Chapter 2
Algorithms for Periodic Functions

In this chapter we show how to compute the Discrete Fourier Transform using a Fast Fourier Transform (FFT) algorithm, including not-so special case situations such as when the data to be transformed are real. In those situations, we speed up the transforms by about a factor of two by exploiting symmetries in the data and the coefficients. We end this chapter by showing how to approximate the derivatives of periodic functions, which are the fundamental approximations that we need to solve partial differential equations with periodic boundary conditions.

2.1 How to Compute the Discrete Fourier Transform

The sums in the Discrete Fourier Transform pair, (1.69), cost $O(N^2)$ operations to compute: N complex multiplications for each of the N values. For large N, they are too expensive to compute by the direct sum. However, we can compute the sums in a significantly more efficient $O(N \log_2(N))$ operations if we use the Fast Fourier Transform. The complex FFT is the core algorithm for the efficient implementation of Fourier spectral methods for large N.

It is beyond the scope of this book to give a thorough presentation of Fast Fourier Transform algorithms. FFTs are ubiquitous and are discussed in detail in numerous books (e.g. [5] and [6], among others), and the basic ideas are found in upper level numerical analysis books. Implementations exist in numerous programming languages tuned for virtually any computer. Rather than use an FFT presented in any book, we recommend libraries provided by one's computer vendor, a well-developed and tested FFT algorithm such as FFTW [12] (http://www.fftw.org/) or that can be found, for example, at netlib (www.netlib.org), or routines sold as part of one of the many commercial numerical libraries (e.g. NAG or IMSL). For completeness, however, we include Temperton's [23] self-sorting in-place complex FFT. Temperton's paper includes a mixed-radix FFT ($N = 2^m 3^q 5^r$), but for reasons of space and simplicity, we present only the radix 2 ($N = 2^m$) algorithm here.

In spectral method applications, the solutions of the differential equations are typically real. When the sequences are real, we can exploit symmetries in the coefficients to save approximately a factor of two in the work to compute the transforms. We present two algorithms that use these symmetries in this section. One simultaneously computes the coefficients of two real sequences with a single complex FFT. The other splits a single real sequence into two sequences of half the original length by putting the even indexed elements into the real part of a complex sequence and the odd indexed elements into the imaginary part.

2.1.1 Fourier Transforms of Complex Sequences

We can recast the Discrete Fourier Transform (DFT) pair (1.69) as

$$\text{DFT} \quad \begin{cases} G_k = \frac{1}{N}\sum_{j=0}^{N-1} g_j e^{-2\pi ijk/N}, & k = 0, 1, \ldots, N-1, \\ g_j = \sum_{k=0}^{N-1} G_k e^{2\pi ijk/N}, & j = 0, 1, \ldots, N-1. \end{cases} \tag{2.1}$$

The first sum, which represents the decomposition of the sequence of physical space values into its Fourier components, is the *forward transform*. It takes an N-periodic sequence $\{g_j\}_{j=0}^{N-1}$ and returns an N-periodic sequence $\{G_j\}_{j=0}^{N-1}$ of *Discrete Fourier coefficients*. (An N-periodic sequence is one for which $g_{j\pm N} = g_j$.) The constant $i = \sqrt{-1}$. The second sum, which represents the synthesis of the Fourier modes back into the physical space, is the *backward transform*. Properties of the transforms can be found in numerous books, including [6].

Except for the sign of the exponents, the forward and backward transforms compute the same sum, namely

$$\text{DFTS:} \quad F_k = \sum_{j=0}^{N-1} f_j e^{\pm 2\pi ijk/N}, \quad k = 0, 1, \ldots, N-1. \tag{2.2}$$

We could compute the *Discrete Fourier Transform Sum*, DFTS of (2.2), directly by Algorithm 6 (DFT). The sign of the input variable s determines whether the procedure computes the forward or backward sum. To avoid having to remember which sign corresponds to which transform, let's define two constants $FORWARD = 1$ and $BACKWARD = -1$ to use as input to the procedure.

The range of wavenumbers, k, used in (2.1) is not the range we need to compute the Fourier interpolant (1.69). For that we need the coefficients in the order Algorithm 6 (DFT), and (2.2) each return a sequence with elements ordered as $k = 0, 1, \ldots, N-1$ instead.

Algorithm 6: *DFT*: Direct (and Slow) Evaluation of the Discrete Fourier Transform

```
Procedure DFT
Input: $\{f_j\}_{j=0}^{N-1}$, $s$
for $k = 0$ to $N-1$ do
    $F_k \leftarrow 0$
    for $j = 0$ to $N-1$ do
        $F_k \leftarrow F_k + f_j * e^{-2s\pi ijk/N}$
    end
end
return $\{F_k\}_{k=0}^{N-1}$
End Procedure DFT
```

For the DFT to be useful in spectral approximations, we must reinterpret the order of the sequences. Equation (1.63) shows that the discrete Fourier coefficient $\tilde{f}_k$ are N-periodic, that is, $\tilde{f}_k = \tilde{f}_{k\pm N}$. Thus, $\tilde{f}_{N/2} = \tilde{f}_{-N/2}$, and therefore the second half of the sequence returned by the transform with $s = -1$ corresponds to the negative values of the index. To use the results of the sum (2.2), i.e. of Algorithm 6 (DFT), we make the following correspondence:

$$\begin{bmatrix} F_0 \\ F_1 \\ \vdots \\ F_{N/2-1} \\ F_{N/2} \\ F_{N/2+1} \\ \vdots \\ F_N \end{bmatrix} \Longleftrightarrow \begin{bmatrix} N\tilde{f}_0 \\ N\tilde{f}_1 \\ \vdots \\ N\tilde{f}_{N/2-1} \\ N\tilde{f}_{-N/2} \\ N\tilde{f}_{-N/2+1} \\ \vdots \\ N\tilde{f}_{-1} \end{bmatrix}. \tag{2.3}$$

To reconstruct the set of values $\{f_j\}_{j=0}^{N-1}$ from the set of coefficients $\{\tilde{f}_k\}_{k=-N/2}^{N/2-1}$, we simply order them using the correspondence (2.3) before calling the transform with $s = +1$. We will show an example in Sect. 2.3 when we compute Fourier interpolation derivatives by FFTs.

