13 The Fast Fourier Transform

13.1 Interpolation on the unit circle

So far in this course, we have considered interpolation of functions at nodes on the real line. When modeling periodic phenomena, it is natural to instead allocate the interpolation points on the unit circle. Figure 1 shows 20 equidistant nodes on the unit circle (in blue) and the image of the unit circle under a real-valued function $f$ (red curve). The dots on the red curve mark the function values to be interpolated.

![Figure 1: A function defined on the unit circle, 20 equidistant interpolation points, and associated function values.](image)

The equidistant points on the unit circle in Figure 1 can be produced in MATLAB/Octave with the code:

```matlab
m=20;
t=[0:m-1]';
plot(cos(2*pi*t/m),sin(2*t*pi/m),'o')
```

This code implies that the unit circle lives in $\mathbb{R}^2$. However, polynomial interpolation is easier in the complex plane, which is introduced in the following section.
13.2 The complex plane

Usually we can add and subtract vectors, but not multiply vectors or divide by a vector. The complex plane can be thought of as the real plane $\mathbb{R}^2$ equipped with rules for multiplying and dividing vectors. A vector with coordinates $x$ and $y$ in the complex plane is written as

$$z = x + iy,$$  \hspace{1cm} x, y \in \mathbb{R}, \tag{1}$$

where the imaginary unit $i$ is a place holder; it multiplies the 2nd coordinate. The vector $z$ is referred to as a complex number. Thus, the complex number (1) is analogous to the point $(x, y)$ in $\mathbb{R}^2$. We add and subtract complex numbers similarly as vectors in $\mathbb{R}^2$. Introduce the complex numbers $z_1 = x_1 + iy_1$, $z_2 = x_2 + iy_2$, $x_1, x_2, y_1, y_2 \in \mathbb{R}$.

Then

$$z_1 + z_2 = x_1 + x_2 + i(y_1 + y_2),$$

$$z_1 - z_2 = x_1 - x_2 + i(y_1 - y_2).$$

The magnitude of a complex number $z$, denoted by $|z|$, is defined to be the norm of the corresponding vector in $\mathbb{R}^2$, i.e., let $z = x + iy$, $x, y \in \mathbb{R}$. Then

$$|z| = \sqrt{x^2 + y^2}.$$  

The complex number $\bar{z} = x - iy$ is said to be the complex conjugate of $z$. Clearly, $|\bar{z}| = |z|$. Moreover,

$$\bar{z}z = |z|^2. \tag{2}$$

Complex numbers differ from vectors in $\mathbb{R}^2$ in that we also can multiply and divide them. Multiplication is defined by

$$z_1z_2 = x_1x_2 - y_1y_2 + i(x_1y_2 + x_2y_1). \tag{3}$$

This rule can be remembered by thinking of the imaginary unit $i$ as a number with the property $i^2 = -1$. Then

$$z_1z_2 = (x_1 + iy_1)(x_2 + iy_2) = x_1(x_2 + iy_2) + iy_1(x_2 + iy_2)$$

$$= x_1x_2 + ix_1y_2 + iy_1x_2 + i^2y_1y_2 = x_1x_2 - y_1y_2 + i(x_1y_2 + x_2y_1),$$

which is equivalent to (3). Thus, when multiplying complex numbers, all we need to remember is that $i$ is a place holder and that $i^2 = -1$ is a real number.

Multiplication of complex numbers on the unit circle is particularly simple. These numbers are of the form

$$z = \cos(\theta) + i\sin(\theta), \hspace{1cm} \theta \in \mathbb{R},$$

similarly as $(\cos(\theta), \sin(\theta))$ is a point on the unit circle in $\mathbb{R}^2$. Let

$$z_1 := \cos(\theta_1) + i\sin(\theta_1), \hspace{0.5cm} z_2 := \cos(\theta_2) + i\sin(\theta_2), \hspace{0.5cm} \theta_1, \theta_2 \in \mathbb{R}. \tag{4}$$

Then it follows from (3) that

$$z_1z_2 = \cos(\theta_1)\cos(\theta_2) - \sin(\theta_1)\sin(\theta_2) + i(\cos(\theta_1)\sin(\theta_2) + \cos(\theta_2)\sin(\theta_1)).$$
which simplifies to

\[ z_1 z_2 = \cos(\theta_1 + \theta_2) + i \sin(\theta_1 + \theta_2). \]

The following MATLAB/Octave code generates 20 equidistant points on the unit circle in the complex plane:

```matlab
m=20;
t=[0:m-1]';
z=cos(2*pi*t/m)+i*sin(2*t*pi/m);
```

Note that MATLAB/Octave know that \( i \) is the imaginary unit (unless you have defined \( i \) to be something else).

