MATH 934 – FAST FOURIER TRANSFORM

INSTRUCTOR: DR. ADAM LARIOS

Let f be a $2\pi-$periodic continuous function on the interval $[0, 2\pi) = \mathbb{R}/(2\pi\mathbb{Z})$. Choose a positive integer $n \in \mathbb{N}$, and let $N = 2n$ be the number of gridpoints. Define the (evenly spaced) interpolation points

$$x_j = \frac{2\pi}{N} j, \qquad j = 0, 1, \cdots, N-1.$$

Let us also denote the functions E_k for $k = 0, 1, \cdots, N-1$, by

$$E_k(x) = e^{ikx} = (e^{ix})^k, \quad \text{where} \quad e^{ix} = \cos(x) + i\sin(x).$$

The set of trigonometric polynomials of degree at most $N-1$ is given by

$$T_N = \Big\{ \sum_{k=0}^{N-1} c_k E_k \Big| c_k \in \mathbb{C} \Big\} \quad \text{on } [0, 2\pi).$$

The problem of trigonometric interpolation is to find a trigonometric polynomial function $P \in T_N$ defined by

$$\begin{cases} P(x) & = \sum_{k=0}^{N-1} c_k E_k(x) \\ & \text{and} \\ P(x_j) & = f(x_j) \quad \text{for } j = 0, 1, \cdots, N-1. \end{cases} \tag{1}$$

We showed in class that such a polynomial always exists, and gave an explicit formula for the coefficients c_k. Indeed, let

$$\omega_N = e^{2\pi i/N}, \quad \omega_n = e^{2\pi i/n}$$

and define a matrix W to have entry ω_N^{kj} at the k^{th} row and j^{th} column. One can show that $\overline{W}^T W = NI$, so W is invertible. Denote by $\vec{f}$ the vector with entries $f_j := f(x_j)$, $j = 0, 1, \cdots, N-1$, and let $\vec{c}$ denote the vector with values c_k, $k = 0, 1, \cdots, N-1$. Then, we showed that $\vec{c} = \frac{1}{N}\overline{W}^T \vec{f}$. Since W is invertible, so there is a one-to-one correspondence between the function values $\{f(x_j)\}_{j=0}^{N-1}$ and the (discrete) Fourier coefficients $\{c_k\}_{k=0}^{N-1}$. The map

$$\mathcal{F}_N : \vec{f} \longmapsto \vec{c} = \frac{1}{N}\overline{W}^T \vec{f}$$

is called the *Discrete Fourier Transform* (DFT). It has a well-defined inverse called the *Inverse Discrete Fourier Transform*. To remind us that the coefficients c_k come from f, we often denote

$$\widehat{f}_k := c_k = \frac{1}{N} \sum_{j=0}^{N-1} f(x_j)\overline{E}_k(x_j).$$

We now find a more efficient algorithm, the Fast Fourier Transform (FFT) for computing these coefficients. Let us subdivide the problem into interpolating over even- and odd-numbered points.

Next, consider the even-numbered points, $\{x_{2j}\}_{j=0}^{n-1}$ and the odd-numbered points, $\{x_{2j+1}\}_{j=0}^{n-1}$. Note that

$$x_{2j} = \frac{2\pi}{N}(2j) = \frac{2\pi}{2n}(2j) = \frac{2\pi}{n}j,$$
$$x_{2j+1} = \frac{2\pi}{N}(2j+1) = \frac{2\pi}{2n}(2j+1) = \frac{2\pi}{n}j + \frac{\pi}{n}.$$

