Spectral Derivatives

Pavel Komarov

January 8, 2025

One of the happiest accidents in all math is the ease of transforming a function to and taking derivatives in the Fourier (i.e. the *frequency*) domain. But in order to exploit this extraordinary fact without serious artefacting, and in order to be able to use a computer, we need quite a bit of extra knowledge and care.

This document sets out the math behind the `spectral-derivatives` Python package, all the way down to the bones, as much as I can manage. I try to get in to the real *whys* behind what we're doing here, touching on fundamental signal processing and calculus concepts as necessary, and building upwards to more general cases.

Contents

1 Bases

A *basis* is a set of functions, call them $\{\xi_k\}$, that can be linearly combined to produce other functions. Often these are chosen to be orthogonal, meaning that if we take the "inner product" of one funtion from the set with itself, we get back a constant (often normalized to be 1), and if we take the inner product of one of these functions with a different member of the set, we get back 0. In this sense the members of an orthogonal basis set are like perpendicular directions on a graph.

The inner product between two functions f and g is a generalization of the inner product between vectors, where instead of summing over a finite number of discrete entries, we integrate over infinitely many infinitesimally-separated points in the domain. We define it as:

$$\langle f, g \rangle = \int_a^b \overline{f(x)} g(x) dx$$

where the overbar $\bar{\circ}$ denotes a complex conjugate.

The inner product is symmetrical, so

$$\langle f, g \rangle = \langle g, f \rangle = \int_a^b f(x) \overline{g(x)} dx$$

Note that if we set a and b at $\pm\infty$, this integral could diverge. If it doesn't diverge with infinite bounds, we say the argument is "Lebesgue integrable"[1]. Some of what we'll do only makes sense for this class of functions, so be aware.

1.1 The Fourier Basis

The most famous basis is the *Fourier* basis,[a] which is the set of complex exponentials:

$$e^{j\omega x} = \cos(\omega x) + j \sin(\omega x) \tag{1}$$

where I use j to represent the imaginary unit ($\sqrt{-1}$), because I'm from Electrical Engineering, and because Python uses j.

Why this identity is true isn't obvious at first but can be seen by Taylor Expanding[3] the exponential function and trigonometric functions:

$$e^a = 1 + a + \frac{a^2}{2!} + \frac{a^3}{3!} + ... = \sum_{n=0}^{\infty} \frac{a^n}{n!}$$

So

$$e^{j\omega x} = 1 + j\omega x + \frac{(j\omega x)^2}{2!} + \frac{(j\omega x)^3}{3!} + ... = 1 + j\omega x - \frac{(\omega x)^2}{2!} - j\frac{(\omega x)^3}{3!} + \frac{(\omega x)^4}{4!} - ...$$

$$\sin(\omega x) = \omega x - \frac{(\omega x)^3}{3!} + \frac{(\omega x)^5}{5!} - \frac{(\omega x)^7}{7!} + ...$$

$$\cos(\omega x) = 1 - \frac{(\omega x)^2}{2!} + \frac{(\omega x)^4}{4!} - \frac{(\omega x)^6}{6!} + ...$$

Notice all the even-power terms appear with alternating sign as in the cosine expansion, and the odd-power terms appear with alternating sign as in the sine expansion, but with an extra j multiplied in.

The presence of complex numbers to make this work can be confusing at first, but don't be scared! All we're really doing is using a compressed representation of a sine plus a cosine, where the real and imaginary parts (orthogonal in the complex plane, and therefore independent and non-interfering) allow us to describe the contributions of sine and cosine simultaneously. In fact, Joseph Fourier originally used only real trigonometric functions[5], and it wasn't until later someone decided it would be easier to work with complex exponentials. Later (subsection 2.1) we'll see that for real signals all the complex numbers cancel, leaving only a *real* sine and *real* cosine, which when added together make *a single, phase-shifted sinusoid!* So think of $e^{j\omega x}$ as oscillations of frequency ω along direction x.

If we inner product sinusoidal wiggles of mismatched frequency, differing by $\Delta\omega$, they misalign and integrate to 0 over a period of length $T = \frac{2\pi}{\Delta\omega}$. But if we inner product matched wiggles, they align, multiply to 1 because of the inner product's complex conjugate, and integrate to a constant, the length of whichever domain we choose to consider, so this basis is orthogonal.

2 Transforms

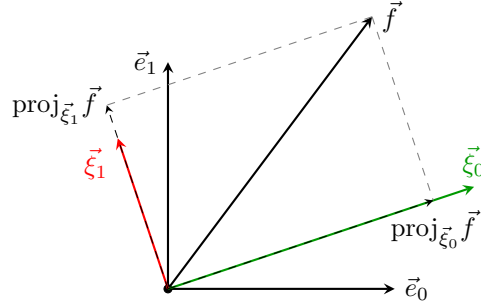We can construct a function from a linear combination of basis functions as:

$$f(x) = \sum_{k=0}^{M-1} c_k \xi_k(x), \quad x \in [a, b] \tag{2}$$

where M is the number of basis functions we're using in our reconstruction and k iterates through them. This is essentially a *recipe*, which tells us *how much* of each basis function ξ_k is present in the signal f on a domain between a and b.

We can find the quantities $\{c_k\}$ by taking the inner product of the above with each of the basis functions to produce a system of M equations, then solving. In the special case where $\{\xi_k\}$ are orthogonal, only one of the inner-producted terms from the sum will be nonzero, so the system's equations untangle to yield the simple relationship:

$$c_k = \frac{\langle \xi_k, f \rangle}{\langle \xi_k, \xi_k \rangle} = \frac{\int_a^b f(x)\overline{\xi_k(x)}dx}{||\xi_k||_2^2} \tag{3}$$