It is convenient to define

\[ e^{i\theta} := \cos(\theta) + i \sin(\theta). \]

Then the numbers (4) can be written as

\[ z_1 = e^{i\theta_1}, \quad z_2 = e^{i\theta_2}, \]

and we can evaluate the product \( z_1 z_2 \) by application of the usual rules for multiplication of exponentials:

\[ z_1 z_2 = e^{i\theta_1} e^{i\theta_2} = e^{i(\theta_1 + \theta_2)}. \]

A graphical illustration is provided by Figure 2.

![Figure 2: Multiplication of complex numbers \( z_1 \) and \( z_2 \) on the unit circle.](image)

Division of complex numbers on the unit circle is equally easy. We have

\[ z_1/z_2 = e^{i\theta_1}/e^{i\theta_2} = e^{i\theta_1} e^{-i\theta_2} = e^{i(\theta_1 - \theta_2)} = \cos(\theta_1 - \theta_2) + i \sin(\theta_1 - \theta_2). \]
We will not need to compute quotients of general complex numbers and therefore omit the discussion of this topic.

13.3 Polynomial interpolation in the complex plane

Let \( \{z_1, z_2, \ldots, z_m\} \) be distinct complex nodes and let \( \{y_1, y_2, \ldots, y_m\} \) be real or complex numbers. We consider the polynomial interpolation problem: determine a polynomial \( p \) of degree at most \( m - 1 \), such that

\[
p(z_j) = y_j, \quad 1 \leq j \leq m.
\]

Express the interpolation polynomial in power form

\[
p(z) = a_1 + a_2 z + a_3 z^2 + \cdots + a_{m-1} z^{m-2} + a_m z^{m-1},
\]

where the coefficients \( a_j \) are real or complex numbers. Then the interpolation conditions (5) give the linear system of equations

This linear system of equations is analogous to the linear system of equations (3) of Lecture 11. Similarly as in Lecture 11, the Vandermonde matrix in (7) is nonsingular when the nodes \( z_j \) are distinct, and then the interpolation problem (5) has a unique solution. Here we are particularly interested in the case when the nodes \( z_j \) are equidistant on the unit circle, i.e., when

\[
z_j = e^{2\pi i (j - 1)/m}, \quad 1 \leq j \leq m.
\]

Then the Vandermonde matrix is quite special.

Exercise 13.1

Investigate the properties of Vandermonde matrices with equidistant nodes on the unit circle using MATLAB/Octave. What are their condition number? What is \( A' \)? (The operation ‘ is different for complex matrices than for real ones.) How do \( A \) and \( A' \) relate? How do \( A^{-1} \) and \( A' \) relate? \( \square \)

Exercise 13.2

Define an inner product such that a scaled version of the Vandermonde matrix with equidistant nodes on the unit circle has orthonormal columns. Thus, the scaled Vandermonde matrix is orthonormal with respect to this inner product. Complex orthonormal matrices are commonly referred to as \textit{unitary matrices}. \( \square \)

Exercise 13.3

Estimate the number of (complex) arithmetic floating-point operations required to solve the interpolation problem (7) without using the structure of the matrix? \( \square \)
Exercise 13.4
Estimate the number of (complex) arithmetic floating point operations required to solve the interpolation problem (7) when using the structure of the matrix uncovered in Exercises 13.1 and 13.2. □