It is highly unlikely that one would want use Algorithm 6 for anything but the simplest of exercises. That algorithm clearly requires $O(N^2)$ complex exponentiations and multiplications. Indeed, the early success of spectral methods was due the fact that one could use a Fast Fourier Transform (FFT) to compute these sums with only $O(N\,\mathrm{Log}_2(N))$ operations.

To help understand how to use an FFT routine, we present Temperton's self-sorting, in-place complex FFT. Like many FFT algorithms, Temperton's pre-computes and stores the complex exponential factors, $e^{-2s\pi ijk/N}$, since those evaluations are so expensive. To that end, we define the arrays $\{w_n^{\pm}\}_{n=0}^{N-1}$ whose elements are $e^{-2s\pi in/N}$ for $s = \pm 1$. Thus, we perform the forward transform when the w's are computed with $s = 1$, and the backward transform when they are computed with $s = -1$. The pre-computation of the trigonometric factors is done by Algorithm 7 (InitializeFFT).

Algorithm 7: *InitializeFFT:* Initialization Routine for FFT

```
Procedure InitializeFFT
Input: $N, s$
$w \leftarrow e^{-2s\pi i/N}$
for $j = 0$ to $N - 1$ do
    $w_j \leftarrow w^j$
end
return $\{w_j\}_{j=0}^{N-1}$
End Procedure InitializeFFT
```

Algorithm 8: *Radix2FFT:* Temperton's Radix 2 Self Sorting Complex FFT

```
Procedure Radix2FFT
Input: $\{f_j\}_{j=0}^{N-1}$, $\{w_j\}_{j=0}^{N-1}$
Integers: $noPtsAtLevel, a, b, c, d, p, N\,\mathrm{div}\,2, m, l, k$
$N\,\mathrm{div}\,2 \leftarrow N/2$
$m \leftarrow \mathrm{Log}_2(N)$
for $l = 1$ to $(m+1)/2$ do
    $noPtsAtLevel \leftarrow 2^{l-1}$
    $a \leftarrow 0$
    $b \leftarrow N\,\mathrm{div}\,2/noPtsAtLevel$
    $p \leftarrow 0$
    for $k = 0$ to $b-1$ do
        $W \leftarrow w_p$
        for $i = k$ to $N-1$ step $N/noPtsAtLevel$ do
            $z \leftarrow W * (f_{a+i} - f_{b+i})$
            $f_{a+i} \leftarrow f_{a+i} + f_{b+i}$
            $f_{b+i} \leftarrow z$
        end
        $p \leftarrow p + noPtsAtLevel$
    end
end
for $l = (m+3)/2$ to $m$ do
    $noPtsAtLevel \leftarrow 2^{l-1}$
    $a \leftarrow 0$
    $b \leftarrow N\,\mathrm{div}\,2/noPtsAtLevel$
    $c \leftarrow noPtsAtLevel$
    $d \leftarrow b + noPtsAtLevel$
    $p \leftarrow 0$
    for $k = 0$ to $b-1$ do
        $W \leftarrow w_p$
        for $j = k$ to $noPtsAtLevel - 1$ step $N/noPtsAtLevel$ do
            for $i = j$ to $N-1$ step $2 * noPtsAtLevel$ do
                $z \leftarrow W * (f_{a+i} - f_{b+i})$
                $f_{a+i} \leftarrow f_{a+i} + f_{b+i}$
                $f_{b+i} \leftarrow f_{c+i} + f_{d+i}$
                $f_{d+i} \leftarrow w * (f_{c+i} - f_{d+i})$
                $f_{c+i} \leftarrow z$
            end
        end
        $p \leftarrow p + noPtsAtLevel$
    end
end
return $\{f_k\}_{j=0}^{N-1}$
End Procedure Radix2FFT
```

We show the FFT algorithm itself in Algorithm 8 (Radix2FFT). The FFT takes a sequence of complex numbers $\{f_j\}_{j=0}^{N-1}$ and the pre-computed complex trigonometric factors $\{w_j\}_{j=0}^{N-1}$ and returns the sum (2.2) by overwriting the original complex sequence. Whether it computes the forward or backward transform depends on the sequence of w's that it is supplied. For details on the algorithm, see [23]. We show examples that use the complex FFT in the sections that follow, where we study special cases of the transforms.

2.1.2 Fourier Transforms of Real Sequences

The complex FFT is about twice as expensive as is necessary when the data to be transformed are real. One of two methods is typically used to compute transforms of real sequences efficiently from the complex transform. The first method solves for the coefficients of two real sequences simultaneously. This is useful if two or more real FFTs have to be computed at once, as happens in multidimensional problems. The second is to use an even-odd decomposition where the transform is computed on a complex sequence of half the length of the original. In that algorithm, half of the values of the real sequence are placed in the real part of the complex sequence and the other half in the imaginary part.

