that $k \equiv \ell \pmod N \iff k = \ell + jN$ for some integer j.)

3 Fast Fourier Transforms

We will now focus our attention on the specific Discrete Fourier Transform (DFT) given in (1). In particular, we investigate the amount of computational effort required to compute a DFT $\hat{f} = (\hat{f}_0, \cdots, \hat{f}_{N-1})^T$ via a 'standard' implementation as well as a much smarter implementation.

When implementing (1), the values of $e^{-2\pi ik/N}$ are assumed to be already stored in a machine, so the computation of these constants can be considered an 'overhead', and will not included in the *operations count*. An operations count is an count of the number of operations (e.g. multiplications/divisions and additions/subtractions) in a computational process. It is not difficult to see from (1) that the computation of each $\hat{f}_k$ will require $O(N)$ multiplications and $O(N)$ additions. Therefore the computation of $\hat{f}$ will require $O(N^2)$ operations in total. It should be noted that since we are multiplying and adding complex numbers in (1), one needs to differentiate between complex operations and real operations. For example, one complex multiplication (i.e. multiplying two complex numbers), requires four real multiplications and two real additions. So although a complex operation is more expensive than a real operation, the computational efforts of the two processes are proportional to each other.

When N is large, as is usually the case in applications, an $O(N^2)$ process will require a prohibitive amount of running time on a computer. However in the 1960s, an implementation of (1) which reduced the operations count from $O(N^2)$ to $O(N\log_2 N)$ was discovered. Since $\log_2 N \ll N$ for large values of N, the new implementation represented a significant improvement. To obtain the improved bound, we will assume for simplicity that N is a power of 2. If N isn't a power of 2, then we can pad f with zeros until its length is a power of 2. However the precise constants in the $O(N\log_2 N)$ bound will depend on the ability to which we can factor N into small primes. Although a factorization of the form $N = 2^\alpha$ is best, factorizations such as $2^\alpha 3^\beta$ or $2^\alpha 3^\beta 5^\gamma$ also give relatively low operations counts.

If N is a power of 2, we can write $N = 2n$ for some integer n. For convenience, define $\omega_N := e^{-2\pi i/N}$ so that $\omega_N^2 = \omega_n$, $\omega_n^n = 1$, and $\omega_N^n = -1$. So from (1) we have that

$$\begin{aligned}
\hat{f}_k &= \frac{1}{N}\sum_{j=0}^{N-1} f_j \omega_N^{jk} \\
&= \frac{1}{2n}\sum_{\ell=0}^{n-1}(f_{2\ell}\omega_N^{2\ell k} + f_{2\ell+1}\omega_N^{(2\ell+1)k}) \\
&= \frac{1}{2n}\sum_{\ell=0}^{n-1} f_\ell^e \omega_n^{\ell k} + \frac{1}{2n}\omega_N^k \sum_{\ell=0}^{n-1} f_\ell^o \omega_n^{\ell k} \\
&= \frac{1}{2}\hat{f}^e + \frac{1}{2}\omega_N^k \hat{f}^o. \qquad (3)
\end{aligned}$$

Similarly it can be shown that

$$\hat{f}_{n+k} = \frac{1}{2}\hat{f}^e - \frac{1}{2}\omega_N^k \hat{f}^o, \qquad (4)$$

where $f^e := (f_0, f_2, \cdots, f_{N-2})^T$ and $f^o := (f_1, f_3, \cdots, f_{N-1})^T$ are n-dimensional vectors containing the even and odd indexed values respectively of f. We have performed what is known as an *even-odd splitting*: we have split a DFT of an N-dimensional vector into two DFTs of n-dimensional vectors. If J_N denotes the number of real operations required to compute a DFT of an N-dimensional vector, it can be shown that $J_N = 2J_{N/2} + 5N$. The final term is due to the n complex multiplications of w_N^k with $\hat{f}^o$ (which is equivalent to $6n$ real operations), as well as the complex addition and subtraction in (3)-(4). (The effect on the operations count of the multiplicative factor of $\frac{1}{2}$ will be ignored, since in practice, (3)-(4) are merely steps in a more complex algorithm, and the multiplication can be embedded in a later computation). Since $J_1 = 0$, we conclude that the operations count is approximately $J_N = 5N\log_2 N$.

The above implementation, which is performed recursively until it is no longer possible to perform a splitting, is called the *Fast Fourier Transform* (FFT). The following Matlab code from [2] computes the FFT of a row vector $x \in \mathbb{C}^N$ whose length is a power of 2. It is NOT the most efficient implementation; it just demonstrates the idea.

```
function y = simple_fft(x)

N = length(x);
if N>1
   ye = simple_fft(x(1:2:end));
   yo = simple_fft(x(2:2:end)).*exp(-2*pi*i*(0:N/2-1)/N);
   y = [ye+yo, ye-yo];
else
   y = x;
end
```

3.1 Numerical Solution of Differential Equations via FFT

As an example of FFT in action, consider the following second order Boundary Value Problem (BVP):

$$u''(x) = f(x), \quad u(0) = u(1) = 0, \quad 0 < x < 1, \tag{5}$$

where $f(x)$ is a given function. This problem is known as the one-dimensional Poisson problem with Dirichlet Boundary Conditions (BCs). The term 'Dirichlet' refers to the fact that at each endpoint of the domain [0,1] we are given the value of the solution $u(x)$, rather than, say, the value of $u'(x)$.

If $f(x)$ is a complicated function, it will not be possible to find the exact (analytical) solution to (5). However we can obtain an approximation to the exact solution by introducing a *finite difference scheme*. We will estimate the values of the exact solution at a finite number of points in [0,1]. For simplicity we will consider a uniform grid consisting of N subintervals. In other words, the domain is discretized by the set of points $\{x_j\}_{j=0}^{N}$ with $x_j = jh$, where $h = 1/N$ is the grid size (i.e. the length of each subinterval).

To represent the value of $u''(x)$ at one of these grid points, it is necessary to find an approximation to it that doesn't involve derivatives. The well-known second order central difference scheme

$$u''(x_j) \approx \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2}, \quad j = 1, 2, \cdots, N-1 \tag{6}$$

is an example of such a formula, where u_j is an approximation to the exact solution at the grid point x_j. Note that the BCs in (5) imply that we can set $u_0 = u_N = 0$ so that at the endpoints, the approximate solution will equal the exact solution. Now define $f_j := f(x_j)$ for $j = 1, 2, \cdots, N-1$. So from (5)-(6) we have the following system of linear equations (one equation for each interior grid point) for the $\{u_j\}$:

$$\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} = f_j, \quad j = 1, 2, \cdots, N-1. \tag{7}$$

Instead of solving the system in (7) via classical matrix methods, we shall transform it into *Fourier space* to find a relation between the $\{\hat{u}_k\}$ and $\{\hat{f}_k\}$ (which are the DSTs of the $\{u_j\}$ and $\{f_j\}$ respectively). As we will see, the relationship in Fourier space is particularly simple. We then transform back into our original space to solve explicitly for the unknowns $\{u_j\}_{j=1}^{N-1}$. Although we are presenting a one-dimensional example here for simplicity, it is important to note that in higher dimensions, solving the linear system in (7) via FFT is cheaper than solving it via classical matrix methods. This is a direct consequence of the $O\left(N \log_2 N\right)$ bound.

