Lecture 2: The Fast Discrete Fourier Transform

Interpolation on the unit circle

So far, we have considered interpolation of functions at nodes on the real line. When we model periodic phenomena, it is natural to put 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. The equidistant points on the unit

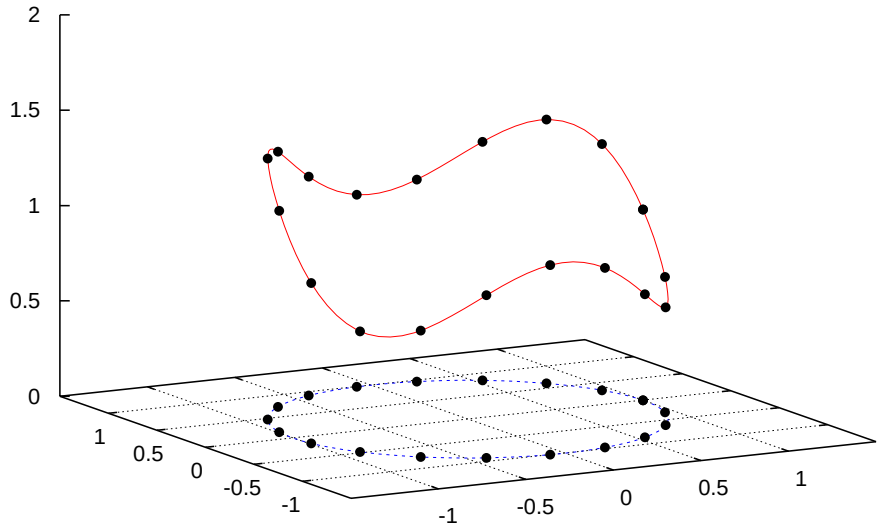

Figure 1: A function defined on the unit circle, 20 equidistant interpolation points, and associated function values.

circle in Figure 1 can be produced in Octave/Matlab with the code

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

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

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

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 like vectors in $\mathbb{R}^2$. Introduce the complex numbers

$$z_1 = x_1 + iy_1, \qquad z_2 = x_2 + iy_2, \qquad x_x, x_2, y_1, y_2 \in \mathbb{R}.$$

Then

$$\begin{aligned} z_1 + z_2 &= x_1 + x_2 + i(y_1 + y_2), \\ z_1 - z_2 &= x_1 - x_2 + i(y_1 - y_2). \end{aligned}$$

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

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

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

$$z_1 z_2 = x_1 x_2 - y_1 y_2 + i(x_1 y_2 + x_2 y_1). \tag{2}$$

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

$$\begin{aligned} z_1 z_2 &= (x_1 + iy_1)(x_2 + iy_2) = x_1(x_2 + iy_2) + iy_1(x_2 + iy_2) \\ &= x_1 x_2 + ix_1 y_2 + iy_1 x_2 + i^2 y_1 y_2 = x_1 x_2 - y_1 y_2 + i(x_1 y_2 + y_1 x_2), \end{aligned}$$

which is equivalent to (2). 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

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

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

$$z_1 := \cos(\theta_1) + i \sin(\theta_1), \qquad z_2 := \cos(\theta_2) + i \sin(\theta_2), \qquad \theta_1, \theta_2 \in \mathbb{R}.$$

Then it follows from (2) that

$$z_1 z_2 = \cos(\theta_1)\cos(\theta_2) - \sin(\theta_1)\sin(\theta_2) + i(\cos(\theta_1)\sin(\theta_2) + \cos(\theta_2)\sin(\theta_1)),$$

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:

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

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

$$z_1 = e^{i\theta_1}, \qquad z_2 = e^{i\theta_2},$$

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.