This is completely analogous to a vector change-of-coordinates,[*] where $\vec{f}$ can be represented in terms of a new orthogonal basis $(\vec{\xi}_0, \vec{\xi}_1)$ instead of axis-aligned unit vectors $(\vec{e}_0, \vec{e}_1)$ via $\vec{f} = \frac{\langle \vec{\xi}_0, \vec{f} \rangle}{||\vec{\xi}_0||_2^2}\vec{\xi}_0 + \frac{\langle \vec{\xi}_1, \vec{f} \rangle}{||\vec{\xi}_1||_2^2}\vec{\xi}_1$:

because:

$$\langle \vec{\xi}_k, \vec{f} - \text{proj}_{\vec{\xi}_k}\vec{f} \rangle = 0 \quad \text{due to orthogonality}$$

$$\langle \vec{\xi}_k, \vec{f} - c_k\vec{\xi}_k \rangle = 0 \quad \text{because } \text{proj}_{\vec{\xi}_k}\vec{f} = c_k\vec{\xi}_k$$

$$\langle \vec{\xi}_k, \vec{f} \rangle - c_k\langle \vec{\xi}_k, \vec{\xi}_k \rangle = 0 \quad \text{by linearity}$$

$$\rightarrow c_k = \frac{\langle \vec{\xi}_k, \vec{f} \rangle}{\langle \vec{\xi}_k, \vec{\xi}_k \rangle}$$

The set of numbers $\{c_k\}$ is now *an alternative representation* of the original function. In some sense it's equally descriptive, so long as we know which basis we're using to reconstruct. The function has been *transformed*.

The $\{c_k\}$ are often said to live in another "domain", although we have to be careful with this terminology, because it technically refers to a "connected" set, not just a collection of M things. To be precise, some authors instead use "series" to describe $\{c_k\}$. However, it is possible for members of the basis set to be related through a continuous parameter which in some sense makes the set dense, even in cases where we only take discrete members of this more general set to be our basis set for a particular scenario. This is the case for the Fourier basis, where we choose $\omega \in \mathbb{R}$, and hence ω really can become a new domain.

---

[*]We are working with functions that live in L^2 space, L for Lebesgue, meaning functions which are square integrable on the domain, so the inner product integrals converge. Essentially, an element of this space is akin to a vector packed infinitely densely with values, until you get a function on a continuum. This is the only L^p space which is also a Hilbert space, i.e. has a nice inner product and "induced norm" from that inner product which gives nice notions of distances analogous to standard Euclidean distance.

2.1 The Fourier Transform

Using Fourier's original real-sinusoid-based formulation, we can write the reconstruction expression as[†]:

$$f(x) = a_0 + \sum_{k=1}^{\infty}(a_k \cos(k\omega_0 x) + b_k \sin(k\omega_0 x))$$

where
- f is periodic with fundamental frequency ω_0, so the kth frequency becomes $k \cdot \omega_0$.
- a_k and b_k are coefficients describing how much cosine and sine to add in, respectively.
- k goes up to ∞ because in general we need an infinite number of ever-higher-frequency sinusoids to reconstruct the function with perfect fidelity.

Let's now use $\cos(x) = \frac{e^{jx}+e^{-jx}}{2}$ and $\sin(x) = \frac{e^{jx}-e^{-jx}}{2j}$, which can be verified by manipulating Euler's formula, Equation 1.

$$f(x) = a_0 + \sum_{k=1}^{\infty}(a_k \frac{e^{jk\omega_0 x} + e^{-jk\omega_0 x}}{2} + b_k \frac{e^{jk\omega_0 x} - e^{-jk\omega_0 x}}{2j})$$

$$= a_0 + \sum_{k=-\infty}^{-1}(\frac{a_{-k}}{2} - \frac{b_{-k}}{2j})e^{jk\omega_0 x} + \sum_{k=1}^{\infty}(\frac{a_k}{2} + \frac{b_k}{2j})e^{jk\omega_0 x} = \sum_{k=-\infty}^{\infty} c_k e^{jk\omega_0 x}$$

So if we choose $c_0 = a_0$ and $c_k = \overline{c_{-k}} = \frac{a_k}{2} + \frac{b_k}{2j}$, then the complex exponential formulation is exactly equivalent to the trigonometric formulation[9]. That is, we can evaluate Equation 3 using the *complex* exponentials as basis functions to find *complex* c_k such that when multiplied in the reconstruction, Equation 2, we get back only *real* signal! Essentially, the relative balance of real and complex in c_k affects how much cosine and sine are present at the kth frequency, thereby accomplishing a phase shift[10]. Without accounting for phase shifts, we would only be able to model *symmetric* signals!

If instead of a fundamental frequency $\omega_0 = \frac{2\pi}{T}$, where T is a period of repetition, the signal contains dense frequencies (because it has no repetition, $T \to \infty$, $\omega_0 \to 0$), and if we care about a domain over the whole of $\mathbb{R}$, then it makes more sense to express both the basis functions and transformed coefficients as functions in ω and to make both our inner product and reconstruction expression integrals from $-\infty$ to $+\infty$:

$$\text{inner product with } \xi: \quad \hat{f}(\omega) = \int_{-\infty}^{\infty} f(x)e^{-j\omega x}dx = \mathcal{F}\{f(x)\}$$

$$\text{reconstruction}: \quad f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega)e^{j\omega x}d\omega = \mathcal{F}^{-1}\{\hat{f}(\omega)\}$$

(4)

where the hat $\hat{o}$ represents a function in the Fourier domain, and the $\frac{1}{2\pi}$ is a scaling factor that corrects for the fact $\hat{f}$ hasn't been normalized like the $\{c_k\}$ in the discrete case. Here, rather than divide by $||e^{j\omega x}||_2^2$, which diverges as the domain edges $\to \pm\infty$, we consider the domain $[\frac{-T}{2}, \frac{T}{2}]$ and let normalizing factor $T = \frac{2\pi}{\omega_0} \to \infty = \frac{2\pi}{d\omega}$[4].

Just like the $\{c_k\}$, $\hat{f}(\omega)$ can be complex, but if the original $f(x)$ is real, then $\hat{f}$'s complexity will perfectly interact with the complex exponentials to produce only a real function in the reconstruction.