13.4 Fourier analysis

The vector $e^{ik\theta}$ traverses the unit circle $k$ times in the counter-clockwise direction when $k > 0$ and in the clockwise direction when $k < 0$ as $\theta$ increases from zero to $2\pi$. Therefore the functions $\theta \to e^{ik\theta}$ represent slow oscillations when $k$ is of small magnitude and rapid oscillations when $k$ is of large magnitude. Substituting $z = e^{i\theta}$ into the polynomial (6) shows that the interpolation polynomial expresses the data \{y_j\}_{j=1}^m as a sum of oscillations,

$$p(e^{i\theta}) = a_1 + a_2 e^{i\theta} + a_3 e^{2i\theta} z^2 + \cdots + a_{m-1} e^{(m-2)i\theta} + a_m e^{(m-1)i\theta}. \quad (9)$$

If, say, $a_2$ is large and the other coefficients are small, then the data \{y_j\}_{j=1}^m stems from a process that is roughly of the form $\theta \to a_2 e^{i\theta}$. The decomposition of a signal (=data) into a linear combination of functions of the form $e^{ik\theta}$ is commonly referred to as Fourier analysis. This decomposition provides valuable information about the process that generated the signal.

We remark, however, that when the nodes $z_j$ are equidistant, then other decompositions of the data with the same coefficients $a_j$ are possible. Let the nodes $z_j$ be defined by (8) and note that

$$z_j^\ell m = e^{2\pi i (j-1)\ell} = \cos(2\pi i (j-1)\ell) + i \sin(2\pi i (j-1)\ell) = 1 + i0 = 1$$

for any integer $\ell$. It follows that

$$z_j^{k+\ell m} = z_j^k z_j^\ell m = z_j^k$$

Therefore, if the polynomial (6) satisfies the interpolation conditions (5), then so does the rational function

$$r(z) = a_1 + a_2 z + a_3 z^2 + \cdots + a_{m-1} z^{-(m-2)} + a_m z^{-(m-1)}, \quad (10)$$

and we obtain the decomposition of the data

$$r(e^{i\theta}) = a_1 + a_2 e^{i\theta} + a_3 e^{2i\theta} + \cdots + a_{m-1} e^{-(m-2)i\theta} + a_m e^{-(m-1)i\theta}$$

with less rapidly oscillating functions. In some applications this decomposition is more meaningful than the decomposition (9).

13.5 The FFT

John Tukey of Princeton University and John Cooley of IBM Research published a paper in 1965 that shows how to compute the coefficients $a_1, a_2, \ldots, a_m$ of the polynomial (6) much more efficiently than the methods outlined in the last section. Their method, based on a divide-and-conquer strategy, is called the fast Fourier transform (FFT). The FFT is one of the most important and widely used mathematical algorithms in existence.

Many variations of the FFT have been developed over the years, including the widely-used FFTW, aka the fastest Fourier transform in the West. It turns out that the basic FFT algorithm had been discovered by several mathematicians (including Gauss) prior to the Cooley-Tukey paper, but the Cooley-Tukey paper profoundly altered the computing landscape. We will investigate the basic idea behind the FFT. Using
coefficients of the interpolating polynomial as
\[ a_{j+1} = \frac{1}{m} \sum_{k=0}^{m-1} y_{k+1} e^{-2\pi i j k/m}, \quad 0 \leq j < m. \] (11)

The FFT is based on the observation that
\[ \left( e^{-2\pi i k/(2m)} \right)^2 = e^{-2\pi i k/m}. \] (12)

We assume for the remainder of this lecture that \( m \) is a power of 2 in order to simplify the presentation. Divide the sum (11) into even and odd terms, and apply (12) to obtain
\[
ma_{j+1} = \sum_{\text{even } k} e^{-2\pi i j k/m} y_{k+1} + \sum_{\text{odd } k} e^{-2\pi i j k/m} y_{k+1}
\]
\[
= \sum_{k=0}^{m/2-1} e^{-2\pi i j (2k)/m} y_{2k+1} + \sum_{k=0}^{m/2-1} e^{-2\pi i j (2k+1)/m} y_{2k+2}
\]
\[
= \sum_{k=0}^{m/2-1} e^{-2\pi i j (2k+1)/m} y_{2k+1} + e^{-2\pi i j m} \sum_{k=0}^{m/2-1} e^{-2\pi i j (2k)/m} y_{2k+2}, \] (13)