We note the following simple by very important observation:

$$\begin{aligned} c_k &= \frac{1}{N}\sum_{j=0}^{N-1} f_j \overline{E}_k(x_j) = \frac{1}{2n}\sum_{j=0}^{2n-1} f(x_j)\overline{E}_k(x_j) \\ &= \frac{1}{2}\frac{1}{n}\sum_{j=0}^{n-1} f(x_{2j})\overline{E}_k(x_{2j}) + \frac{1}{2}\frac{1}{n}\sum_{j=0}^{n-1} f(x_{2j+1})\overline{E}_k(x_{2j+1}) \\ &= \frac{1}{2}\frac{1}{n}\sum_{j=0}^{n-1} f(x_{2j})\overline{E}_k(x_{2j}) + \frac{1}{2}e^{-i\pi k/n}\frac{1}{n}\sum_{j=0}^{n-1} f(x_{2j+1})\overline{E}_k(x_{2j}) \end{aligned}$$

Thus, the Fourier coefficients from the interpolations of odd and even points can be used to find the Fourier coefficients of all the points.

For the sake of concreteness, let us try this in the case $N = 4 = 2n$, $n = 2$. Then

$$\begin{aligned} c_k &= \tfrac{1}{4}\sum_{j=0}^{4-1} f(x_j)\overline{E}_k(x_j) \\ &= \tfrac{1}{4}f(x_0)e^{-0\frac{2\pi i}{4}k} + \tfrac{1}{4}f(x_1)e^{-1\frac{2\pi i}{4}k} + \tfrac{1}{4}f(x_2)e^{-2\frac{2\pi i}{4}k} + \tfrac{1}{4}f(x_3)e^{-3\frac{2\pi i}{4}k} \\ &= \tfrac{1}{4}f(x_0) + \tfrac{1}{4}f(x_1)e^{-\frac{\pi i}{2}k} + \tfrac{1}{4}f(x_2)e^{-\frac{2\pi i}{2}k} + \tfrac{1}{4}f(x_3)e^{-3\frac{\pi i}{2}k} \\ &= \tfrac{1}{2}\underbrace{\left(\tfrac{1}{2}f(x_0) + \tfrac{1}{2}f(x_2)e^{-\frac{2\pi i}{2}k}\right)}_{\widetilde{c}_k^{\text{even}}} + \tfrac{1}{2}e^{-\frac{\pi i}{2}k}\underbrace{\left(\tfrac{1}{2}f(x_1) + \tfrac{1}{2}f(x_3)e^{-\frac{2\pi i}{2}k}\right)}_{\widetilde{c}_k^{\text{odd}}}. \end{aligned}$$

We see that $\widetilde{c}_k^{\text{even}}$ is the k^{th} Fourier coefficient that would be generated by generating the Fourier coefficients from just the data $f(x_0)$ and $f(x_2)$, and $\widetilde{c}_k^{\text{odd}}$ is the k^{th} Fourier coefficient that would be generated by generating the Fourier coefficients from just the data $f(x_1)$ and $f(x_3)$.

We can see this another way as well, using the following trick. Let $p(x)$ be the interpolation at the even-numbered points $\{x_{2j}\}_{j=0}^{n-1}$ of the even-numbered values $\{f(x_{2j})\}_{j=0}^{n-1}$. Let $q(x)$ by the interpolation *again at the even-numbered* $\{x_{2j}\}_{j=0}^{n-1}$ points, but with the *odd-numbered* values $\{f(x_{2j+1})\}_{j=0}^{n-1}$. That is,

$$p(x_{2j}) = f(x_{2j}) \qquad \text{and} \qquad q(x_{2j}) = f(x_{2j+1}).$$

Then, we define

$$\widetilde{P}(x) = \tfrac{1}{2}(1 + E_n(x))p(x) + \tfrac{1}{2}(1 - E_n(x))q(x - \tfrac{\pi}{n})$$

We will now show that $\widetilde{P} = P$, where P is defined by (1).