2.2 A Whole Family

Part of what makes Fourier transforms confusing is the proliferation of different variants for different situations, so it's worth categorizing them[11]. First off, are we dealing with a periodic signal (which has an ω_0) or an aperiodic signal (which doesn't)? And second, are we dealing with a continuous function or discrete samples? Here is a diagram:

[†]It's worth considering how weird it is this works to express arbitrary functions, even non-smooth ones (so long as they meet the Dirichlet conditions[4], i.e. aren't pathological cases), a fact so counter-intuitive that Joseph Lagrange publicly declared Fourier was wrong at a meeting of the Paris Academy in 1807[6] and rejected Fourier's paper, which then went unpublished until after Lagrange died![4] It's valuable to ask why this works[7] and sift through some analysis[8].
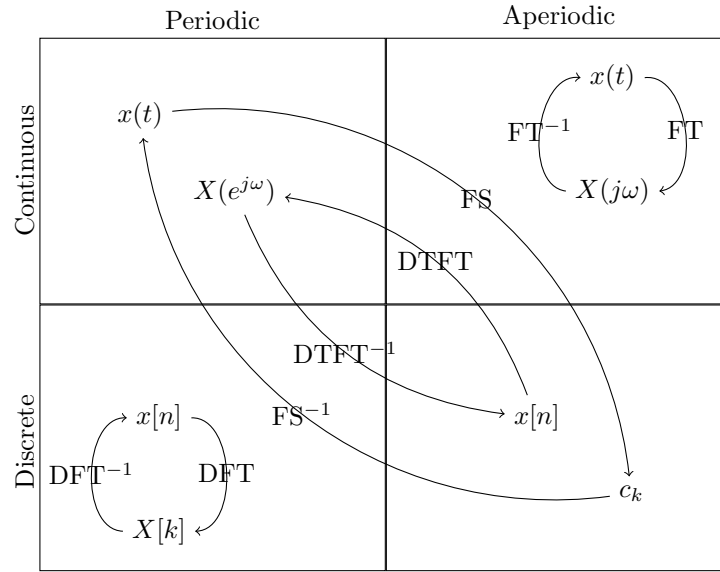
Note that, following a more signal-processing-ish convention[4], the function we're transforming is now called x, and the independent variable, since it can no longer be x, is named t. For discrete signals, we use independent variable n in square brackets.

Here FS stands for "Fourier Series", which is the first situation covered above. FT stands for "Fourier Transform", which is given by the integral pair, Equation 4. But these are not the only possibilities! DTFT stands for "Discrete Time Fourier Transform", where the signal we want to analyze is discrete but the transform is continuous. And finally DFT stands for "Discrete Fourier Transform", not to be confused with the DTFT, which we use when *both* the original and transformed signals are sampled.

All of these can be considered Fourier transforms, but often when people talk about *the* canonical "Fourier Transform", they are referring to the continuous, aperiodic case in the upper righthand cell.

The notation of all these different functions and transforms is easy to mix up and made all the more confusing by the reuse of symbols. But it's important to keep straight which situation we're in. I can only apologize. For more on all these, see [4].

3   Taking Derivatives in the Fourier Domain

Let's take a Fourier transform of the derivative of a function[12]:

$$\mathcal{F}\{\frac{d}{dx}f(x)\} = \int\limits_{-\infty}^{\infty} \underbrace{\frac{df}{dx}}_{dv}\underbrace{e^{-j\omega x}}_{u}dx = \underbrace{f(x)e^{-j\omega x}\Big|_{-\infty}^{\infty}}_{\substack{0 \text{ for Lebesgue-}\\ \text{integrable}\\ \text{functions}}} - \int\limits_{-\infty}^{\infty} f(x)(-j\omega)e^{-j\omega x}dx = j\omega \cdot \hat{f}(\omega)$$

We can use the inverse transform equation to see the same thing:

$$\frac{d}{dx}f(x) = \frac{d}{dx}\frac{1}{2\pi}\int\limits_{-\infty}^{\infty}\hat{f}(\omega)e^{j\omega x}d\omega = \frac{1}{2\pi}\int\limits_{-\infty}^{\infty}\hat{f}(\omega)\frac{d}{dx}e^{j\omega x}d\omega = \mathcal{F}^{-1}\{j\omega \cdot \hat{f}(\omega)\}$$

So a derivative in the x domain can be accomplished by a *multiplication* in the frequency domain. We can raise to higher derivatives simply by multiplying by $j\omega$ more times.

This is great because taking derivatives in the spatial domain is actually pretty hard, especially if we're working with discrete samples of a signal, whereas taking the derivative this way in the frequency domain, the *spectral derivative*, gives us much better fidelity.[13, 14] The cost is that we have to do a Fourier transform and inverse Fourier transform to sandwich the actual differentiation, but there is an $O(N \log N)$ algorithm to accomplish the DFT (subsection 2.2 and Equation 5) for discrete signals called the Cooley-Tukey algorithm, also known as the Fast Fourier Transform (FFT)[14].

3.1 Taking Derivatives in the Discrete Case

Because we're going to want to use a computer, and a computer can only operate on discrete representations, we really need to talk about the DFT and what it means to take a derivative in this discrete paradigm. It has a connection to the above continuous case but is far more subtle, worth going in to *at some length*.

3.1.1 The DFT Pair

$$\text{DFT:} \quad Y_k = \sum_{n=0}^{M-1} y_n e^{-j\frac{2\pi}{M}nk}$$

$$\text{DFT}^{-1}: \quad y_n = \frac{1}{M} \sum_{k=0}^{M-1} Y_k e^{j\frac{2\pi}{M}nk}$$

(5)

where
- n iterates samples in the original domain (often spatial)
- k iterates samples in the frequency domain (wavenumbers)
- M is the number of samples in the signal, often given as N by other sources[15], but I'll use N for something else later and want to be consistent
- y denotes the signal in its original domain
- Y denotes the signal in the frequency domain

We can express this as the linear inverse problem:

$$\frac{1}{M} \begin{bmatrix} e^{j\frac{2\pi}{M}00} & e^{j\frac{2\pi}{M}01} & \cdots & e^{j\frac{2\pi}{M}0(M-1)} \\ e^{j\frac{2\pi}{M}10} & e^{j\frac{2\pi}{M}11} & \cdots & e^{j\frac{2\pi}{M}1(M-1)} \\ \vdots & \vdots & \ddots & \vdots \\ e^{j\frac{2\pi}{M}(M-1)0} & e^{j\frac{2\pi}{M}(M-1)1} & \cdots & e^{j\frac{2\pi}{M}(M-1)(M-1)} \end{bmatrix} \begin{bmatrix} Y_0 \\ Y_1 \\ \vdots \\ Y_{M-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{M-1} \end{bmatrix}$$

which, thanks the special structure of its matrix, can be solved by a divide-and-conquer strategy that recursively builds the solution to a larger problem out of smaller ones (FFT)[14].