Denote the imaginary part of a complex number z by $\mathrm{Im}(z)$. Since $e^{i\theta} = \cos(\theta) + i\sin(\theta)$ for $\theta \in \mathbb{R}$, then $\sin(\theta) = \mathrm{Im}(e^{i\theta})$ and $e^{i\theta} - e^{-i\theta} = 2i\sin(\theta)$. So if we express (7) in terms of the inverse DST in (2), we have (for $j = 1, 2, \cdots, N-1$):

$$\begin{aligned}
\sum_{k=1}^{N-1} \hat{f}_k \sin\left(\frac{jk\pi}{N}\right) &= \sum_{k=1}^{N-1} \hat{u}_k \, \frac{1}{h^2} \, \mathrm{Im}\left(e^{ik\pi(j+1)/N} - 2e^{ik\pi j/N} + e^{ik\pi(j-1)/N}\right) \\
&= \sum_{k=1}^{N-1} \hat{u}_k \, \frac{1}{h^2} \, \mathrm{Im}\left(e^{ijk\pi/N}\left(e^{ik\pi/2N} - e^{-ik\pi/2N}\right)^2\right) \\
&= \sum_{k=1}^{N-1} \hat{u}_k \, \frac{1}{h^2} \, \mathrm{Im}\left(e^{ijk\pi/N}\left(2i\sin\left(\frac{k\pi}{2N}\right)\right)^2\right) \\
&= \sum_{k=1}^{N-1} \hat{u}_k \, \frac{-4}{h^2}\sin^2\left(\frac{k\pi}{2N}\right) \, \sin\left(\frac{jk\pi}{N}\right) \\
&= \sum_{k=1}^{N-1} \hat{u}_k \lambda_k \sin\left(\frac{jk\pi}{N}\right) \;, \qquad (8)
\end{aligned}$$

where

$$\lambda_k := \frac{-4}{h^2}\sin^2\left(\frac{k\pi}{2N}\right), \quad k = 1, 2, \cdots, N-1 \qquad (9)$$

are the *Fourier multipliers*. By equating the coefficients of $\sin(\frac{jk\pi}{N})$ we have the following relation in Fourier space:

$$\hat{f}_k = \lambda_k \hat{u}_k, \quad k = 1, 2, \cdots, N-1.$$

So we have the following algorithm:

• Transform the right-hand side values $\{f_j\}$ (via the FFT implementation of the DST) to obtain $\{\hat{f}_k\}$ in Fourier Space:

$$\hat{f}_k = \frac{1}{N}\sum_{j=1}^{N-1} f_j \sin\left(\frac{jk\pi}{N}\right), \quad k = 1, 2, \cdots, N-1.$$

• Convert the $\{\hat{f}_k\}$ into the $\{\hat{u}_k\}$ via the Fourier multipliers:

$$\lambda_k = \frac{-4}{h^2}\sin^2\left(\frac{k\pi}{2N}\right), \quad \hat{u}_k = \frac{\hat{f}_k}{\lambda_k}, \quad k = 1, 2, \cdots, N-1.$$

• Take the inverse DST (implemented as a FFT) to obtain the numerical solution $\{u_j\}$ of (5):

$$u_j = \sum_{k=1}^{N-1} \hat{u}_k \sin\left(\frac{jk\pi}{N}\right), \quad j = 1, 2, \cdots, N-1.$$

In Matlab, the commands `dst` and `idst` can be used to determine the DST and its inverse.

3.1.1 Generalization to nonhomogeneous Dirichlet BCs and domain $[a,b]$

We now extend the original Dirichlet problem to the case where the BCs are no longer homogeneous, and the domain is no longer the unit domain. For the BVP

$$u''(x) = f(x), \quad u(a) = u_0, \; u(b) = u_N, \quad a < x < b, \tag{10}$$

we discretize the domain uniformly according to $x_j = a + jh, \; j = 0, 1, \cdots, N$, where $h = (b-a)/N$. A further modification is required to handle the nonhomogeneous BCs. For $j = 2, 3, \cdots, N-2$ in (7), the boundary values u_0 and u_N are not present. However when $j = 1, N-1$ we have

$$\frac{u_2 - 2u_1 + u_0}{h^2} = f_1, \quad \frac{u_N - 2u_{N-1} + u_{N-2}}{h^2} = f_{N-1}. \tag{11}$$

By moving the known boundary data to the right-hand sides of the formulas in (11), we have *homogenized* the BCs:

$$\frac{u_2 - 2u_1}{h^2} = f_1 - \frac{u_0}{h^2}, \quad \frac{-2u_{N-1} + u_{N-2}}{h^2} = f_{N-1} - \frac{u_N}{h^2}. \tag{12}$$

Therefore we can solve for $\{u_j\}_{j=1}^{N-1}$ using the same steps that were used in solving the homogeneous BVP on the unit domain, except for the following modifications:
(1) The right-hand side values $\{f_j\}_{j=1}^{N-1}$ are those of $f(x)$ evaluated at $x_1 = a + h, \; x_2 = a + 2h, \cdots, \; x_{N-1} = a + (N-1)h$;
(2) In view of (12), the first and last elements of the vector of f_j values need to be changed from f_1 and f_{N-1} to $f_1 - \frac{u_0}{h^2}$ and $f_{N-1} - \frac{u_N}{h^2}$ respectively.

3.1.2 Generalization to nonhomogeneous Neumann BCs

We now change the Dirichlet BCs of (10) so that instead of knowing the solution $u(x)$ on the boundary, we only know the values of $u'(x)$. Such BCs are known as *Neumann* BCs:

$$u''(x) = f(x), \quad u'(a) = g_0, \; u'(b) = g_N, \quad a < x < b, \tag{13}$$

We have $N+1$ unknowns $\{u_j\}_{j=0}^{N}$. To be able to write out a finite difference scheme in the same way that we have previously, it is necessary to convert the derivative terms $u'(a), u'(b)$ into formulas involving $u(x)$ only. This can be done via a first order central difference formula: $u'(x_j) \approx (u(x_{j+1}) - u(x_{j-1}))/2h$. In our discretization of the domain $[a,b]$, we have $a = x_0$ and $b = x_N$, so the Neumann BCs can be approximated by

$$u'(a) \approx \frac{u_1 - u_{-1}}{2h} = g_0, \quad \text{and} \quad u'(b) \approx \frac{u_{N+1} - u_{N-1}}{2h} = g_N. \tag{14}$$

Since we are only interested in determining $\{u_j\}_{j=0}^{N}$, and not u_{-1} or u_{N+1}, we can eliminate the latter two values by combining (7) for $j = 0$ and N, with (14). Thus

$$\frac{2u_1 - 2u_0}{h^2} = f_0 + \frac{2}{h}g_0, \quad \frac{-2u_N + 2u_{N-1}}{h^2} = f_N - \frac{2}{h}g_N. \tag{15}$$

The $N+1$ unknowns are then determined by solving $N+1$ equations consisting of (7) for $j = 1, 2, \cdots, N-1$, and (15). By applying a DCT to this system of equations after modifying the right-hand side vector to $\tilde{f} := (f_0 + \frac{2}{h} g_0, f_1, \cdots, f_{N-1}, f_N - \frac{2}{h} g_N)^T$, we find that the process of determining a numerical solution to (13) is similar to the algorithm given earlier for finding a numerical solution to the Dirichlet problem (5).

Since (13) only contain derivatives of $u(x)$, but not $u(x)$ itself, the solution is only unique up to an additive constant. That is, if $u(x)$ is a solution, then so is $u(x) + C$ for an arbitrary constant C. Therefore we have one degree of freedom when computing the solution. Typically we choose $\hat{u}_0 = 0$.