2.1.2.1 Simultaneous Fourier Transformation of Two Real Sequences

We can use the complex FFT to compute the DFT of two real sequences simultaneously. Suppose that $\{x_j\}_{j=0}^{N-1}$ and $\{y_j\}_{j=0}^{N-1}$ are real sequences whose discrete transform coefficients we need. Then

$$\begin{aligned} X_k &= \frac{1}{N}\sum_{j=0}^{N-1} x_j e^{-2\pi ijk/N}, \\ Y_k &= \frac{1}{N}\sum_{j=0}^{N-1} y_j e^{-2\pi ijk/N}, \end{aligned} \qquad k = 0, 1, \ldots, N-1. \tag{2.4}$$

If we combine these into a single complex vector, $z = x + iy$, the discrete Fourier coefficients of z are

$$Z_k = \frac{1}{N}\sum_{j=0}^{N-1} z_j e^{-2\pi ijk/N} = \frac{1}{N}\sum_{j=0}^{N-1} x_j e^{-2\pi ijk/N} + \frac{i}{N}\sum_{j=0}^{N-1} y_j e^{-2\pi ijk/N} \tag{2.5}$$

or

$$Z_k = X_k + iY_k. \tag{2.6}$$

Since x_j and y_j are real,

$$Z^*_{-k} = \frac{1}{N}\sum_{j=0}^{N-1} x_j e^{-2\pi ijk/N} - \frac{i}{N}\sum_{j=0}^{N-1} y_j e^{-2\pi ijk/N} \tag{2.7}$$

so

$$Z^*_{-k} = X_k - iY_k. \tag{2.8}$$

When we solve for X_k and Y_k from (2.6) and (2.8)

$$\begin{aligned} X_k &= \frac{1}{2}\left(Z_k + Z^*_{-k}\right), \\ Y_k &= \frac{-i}{2}\left(Z_k - Z^*_{-k}\right), \end{aligned} \quad k = 0, 1, \ldots, N-1. \tag{2.9}$$

The discrete coefficients are N-periodic, so $Z_{-k} = Z_{N-k}$ and $Z_0 = Z_N$. Therefore,

$$\begin{aligned} X_k &= \frac{1}{2}\left(Z_k + Z^*_{N-k}\right), \\ Y_k &= \frac{-i}{2}\left(Z_k - Z^*_{N-k}\right), \end{aligned} \quad k = 0, 1, \ldots, N-1. \tag{2.10}$$

We show how to compute the forward DFT of two real sequences in Algorithm 9 (FFFTOfTwoRealVectors). It takes as input the trigonometric factors pre-computed with $s = FORWARD$, the two real sequences, and the length of the sequences. It returns the scaled discrete Fourier coefficients of the two sequences. Since the complex FFT does not scale the forward transform by the $1/N$ factor, the procedure does the scaling when it extracts the coefficients. It is possible to modify the weights w^+ computed by Algorithm 7 (InitializeFFT) to include the factor of $1/N$ to save the cost of the divisions if multiple transforms are needed. We have added the scaling

Algorithm 9: *FFFTOfTwoRealVectors:* Simultaneous Computation of the DFT of Two Real Sequences. The Forward Transform

```
Procedure FFFTOfTwoRealVectors
Input: $\{x_j\}_{j=0}^{N-1}, \{y_j\}_{j=0}^{N-1}, \{w_j^+\}_{j=0}^{N-1}$
Uses Algorithms:
   Algorithm 8 (Radix2FFT)
for $j = 0$ to $N-1$ do
    $Z_j \leftarrow x_j + iy_j$
end
$\{Z_k\}_{k=0}^{N-1} \leftarrow Radix2FFT\left(\{Z_j\}_{j=0}^{N-1}, \{w_j^+\}_{j=0}^{N-1}\right)$
$X_0 \leftarrow \mathrm{Re}(z_0)/N$
$Y_0 \leftarrow \mathrm{Im}(z_0)/N$
for $k = 1$ to $N-1$ do
    $X_k \leftarrow \frac{1}{2}(Z_k + Z^*_{N-k})/N$
    $Y_k \leftarrow \frac{-i}{2}(Z_k - Z^*_{N-k})/N$
end
return $\{X_k\}_{k=0}^{N-1}, \{Y_k\}_{k=0}^{N-1}$
End Procedure FFFTOfTwoRealVectors
```

Algorithm 10: *BFFTForTwoRealVectors:* Simultaneous Computation of the DFT of Two Real Sequences. The Backward Transform

```
Procedure BFFTForTwoRealVectors
Input: $\{X_j\}_{j=0}^{N-1}, \{Y_j\}_{j=0}^{N-1}, \{w_k^-\}_{k=0}^{N-1}$
Uses Algorithms:
   Algorithm 8 (Radix2FFT)
for $j = 0$ to $N - 1$ do
    $Z_j \leftarrow X_j + iY_j$
end
$\{Z_k\}_{k=0}^{N-1} \leftarrow Radix2FFT(\{Z_j\}_{j=0}^{N-1}, \{w_k^-\}_{k=0}^{N-1})$
for $k = 0$ to $N - 1$ do
    $x_k \leftarrow \mathrm{Re}(Z_k)$
    $y_k \leftarrow \mathrm{Im}(Z_k)$
end
return $\{x_k\}_{k=0}^{N-1}, \{y_k\}_{k=0}^{N-1}$
End Procedure BFFTForTwoRealVectors
```

here, since a modification of the trigonometric factors routine might not be possible if a library FFT is used.

To reverse the operation, we use (2.6) and the inverse FFT. The desired solutions are simply the real and imaginary parts of the complex sequence returned by the FFT. We present the procedure to compute the inverse transform in Algorithm 10 (BFFTForTwoRealVectors).

2.1.2.2 Fourier Transformation of a Real Sequence by Even-Odd Decomposition

We can also evaluate the DFT of a single sequence of real values efficiently with a complex FFT. The FFT will operate on a new sequence of half the original length that we create by putting half the original data into the real part and the other half into the complex part. Let the even and odd elements of a sequence $\{f_j\}_{j=0}^{N-1}$ be

$$\begin{aligned} e_j &= f_{2j}, \\ o_j &= f_{2j+1}, \end{aligned} \qquad j = 0, 1, \ldots, M-1, \tag{2.11}$$

where $M = N/2$. The forward Fourier transforms of these two sequences are

$$\begin{aligned} E_k &= \frac{1}{M}\sum_{j=0}^{M-1} e_j e^{-2\pi ijk/M}, \\ O_k &= \frac{1}{M}\sum_{j=0}^{M-1} o_j e^{-2\pi ijk/M}, \end{aligned} \qquad j = 0, 1, \ldots, M-1. \tag{2.12}$$

