# EECS 16B Designing Information Devices and Systems II
# Fall 2019 Discussion Worksheet Discussion 15A

## Questions

**1. The DFT basis as a polynomial basis**

Earlier we thought about interpolation as learning about a function globally from its samples, and using that data to recover the parameters underlying the function. Given the parameters, we could predict what the function's value is even where we didn't take a sample. We thought about polynomials as being a reasonable way to approximate functions, and this naturally led us to a set of basis functions that corresponded to the monomials $x^k$. Lagrange interpolation told us that the feature vectors that would result from sampling such monomials would be linearly independent as long as the vectors satisfied the trivial condition of being long enough (longer than the highest degree being considered) and that the samples were taken at different points. Unfortunately, while they would be linearly independent, the challenge was that even for simple univariate functions of time, the monomials would be numerically poorly behaved over the real line. High-degree monomials tend to blow up to ginormous numbers when given an input with magnitude greater than 1 and tend to go to tiny numbers close to zero when given an input with magnitude less than 1.

The Discrete Fourier Transform (DFT) bases (yes *plural*) can be viewed as a clever way around this difficulty with monomials. Instead of evaluating the monomials on the real line, where they are generically badly behaved, we find a way of evaluating them on the complex unit circle $e^{j\theta}$ for $\theta \in [0, 2\pi)$, where they are well behaved. The $k$-th monomial $x^k$ becomes $e^{jk\theta} = \cos(k\theta) + j\sin(k\theta)$ when evaluated at points $x = e^{j\theta}$ on the unit circle. This manifests the idea of increased "wiggliness" that higher-degree polynomials also invoke, without the blowing-up or shrinking-to-zero behaviors.

So how does this work? We take a list of $n$ regularly-spaced samples of our unknown function $f$ over some interval of interest. Then, we identify the interval with the unit circle, or $\theta \in [0, 2\pi)$. This means that we pretend as though the $i$-th sample point was at $x_i = e^{j\frac{2\pi i}{n}}$ for $i = 0, 1, \ldots, n-1$ — the $n$-th roots of unity, evenly spaced around the unit circle. The $y_i$ value is whatever the function returned for that sample. The goal now becomes learning what the coefficients of our underlying basis functions are, and this can be solved by solving an appropriate system of equations given by $B\vec{F} = \vec{y}$ where the columns of $B$ represent our different basis functions evaluated at the sampling points $x_0, x_1, \ldots, x_{n-1}$. To avoid a proliferation of notation, instead of $\vec{y}$, we will just use $\vec{f}$ because we want to remind ourselves that these are samples from our underlying function $f$ that we want to learn.

Because of the interesting properties of exponentials and powers, the $k$th DFT basis vector represents a counterclockwise path through the $n$-th roots of unity starting at 1 and then moving by $k$ roots at a time. So the 0th DFT basis vector just stays at 1. The 1st DFT basis vector goes through the $n$ roots of unity one at a time clockwise starting with 1. The 2nd DFT basis vector takes two trips around the unit circle starting at 1 and moving two at a time. And so on.

There are several different ways to normalize the DFT basis vector, each coming from different motivations. Below is a table of three such normalizations in widespread use. They don't have standardized names, and so we will give them names that we like.

 

| Variant | Normalizing Factor | Notation | entries |
|---|---|---|---|
| Polynomial DFT basis | 1 | $\vec{b}_k$ | $b_k[i] = e^{j\frac{2\pi ik}{n}}$ |
| Orthonormal DFT basis | $\frac{1}{\sqrt{n}}$ | $\vec{u}_k$ | $u_k[i] = \frac{1}{\sqrt{n}} e^{j\frac{2\pi ik}{n}}$ |
| Traditional/Classic DFT basis | $\frac{1}{n}$ | $\vec{d}_k$ | $d_k[i] = \frac{1}{n} e^{j\frac{2\pi ik}{n}}$ |

The table above defines the relevant matrices $B = [\vec{b}_0, \vec{b}_1, \cdots, \vec{b}_{n-1}]$ or $U = [\vec{u}_0, \vec{u}_1, \cdots, \vec{u}_{n-1}]$ or $D = [\vec{d}_0, \vec{d}_1, \cdots, \vec{d}_{n-1}]$ and correspondingly result in relationships (e.g. $B\vec{F} = \vec{f}$) between the learned parameters $\vec{F}$ and the original data $\vec{f}$. They clearly all differ only by scaling[1].

The goal here is to understand how to use the DFT basis to help us actually do interpolation.

**Notation:** We can think of a real-world signal that is a function of time $f(t)$. By recording its values at regular intervals, we can represent it as a vector of discrete samples $\vec{f}$, of length $n$.

$$\vec{f} = \begin{bmatrix} f[0] \\ f[1] \\ \vdots \\ f[n-1] \end{bmatrix} \quad \text{and} \quad \vec{F}_U = \begin{bmatrix} F[0] \\ F[1] \\ \vdots \\ F[n-1] \end{bmatrix} \tag{1}$$

Here $\vec{F}_U$ are the parameters defining $\vec{f}$ after a change of basis into the DFT basis $U$, that is

$$\vec{F}_U = U^{-1}\vec{f} = U^*\vec{f} \tag{2}$$

where $U$ is a matrix of the normalized DFT basis vectors.