So a numerical solution to (13) is found as follows:

- Transform the modified right-hand side values $\{\tilde{f}_j\}$ (via the FFT implementation of the DCT) to obtain $\{\hat{f}_k\}$:

$$\hat{f}_k = \frac{1}{N} \sum_{j=0}^{N}{}' \, \tilde{f}_j \cos\left(\frac{jk\pi}{N}\right), \quad k = 1, 2, \cdots, N.$$

- Convert the $\{\hat{f}_k\}$ into the $\{\hat{u}_k\}$ via the Fourier multipliers:

$$\hat{u}_0 = 0, \quad \lambda_k = \frac{-4}{h^2} \sin^2\left(\frac{k\pi}{2N}\right), \quad \hat{u}_k = \frac{\hat{f}_k}{\lambda_k}, \quad k = 1, 2, \cdots, N.$$

- Take the inverse DCT (implemented as a FFT) to obtain $\{u_j\}$:

$$u_j = \sum_{k=0}^{N}{}' \, \hat{u}_k \cos\left(\frac{jk\pi}{N}\right), \quad j = 0, 1, \cdots, N.$$

In Matlab, the commands `dct` and `idct` can be used to determine the DCT and its inverse.

Another variation on the type of BCs one may encounter when attempting to solve a BVP is a mixed Dirichlet/Neumann BC, e.g., $u(0) = u'(1) = 0$. It can be shown that the numerical solution obeys a sine expansion of the form

$$u_j = \sum_{k=1}^{N} \hat{u}_k \sin\left(\frac{(2k+1)j}{2N}\pi\right). \tag{16}$$

This is known as a *quarter-wave* expansion.

3.1.3 Generalization to 2-dimensional BVPs

Now consider the following two-dimensional Poisson Problem, in which we wish to solve a BVP for a function $u(x, y)$ in two-dimensional space:

$$\begin{gathered} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (x, y) \in \Omega \equiv (0,1) \times (0,1), \\ u(x, y) = 0 \text{ on the boundary of } \Omega. \end{gathered} \tag{17}$$

Discretize the domain Ω into rectangles of width h_x and height h_y by introducing the grid points

$$x_i = ih_x,\ i = 0, 1, \cdots, N_x = \frac{1}{h_x}, \quad y_j = jh_y,\ j = 0, 1, \cdots, N_y = \frac{1}{h_y}.$$

In analogue to (6), define the following central difference approximations to the second derivative terms in (17):

$$D_x^2 u_{i,j} := \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2}, \quad D_y^2 u_{i,j} := \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2}.$$

This gives rise to the following linear system (c.f. (7)):

$$(D_x^2 + D_y^2) u_{i,j} = f_{i,j}, \quad i = 1, \cdots, N_x - 1,\ j = 1, \cdots, N_y - 1, \tag{18}$$

where $u_{i,j}$ and $f_{i,j}$ are approximations to the exact solution $u(x, y)$ and right-hand side $f(x, y)$ respectively at the point (x_i, y_j). Since the exact solution vanishes on the boundary, we also impose the BCs $u_{0,j} = u_{N_x,j} = 0,\ j = 0, 1, \cdots, N_y$, and $u_{i,0} = u_{i,N_y} = 0,\ i = 0, 1, \cdots, N_x$.

Analogously to (2), we can define the *two-dimensional DST* that transforms $\{u_{i,j}\}$ to $\{\hat{u}_{k,\ell}\}$, and its inverse:

$$\hat{u}_{k,\ell} = \frac{1}{N_x N_y} \sum_{i=1}^{N_x-1} \sum_{j=1}^{N_y-1} u_{i,j} \sin\left(\frac{ik\pi}{N_x}\right) \sin\left(\frac{j\ell\pi}{N_y}\right),\ k = 1, \cdots, N_x - 1,\ \ell = 1, \cdots, N_y - 1, \tag{19}$$

$$u_{i,j} = \sum_{k=1}^{N_x-1} \sum_{\ell=1}^{N_y-1} \hat{u}_{k,\ell} \sin\left(\frac{ik\pi}{N_x}\right) \sin\left(\frac{j\ell\pi}{N_y}\right), \quad i = 1, \cdots, N_x - 1,\ j = 1, \cdots, N_y - 1. \tag{20}$$

So

$$D_x^2 u_{i,j} = \sum_{k=1}^{N_x-1} \sum_{\ell=1}^{N_y-1} \hat{u}_{k,\ell} D_x^2 \sin\left(\frac{ik\pi}{N_x}\right) \sin\left(\frac{j\ell\pi}{N_y}\right) = \sum_{k=1}^{N_x-1} \sum_{\ell=1}^{N_y-1} \hat{u}_{k,\ell} \left(\lambda_k \sin\left(\frac{ik\pi}{N_x}\right)\right) \sin\left(\frac{j\ell\pi}{N_y}\right),$$

where

$$\lambda_k := \frac{-4}{h_x^2} \sin^2\left(\frac{k\pi}{2N_x}\right), \quad k = 1, \cdots, N_x - 1 \quad \text{(c.f. (8)-(9))}. \tag{21}$$

Similarly,

$$D_y^2 u_{i,j} = \sum_{k=1}^{N_x-1} \sum_{\ell=1}^{N_y-1} \hat{u}_{k,\ell} \left(\tilde{\lambda}_\ell \sin\left(\frac{j\ell\pi}{N_y}\right)\right) \sin\left(\frac{ik\pi}{N_x}\right),$$

where

$$\tilde{\lambda}_\ell := \frac{-4}{h_y^2} \sin^2\left(\frac{\ell\pi}{2N_y}\right), \quad \ell = 1, \cdots, N_y - 1. \tag{22}$$

So in terms of two-dimensional DSTs, (18) becomes

$$\sum_{k=1}^{N_x-1}\sum_{\ell=1}^{N_y-1} \hat{u}_{k,\ell}(\lambda_k+\tilde{\lambda}_\ell)\ \sin\left(\frac{ik\pi}{N_x}\right)\sin\left(\frac{j\ell\pi}{N_y}\right) = \sum_{k=1}^{N_x-1}\sum_{\ell=1}^{N_y-1} \hat{f}_{k,\ell}\ \sin\left(\frac{ik\pi}{N_x}\right)\sin\left(\frac{j\ell\pi}{N_y}\right),$$

$$\text{i.e.}\quad \hat{u}_{k,\ell}(\lambda_k+\tilde{\lambda}_\ell) = \hat{f}_{k,\ell}\ ,\quad k=1,\cdots,N_x-1,\ \ell=1,\cdots,N_y-1. \tag{23}$$

In an similar way to the one-dimensional case, we then have the following algorithm:

- Find the DST of $\{f_{i,j}\}$ using (19) – with $f,\hat{f}$ replacing $u,\hat{u}$.
- Compute $\{\lambda_k\}$, $\{\tilde{\lambda}_l\}$ using (21)-(22), and then $\{\hat{u}_{k,l}\}$ using (23).
- Compute the numerical solution $\{u_{i,j}\}$ using (20).

4 Image Processing

One of the many applications of Fourier Transforms is in Image Processing. An 'image' can be thought of as a two-dimensional function defined inside some rectangular domain. The function values measure the 'brightness' of the image at certain points in the domain. Although images are continuous functions, they need to be discretized in order to be stored in memory. Hence the need to divide the domain into small subsquares, called 'pixels', and store one representative brightness value from each pixel. However there are some practical difficulties associated with this approach. If we use a 256 ($= 2^8$) color level (so that the brightness values are measured on an integer scale from 0 to 255), then 8 bits of memory are required to store each brightness value. If an accurate discretization of the image is to be obtained, an excessive amount of memory would be required to store the large number of brightness values. In addition, one has to figure how to reverse the discretization process, i.e. how to recover the image (or at least a good approximation to it) from the discrete brightness values. In what follows, we will attempt to address these two concerns.