But

$$
\begin{aligned}
F_k &= \frac{1}{N}\sum_{j=0}^{N-1} f_j e^{-2\pi ijk/N} \\
&= \frac{1}{N}\sum_{j=0}^{2M-1} f_j e^{-2\pi ijk/2M} \\
&= \frac{1}{N}\sum_{j=0}^{M-1} f_{2j} e^{-2\pi ijk/M} + \frac{1}{N}\sum_{j=0}^{M-1} f_{2j+1} e^{-2\pi i(2j+1)k/2M} \\
&= \frac{1}{2M}\sum_{j=0}^{M-1} f_{2j} e^{-2\pi ijk/M} + \frac{e^{-2\pi ik/N}}{2M}\sum_{j=0}^{M-1} f_{2j+1} e^{-2\pi ijk/M} \\
&= \frac{1}{2}\left(E_k + e^{-2\pi ik/N} O_k\right), \quad k = 0, 1, \ldots, N/2-1. \qquad (2.13)
\end{aligned}
$$

Thus, the complex DFT of the even-odd decomposition of the real sequence produces the first half of the discrete coefficients of the original sequence. We find the second half of the sequence of coefficients by recalling that if f_k is real and N-periodic, $F_{N-k} = F_k^*$.

We could compute coefficients E_k and O_k simultaneously using Algorithm 9 (FFFTOfTwoRealVectors), but it is probably best to combine the two algorithms. To combine them, we let $z_j = e_j + io_j$ and use (2.9) to define

$$
\begin{aligned}
F_k = \frac{1}{2N}&\left\{\left(Z_k + Z^*_{N/2-k}\right) - ie^{-2\pi ijk/N}\left(Z_k - Z^*_{N/2-k}\right)\right\}, \\
&k = 0, 1, \ldots, N/2-1. \qquad (2.14)
\end{aligned}
$$

We find the second half of the F_k array from the symmetry relation, the first value from

$$F_0 = \left(\mathrm{Re}(Z_0) + \mathrm{Im}(Z_0)\right)/N \qquad (2.15)$$

and the center from

$$F_{N/2} = \left(\mathrm{Re}(Z_0) - \mathrm{Im}(Z_0)\right)/N. \qquad (2.16)$$

Algorithm 11 (FFFTEO) shows one way to implement the Even-Odd decomposition. It uses the fact that

$$e^{\pm 2\pi ij/M} = e^{\pm 2\pi i(2j)/N} \qquad (2.17)$$

so that the factors w^+ required by the complex FFT of length M are simply the even indexed components of the factors computed for the full length transform, factors that are also required by (2.13). This saves us the re-computation of the trigonometric factors.

Algorithm 11: *FFFTEO:* The Forward DFT by Even-Odd Decomposition

```
Procedure FFFTEO
Input: $\{f_j\}_{j=0}^{N-1}, \{w_j^+\}_{j=0}^{N-1}$
Uses Algorithms:
   Algorithm 8 (Radix2FFT)
for $j = 0$ to $N/2-1$ do
    $Z_j \leftarrow f_{2j} + if_{2j+1}$
    $w_j \leftarrow w_{2j}^+$
end
$\{Z_k\}_{k=0}^{N/2-1} \leftarrow Radix2FFT\left(\{Z_j\}_{j=0}^{N/2-1}, \{w_j\}_{j=0}^{N/2-1}\right)$
$F_0 \leftarrow (\mathrm{Re}(Z_0) + \mathrm{Im}(Z_0))/N$
$F_{N/2} \leftarrow (\mathrm{Re}(Z_0) - \mathrm{Im}(Z_0))/N$
for $k = 1$ to $N/2-1$ do
    $F_k \leftarrow \frac{1}{2N}\left\{\left(Z_k + Z_{N/2-k}^*\right) - iw_k^+\left(Z_k - Z_{N/2-k}^*\right)\right\}$
end
for $k = 1$ to $N/2-1$ do
    $F_{N-k} \leftarrow F_k^*$
end
return $\{F_k\}_{k=0}^{N-1}$
End Procedure FFFTEO
```

We can also evaluate the backward transform using the even-odd decomposition. Since $F_{k+N/2} = F_{N/2-k}^*$, (2.13) implies that

$$F_{N/2-k}^* = \frac{1}{2N}\left[\left(Z_k + Z_{M-k}^*\right) + ie^{-2\pi ik/N}\left(Z_k - Z_{N/2-k}^*\right)\right]. \tag{2.18}$$

If we add and subtract this and (2.13) we get

$$E_k = \left(F_k + F_{N/2-k}^*\right), \quad k = 0, 1, \ldots, N/2-1 \tag{2.19}$$

and

$$O_k = e^{2\pi ik/N}\left(F_k - F_{N/2-k}^*\right), \quad k = 0, 1, \ldots, N/2-1. \tag{2.20}$$

When we combine these into a single complex sequence

$$Z_k = E_k + iO_k, \quad k = 0, 1, \ldots, N/2-1, \tag{2.21}$$

the backward FFT produces

$$z_j = e_j + io_j, \quad k = 0, 1, \ldots, N/2-1. \tag{2.22}$$

We extract the full sequence from the real and the imaginary parts,

$$\left.\begin{aligned} f_{2j} &= \mathrm{Re}(z_j) \\ f_{2j+1} &= \mathrm{Im}(z_j) \end{aligned}\right\}, \quad j = 0, 1, \ldots, N/2-1. \tag{2.23}$$

Algorithm 12: *BFFTEO:* The Backward DFT by Even-Odd Decomposition

```
Procedure BFFTEO
Input: $\{F_k\}_{k=0}^{N-1}$, $\{w_k^-\}_{k=0}^{N-1}$
Uses Algorithms:
  Algorithm 8 (Radix2FFT)
$M \leftarrow N/2$
for $k = 0$ to $M-1$ do
    $E \leftarrow \left(F_k + F_{M-k}^*\right)$
    $O \leftarrow w_k^- \left(F_k - F_{M-k}^*\right)$
    $Z_k \leftarrow E_k + i\, O_k$
    $w_k \leftarrow w_{2k}^-$
end
$\{Z_j\}_{j=0}^{M-1} \leftarrow Radix2FFT(\{Z_k\}_{k=0}^{M-1}, \{w_k\}_{k=0}^{M-1})$
for $j \leftarrow 0$ to $M-1$ do
    $f_{2j} \leftarrow \mathrm{Re}(Z_j)$
    $f_{2j+1} \leftarrow \mathrm{Im}(Z_j)$
end
return $\{f_j\}_{j=0}^{N-1}$
End Procedure BFFTEO
```

We present the procedure for using the even-odd decomposition to compute the backward transform in Algorithm 12 (BFFTEO). As before, we use the periodicity of the trigonometric factors to avoid their re-computation.