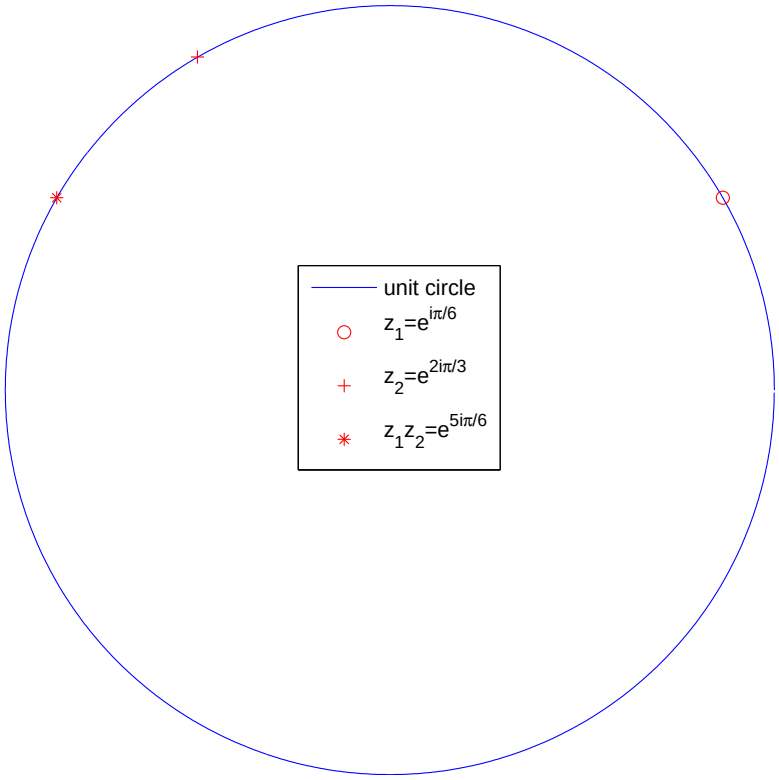

Figure 2: Multiplication of complex numbers z_1 and z_2 on the unit circle.

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.

Polynomial interpolation in the complex plane

Let $\{z_1, z_2, \ldots, z_n\}$ be distinct complex nodes and let $\{y_1, y_2, \ldots, y_n\}$ real or complex numbers. We consider the polynomial interpolation problem: Determine a polynomial p of degree at most $n-1$, such that

$$p(z_j) = y_j, \qquad 1 \le j \le n. \tag{3}$$

Express the interpolation polynomial in power form

$$p(z) = a_1 + a_2 z + a_3 z^2 + \cdots + a_{n-1} z^{n-2} + a_n z^{n-1}, \tag{4}$$

where the coefficients a_j are real or complex numbers. Then the interpolation conditions (3) give the linear system of equations

$$\begin{bmatrix} 1 & z_1 & z_1^2 & \ldots & z_1^{n-2} & z_1^{n-1} \\ 1 & z_2 & z_2^2 & \ldots & z_2^{n-2} & z_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & z_{n-1} & z_{n-1}^2 & \ldots & z_{n-1}^{n-2} & z_{n-1}^{n-1} \\ 1 & z_n & z_n^2 & \ldots & z_n^{n-2} & z_n^{n-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}. \tag{5}$$

The interpolation problem (3) has a unique solution if and only if the above Vandermonde matrix is nonsingular. We are particularly interested in the case when the nodes z_j are equidistant on the unit circle, i.e., when

$$z_j = e^{2\pi i (j-1)/n}, \qquad 1 \le j \le n. \tag{6}$$

Then the Vandermonde matrix is quite special.

Exercises

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? The matrix A^{-1} can be expressed in terms of A. How? Report your findings.
2. Estimate the number of (complex) floating-point operations that are needed to compute solve the interpolation problem (5) without using the structure of the matrix?
3. Estimate the number of (complex) floating-point operations that are needed to solve the interpolation problem (5) when using the structure of the matrix uncovered in Exercise 1.