4.1 Image Compression

Suppose that we divide our domain (which is assumed to be the unit square [0,1] $\times$ [0,1], for simplicity) into N^2 subsquares of equal size. Label the rows and columns $0,1,\cdots,N-1$, and let f_{ij} be the brightness value at the center of the subsquare located in the i^{th} row and j^{th} column. By taking the *two-dimensional DCT* of the $\{f_{ij}\}$ (in Matlab, use the `dct2` command), we obtain the Fourier coefficients $\{\hat{f}_{k\ell}\}$:

$$\hat{f}_{k\ell} = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_{ij}\ \cos\left(\frac{(2i+1)k\pi}{2N}\right)\cos\left(\frac{(2j+1)\ell\pi}{2N}\right),\quad k,\ell=0,\cdots,N-1\ . \tag{24}$$

The two-dimensional DCT is used because its inverse (`idct2` in Matlab)

$$f_{ij} = \sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1} \hat{f}_{k\ell}\ \cos\left(\frac{(2i+1)k\pi}{2N}\right)\cos\left(\frac{(2j+1)\ell\pi}{2N}\right), \quad i,j = 0,\cdots,N-1 \qquad (25)$$

allows for an even extension of the $\{f_{ij}\}$ to $\mathbb{R}^2$. In other words, we can extend the function in the x direction to [0,2] $\times$ [0,1] via an even extension about the line $x = 1$. Now extend the resulting image in the y direction to [0,2] $\times$ [0,2] via an even extension about the line $y = 1$. By continually repeating this process, we obtain the required extension to $\mathbb{R}^2$. There will be no discontinuities at the edges in the even extension to $\mathbb{R}^2$, which is a desirable property. The other reason why we wish to transform the image data $\{f_{ij}\}$ into the Fourier coefficients $\{\hat{f}_{k\ell}\}$ is that the quality of the image can be reasonably retained even if we only use a few of the Fourier coefficients in (25) to reconstruct the image. This idea is known as *image compression*. After computing the Fourier coefficients via (24), we throw away a given proportion of the coefficients whose magnitudes are smallest, i.e. we set these coefficients to zero. The compute the compressed image via (25). Since we have kept the coefficients with the largest influence on the image, the compressed image should approximate the original image reasonably well. (Of course if we throw away too many coefficients, the image will be compressed too much, and we will not be able to satisfactorily recover the original image.)

The image compression process can be implemented without too much difficulty in Matlab. BMP, TIFF, or GIF images are suitable for compression, but JPEG images are not, since they have already been significantly compressed. We can use the `xv` software (which exists on UNIX Operating Systems - simply type `xv &` at the command line to launch the viewer) to cut a portion of an image that can be used for compression. A portion that is sharp and strong in contrast is preferable. This can be done in `xv` as follows: After launching `xv`, click the right mouse button to bring up the controls and load the image. Select "Image Info" from the "Windows" menu. While holding the left mouse button, drag the mouse from one corner to the diagonally opposite corner while the current size is displayed in the "Info" window. When satisfied with the selected area, release the mouse button, click "Crop", and then "Save". When saving, choose the format "TIFF", the color "Greyscale", and no Compression.

Import the cropped portion of the image into Matlab via the Matlab command `im = double(imread('filename.tiff','tiff'))`. The image is now represented as the matrix `im`. Now cut from this portion a smaller portion containing a fixed number of subsquares: `im = im(1:N,1:N)`. The image now contains N^2 subsquares and can be viewed in Matlab via the command `imshow(im,[])`. We now proceed with the image compression by cutting off $x\%$ of the DCT coefficients. In Matlab, this is done as follows:

```
dctim = dct2(im);                   % Compute the two-dimensional DCT of the image.
dctim1 = reshape(dctim,N^2,1);      % Reshape dctim into a vector before sorting.
[s,index] = sort(abs(dctim1));      % Sort the magnitudes of the Fourier coefficients.
cutoff = ceil((x/100)*N^2);         % Compute the number of coefficients to be cut off.
dctim1(index(1:cutoff)) = 0;
dctim0 = reshape(dctim1,N,N));      % Reshape the compressed data into a matrix.
im0 = idct2(dctim0);                % Compute the compressed image via the Inverse DCT.
```

4.2 The Weiner Filter

We now investigate the problem of recovering an original image, given a 'blurred' version of the image that has also been affected by 'noise'. The blurriness in an image is a consequence of the fact that the technology used to obtain the image is not perfect; it still exists even if not visible to the naked eye. The technology that is used to capture an image essentially applies a *response function* that convolves the original (unblurred) image. The response function blurs the original image by smoothing out the sharp features of the image. Since the behavior of a response function can usually be quantified, in principal we could apply the inverse of this response function through a process called *deconvolution* to recover the original image. However we must also take into account the effect of noise, which is always added to the original image in a nondeterministic way. For example, the noise values added to each pixel might be random variables belonging to a Uniform or Normal distribution. This makes it difficult to recover the original image. Suppose that the original image f is given by an N by N matrix of brightness values, as in the previous section. Then the response r and the noise n are also given by matrices of the same size. The blurred noisy image g is then given by

$$g = r * f + n \ , \tag{26}$$

where $r * f$ is the *convolution* of r with f. The (discrete) convolution is given by the matrix with components

$$(r * f)_{i,j} = \sum_{k,\ell} r_{i-k,j-\ell} \, f_{k,\ell} \ .$$

It can then be shown that in the DCT (24), convolution transforms to multiplication:

$$\widehat{(r * f)} = \hat{r}\hat{f} \ ,$$

where all hatted quantities represent two-dimensional DCTs, and on the right-hand side, we have *componentwise* multiplication (NOT matrix multiplication) of the two matrices $\hat{r}$ and $\hat{f}$. In other words, $(\hat{r}\hat{f})_{i,j} \equiv \hat{r}_{i,j} \times \hat{f}_{i,j}$. In Matlab, the syntax is `r_hat.*f_hat` . So after taking the two-dimensional DCT of (26), we obtain

$$\hat{g} = \hat{r}\hat{f} + \hat{n} \ . \tag{27}$$

An approximation to the DCT of the original image, $\hat{f}$, can be obtained by ignoring the effect of noise, and applying deconvolution to (27):

$$\hat{g} \approx \hat{r}\hat{f} \implies \hat{f} \approx \frac{\hat{g}}{\hat{r}} \ . \tag{28}$$

(The division operator in the right-hand side of (28) denotes componentwise division - in Matlab, use `g_hat./r_hat` .) Since our objective is to recover the original image f (via $\hat{f}$) as accurately as possible, we will now investigate whether it is possible to apply a special *filter* to the approximation $\hat{g}/\hat{r}$ so that it better approximates $\hat{f}$. Let the filter (which mathematically speaking, will be a matrix of size N by N) be Q. Then our improved approximation to $\hat{f}$ can be expressed as

$$\hat{h} := \frac{\hat{g}}{\hat{r}}\hat{Q} \ . \tag{29}$$

We hope that if Q is chosen well, then after taking the Inverse DCT of $\hat{h}$, h will be a good approximation to the original image f. One way of achieving this is to minimize the error $h - f$ in the Least Squares sense: $\sum_{i,j=0}^{N-1} \mathrm{E}\left((h_{i,j} - f_{i,j})^2\right)$. (The Least Squares error is also known as the Mean Square error.) $\mathrm{E}(\cdot)$ denotes the Expectation (i.e. mean). This is required because of the nondeterministic nature of the noise, which is present in the $h_{i,j}$ terms. It can be shown (by a result called 'Parseval's identity') that $\mathrm{E}\left((h_{i,j} - f_{i,j})^2\right) = \mathrm{E}((\hat{h}_{i,j} - \hat{f}_{i,j})^2)$, so in the least squares sense, our goal is to find the matrix Q (or $\hat{Q}$) which minimizes $\sum_{k,\ell=0}^{N-1} \mathrm{E}((\hat{h}_{k,\ell} - \hat{f}_{k,\ell})^2)$. This is a separable optimization problem in the sense that the minimum can be found by minimizing each term in the summation separately. In other words, for each k and ℓ, we wish to minimize