We define $\omega = e^{j\frac{2\pi}{n}}$ for convenience so we can write:

$$U = \begin{bmatrix} | & & | \\ \vec{u}_0 & \cdots & \vec{u}_{n-1} \\ | & & | \end{bmatrix} = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \tag{3}$$

Alternatively, we have that $\vec{f} = U\vec{F}_U$ or more explicitly

$$\vec{f} = F_U[0]\vec{u}_0 + \cdots + F_U[n-1]\vec{u}_{n-1} \tag{4}$$

[1] FYI: Numpy given normalization "none" will compute the Traditional/Classic map and solve the equation $D\vec{F} = \vec{f}$ to return $\vec{F}$. If you give Numpy the normalization "ortho", then it will compute the Orthonormal DFT basis version and solve $U\vec{F} = \vec{f}$ and return $\vec{F}$. Nobody bothered to define a separate normalization for the Polynomial version because it turns out that it can be computed by calling the inverse classic DFT on a "flipped" vector $\vec{f}$. By the end of this discussion, you will hopefully be in a place to understand that. But since this is an EECS course and the FFT is literally considered one of the most important algorithms of all time, we felt that it was important to connect to numpy here.

 

In other words, $\vec{f}$ is a linear combination of the weighted complex exponentials $\vec{u}_i$ with coefficients $F_U[i]$. We can similarly define $\vec{F}_B$ and $\vec{F}_D$ for the other bases, although because those are not normalized, the inverse isn't just the complex conjugate. Some scaling is also required.

Let $s_k = \omega^k$ be the $k$th $n$th-root-of-unity. The $n$ points $s_0, s_1, \ldots, s_{n-1}$ are the $n$th-roots-of-unity.

(a) For $n = 7, 8$, **please sketch $s_0, s_1, \ldots, s_{n-1}$ on the complex plane.**

(b) For $n = 8$, create vectors $\vec{b}_k$ for $k = 0, 1, 2, 6, 7$ by stacking up the evaluations of $x^k$ over the $s_i$. The $i$-th entry in $\vec{b}_j$ should be $s_i^k$.
**What do you observe?**

(c) **Once you have an orthonormal basis $U$, how would you use it to transform $\vec{f}$ into this basis? How would you use this to get $\vec{F}_B$ for example? Or $\vec{F}_D$?**

(d) Now, suppose we want to use a DFT basis to do interpolation. That means that instead of evaluating $f(x_i)$ for one the $x_i$ that we effectively sampled it at, we want to ask for an estimate of $f(e^{j\theta})$ for a $\theta \neq \frac{2\pi i}{n}$ for some $i$. That was the point — we want to recover what is happening between samples. For this, we will try to use the natural $B$ basis corresponding to the natural monomials. We have learned coefficients $F_B[0], F_B[1], \ldots, F_B[n-1]$. We will drop the subscript $B$ from now on. Suppose we estimate the value of $f$ at a point $e^{j\theta}$ by the most natural *interpolation*:

$$\hat{f}(e^{j\theta}) = F[0] + F[1]e^{j\theta} + \cdots + F[n-1]e^{j(n-1)\theta}.$$

Here, consider the three dimensional basis matrix

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{j\omega} & e^{j2\omega} \\ 1 & e^{j2\omega} & e^{j4\omega} \end{bmatrix}$$

with $\omega = \frac{2\pi}{3}$.
Suppose $\vec{f} = [1, 1, 1]^\top$, **compute $\hat{f}(e^{j\theta})$? Does this make sense?**

 

(e) Continuing the previous part, suppose $\vec{f} = [0,1,1]^\top$ and $\vec{f} = [0,1,2]^\top$, **compute $\hat{f}(e^{j\theta})$? What is it evaluated at $\theta = \pi$, which corresponds to a time halfway between the last two samples. Does this make sense?**

(f) Continuing the previous part, **Show that for general $\vec{f}$ and $\theta \in [0, 2\pi)$, the predicted $\hat{f}$ is complex even if $\vec{f}$ were all real.**

(g) One way to bypass this difficulty is to rethink the functions. Recall from part (b) that we saw how $\vec{b}_1 = \vec{\bar{b}}_7$ and $\vec{b}_2 = \vec{\bar{b}}_6$.

**Show that under fixed $n$, we have $\vec{b}_{-k} = \vec{b}_{n-k}$ for all $k$.**

This phenomenon is called "aliasing[2]" — the same vector of samples has many "names" (true underlying functions) that it corresponds to sampling.

(h) From this we see that the basis vectors haven't changed as far as their samples go, but they can be reinterpreted as corresponding to different basis functions. Effectively, this means we can write complex exponentials in pairs of positive and negative frequencies, giving us interpolations that are real. Essentially, we can choose the most suitable backstory for our purposes.

Here, the evenness and oddness of $n$ is going to matter. Let's start with the easier to understand case of $n$ being odd.