2.1.3 The Fourier Transform in Two Space Variables

Multidimensional transforms take advantage of the tensor product basis, $\phi_{n,m}(x,y) = \phi_n(x)\phi_m(y) = e^{2\pi inx}e^{2\pi imy}$. In two space variables, the DFT of a two dimensional sequence $\{f_{j,k}\}_{j,k=0}^{N-1,M-1}$ is

$$F_{nm} = \frac{1}{NM}\sum_{k=0}^{M-1}\sum_{j=0}^{N-1} f_{j,k}e^{-2\pi ijn/N}e^{-2\pi ikm/M},$$

$$f_{j,k} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F_{nm}e^{2\pi ijn/N}e^{2\pi ikm/M}. \tag{2.24}$$

In this section, since it is apropos to most spectral approximations of PDEs, we assume that f is real. Furthermore, we assume that N and M are even.

We can factor the forward transform into

$$F_{nm} = \frac{1}{NM}\sum_{k=0}^{M-1}\left\{\sum_{j=0}^{N-1} f_{j,k}e^{-2\pi ijn/N}\right\}e^{-2\pi ikm/M}. \tag{2.25}$$

So let us define an intermediate array

$$\bar{F}_{n,k} = \frac{1}{N} \sum_{j=0}^{N-1} f_{j,k} e^{-2\pi i j n/N}, \quad \begin{array}{l} n = 0, 1, \ldots, N-1, \\ k = 0, 1, \ldots, M-1, \end{array} \tag{2.26}$$

so that

$$F_{nm} = \frac{1}{M} \sum_{k=0}^{M-1} \bar{F}_{n,k} e^{-2\pi i k m/M}, \quad \begin{array}{l} j = 0, 1, \ldots, N-1, \\ m = 0, 1, \ldots, M-1. \end{array} \tag{2.27}$$

Thus, the tensor product basis reduces the two dimensional transform to a sequence of one dimensional transforms.

As before, direct application of a complex FFT to compute the two dimensional transform of the real sequence would amount to doing twice as much work and use twice as much storage as is necessary. Fortunately, we can combine the ideas of the previous two sections to develop a more efficient algorithm than that.

Since we assume the initial sequence is real, only half the number of FFTs is required in the first coordinate direction. To that end, let us define the complex sequence

$$z_{j,l} = f_{j,2l} + i f_{j,2l+1}, \quad \begin{array}{l} j = 0, 1, \ldots, N-1, \\ l = 0, 1, \ldots, M/2-1, \end{array} \tag{2.28}$$

whose Fourier transform in the first index is

$$\bar{Z}_{n,l} = \frac{1}{N} \sum_{j=0}^{N-1} \left(f_{j,2l} + i f_{j,2l+1} \right) e^{-2\pi i j n/N}, \quad \begin{array}{l} n = 0, 1, \ldots, N-1, \\ l = 0, 1, \ldots, M/2-1. \end{array} \tag{2.29}$$

Note that the intermediate array $\bar{Z}_{n\,l}$ is half the size of the $\bar{F}$ array. Now, we know the intermediate transform of the original data, namely,

$$\begin{aligned} \bar{F}_{n,2l} &= \frac{1}{2}\left(\bar{Z}_{n,l} + \bar{Z}^*_{n,M-l}\right), \\ \bar{F}_{n,2l+1} &= \frac{-i}{2}\left(\bar{Z}_{n,l} - \bar{Z}^*_{n,M-l}\right), \end{aligned} \quad \begin{array}{l} n = 0, 1, \ldots, N-1, \\ l = 0, 1, \ldots, M/2-1, \end{array} \tag{2.30}$$

but there is no need to compute it at this stage. Instead, we go back to (2.27) and split it into even and odd components as in the previous subsection

$$F_{nm} = \frac{1}{M} \sum_{l=0}^{M/2-1} \left\{ \bar{F}_{n,2l} e^{-2\pi i (2l) m/M} + e^{-2\pi i m/M} \bar{F}_{n,2l+1} e^{-2\pi i (2l) m/M} \right\}. \tag{2.31}$$

Then the transforms to be computed on the second index are the half-length transforms

$$F_{nm} = \frac{1}{M} \sum_{l=0}^{M/2-1} X_{n,l} e^{-2\pi i l m/(M/2)}, \quad \begin{array}{l} n = 0, 1, \ldots, N-1, \\ m = 0, 1, \ldots, M/2-1, \end{array} \tag{2.32}$$