$$
\begin{aligned}
\mathrm{E}\left((\hat{h}_{k,\ell} - \hat{f}_{k,\ell})^2\right) &= \mathrm{E}\left(\left(\frac{\hat{g}_{k,\ell}}{\hat{r}_{k,\ell}}\hat{Q}_{k,\ell} - \hat{f}_{k,\ell}\right)^2\right) \\
&= \mathrm{E}\left(\left(\frac{\hat{r}_{k,\ell}\hat{f}_{k,\ell} + \hat{n}_{k,\ell}}{\hat{r}_{k,\ell}}\hat{Q}_{k,\ell} - \hat{f}_{k,\ell}\right)^2\right) \\
&= \mathrm{E}\left(\left(\hat{f}_{k,\ell}(\hat{Q}_{k,\ell} - 1) + \frac{\hat{Q}_{k,\ell}}{\hat{r}_{k,\ell}}\hat{n}_{k,\ell}\right)^2\right) \\
&= (\hat{f}_{k,\ell}(\hat{Q}_{k,\ell} - 1))^2 + 2\hat{f}_{k,\ell}(\hat{Q}_{k,\ell} - 1)\frac{\hat{Q}_{k,\ell}}{\hat{r}_{k,\ell}}\mathrm{E}(\hat{n}_{k,\ell}) + \left(\frac{\hat{Q}_{k,\ell}}{\hat{r}_{k,\ell}}\right)^2 \mathrm{E}(\hat{n}_{k,\ell}^2), \qquad (30)
\end{aligned}
$$

since the values of $\hat{f}_{k,\ell}$, $\hat{Q}_{k,\ell}$, and $\hat{r}_{k,\ell}$ are deterministic, but the $\hat{n}_{k,\ell}$ are random variables. For simplicity, we will assume that the noise values on each pixel have mean zero, are uncorrelated, and have the same standard deviation σ, i.e.

$$
\mathrm{E}(n_{i,j}) = 0 \;\; \forall\; i,j, \quad \text{and} \quad \mathrm{Cov}(n_{i_1,j_1}, n_{i_2,j_2}) = \sigma^2 \delta_{i_1 i_2}\delta_{j_1 j_2} \;\; \forall\; i_1, j_1, i_2, j_2. \qquad (31)
$$

(Given two random variables X and Y, $\mathrm{Cov}(X,Y) := \mathrm{E}[(X - \mathrm{E}(X))(Y - \mathrm{E}(Y))]$ denotes their Covariance, and δ denotes the Kronecker delta: $\delta_{ij} = 1$ if $i = j$, and 0 otherwise.) The 'mean zero' assumption simply says that in some pixels, the noise is expected to add brightness to the image, and the total increase in brightness is balanced by the decrease in brightness that the noise inflicts on the other pixels.

It can be shown that the DCT of the noise satisfies the same properties in (31) as the noise itself. For example

$$
\begin{aligned}
\mathrm{E}(\hat{n}_{k,\ell}) &= \mathrm{E}\left(\frac{1}{N^2}\sum_{i,j} n_{i,j} \cos\left(\frac{(2i+1)k\pi}{2N}\right)\cos\left(\frac{(2j+1)\ell\pi}{2N}\right)\right), \\
&= \frac{1}{N^2}\sum_{i,j} \mathrm{E}(n_{i,j}) \cos\left(\frac{(2i+1)k\pi}{2N}\right)\cos\left(\frac{(2j+1)\ell\pi}{2N}\right), \\
&= 0 \qquad \text{from (31).}
\end{aligned}
$$

It then follows from (30) that

$$\mathrm{E}\left((\hat{h}_{k,\ell}-\hat{f}_{k,\ell})^2\right) = (\hat{f}_{k,\ell}(\hat{Q}_{k,\ell}-1))^2 + \left(\frac{\hat{Q}_{k,\ell}}{\hat{r}_{k,\ell}}\right)^2 \sigma^2 \ .$$

The right-hand side is a quadratic function of $\hat{Q}_{k,\ell}$ which achieves its minimum value when

$$\frac{d\mathrm{E}\left((\hat{h}_{k,\ell}-\hat{f}_{k,\ell})^2\right)}{d\hat{Q}_{k,\ell}} = 2\hat{f}^2_{k,\ell}(\hat{Q}_{k,\ell}-1) + 2\frac{\hat{Q}_{k,\ell}}{\hat{r}^2_{k,\ell}}\sigma^2 = 0 \ ,$$

$$\text{i.e.} \quad \hat{Q}_{k,\ell} = \frac{1}{1+\left(\frac{\sigma}{\hat{f}_{k,\ell}\hat{r}_{k,\ell}}\right)^2} \ . \tag{32}$$

The matrix Q whose DCT has the elements given in (32) is known as the 'Weiner filter' or 'Optimum Filter'. In practice the values of $\hat{Q}_{k,\ell}$ cannot be computed precisely since $\hat{f}_{k,\ell}$ is not known. Instead, we set $\hat{Q}_{k,\ell} = \frac{1}{1+\alpha^2}$, and experiment with different values of α to obtain the best approximation to the original image f. In other words, for a range of α values, we compute $\hat{Q}_{k,\ell} = \frac{1}{1+\alpha^2}$ for all k, ℓ, then $\hat{h}$ via (29), and finally we apply the Inverse DCT to $\hat{h}$ to obtain $h \approx f$.

5 Wavelets

In the previous section, we expressed a two-dimensional image $f(x)$ in terms of the basis functions $\{\cos\left(\frac{(2j+1)\ell\pi}{2N}\right)\}$ - c.f. (25). We shall now assume for simplicity that f is one-dimensional. It is necessary to assume that f is L^2-integrable on $\mathbb{R}$, viz., $\int_{-\infty}^{\infty} f(x)^2\, dx < \infty$ (so f is not required to be continuous). We will now consider a new basis involving *Wavelet functions* in which we can represent f. Roughly speaking, a Wavelet function has the property that under certain types of translation and scaling, we can form a basis for the space of all L^2-integrable functions. Such functions provide a good basis for approximating signals and images. We will study both the properties and construction of these functions.

As an example of a simple wavelet, consider the approximation of a function f by a piecewise constant function. Firstly we choose an integer j such that f is sampled at the grid points $x_{j,k} = kh_j$ for $k \in \mathbb{Z}$, where $h_j = 1/2^j$, is the grid size. We call j the 'level of the grid'. One possible form for the piecewise constant approximant of f is

$$f_{j,k} = \frac{1}{h_j}\int_{x_{j,k}}^{x_{j,k+1}} f(x)\, dx \ , \quad x_{j,k} < x < x_{j,k+1} \ , \quad k \in \mathbb{Z} \ . \tag{33}$$

Note that $f_{j,k}$ is simply the mean value of f on the interval $[x_{j,k}\, , x_{j,k+1}]$. In view of $x_{j-1,k} = k/2^{j-1} = 2k/2^j = x_{j,2k}$, we see that when we increase the level of the grid, from $j-1$ to j, we are doubling the density of grid points. The grid at level j is the grid at level $j-1$ supplemented by new grid points halfway between each pair of grid points in level $j-1$. It is not difficult to show using (33) that the heights of the piecewise constant approximants at levels $j-1$ and j are related by

$$f_{j-1,k} = \frac{1}{2}(f_{j,2k} + f_{j,2k+1}) \ . \tag{34}$$

Now define

$$g_{j-1,k} = \frac{1}{2}(f_{j,2k} - f_{j,2k+1}) \ , \quad c_{j,k} = 2^{j/2} f_{j,k} \ , \quad d_{j,k} = 2^{j/2} g_{j,k} \ . \tag{35}$$

We than have the following recursive relations between the $c_{\cdot,\cdot}$ and $d_{\cdot,\cdot}$ values at levels $j-1$ and j:

$$c_{j-1,k} = \frac{1}{\sqrt{2}}(c_{j,2k} + c_{j,2k+1}) \ , \quad d_{j-1,k} = \frac{1}{\sqrt{2}}(c_{j,2k} - c_{j,2k+1}), \tag{36}$$

$$c_{j,2k} = \frac{1}{\sqrt{2}}(c_{j-1,k} + d_{j-1,k}) \ , \quad c_{j,2k+1} = \frac{1}{\sqrt{2}}(c_{j-1,k} - d_{j-1,k}). \tag{37}$$

((36) and (37) are known as the Haar Wavelet Transform (WT) and the Haar Inverse Wavelet Transform (IWT).)