For simplicity, we can collect $\frac{2\pi}{M}n$ as a single term, $\theta_n \in [0, 2\pi)$, or $\frac{2\pi}{M}k$ as a single term, ω_k. We then get $y_n = y(\theta_n)$ and $Y_k = Y(\omega_k)$. This may help highlight the fact the original signal and transformed signal live on a domain which maps to the unit circle[16] (hence periodicity and aliasing) and are being sampled at equally-spaced angles/angular velocities.

3.1.2 Interpolation

I now quote Steven Johnson[17], with some of my own symbols and notation sprinkled in:

> "In order to compute derivatives like $y'(\theta)$, we need to do more than express y_n. We need to use the DFT^{-1} expression to define a continuous interpolation between the samples y_n—this is called *trigonometric interpolation*—and then differentiate this interpolation. At first glance, interpolating seems very straightforward: one simply evaluates the DFT^{-1} expression at non-integer $n \in \mathbb{R}$. This indeed defines *an* interpolation, but it is not the *only* interpolation, nor is it the *best* interpolation for this purpose. The reason there is more than one interpolation is due to *aliasing*: any term $e^{+j\theta_n k}Y_k$ in the DFT^{-1} can be replaced by $e^{+j\theta_n(k+mM)}Y_k$ for any integer m and still give the *same* samples y_n, since $e^{j\frac{2\pi}{M}nmM} = e^{j2\pi nm} = 1$ for any integers m and n. Essentially, adding the mM term to k means that the interpolated function $y(\theta)$ just oscillates m extra times between the sample points, which has no effect on y_n but has a huge effect on derivatives. To resolve this ambiguity, one imposes additional criteria—e.g. a bandlimited spectrum and/or minimizing some derivative of the interpolated $y(\theta)$"

We can now posit a slightly more general formula for the underlying continuous, periodic (over interval length M) signal:

$$y(\theta) = \frac{1}{M} \sum_{k=0}^{M-1} Y_k e^{j\theta(k+m_k M)}, \quad m_k \in \mathbb{Z}$$

"In order to uniquely determine the m_k, a useful criterion is that we wish to *oscillate as little as possible* between the sample points y_n. One way to express this idea is to assume that $y(\theta)$ is *bandlimited* to frequences $|k + m_k M| \leq \frac{M}{2}$. Another approach, that gives the same result ... is to *minimize the mean-square slope*"[‡]

$$\frac{1}{2\pi}\int_0^{2\pi} |y'(\theta)|^2 d\theta = \frac{1}{2\pi}\int_0^{2\pi}\left|\frac{1}{M}\sum_{k=0}^{M-1} j(k+m_k M)Y_k e^{j\theta(k+m_k M)}\right|^2 d\theta$$

$$= \frac{1}{2\pi M^2}\int_0^{2\pi}\left(\sum_{k=0}^{M-1} j(k+m_k M)Y_k e^{j\theta(k+m_k M)}\right)\overline{\left(\sum_{k=0}^{M-1} j(k+m_k M)Y_k e^{j\theta(k+m_k M)}\right)} d\theta$$

$$= \frac{1}{2\pi M^2}\int_0^{2\pi}\sum_{k=0}^{M-1}\sum_{k'=0}^{M-1}\left(j(k+m_k M)Y_k e^{j\theta(k+m_k M)}\right)\overline{\left(j(k'+m_{k'} M)Y_{k'} e^{j\theta(k'+m_{k'} M)}\right)} d\theta$$

$$= \frac{1}{M^2}\sum_{k=0}^{M-1}\sum_{k'=0}^{M-1}(k+m_k M)\overline{(k'+m_{k'} M)}Y_k\overline{Y_{k'}}\underbrace{\frac{1}{2\pi}\int_0^{2\pi} e^{j\theta(k+m_k M)}e^{-j\theta(k'+m_{k'} M)} d\theta}_{=\begin{cases} 0 & \text{if } k+m_k M \neq k'+m_{k'}M \\ & \iff k \neq k' \text{ for } 0 \leq k, k' < M \\ 1 & \text{if } k = k' \end{cases}}$$

$$= \frac{1}{M^2}\sum_{k=0}^{M-1}|Y_k|^2(k+m_k M)^2$$

We now seek to minimize this by choosing m_k for each k. Only the last term depends on m_k, so it's sufficient to minimize only this:

$$\begin{aligned}\underset{m_k}{\text{minimize}}\quad & (k+m_k M)^2 \\ \text{s.t.}\quad & 0 \leq k < M \\ & m_k \in \mathbb{Z}\end{aligned}$$

This problem actually admits of good ol' calculus plus some common sense:

$$\frac{d}{dm_k}(k+m_k M)^2 = 2(k+m_k M)M = 0 \longrightarrow m_k^* = \frac{-k}{M} \in (-1, 0]$$

where * denotes optimality. But we additionally need to choose $m_k \in \mathbb{Z}$. Let's plot it to see what's going on.
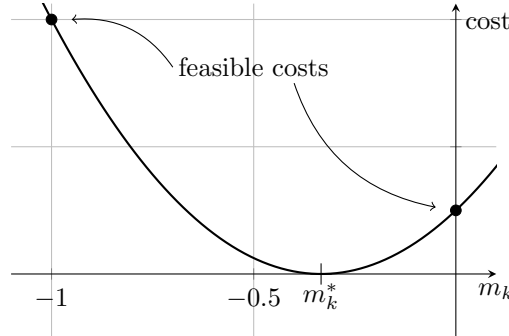
As we change the values of M and k, the parabola shifts around, getting taller for larger M and shifting leftward as $k \to M$.