where we have written even indices as \( 2k \) and odd ones as \( 2k+1 \). The above manipulations form the basis for the FFT. The above splitting of a sum into two sums over components with even and odd indices is applied by the FFT for all coefficients \( \{a_1, a_2, \ldots, a_m\} \). Specifically, the last equation above (i.e., (13)) suggests that the FFT for \( m \) nodes can be computed by evaluating 2 FFTs with \( m/2 \) nodes and one multiplication. Since \( m \) is a power of 2, we may continue to partition the summation task into 4 FFTs of length \( m/4 \), 8 FFTs of length \( m/8 \), and so on, until we obtain \( m \) FFTs of length one. The 1-node FFT requires requires no summation; it is just a number. Since \( m \) is assumed to be a power of 2, there are \( \log_2 m \) steps in this divide-and-conquer strategy. The total computational effort required is \( O(m \log_2 m) \) arithmetic floating point operations, which stem from the multiplication by factors of the form \( e^{-2\pi i j m} \) and the addition of partial sums in the conquer phase of the algorithm.

Exercise 13.5

Time the MATLAB FFT function `fft` when applied to long vectors with \( m = 2^k \) elements for \( k = 6, 7, 8, \ldots \). Plot your result. How does the computing time grow with \( m \)? Use the MATLAB functions `tic` and `toc` for timing. Also time the MATLAB function `fft` when applied to long vectors with \( 2^k + 1 \) entries for \( k = 6, 7, 8, \ldots \). Plot your timings. Are the computing times longer than for vectors of length \( 2^k \)?

13.6 Polynomial least-squares approximation

Let the nodes \( z_j, 1 \leq j \leq m \), be defined by (8) and consider the least-squares problem
\[
\min_{a_1, a_2, \ldots, a_n} \left\| \begin{array}{cccc}
1 & z_1 & \ldots & z_1^{n-1} \\
1 & z_2 & \ldots & z_2^{n-1} \\
\vdots & \vdots & \ddots & \vdots \\
1 & z_{m-1} & \ldots & z_{m-1}^{n-1} \\
1 & z_m & \ldots & z_m^{n-1} \\
\end{array} \right\| \left[ \begin{array}{c}
a_1 \\
a_2 \\
\vdots \\
a_n \\
\end{array} \right] - \left[ \begin{array}{c}
y_1 \\
y_2 \\
\vdots \\
y_{m-1} \\
y_m \\
\end{array} \right], \] (14)
where $1 \leq n \leq m$. This kind of least-squares problems arise when approximating data given at $m$ equidistant points on the unit circle by a polynomial of degree at most $n - 1$. Here $\| \cdot \|$ denotes the norm

$$\| w \| = \left( \sum_{j=1}^{m} |w_j|^2 \right)^{1/2}$$

for $m$-vectors $w = [w_1, w_2, \ldots, w_m]^T$ with complex entries. Differently from the definition (6) of Lecture 1 of the norm of a vector with real entries, we square the magnitude of the entries. The reason for this is that $w_j^2$ may be negative when $w_j$ is a complex number. This vector norm is invariant under multiplication by a unitary matrix. The fact that the columns of the matrix in (14), after suitable scaling, are the first $n$ columns of a unitary matrix can be used to show that the solution $a_1, a_2, \ldots, a_n$ of the least-squares problem is given by (11); see Exercise 13.7. Hence, the FFT can also be attractive to use when solving least-squares problems.

**Exercise 13.6**

Show the property (2) of complex numbers. Let $w = [w_1, w_2, \ldots, w_m]^T$ and define $w^H = [\bar{w}_1, \bar{w}_2, \ldots, \bar{w}_m]$. This vector is referred to as the conjugate transpose of $w$. Show that

$$w^H w = \| w \|^2.$$

How is conjugate transposition related to the operation $\dagger$ in MATLAB/Octave?

**Exercise 13.7**

Show that the solution of (14) is determined by (11). Use the fact that a matrix $W$ with orthonormal complex columns satisfies $W^H W = I$. □