To see this, first note that, since j is an integer,

$$E_n(x_{2j}) = e^{in\frac{2\pi}{n}j} = e^{2\pi ij} = 1,$$
$$E_n(x_{2j+1}) = e^{in\left(\frac{2\pi}{n}j + \frac{\pi}{n}\right)} = e^{2\pi ij}e^{i\pi} = -1.$$

Thus,

$$\widetilde{P}(x_{2j}) = \tfrac{1}{2}(1+1)p(x_{2j}) + \tfrac{1}{2}(1-1)q(x_{2j} - \tfrac{\pi}{n}) = p(x_{2j}) = f(x_{2j}),$$

and

$$\widetilde{P}(x_{2j+1}) = \tfrac{1}{2}(1+(-1))p(x_{2j+1}) + \tfrac{1}{2}(1-(-1))q(x_{2j+1} - \tfrac{\pi}{n}) = q(x_{2j}) = f(x_{2j+1}).$$

Moveover, let

$$p(x) = \sum_{j=0}^{n-1} \alpha_j E_j(x) \qquad \text{and} \qquad q(x) = \sum_{j=0}^{n-1} \beta_j E_j(x)$$

Then, since $E_n(x)E_j(x) = E_{n+j}(x)$, we find

$$\begin{aligned}
\widetilde{P}(x) &= \tfrac{1}{2}(1 + E_n(x)) \sum_{j=0}^{n-1} \alpha_j E_j(x) + \tfrac{1}{2}(1 - E_n(x)) \sum_{j=0}^{n-1} \beta_j E_j(x - \tfrac{\pi}{n}) \\
&= \tfrac{1}{2}(1 + E_n(x)) \sum_{j=0}^{n-1} \alpha_j E_j(x) + \tfrac{1}{2}(1 - E_n(x)) \sum_{j=0}^{n-1} \beta_j E_j(x) e^{-\pi i/n} \\
&= \tfrac{1}{2} \sum_{j=0}^{n-1} (\alpha_j E_j(x) + \beta_j E_j(x) e^{-\pi i/n}) + \tfrac{1}{2} \sum_{j=0}^{n-1} (\alpha_j E_{n+j}(x) - \beta_j E_{n+j}(x) e^{-\pi i/n})
\end{aligned}$$

Comparing this with (1), we see that if we set

$$\begin{aligned}
c_j &= \tfrac{1}{2}\alpha_j + \tfrac{1}{2}e^{-i\pi/n}\beta_j, \qquad j = 0, 1, \ldots, n-1, \\
c_{n+j} &= \tfrac{1}{2}\alpha_j - \tfrac{1}{2}e^{-i\pi/n}\beta_j, \qquad j = 0, 1, \ldots, n-1,
\end{aligned}$$

then we can write

$$\widetilde{P}(x) = \sum_{j=0}^{N-1} c_j E_j(x) = P(x).$$

This gives us the following algorithm for computing the coefficients, written in Matlab code. It uses *recursion*, which means that the function calls itself.

```
1  function f = demoFFT(f)
2  N = length(f);
3
4  if (N <= 1)
5      return;
6  end
7
8  odd  = demoFFT(f(1:2:N));
9  even = demoFFT(f(2:2:N));
10 for j = 1:(N/2)
11     E = exp(-2i*pi*(j-1)/N);
12     f(j)     = 0.5*(odd(j) + E*even(j));
13     f(j+N/2) = 0.5*(odd(j) - E*even(j));
14 end
15
16 end
```

Matlab's built-in function `fft` leaves off the `0.5` factors on lines 12 and 13. This has the result of multiplying the final result by a factor of N. To check that the above code agrees with Matlab's fft, you can save the code above as `demoFFT` and try the following.

```
>> N = 32; f = rand(1,N); norm(demoFFT(f)*N - fft(f))
```

This is not the fastest way to code the Fast Fourier Transform (although it is far faster than computing the Discrete Fourier Transform directly, i.e., by matrix multiplication). It is just a demo to illustrate the core idea. Many optimizations can be made; some obvious, so not so obvious. In your codes, it is best if you call Matlab's built-in function `fft`, which is a highly optimized Fast Fourier Transform.