We now extend the ideas used in formulating the Haar WT and IWT by introducing the concept of a *filter bank*. A filter bank is simply a collection of sets of coefficients in any generalized WT and IWT: $\{a_\ell\}$, $\{b_\ell\}$, $\{\tilde{a}_\ell\}$, $\{\tilde{b}_\ell\}$, where

$$c_{j-1,k} = \sum_\ell a_\ell c_{j,2k+\ell} \ , \quad d_{j-1,k} = \sum_\ell b_\ell c_{j,2k+\ell} \ , \tag{38}$$

and

$$c_{j,2k} = \sum_\ell (\tilde{a}_{-2\ell} c_{j-1,k+\ell} + \tilde{b}_{-2\ell} d_{j-1,k+\ell}) \ , \quad c_{j,2k+1} = \sum_\ell (\tilde{a}_{-2\ell+1} c_{j-1,k+\ell} + \tilde{b}_{-2\ell+1} d_{j-1,k+\ell}) \tag{39}$$

represent the generalized WT and IWT respectively. The coefficients $\{a_\ell\}$, $\{b_\ell\}$ are said to correspond to 'low-pass' and 'high-pass' filters respectively. One of the main features of modern Wavelet theory is that each of the four sequences of coefficients in the filter bank have *compact support*. This means that all of the coefficients are zero except for a small number of consecutive nonzero coefficients. Therefore when we have to compute infinite series with the filter bank coefficients serving as the coefficients of that series, we need only evaluate a small number of terms. In fact a comparison of (36)-(37) with (38)-(39) shows that for the Haar Wavelets, we have

$$a_0 = \tilde{a}_0 = a_1 = \tilde{a}_1 = b_0 = \tilde{b}_0 = \frac{1}{\sqrt{2}}, \quad b_1 = \tilde{b}_1 = \frac{-1}{\sqrt{2}} \ . \tag{40}$$

Now it is not difficult to verify that the IWT in (39) can be written as

$$c_{j,k} = \sum_\ell (\tilde{a}_{k-2\ell} c_{j-1,\ell} + \tilde{b}_{k-2\ell} d_{j-1,\ell}) \ , \quad k \in \mathbb{Z}. \tag{41}$$

We will now find conditions on the filter bank, so that in going from level $j-1$ to j and then back to level $j-1$ (or vice-versa), the values of $c_{\cdot,\cdot}$ and $d_{\cdot,\cdot}$ are preserved. This is necessary for consistency. To do this, we define the following Fourier series, also known as 'mask functions':

$$a(z) = \sum_\ell a_\ell z^\ell \ , \quad b(z) = \sum_\ell b_\ell z^\ell \ , \quad \tilde{a}(z) = \sum_\ell \tilde{a}_\ell z^\ell \ , \quad \tilde{b}(z) = \sum_\ell \tilde{b}_\ell z^\ell \ ,$$

$$\text{and} \quad c_j(z) = \sum_k c_{j,k} z^k \ , \tag{42}$$

where $z = e^{2\pi i\theta}$ is a complex number of unit length. From (41)-(42) we see that

$$c_j(z) = \sum_{k,\ell} (\tilde{a}_{k-2\ell} c_{j-1,\ell} + \tilde{b}_{k-2\ell} d_{j-1,\ell}) z^k = \tilde{a}(z) c_{j-1}(z^2) + \tilde{b}(z) d_{j-1}(z^2), \tag{43}$$

and

$$\begin{aligned} c_{j-1}(z^2) &= \sum_k c_{j-1,k} z^{2k} = \sum_{k,\ell} a_\ell c_{j,2k+\ell} z^{2k} \frac{1+(-1)^{2k}}{2} = \sum_{m,\ell} a_\ell c_{j,m+\ell} z^m \frac{1+(-1)^m}{2} \\ &= \frac{1}{2} \sum_{m,\ell} a_\ell c_{j,m+\ell} z^m + \frac{1}{2} \sum_{m,\ell} a_\ell c_{j,m+\ell} (-z)^m \\ &= \frac{1}{2} a(z^{-1}) c_j(z) + \frac{1}{2} a(-z^{-1}) c_j(-z). \end{aligned} \tag{44}$$

Similarly,

$$d_{j-1}(z^2) = \frac{1}{2} b(z^{-1}) c_j(z) + \frac{1}{2} b(-z^{-1}) c_j(-z). \tag{45}$$

Combining (43)-(45), gives

$$c_j(z) = \frac{1}{2} (\tilde{a}(z) a(z^{-1}) + \tilde{b}(z) b(z^{-1})) c_j(z) + \frac{1}{2} (\tilde{a}(z) a(-z^{-1}) + \tilde{b}(z) b(-z^{-1})) c_j(-z) \ . \tag{46}$$

Since this must hold for any z (of unit length), we can equate the coefficients of $c_j(z)$ and $c_j(-z)$ to obtain the following conditions on the mask functions:

$$\tilde{a}(z) a(z^{-1}) + \tilde{b}(z) b(z^{-1}) = 2 \ , \quad \tilde{a}(z) a(-z^{-1}) + \tilde{b}(z) b(-z^{-1}) = 0. \tag{47}$$

These conditions are well-known in the signal processing engineering. They are called the perfect (or exact) reconstruction conditions. These conditions alone are not enough to uniquely determine the mask functions. Therefore we impose some more conditions. (See [1] - page 162 for a discussion of the different types of conditions that can be imposed.) We will consider the case where

$$\tilde{a}(z) = a(z), \quad \tilde{b}(z) = b(z), \quad b(z) = -z a(-z^{-1}) \ . \tag{48}$$

The filters chosen as above are called quadrature mirror filters (QMF). (use Conjugate Quadrature Filters to be associated with wavelet functions. See page 163 of [1].) Hence

$$b(z) = -z \sum_\ell a_\ell (-z)^{-\ell} = \sum_\ell a_\ell (-z)^{1-\ell} = \sum_m (-1)^m a_{1-m} z^m \ .$$

So the relationships between the filter bank coefficients are given by

$$b_k = (-1)^k a_{1-k}, \quad \tilde{a}_k = a_k, \quad \tilde{b}_k = b_k \ , \tag{49}$$

and from (47)-(48), we have that the $\{a_k\}$ are restricted by

$$a(z) a(z^{-1}) + a(-z) a(-z^{-1}) = 2. \tag{50}$$

In the case of a real filter bank, this reduces to

$$|a(z)|^2 + |a(-z)|^2 = 2. \tag{51}$$

As an example,

$$\begin{aligned} 2 &= 2(\cos^2(\pi\theta) + \sin^2(\pi\theta)) \\ &= |\sqrt{2}e^{i\pi\theta}\cos(\pi\theta)|^2 + |-\sqrt{2}ie^{i\pi\theta}\sin(\pi\theta)|^2 \\ &= |a(z)|^2 + |a(-z)|^2 \ , \end{aligned}$$

where

$$a(z) = \sqrt{2}e^{i\pi\theta}\cos(\pi\theta) = \sqrt{2}e^{i\pi\theta}\left(\frac{e^{-i\pi\theta} + e^{i\pi\theta}}{2}\right) = \frac{1}{\sqrt{2}}(1 + e^{2i\pi\theta}) = \frac{1}{\sqrt{2}}(1 + z) \ .$$