We can see that for $k \in [0, \frac{M}{2})$, the $m_k = 0$ solution is lower down the cost curve, and for $k \in (\frac{M}{2}, M)$, the $m_k = -1$ solution is more optimal. "If $k = \frac{M}{2}$ (for even M), however, there is an ambiguity: either $m_k = 0$ or -1 gives the

---

[‡]It's due to this ambiguity and constraint that spectral methods are only suitable for smooth functions!

same value $(k + m_k M)^2 = (\frac{M}{2})^2$. For this $Y_{M/2}$ term (the "Nyquist" term), we can arbitrarily split up the $Y_{M/2}$ term between $m = 0$ [$j\frac{M}{2}\theta$, positive frequency] and $m = -1$ [$j(\frac{M}{2} - M)\theta = -j\frac{M}{2}\theta$, negative frequency]:"

$$Y_{M/2}(ue^{j\frac{M}{2}\theta} + (1-u)e^{-j\frac{M}{2}\theta})$$

where $u \in \mathbb{C}$ s.t. at sample points θ_n we get $Y_{M/2}(ue^{j\frac{M}{2}\frac{2\pi}{M}n} + (1-u)e^{-j\frac{M}{2}\frac{2\pi}{M}n}) = Y_{M/2}(u\overbrace{e^{j\pi n}}^{(-1)^n} + (1-u)\overbrace{e^{-j\pi n}}^{(-1)^n}) = Y_{M/2}(-1)^n$ "and so recover the DFT^{-1}."

If we use the above in the mean-squared slope derivation instead of $Y_k e^{j\theta(k + m_k M)}$ and $Y_{k'} e^{j\theta(k' + m_{k'} M)}$, then the integral portion becomes:

$$Y_{M/2}\overline{Y_{M/2}}\frac{1}{2\pi}\int_0^{2\pi}(ue^{j\frac{M}{2}\theta} + (1-u)e^{-j\frac{M}{2}\theta})\overline{(ue^{j\frac{M}{2}\theta} + (1-u)e^{-j\frac{M}{2}\theta})}d\theta$$

$$= |Y_{M/2}|^2\frac{1}{2\pi}\left(u\bar{u}\int_0^{2\pi}\underbrace{e^{j\frac{M}{2}\theta}e^{-j\frac{M}{2}\theta}}_{=1}d\theta + u\overline{(1-u)}\int_0^{2\pi}\underbrace{e^{j\frac{M}{2}\theta}e^{j\frac{M}{2}\theta}}_{\text{periodic!}}d\theta + (1-u)\bar{u}\int_0^{2\pi}\underbrace{e^{-j\frac{M}{2}\theta}e^{-j\frac{M}{2}\theta}}_{\text{periodic!}}d\theta\right.$$

$$\left. + (1-u)\overline{(1-u)}\int_0^{2\pi}\underbrace{e^{-j\frac{M}{2}\theta}e^{j\frac{M}{2}\theta}}_{=1}d\theta\right)$$

$$= |Y_{M/2}|^2\frac{1}{2\pi}(|u|^2 2\pi + |1-u|^2 2\pi) = |Y_{M/2}|^2(|u|^2 + |1-u|^2)$$

because integrating something periodic over a multiple of its period yields its mean, which is 0 in this case.

We now know that the contribution to the mean-squared slope from the $\frac{M}{2}^{th}$ term $\propto |u|^2 + |1-u|^2$. What's the optimal u?

$$\frac{d}{du}|u|^2 + |1-u|^2 = 2u - 2(1-u) = 0 \longrightarrow u = \frac{1}{2}$$

So "the $Y_{M/2}$ term should be *equally split* between the frequencies $\pm\frac{M}{2}\theta$, giving a $\cos(\frac{M}{2}\theta)$ term." Note that if M is odd, there is no troublesome $\frac{M}{2}$ term like this, but later we'll use the Discrete Cosine Transform[20] type I (DCT-I), which is equivalent to the FFT with even M and $Y_k = Y_{M-k}$, so we *do* have to worry about the Nyquist term.

Now if we put it all together we get "the **unique "minimal-oscillation" trigonometric interpolation** of order M":

$$y(\theta) = \frac{1}{M}\left(Y_0 + \sum_{0 < k < \frac{M}{2}}\left(Y_k e^{jk\theta} + Y_{M-k}e^{-jk\theta}\right) + Y_{M/2}\cos(\frac{M}{2}\theta)\right) \tag{6}$$

"As a useful side effect, this choice of trigonometric interpolation has the property that real-valued samples y_n (for which Y_0 is real and $Y_{M-k} = \overline{Y_k}$) will result in a purely real-valued interpolation $y(\theta)$ for all θ."

3.1.3 Taking Derivatives of the Interpolant

Now at last, with this interpolation between integer n in hand, we can take a derivative w.r.t. the spatial variable:

$$\frac{d}{d\theta}y(\theta) = \frac{1}{M}\left(\sum_{0 < k < \frac{M}{2}}jk(Y_k e^{jk\theta} - Y_{M-k}e^{-jk\theta}) - \frac{M}{2}Y_{M/2}\sin(\frac{M}{2}\theta)\right)$$

Evaluating at $\theta_n = \frac{2\pi}{M}n, n \in \mathbb{Z}$, we get:

$$y_n' = \frac{1}{M}\left(\sum_{0 < k < \frac{M}{2}}jk(Y_k e^{jk\frac{2\pi}{M}n} - Y_{M-k}e^{-jk\frac{2\pi}{M}n}) - \frac{M}{2}\overbrace{Y_{M/2}\sin(\pi n)}^{0}\right) = \frac{1}{M}\sum_{k=0}^{M-1}Y_k'e^{j\frac{2\pi}{M}kn}$$

$$\text{where} \quad Y_k' = \begin{cases} jk \cdot Y_k & k < \frac{M}{2} \\ 0 & k = \frac{M}{2} \\ j(k-M) \cdot Y_k & k > \frac{M}{2} \end{cases} \leftarrow \text{comes from: } k_{new} = M - k_{old}, 0 < k_{old} < \frac{M}{2}$$

$$\rightarrow \frac{M}{2} < k_{new} < M; -jk_{old} \cdot Y_{M-k_{old}} \rightarrow -j(M - k_{new}) \cdot Y_{k_{new}}$$

Easy! Now let's do the second derivative:

$$\frac{d^2}{d\theta^2} y(\theta) = \frac{1}{M} \Big(\sum_{0 < k < \frac{M}{2}} (jk)^2 (Y_k e^{jk \frac{2\pi}{M} n} + Y_{M-k} e^{-jk \frac{2\pi}{M} n}) - \Big(\frac{M}{2}\Big)^2 Y_{M/2} \cos(\frac{M}{2}\theta) \Big)$$