Algorithm 13: *Forward2DFFT:* A Two-Dimensional Forward FFT of a Real Array with an Even Number of Points in Each Direction

```
Procedure Forward2DFFT
Input: $\{f_{j,k}\}_{j,k=0}^{N-1,M-1}$, $\{w_j^{+,1}\}_{j=0}^{N-1}$, $\{w_j^{+,2}\}_{j=0}^{M-1}$
Uses Algorithms:
   Algorithm 9 (FFFTOfTwoRealVectors)
   Algorithm 8 (Radix2FFT)
for $k = 0$ to $M-2$ step 2 do
    $\left\{\{\bar F_{n,k}\}_{n=0}^{N-1}, \{\bar F_{n,k+1}\}_{n=0}^{N-1}\right\} \leftarrow$
    $FFFTOfTwoRealVectors\left(\{f_{j,k}\}_{j=0}^{N-1}, \{f_{j,k+1}\}_{j=0}^{N-1}, \{w_j^{+,1}\}_{j=0}^{N-1}\right)$
end
for $n = 0$ to $N-1$ do
    $\{F_{n,m}\}_{m=0}^{M-1} \leftarrow Radix2FFT\left(\{\bar F_{n,k}\}_{k=0}^{M-1}, \{w_j^{+,2}\}_{j=0}^{M-1}\right)$
end
for $m = 0$ to $M-1$ do
    for $n = 0$ to $N-1$ do
        $F_{n,m} \leftarrow F_{n,m}/M$
    end
end
return $\{F_{n,m}\}_{n,m=0}^{N-1,M-1}$
End Procedure Forward2DFFT
```

where the values to be transformed are

$$X_{n,l} = \frac{1}{2}\left(\bar Z_{n,l} + \bar Z^*_{n,N-l}\right) - \frac{ie^{-2\pi i m/M}}{2}\left(\bar Z_{n,l} - \bar Z^*_{n,N-l}\right). \tag{2.33}$$

We compute the second half of the array from the symmetry $F_{N-n\,M-m} = F^*_{n,m}$. Overall we see that approximately one half the work of the direct use of the complex FFT is done. Under the assumption that the initial data are real and that N and M are even, we use Algorithms 9 (FFFTOfTwoRealVectors) and 10 (BFFTForTwoRealVectors) to compute the two dimensional transforms efficiently. We show the procedures in Algorithm 13 (Forward2DFFT) for the forward transform and Algorithm 14 (Backward2DFFT) for the backward transform.

2.2 The Real Fourier Transform

Though we do not directly need it in typical Fourier spectral computations, we derive the real transform for the purposes of the next chapter where we study Chebyshev transforms. We can compute the real transform from the complex transform,

Algorithm 14: *Backward2DFFT:* Two-Dimensional Backward FFT of a Real Array with an Even Number of Points in Each Direction

```
Procedure Backward2DFFT
Input: $\{F_{n,m}\}_{n,m=0}^{N-1,M-1}$, $\{w_j^{-,1}\}_{j=0}^{N-1}$, $\{w_j^{-,2}\}_{j=0}^{M-1}$
Uses Algorithms:
   Algorithm 8 (Radix2FFT)
   Algorithm 10 (BFFTForTwoRealVectors)
for $n = 0$ to $N-1$ do
    $\{F_{n,k}\}_{k=0}^{M-1} \leftarrow Radix2FFT(\{\bar{F}_{n,m}\}_{m=0}^{M-1}, \{w_j^{-,2}\}_{j=0}^{M-1})$
end
for $k = 0$ to $M-2$ step 2 do
    $\left\{\{f_{j,k}\}_{j=0}^{N-1}, \{f_{j,k+1}\}_{j=0}^{N-1}\right\} \leftarrow$
    $BFFTForTwoRealVectors(\{F_{n,k}\}_{n=0}^{N-1}, \{F_{n,k+1}\}_{n=0}^{N-1}, \{w_j^{-,1}\}_{j=0}^{N-1})$
end
return $\{f_{j,k}\}_{j,k=0}^{N-1,M-1}$
End Procedure Backward2DFFT
```

too. If N is even, the real transform takes the form

$$x_j = \frac{a_0}{2} + \sum_{k=1}^{N/2-1} \left\{ a_k \cos\left(\frac{2\pi jk}{N}\right) + b_k \sin\left(\frac{2\pi jk}{N}\right) \right\} + \frac{(-1)^j a_{N/2}}{2}. \tag{2.34}$$

In terms of the complex transform,

$$x_j = \sum_{k=0}^{N-1} X_k e^{2\pi ijk/N} = \sum_{k=0}^{N/2-1} X_k e^{2\pi ijk/N} + \sum_{k=1}^{N/2} X_{N-k} e^{2\pi ij(N-k)/N}. \tag{2.35}$$

Since $X_k = X_{N-k}^*$,

$$x_j = X_0 + 2 \sum_{k=0}^{N/2-1} \mathrm{Re}\left\{X_k e^{2\pi ijk/N}\right\} + (-1)^j X_{N/2}. \tag{2.36}$$

Using the fact that

$$\mathrm{Re}\left\{X_k e^{2\pi ijk/N}\right\} = \mathrm{Re}(X_k)\cos\left(\frac{2\pi jk}{N}\right) - \mathrm{Im}(X_k)\sin\left(\frac{2\pi jk}{N}\right), \tag{2.37}$$

$$x_j = X_0 + 2 \sum_{k=0}^{N/2-1} \left\{ \mathrm{Re}(X_k)\cos\left(\frac{2\pi jk}{N}\right) - \mathrm{Im}(X_k)\sin\left(\frac{2\pi jk}{N}\right) \right\} + (-1)^j X_{N/2}. \tag{2.38}$$

Algorithm 15: *ForwardRealFFT:* The Forward Real Transform

```
Procedure ForwardRealFFT
Input: $\{x_j\}_{j=0}^{N-1}$, $\{w_j^+\}_{j=0}^{N-1}$
Uses Algorithms:
   Algorithm 11 (FFFTEO)
$\{X_k\}_{k=0}^{N-1} \leftarrow FFFTEO(\{x_j\}_{j=0}^{N-1}, \{w_j^+\}_{j=0}^{N-1})$
for $k = 0$ to $N/2$ do
    $a_k \leftarrow 2\,\mathrm{Re}(X_k)$
    $b_k \leftarrow -2\,\mathrm{Im}(X_k)$
end
$b_0 \leftarrow 0$
$b_{N/2} \leftarrow 0$
return $\{a_k\}_{k=0}^{N/2}, \{b_k\}_{k=0}^{N/2}$
End Procedure ForwardRealFFT
```