Comparing this with $a(z) = \sum_\ell a_\ell z^\ell$ and (49), we obtain the coefficients in (40). (Note that $a(z)$ does not uniquely satisfy (51).) The other filter bank coefficients are given by (49). This is actually the Haar wavelet, the simplest type of wavelet. A slightly more complicated wavelet known as the 'db2 wavelet' (named after Daubechies, who is one of pioneers in developing Wavelet theory). The wavelet is more complicated in the sense that each of the mask functions consists of four nonzero terms (as opposed to two in the Haar wavelet). However the extra nonzero terms in the mask functions allow for a more accurate representation of a function by continuous piecewise linear basis functions, instead of the piecewise constant basis functions used for Haar wavelets. The coefficients in the db2 filter bank are determined as follows:

$$\begin{aligned} 2 &= 2(\cos^2(\pi\theta) + \sin^2(\pi\theta))^3 \\ &= 2\cos^4(\pi\theta)(\cos^2(\pi\theta) + 3\sin^2(\pi\theta)) + 2\sin^4(\pi\theta)(\sin^2(\pi\theta) + 3\cos^2(\pi\theta)) \\ &= |a(z)|^2 + |a(-z)|^2 \ , \end{aligned}$$

where

$$\begin{aligned} a(z) &= \sqrt{2}e^{3\pi i\theta}\cos^2(\pi\theta)(\cos(\pi\theta) - i\sqrt{3}\sin(\pi\theta)) \\ &= \sqrt{2}e^{3\pi i\theta}\left(\frac{e^{\pi i\theta} + e^{-\pi i\theta}}{2}\right)^2\left(\frac{e^{\pi i\theta} + e^{-\pi i\theta}}{2} - i\sqrt{3}\frac{e^{\pi i\theta} - e^{-\pi i\theta}}{2i}\right) \\ &= \frac{1}{4\sqrt{2}}\left(e^{2\pi i\theta} + 1\right)^2\left(e^{2\pi i\theta} + 1 - \sqrt{3}(e^{2\pi i\theta} - 1)\right) \\ &= \frac{1}{4\sqrt{2}}(z+1)^2(z + 1 - \sqrt{3}(z-1)) \ . \end{aligned}$$

Upon expanding $a(z)$ - which is nonunique - in powers of z, we obtain

$$a_0 = \frac{1+\sqrt{3}}{4\sqrt{2}}, \quad a_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}, \quad a_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}, \quad a_3 = \frac{1-\sqrt{3}}{4\sqrt{2}}. \tag{52}$$

In general, the coefficients in the Daubechies wavelets of order n can be determined similarly as above:

$$\begin{aligned}
2 &= 2(\cos^2(\pi\theta) + \sin^2(\pi\theta))^{2n-1} \\
&= 2\cos^n(\pi\theta)\sum_{k=0}^{n-1}\binom{2n-1}{k}\cos^{2(n-1-k)}(\pi\theta)\sin^{2k}(\pi\theta)) + 2\sin^n(\pi\theta)\sum_{k=0}^{n-1}\binom{2n-1}{k}\sin^{2(n-1-k)}(\pi\theta)\cos^{2k}(\pi \\
&= |a(z)|^2 + |a(-z)|^2 \ ,
\end{aligned}$$

where $a(z)$ is a square root of $2\cos^n(\pi\theta)\sum_{k=0}^{n-1}\binom{2n-1}{k}\cos^{2(n-1-k)}(\pi\theta)\sin^{2k}(\pi\theta))$ by using Riesz's lemma (cf. [1]).

As usual, the other filter bank coefficients are given by (49). In Matlab, the commands `dbwavf('haar')` and `dbwavf('db2')` generate the coefficients a_ℓ in the filter bank. However they are different from those given in (40) and (52), since in Matlab, they are linearly scaled so that they sum to one.

As we have seen previously, associated with a filter bank is a Wavelet Transform. We now turn to the problem of computing basis functions. Consider the following basis functions: The *Refinable function*

$$\phi(x) = \sqrt{2}\sum_\ell a_\ell \phi(2x-\ell) \ , \tag{53}$$

the *Scaling function* $\phi(x)$ is refinable and orthonormal

$$\int_{-\infty}^{\infty}\phi(x)\phi(x-\ell)dx = \begin{cases} 1, & \text{if } \ell = 0, \\ 0, & \text{otherwise} \ , \end{cases} \tag{54}$$

and the *Wavelet function*

$$\psi(x) = \sqrt{2}\sum_\ell b_\ell \phi(2x-\ell) \tag{55}$$

if ϕ is a scaling function. Usually we will not be able to find a simple analytic formula for ϕ (or ψ), but it is possible to compute ϕ to arbitrarily high precision using what is known as the *Cascade Algorithm*. The idea is simply to iterate on (53):

$$\phi^{n+1}(x) = \sqrt{2}\sum_\ell a_\ell \phi^n(2x-\ell) \ , \quad n = 0, 1, \cdots .$$

where some initial guess $\phi^0(x)$ has been prescribed, e.g. $\phi^0(x) = \chi_{[0,1)}(x)$, where

$$\chi_{[a,b)}(x) := \begin{cases} 1 & : \ \text{if } a \le x < b \\ 0 & : \ \text{else} \end{cases} .$$

After a suitable approximation has been found to $\phi(x)$ by iterating the Cascade Algorithm, we can compute an approximation to the $\psi(x)$ via (55).

In the case of the Haar wavelet (with coefficients from (40)), it *is* possible to find a simple analytic formula for ϕ. It is easy to verify that it is given by

$$\phi(x) = \chi_{[0,1)}(x) \ , \tag{56}$$

$$\text{since} \quad \phi(2x) = \chi_{[0,1/2)}(x), \quad \text{and} \quad \phi(2x-1) = \chi_{[1/2,1)}(x),$$

so that

$$\phi(x) = \phi(2x) + \phi(2x-1),$$

which is (53). The Wavelet function is given by

$$\psi(x) = \chi_{[0,1/2)}(x) - \chi_{[1/2,1)}(x) = \begin{cases} 1 & : \text{ if } 0 \le x < 1/2 \\ -1 & : \text{ if } 1/2 \le x < 1 \\ 0 & : \text{ else} \end{cases} .$$

In the case of the db2 wavelet however, things are more complicated. The following code plots the Scaling and Wavelet functions (see Figure 1) with 10 iterations (levels) of the db2 filter. Unlike in (56), there is no simple formula for the Scaling (or Wavelet) function. As mentioned previously, the $\{a_\ell\}$ values given in Figure 1 are different from those in Matlab. For consistency, Matlab also scales the Scaling and Wavelet functions: in (53)-(55), $\sqrt{2}$ becomes 2.

```
% Compute the db2 coefficients:
[PHI,PSI,XVAL] = wavefun('db2',10);

% Plot the scaling function:
subplot(2,1,1), plot(XVAL,PHI);

% Plot the wavelet function:
subplot(2,1,2), plot(XVAL,PSI);
```

We now explain how Scaling and Wavelet functions can be used to obtain (38)-(39). Suppose that these functions take on the following form:

$$\phi_{j,k}(x) = 2^{j/2}\phi(2^j x - k), \quad \psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k) \ . \tag{57}$$