And again evaluating at $\theta_n = \frac{2\pi}{M} n, n \in \mathbb{Z}$:

$$y_n'' = \frac{1}{M} \Big(\sum_{0 < k < \frac{M}{2}} (jk)^2 (Y_k e^{jk \frac{2\pi}{M} n} + Y_{M-k} e^{-jk \frac{2\pi}{M} n}) - \Big(\frac{M}{2}\Big)^2 Y_{M/2}(-1)^n \Big) = \frac{1}{M} \sum_{k=0}^{M-1} Y_k'' e^{j \frac{2\pi}{M} kn}$$

$$\text{where} \quad Y_k'' = \begin{cases} (jk)^2 \cdot Y_k & k < \frac{M}{2} \\ \Big(j\frac{M}{2}\Big)^2 \cdot Y_k & k = \frac{M}{2} \\ (j(k - M))^2 \cdot Y_k & k > \frac{M}{2} \end{cases} \quad \text{or equivalently} \quad Y_k'' = \begin{cases} (jk)^2 \cdot Y_k & k \leq \frac{M}{2} \\ (j(k - M))^2 \cdot Y_k & k > \frac{M}{2} \end{cases}$$

It's important to realize "this [second derivative] procedure is *not* equivalent to performing the spectral first-derivative procedure twice (unless M is odd so that there is no $Y_{M/2}$ term) because the first derivative operation omits the $Y_{M/2}$ term entirely."[17]

We can repeat for higher derivatives, but the punchline is that for odd derivatives the $\frac{M}{2}$ term goes away,[§] and for even derivatives it comes back. In general:

$$Y_k^{(\nu)} = \begin{cases} (jk)^\nu \cdot Y_k & k < \frac{M}{2} \\ (j\frac{M}{2})^\nu \cdot Y_k & k = \frac{M}{2} \text{ and } \nu \text{ even} \\ 0 & k = \frac{M}{2} \text{ and } \nu \text{ odd} \\ (j(k - M))^\nu \cdot Y_k & k > \frac{M}{2} \end{cases} \tag{7}$$

This has definite echoes of the standardly-given, continuous-time case covered in section 3, but it's emphatically *not* as simple as just multiplying by $j\omega$ or even by jk. However, the final answer is thankfully super compact to represent in math and in code.

3.2 Limitations

So far it has all been good news, but there is a serious caveat to using the Fourier basis, especially for derivatives.

Although a Fourier transform tends to have more "mass" at lower frequencies and fall off as we go to higher ones (otherwise the reconstruction integral would diverge), and therefore we can get really great reconstructions by leaving off higher modes[14] (or equivalently only sampling and transforming M components), we in fact need *all* the infinite modes to reconstruct an *arbitrary* signal[4]. Even then, the Fourier basis can not represent true discontinuities nor non-smooth corners, instead converging "almost everywhere", which is math speak for the "measure" or volume of the set where it doesn't work being 0, meaning it only doesn't work *at* the discontinuities or corners themselves.[4]

If there are discontinuities or corners, we get what's called the Gibbs Phenomenon[4] (see next figure), essentially overshoot as the set of basis functions tries to fit a sudden change. These extra wiggles are bad news for function approximation but even worse news for taking derivatives: If we end up on one of those oscillations, the slope might wildly disagree with that of the true function!

This is a bigger problem than it may first appear, because when we do this on a computer, we're using the DFT, which implicitly periodically extends the function (subsection 2.2, Equation 5). So we not only need the function to have no jumps or corners internal to its domain; we need it to match up smoothly *at the edges* of its domain too!

This rules out the above spectral method for all but "periodic boundary conditions"[14]. But if the story ended right there, I wouldn't have thought it worth building this package.

[§]For *real* signals it is common (e.g. in the kind of code ChatGPT might generate to do this) not to worry about zeroing out the Nyquist term and *instead* throw away the imaginary part of the inverse transform. This works because $Y_{M/2} = \sum_{n=0}^{M-1} y_n e^{-j \frac{2\pi}{M} n \frac{M}{2}} = \sum_{n=0}^{M-1} y_n (-1)^n$, which will be purely real for real y_n, and when we mulitply by $(jk)^\nu$ for odd ν, then $Y_{M/2}^{(\nu)}$ gets a constituent odd power of j, which makes it purely imaginary. Then its contribution to the inverse transform (i.e. to each sample of the derivative $y_n^{(\nu)}$) is $+Y_{M/2}^{(\nu)} e^{j \frac{2\pi}{M} n \frac{M}{2}} = +Y_{M/2}^{(\nu)}(-1)^n$, which will also be purely imaginary. Whereas other imaginary components in the transform of a real signal have negated twins at negative frequency to pair with and become *real* sines in the inverse transform, the Nyquist term has no twin, so it's the *only* imaginary thing left over. Thus for *real* signals keeping only the `ifft().real` is *equivalent* to zeroing out the Nyquist term.
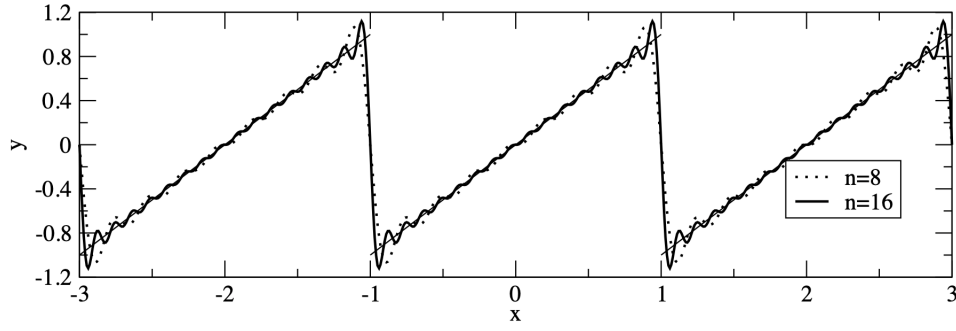
An example of the Gibbs phenomenon, from [14]. Using more modes allows the solution to fit better in the middle of the domain but makes peaks at the edges more extreme.