Algorithm 16: *BackwardRealFFT*: The Backward Real Transform

```
Procedure BackwardRealFFT
Input: $\{a_k\}_{k=0}^{N/2}, \{b_k\}_{k=0}^{N/2}$
Uses Algorithms:
   Algorithm 12 (BFFTEO)
   Algorithm 7 (InitializeFFT)
$\{w_j^+\}_{j=0}^{N/2-1} \leftarrow InitializeFFT(N/2, FORWARD)$
$X_0 \leftarrow a_0/2$
$X_{N/2} \leftarrow a_{N/2}/2$
for $k = 1$ to $N/2 - 1$ do
    $X_k \leftarrow (a_k + i b_k)/2$
end
for $k = 1$ to $N/2 - 1$ do
    $X_{N-k} \leftarrow X_k^*$
end
$\{x_j\}_{j=0}^{N-1} \leftarrow BFFTEO(\{X_k\}_{k=0}^{N-1}, \{w_k^+\}_{k=0}^{N-1})$
return $\{x_j\}_{j=0}^{N-1}$
End Procedure BackwardRealFFT
```

When we match terms with (2.34) we get the relation between the coefficients of the real and complex transforms

$$\begin{aligned} a_k &= 2\,\mathrm{Re}(X_k), \\ b_k &= -2\,\mathrm{Im}(X_k), \end{aligned} \qquad k = 0, 1, \ldots, N/2 - 1. \tag{2.39}$$

Thus, we can compute the forward and backward real transforms from the complex transform, and we show the procedures in Algorithms 15 (ForwardRealFFT) and 16 (BackwardRealFFT). Note that half of the X_k's returned by the forward FFT are not used in Algorithm 15. We can get additional savings if we explicitly incorporate Algorithm 11 (FFFTEO) minus the final loop that produces that second half of the coefficients (Problem 2.2).

2.3 How to Evaluate the Fourier Interpolation Derivative by FFT

For large N, it is efficient to use the FFT to compute the derivative of Fourier interpolants written in the form (1.73). Exactly what is that value of N is very implementation dependent. It depends on the FFT code, the architecture of the machine and the matrix multiplication code. Crossover points, where the FFT becomes more efficient than the matrix multiplication method that we discuss in the next section, have been reported to vary from eight to 128. Before deciding on which method to incorporate into production codes, it is probably best to program both and test for the particular computer architecture to be used.

To use the FFT to compute the derivative of the interpolant at the nodes, we must put the derivative in the form of the DFT. Let's rewrite the derivative of the interpolant by separating the $N/2$ mode

$$(I_N f)'\left(x_j\right) = \sum_{k=-N/2}^{N/2} \frac{ik\tilde{f}_k}{\bar{c}_k} e^{ikx_j} = \sum_{k=-N/2}^{N/2-1} \frac{ik\tilde{f}_k}{\bar{c}_k} e^{ikx_j} + \frac{i(\frac{N}{2})\tilde{f}_{N/2}}{\bar{c}_{N/2}} e^{i(\frac{N}{2})x_j}. \tag{2.40}$$

Since $\tilde{f}_{N/2} = \tilde{f}_{-N/2}$ and $e^{i(N/2)x_j} = e^{-i(N/2)x_j}$, we can rewrite the derivative as

$$\begin{aligned}(I_N f)'\left(x_j\right) &= \sum_{k=-N/2}^{N/2-1} \frac{ik\tilde{f}_k}{\bar{c}_k} e^{ikx_j} - \frac{i(\frac{-N}{2})\tilde{f}_{-N/2}}{\bar{c}_{-N/2}} e^{i(-\frac{N}{2})x_j} \\ &= \sum_{k=-N/2+1}^{N/2-1} ik\tilde{f}_k e^{ikx_j}. \end{aligned} \tag{2.41}$$

Therefore, we can use the DFT to evaluate the derivative of the Fourier interpolant at the nodes by setting $\tilde{f}_{-N/2} = 0$.

We show how to compute the interpolation derivative using the FFT in Algorithm 17 (FourierDerivativeByFFT). The procedure first computes the discrete Fourier coefficients by Algorithm 11 (FFFTEO), with the assumption that the exponential factors have been pre-computed and stored. It then uses the correspondence we made in (2.3) to compute the discrete Fourier coefficients of the derivative. Finally, it computes the derivative values at the nodes by the backward FFT, Algorithm 12 (BFFTEO). Note that the procedure sets the $-N/2$ coefficient to zero to represent (2.41).

Algorithm 17 easily generalizes to higher order derivatives. To modify Algorithm 17 to compute the approximation of the mth derivative, we merely need to

Algorithm 17: *FourierDerivativeByFFT:* Fast Evaluation of the Fourier Polynomial Derivative

```
Procedure FourierDerivativeByFFT
Input: $\{f_j\}_{j=0}^{N-1}, \{w_j^+\}_{j=0}^{N-1}, \{w_j^-\}_{j=0}^{N-1}$
Uses Algorithms:
    Algorithm 11 (FFFTEO)
    Algorithm 12 (BFFTEO)
$\{F_j\}_{j=0}^{N-1} \leftarrow FFFTEO(\{F_j\}_{j=0}^{N-1}, \{w_j^+\}_{j=0}^{N-1})$
for $k = 0$ to $N/2 - 1$ do
    $F_k \leftarrow ik * F_k$
end
$F_{-N/2} \leftarrow 0$
for $k = N/2 + 1$ to $N - 1$ do
    $F_k \leftarrow i(k - N) * F_k$
end
$\{Df_{\mathrm{j}}\}_{j=0}^{N-1} \leftarrow BFFTEO(\{F_j\}_{j=0}^{N-1}, \{w_j^-\}_{j=0}^{N-1})$
return $\{(Df)_j\}_{j=0}^{N-1}$
End Procedure FourierDerivativeByFFT
```

add m to the input list and replace ik by $(ik)^m$ in the first loop and $i(k - N)$ by $(i(k - N))^m$ in the second. However, we do not set $\tilde{f}_{-N/2} = 0$ for even derivatives. We leave the difference between even and odd derivatives to Problem 2.4. The difference also implies that computing the first derivative twice is not the same as computing the second derivative (Problem 2.5).