Here k represents the translation of $\phi(x)$ and $\psi(x)$, and 2^j represents a scaling. Then from (53)-(55) we have that

$$\phi_{j-1,k}(x) = \sum_\ell a_\ell \phi_{j,2k+\ell}(x), \quad \psi_{j-1,k}(x) = \sum_\ell b_\ell \phi_{j,2k+\ell}(x) \ . \tag{58}$$

Now for two functions $f(x), g(x)$ that are L^2-integrable on $\mathbb{R}$, define the following inner product:

$$\langle f(x), g(x) \rangle = \int_{-\infty}^{\infty} f(x)g(x)\ dx \ .$$

It can then be shown that the $\{\phi_{j,k}\}$ from (57) satisfy the following properties:

$$\langle \phi_{j,k}, \phi_{j,\ell} \rangle = \delta_{k\ell} \quad \forall\, j,k,\ell \in \mathbb{Z} \ ,$$

$$\langle \phi_{j,k}, \psi_{j,\ell} \rangle = 0 \quad \forall\, j,k,\ell \in \mathbb{Z} \ ,$$

$$\langle \psi_{j,k}, \psi_{\ell,m} \rangle = \delta_{j\ell}\delta_{km} \quad \forall\, j,k,\ell \in \mathbb{Z} \ . \tag{59}$$

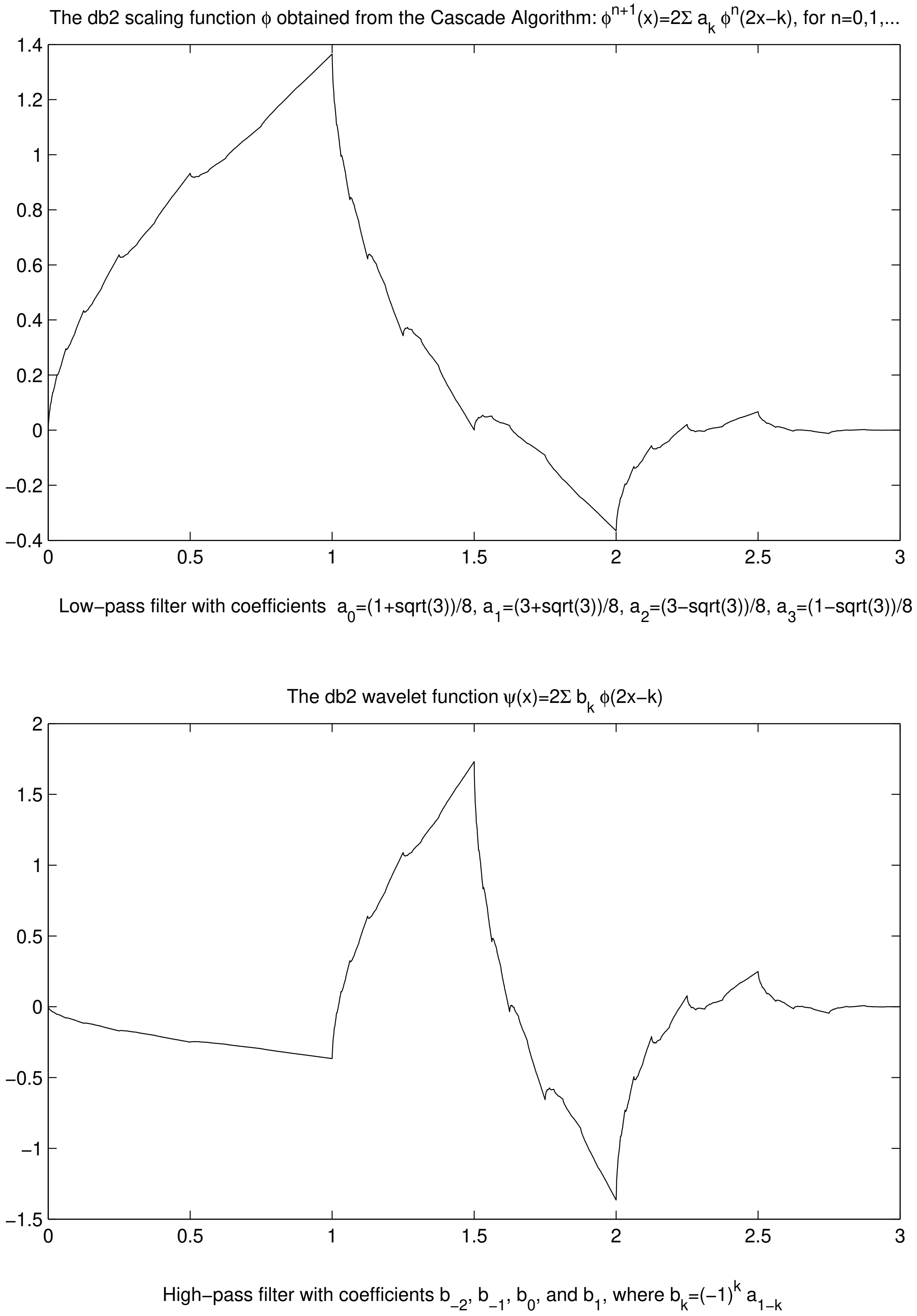

Figure 1: Plots of the Scaling and Wavelet functions for the db2 filter.

From (59) we have that the $\{\psi_{j,k}\}$ are orthornormal. If $\int_{\infty}^{\infty} \phi(x)dx = 1$, i.e., the coefficients a_ℓ's sum to one, we can show that $\{\psi_{j,k}\}$ is an orthonormal basis for L^2. Hence any function $f \in L^2$ can be represented as

$$f(x) = \sum_{j,k} \langle f, \psi_{j,k} \rangle \psi_{j,k}(x) \ . \tag{60}$$

In analogue to (35), define

$$c_{j,k} = \langle f, \phi_{j,k} \rangle, \quad d_{j,k} = \langle f, \psi_{j,k} \rangle \ . \tag{61}$$

In practice, we do not compute the formulas in (61), since we would be required to numerically evaluate the integrals. Rather, we use the formulas for $c_{j,k}$ and $d_{j,k}$ in (35) as approximations. Now from (58), we have that

$$c_{j-1,k} = \sum_\ell a_\ell c_{j,2k+\ell} \ , \quad d_{j-1,k} = \sum_\ell b_\ell c_{j,2k+\ell} \ . \tag{62}$$

This is known as the *single-level* Discrete Wavelet Transform (DWT). The process of iterating this transform from level j to 0 in increments of 1 is known as *wavelet decomposition.* The result is the *multi-level* DWT:

$$\{c_{j,k}\}_{k\in\mathbb{Z}} \to \{c_{0,k}, d_{0,k}, d_{1,k}, \cdots, d_{j-1,k}\}_{k\in\mathbb{Z}} \ . \tag{63}$$

In other words, we compute the $\{c_{j,k}\}$, and then $\{d_{i,k}\}$ recursively for $i = 0, 1, \cdots, j-1$. These are precisely the coefficients in (60) and gives us a representation of f by wavelets.

The inverse of this transform is obtained recursively by a process known as *wavelet reconstruction*:

$$c_{j,2k} = \sum_\ell (\tilde{a}_{-2\ell} c_{j-1,k+\ell} + \tilde{b}_{-2\ell} d_{j-1,k+\ell}), \quad c_{j,2k+1} = \sum_\ell (\tilde{a}_{-2\ell+1} c_{j-1,k+\ell} + \tilde{b}_{-2\ell+1} d_{j-1,k+\ell}). \tag{64}$$

This is (38)-(39). The Matlab commands `dwt`, `idwt`, `waverec`, and `wavedec` implement single and multi-level one-dimensional DWTs.

References

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

[2] T. von Petersdorff's webpage: `http://www.glue.umd.edu/~tvp/660/` .

[3] J. S. Walker, *A Primer on Wavelets and their Scientific Applications*, 1999.