4   The Chebyshev Basis

There is another basis which we can use to represent arbitrary functions, called the Chebyshev polynomials[18], which has a really neat relationship to the Fourier basis.
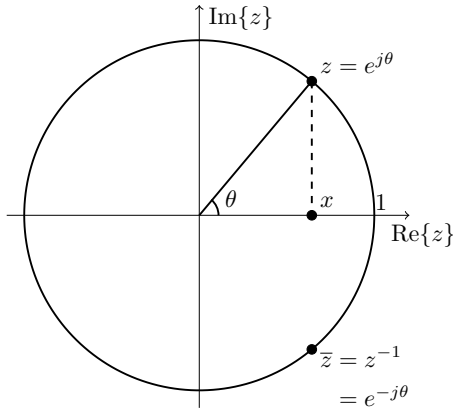
$$\text{Let} \quad x \in [-1, 1] \qquad \text{Chebyshev}$$
$$= \cos(\theta), \ \theta \in [0, \pi] \qquad \text{Fourier} \qquad (8)$$
$$= \frac{1}{2}(z + z^{-1}), \ |z| = 1 \qquad \text{Laurent}$$

The kth Chebyshev polynomial is defined as $T_k(x) = Re\{z^k\} = \frac{1}{2}(z^k + z^{-k}) = \cos(k\theta)$ by Euler formula:

$$T_0(x) = Re\{z^0\} = 1$$
$$T_1(x) = Re\{z^1\} = \frac{1}{2}(e^{j\theta} + e^{-j\theta}) = \cos(\theta) = x$$
$$T_2(x) = \frac{1}{2}(e^{j2\theta} + e^{-j2\theta}) = \cos(2\theta)$$
$$\text{but also } = \frac{1}{2}\underbrace{(z^2 + 2 + z^{-2})}_{\text{perfect square}} - 1 = \left(\sqrt{\frac{1}{2}}(z + z^{-1})\right)^2 - 1 = \underbrace{\left(\frac{2}{\sqrt{2}}\right)^2}_{2} \underbrace{\frac{z + z^{-1}}{2}}_{\cos(\theta)} - 1 = 2x^2 - 1$$
$$T_3(x) = \frac{1}{2}(e^{j3\theta} + e^{-j3\theta}) = \cos(3\theta)$$
$$\text{but also } = \frac{1}{2}(z + z^{-1})^3 - \frac{3}{2}(z + z^{-1}) = 4x^3 - 3x$$
$$\dots$$

It turns out there is a recurrent pattern:

$$T_{k+1} = \frac{1}{2}(z^{k+1} + z^{-(k+1)}) = \frac{1}{2}(z^k + z^{-k})(z + z^{-1}) - \frac{1}{2}(z^{k-1} + z^{-(k-1)}) = 2xT_k(x) - T_{k-1}(x)$$

Due to the relationship between θ and x on their respective domains, you can think of these polynomials as cosine waves "wrapped around a cylinder and viewed from the side."[18]

10
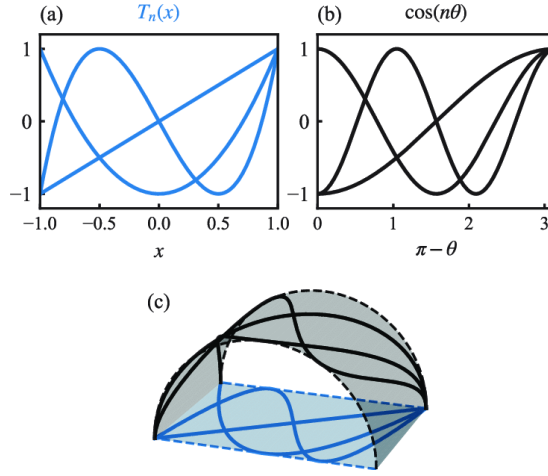
Relationship of Chebyshev domain and Fourier Domain, from [19]. Notice the cosines are horizontally flipped. The authors use n instead of k, which is common for Chebyshev polynomials (e.g. [18]), but I prefer k to enumerate basis modes, for consistency.

Essentially, on the domain $[-1, 1]$ each of these polynomials has ever more wiggles in the range $[-1, 1]$, and they perfectly coincide with the *shadows* of horizontally-reversed 2π-periodic cosines in the domain $[0, \pi]$. If we trace a function's value over $x = \cos(\theta)$ for linearly-increasing $\theta \in [0, \pi]$ instead of tracing it for linearly-decreasing $x \in [-1, 1]$, it's as if we're walking along the arc of the cylinder instead of along the shadow. We're effectively moving, horizontally flipping, and *warping* the function (by expanding near the edges and compressing in the middle) to a new θ domain.

We can reconstruct a function using the different variables/basis formulations, and as long as our variables are related as in Equation 8, these reconstructions are *equivalent*:

$$y(x) = \sum_{k=0}^{N} a_k T_k(x) \;;\quad y(z) = \sum_{k=0}^{N} a_k \frac{1}{2}(z^k + z^{-k}) \;;\quad y(\theta) = \sum_{k=0}^{N} a_k \cos(k\theta) \tag{9}$$

Note the set of $\{a_k\}$ is for $k \in \{0, ...N\}$ and therefore has cardinality $N + 1$.

4.1 The Advantage of Chebyshev

Why might we prefer this basis to the Fourier basis? Well, the advantage of a polynomial basis is we can avert the need for periodicity at the boundaries. Polynomial fits don't suffer the Gibbs Phenomenon, however they do suffer from the also-bad Runge Phenomenon[14]:
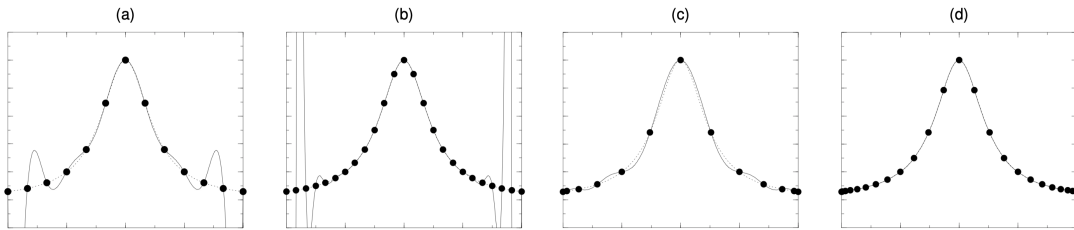
The Runge phenomenon, demonstrated in (a) and (b), mitigated in (c) and (d), from [14]

However, there is something we can do about the Runge phenomenon: By clustering fit points at the edges of the domain, the wild wobbles go away.