2.4 How to Compute Derivatives by Matrix Multiplication

For small enough N, we can compute the derivative of the Fourier interpolant at the interpolation points efficiently by matrix vector multiplication. Differentiation of the interpolant at the nodes is

$$(I_N f)'_n = \sum_{j=0}^{N-1} D_{nj} f_j. \tag{2.42}$$

The matrix D, whose elements are D_{nj}, is called the *Fourier derivative matrix*. Formally, the elements are the values of $h'_j(x_n)$ presented in (1.75). In practice, we pre-compute and store this matrix.

The first issue in the use of matrix multiplication to compute the Fourier derivative approximation is how to construct the derivative matrix itself. The construction of spectral derivative matrices has been the subject of much discussion since it was noticed that derivatives computed with the Chebyshev differentiation matrix (which we discuss in Sect. 3.5) were very sensitive to rounding errors. To reduce the effects of rounding errors, several modifications to the matrices have been proposed,

Algorithm 18: *FourierDerivativeMatrix:* Computation of the Fourier Derivative Matrix Using the Negative Sum Trick

```
Procedure FourierDerivativeMatrix
Input: $N$
for $i = 0$ to $N-1$ do
    $D_{i,i} \leftarrow 0$
    for $j = 0$ to $N-1$ do
        if $j \neq i$ then
            $D_{i,j} \leftarrow \frac{1}{2}(-1)^{i+j}\cot\left[\frac{(i-j)\pi}{N}\right]$
            $D_{i,i} \leftarrow D_{i,i} - D_{i,j}$
        end
    end
end
return $\{D_{i,j}\}_{i,j=0}^{N-1}$
End Procedure FourierDerivativeMatrix
```

including preconditioning, use of the symmetry properties, and the use of what has come to be called the "Negative Sum Trick" to compute the diagonal entries. Comparisons of the numerous approaches have favored the use of the Negative Sum Trick to evaluate the derivative matrix.

The Negative Sum Trick comes from the observation that the derivative of a constant must be zero. This means that $\sum_{j=0}^{N-1} D_{nj} = 0$ for $n = 0, 1, \ldots, N-1$. To enforce that condition we compute the diagonal elements to satisfy it explicitly, i.e. we evaluate the diagonal entries of the matrix to satisfy

$$D_{nn} = -\sum_{\substack{j=0 \\ j\neq n}}^{N-1} D_{nj}. \tag{2.43}$$

The diagonal elements computed with (2.43) will not be exactly equal to zero, but overall the effects of rounding errors are minimized.

Algorithm 18 (FourierDerivativeMatrix) shows how to pre-compute the Fourier derivative matrix using the Negative Sum Trick. For the best roundoff error properties, however, the diagonal entries should be computed separately, after the rest of the matrix has been computed, since the order in which the sum (2.43) is computed is important. The off-diagonal terms should be sorted and summed from smallest in magnitude to largest. For the sake of simplicity, we present Algorithm 18 without the ordered sum.

The next issue is to decide how to implement the matrix multiplication. Since this is also an issue for the polynomial approximations discussed in the next chapter, and is not specifically related to spectral methods *per se*, we leave the detailed discussion of matrix multiplication for later. If efficiency is not of the utmost importance, of course, we would use the standard implementation shown in Algorithm 19 (MxVDerivative), with $s = 0$ and $e = N-1$. That algorithm takes the derivative

Algorithm 19: *MxVDerivative:* A Matrix-Vector Multiplication Procedure

```
Procedure MxVDerivative
Input: $\{D_{i,j}\}_{i,j=s}^{e}$, $\{f_j\}_{j=s}^{e}$
for $i = s$ to $e$ do
    $t = 0$
    for $j = s$ to $e$ do
        $t \leftarrow t + D_{i,j} * f_j$
    end
    $(I_N f)'_i \leftarrow t$
end
return $\{(I_N f)'_i\}_{i=s}^{e}$
End Procedure MxVDerivative
```

matrix, pre-computed by Algorithm 18 (FourierDerivativeMatrix) and the sequence of values at the nodes, and returns derivative of the interpolant, evaluated at the nodes.

Exercises

2.1 Compare the speed of the simple, direct DFT, Algorithm 6 to the FFT, Algorithm 8 for various values of N. Above what value of N is the FFT faster? Are there any advantages that the direct DFT has over the FFT? Make the same comparisons to a library FFT.

2.2 Half of the X_k's returned by the forward FFT are not used in Algorithm 15 (ForwardRealFFT). Modify the procedure to get additional savings by explicitly incorporating Algorithm 11 (FFFTEO) minus the final loop that produces that second half of the coefficients.

2.3 Compute and plot as a function of x the Fourier interpolation derivative of the functions

$$f(x) = \sin(x/2),$$
$$f(x) = e^{\sin(x)}$$

on the interval $[0, 2\pi]$ for several values of N. Observe the behavior or the derivative approximations and explain their differences. Also, plot the maximum error at the nodes as a function of N and compare.

2.4 Show that

$$(I_N f)^{(m)}(x_j) = \begin{cases} \sum_{k=-N/2+1}^{N/2-1} (ik)^m \tilde{f}_k e^{ikx_j}, & m \text{ odd}, \\ \sum_{k=-N/2}^{N/2-1} (ik)^m \tilde{f}_k e^{ikx_j}, & m \text{ even}. \end{cases}$$

Use the result to modify Algorithm 17 (FourierDerivativeByFFT) to compute mth order derivatives.

2.5 Use the result of Problem 2.4 to show that taking the first derivative twice is not the same as taking the second derivative. By how much do they differ? How significant is that difference for smooth functions?

2.6 For what values of N is it faster to compute the Fourier interpolation derivative of a function by FFT than by matrix multiplication?