Fourier analysis

The vector $e^{ik\theta}$ traverses the unit circle k times in the counter-clockwise direction when θ increases from zero to 2π. 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 (4) shows that the interpolation polynomial expresses the data $\{y_j\}_{j=1}^n$ as a sum of oscillations,

$$p(e^{i\theta}) = a_1 + a_2 e^{i\theta} + a_3 e^{i2\theta} z^2 + \cdots + a_{n-1} e^{i(n-2)\theta} + a_n e^{i(n-1)\theta}. \tag{7}$$

If, say, a_2 is large and the other coefficients are small, then the data $\{y_j\}_{j=1}^n$ 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 (6) and note that

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

for any integer ℓ. It follows that

$$z_j^{k+\ell n} = z_j^k \, z_j^{\ell n} = z_j^k$$

Therefore, if the polynomial (4) satisfies the interpolation conditions, then so does the rational function

$$r(z) = a_1 + a_2 z + a_3 z^2 + \cdots + a_{n-1} z^{-2} + a_n z^{-1}, \tag{8}$$

and we obtain the decomposition of the data

$$r(e^{i\theta}) = a_1 + a_2 e^{i\theta} + a_3 e^{i2\theta} + \cdots + a_{n-1} e^{-i2\theta} + a_n e^{-i\theta}$$

with less rapidly oscillating functions. In many, but not all, applications this decomposition is more meaningful than the decomposition (7).

The FFT

John Tukey of Princeton University and John Cooley of IBM Research published a paper in 1965 that shows how to compute the DFT much more efficiently than the methods outlined in the last section. Their method, based on a *divide and conquer* strategy, is called the fast discrete Fourier transform, or 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 in fact discovered by many others (including Lanczos and Gauss) prior to the Cooley-Tukey paper, but it was that paper that profoundly altered the computing landscape. We will investigate the basic idea behind the FFT.

Using the structure of the Vandermonde matrix unveiled in Exercise 3, we can express the Fourier coefficients a_j in the interpolating polynomial as

$$a_{j+1} = \frac{1}{n} \sum_{k=0}^{n-1} y_{k+1} e^{-2\pi i jk/n}, \qquad 0 \le j < n. \tag{9}$$

The FFT is based on the observation that (note the $2n$ in the denominator on the left)

$$\left(e^{-2i\pi k/2n}\right)^2 = e^{-2i\pi k/n}.$$

We will assume for the remainder of these lecture notes that n is a power of 2 in order to simplify things. Apply the above observation to the DFT coefficient formula defined by Equation (9), dividing the sum into even and odd parts:

$$\begin{aligned} na_{j+1} &= \sum_{even\ k} e^{-2i\pi jk/n} y_{k+1} + \sum_{odd\ k} e^{-2i\pi jk/n} y_{k+1} \\ &= \sum_{k=0}^{n/2-1} e^{-2i\pi j(2k)/n} y_{2k+1} + \sum_{k=0}^{n/2-1} e^{-2i\pi j(2k+1)/n} y_{2k+2} \\ &= \sum_{k=0}^{n/2-1} e^{-2i\pi j(2k+1)/n} y_{2k+1} + e^{-2i\pi jn} \sum_{k=0}^{n/2-1} e^{-2i\pi j(2k)/n} y_{2k+2}. \end{aligned} \tag{10}$$

Despite the complicated-looking formulas, we are again only using basic high-school algebra and writing even indices as $2k$ and odd ones as $2k+1$. Note that equation (10) outlines the computation of one DFT coefficient, a_{j+1}. The whole DFT computes all n coefficients $\{a_1, a_2, \ldots, a_n\}$.

The last equation above, equation (10), suggests that the we can divide the DFT computation of length n nodes into two DFT computations each of length $n/2$ nodes, plus one multiplication. Recall that we assumed n is a power of 2. That means that we can continue to partition the DFT problem into four DFTs of length $n/4$, eight DFTs of length $n/8$, and so on, up to n DFTs of length 1. A DFT of length 1 has no summation–it's just a number. If n is a power of 2, then there are $\log_2 n$ steps in this divide and conquer strategy. Thus we require about $O(\log_2 n)$ computations to compute each coefficient of the DFT with this method. The total computational requirement of this method for all n coefficients is about $O(n \log_2 n)$.

Exercise

4. Experiment with timings using long vectors, whose lengths are a power of two. Compare the two DFT methods from the last section with the FFT (just use the fft command).