[2]Notice how this classical terminology is somewhat curiously sample-centric. It views the sampled vectors as being the objects and the functions as being "names" — and so the same vector has many aliases. The reality is actually reversed and the phenomenon is actually one of Dopplegangers. Many different functions can look the same if you only sample them at those points. The name of the game here is in choosing among these Dopplegangers and those choices have consequences as we are seeing here. This turns out to be a major issue in machine learning far beyond interpolation.

 

**Show that**

$$B = \begin{bmatrix} | & | & | & & | & | \\ \vec{b}_0 & \vec{b}_1 & \vec{b}_2 & \cdots & \vec{b}_{n-2} & \vec{b}_{n-1} \\ | & | & | & & | & | \end{bmatrix}$$

**can be equivalently written as**

$$B = \begin{bmatrix} | & | & & | & | & & | & | \\ \vec{b}_0 & \vec{b}_1 & \cdots & \vec{b}_{\lfloor \frac{n}{2} \rfloor} & \vec{b}_{-\lfloor \frac{n}{2} \rfloor} & \cdots & \vec{b}_{-2} & \vec{b}_{-1} \\ | & | & & | & | & & | & | \end{bmatrix}.$$

(i) Hence, instead of doing interpolation naively as above, we can do interpolation by

$$\begin{aligned} \hat{f}(e^{j\theta}) =& F[0] + \\ & F[1]e^{j\theta} + \cdots + F\left[\lfloor \frac{n}{2} \rfloor\right] e^{j\lfloor \frac{n}{2} \rfloor \theta} + \\ & F\left[n - \lfloor \frac{n}{2} \rfloor\right] e^{j\left(-\lfloor \frac{n}{2} \rfloor\right)\theta} + \cdots + F[n-2]e^{-j2\theta} + F[n-1]e^{-j\theta} \\ =& F[0] + \sum_{k=1}^{\lfloor \frac{n}{2} \rfloor} (F[k]e^{jk\theta} + F[n-k]e^{-jk\theta}) \end{aligned} \tag{5}$$

**Repeat the earlier examples of** $[1,1,1]^\top$**,** $[0,1,1]^\top$ **and** $[0,1,2]^\top$ **with the above rethinking of what the functions are.**

(j) Now, let's show this is true in general (for odd $n$). **Show that the interpolation method of** (5) **for** $e^{j\theta}$ **gives rise to real numbers whenever the samples** $\vec{f}$ **are real.**

 

(k) This leaves the case of even $n$. For even $n$, there is a special vector at position $\frac{n}{2}$. **Show that $\vec{b}_{\frac{n}{2}}$ consists entirely of $+1$ and $-1$ in alternating order.**

(l) For even $n$, we need to decide how to interpret this special position $\frac{n}{2}$. In keeping with the spirit of the odd case, we decide to split our ambiguity. We pretend that it comes from both $e^{j\frac{n}{2}\theta}$ and $e^{-j\frac{n}{2}\theta}$ and was in fact samples of the average of those two. **Show that on the $n$-th roots of unity where $\theta = \frac{2\pi i}{n}$, both of those functions must agree with each other.**
This results in the interpolation rule:

$$\begin{aligned}\hat{f}(e^{j\theta}) =& F[0]+ \\ & \sum_{k=1}^{\frac{n}{2}-1}(F[k]e^{jk\theta}+F[n-k]e^{-jk\theta})+ \\ & \frac{1}{2}F[\frac{n}{2}](e^{j\frac{n}{2}\theta}+e^{-j\frac{n}{2}\theta}) \end{aligned} \tag{6}$$

which is also always real by the same reasons as above, together with the fact that from what you've shown above, $F[\frac{n}{2}]$ is real and the function it multiplies is a pure cosine.

(m) Compute the DFT coefficients $\vec{F}$ for the following signal with respect to the polynomial basis:

$$\vec{f} = \begin{bmatrix} \sin\left(\frac{2\pi}{3}(0)\right) & \sin\left(\frac{2\pi}{3}(1)\right) & \sin\left(\frac{2\pi}{3}(2)\right) & \sin\left(\frac{2\pi}{3}(3)\right) & \sin\left(\frac{2\pi}{3}(4)\right) & \sin\left(\frac{2\pi}{3}(5)\right) \end{bmatrix}^T.$$

(n) Now, consider real six-dimensional vectors. Suppose we knew

$$\vec{f} = \begin{bmatrix} 1 \\ f[1] \\ f[2] \\ 1 \\ 1-\frac{\sqrt{3}}{2} \\ f[5] \end{bmatrix}.$$

 

where the $f[j]$ variables indicate values that we do not know. Suppose we further knew that in the polynomial DFT domain that $F[m] = 0$ for $|m| \geq 2$.

Find the missing values that we do not know. What is $\vec{F}$?

(o) What if we didn't know that $f[4]$ is $1 - \frac{\sqrt{3}}{2}$? Would there be a unique $\vec{f}$ that is compatible with the given information?

**Contributors:**

- Yen-Sheng Ho.
- Yu-Yun Dai.
- Kuan-Yun Lee.
- Anant Sahai.

 