If we take $x_n = \cos(\theta_n)$ with θ_n equispaced, $n \in \{0, ...N\}$, then we get a very natural clustering at the edges of $[-1, 1]$. What's more, if we have equispaced θ_n and a reconstruction expression built up out of sinusoids, we're back in a Fourier paradigm (at least in variable θ) and can exploit the efficiency of the FFT, or, better, the discrete cosine and sine transforms![20, 21]

Notice too that the polynomials/projected cosines are *asymmetrical*, so we can natually use this basis to model arbitrary, lopsided functions without having to worry about phase shifts like we did for a Fourier basis of discrete harmonics.

5 Chebyshev Derivatives by Reduction to the Fourier Case

This all suggests a solution procedure:

Chebyshev Derivative via Fourier

1: Sample y at $\{x_n = \cos(\theta_n)\}$ rather than at equally spaced $\{x_n\}$, thereby warping the function over the arc of a cylinder.
2: Use the DCT to transform to frequency domain.
3: Multiply by appropriate $(jk)^\nu$ to accomplish differentiation in the θ domain.
4: Inverse transform using the DST if odd function, DCT if even function.
5: Change variables back from θ to x, the Chebyshev variable, taking care that this entails an extra chain rule.
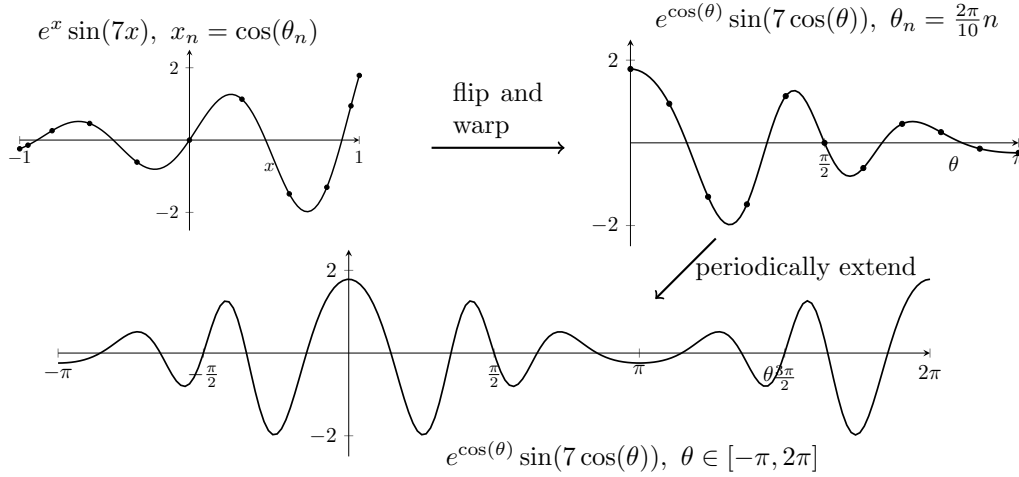

Illustration of implicit function manipulations in the first two steps of the algorithm. The edges of the aperiodic function can be made to match by "periodic extension", but this operation alone only fixes discontinuity; corners are created at 0 and π, resulting in Gibbs phenomenon when we frequency transform. Warping *stretches* corners into smooth transitions.

There are still a lot of details left to be worked out here, which we'll tackle in sequence.

5.1 The Discrete Cosine Transform

Because the reconstruction of $y(\theta)$ (Equation 9) only contains cosines, doing a full FFT/DFT, which tries to fit sines as well as cosines, would be doing extra work. Instead we can use the DCT.

There are actually several possible definitions of the DCT[20], based on different discrete periodic extensions. The DCT-II is the default in `scipy`[20], because it most closely resembles the Karhunen–Loève Transform[22], which is provably optimal to represent signals generated by a stationary (statistical properties not varying over time) Markov process (where values are related only to *immediate* previous values, through an autocorrelation coefficient). This class of signals is not exactly the same as the class of smooth functions, but it's similar in spirit. Thus we can often get better "energy compaction" with this basis and represent signals in fewer coefficients[23], especially when they have steep slopes at their boundaries. Indeed, common compression standards like JPEG choose to use the DCT-II.

However, we are dealing with *warped* functions with *flattened* edges, which are best made periodic by direct stacking-and-mirroring around the ends of the domain; they do not need any "wiggle room" between repeats of the endpoints. To achieve this with the DCT-II, we have to sample at "half-index" points, $\theta_{n,\text{II}} = \frac{1}{N+1}(n + \frac{1}{2})$, which do not go all the way to the the domain boundaries. But for compatibility with boundary value problems, we really want a sampling that includes the boundaries, like $\theta_n = \frac{\pi n}{N}$. The DCT-I implies the periodic extension we want with the sampling we want, so it is actually our best choice here, and it mercifully comes with the added benefit of being the least confusing variant to derive from the DFT:

Say we periodically extend y, i.e. stack $y(\theta)$, $\theta \in [0, \pi]$ next to a horizontal flip of itself on $(\pi, 2\pi)$, and then sample at the canonical DFT points, $\theta_n = \frac{2\pi}{M}n$, $n \in \{0, ...M - 1\}$ (Equation 5). We get:

$$\vec{y}_{\text{ext}} = [\underbrace{y_0, y_1, ...y_{N-1}, y_N, \underbrace{y_{N-1}, ...y_1}_{\substack{\text{redundant} \\ \text{information}}}}_{\substack{\text{original vector,} \\ \text{length N+1}}}], \text{ that is: } y_n = y_{M-n}, \ 0 \le n \le N$$

$$\underbrace{}_{\text{length M = 2N, necessarily even!}}$$

Then using $M - k$ for k in the DFT equation, we get:

$$\text{let } n_{new} = M - n_{old}$$

$$Y_{M-k} = \sum_{n=0}^{M-1} y_n e^{-j\frac{2\pi}{M}n(M-k)} = \sum_{n=0}^{M-1} y_n \underbrace{e^{-j2\pi n}}_{1} e^{j\frac{2\pi}{M}nk} \overset{\downarrow}{=} \sum_{n=M}^{1} y_{M-n} e^{j\frac{2\pi}{M}(M-n)k}$$

$$= \sum_{n=1}^{M} \underbrace{y_{M-n}}_{=y_n} \underbrace{e^{j2\pi k}}_{1} e^{-j\frac{2\pi}{M}nk} = \sum_{n=1}^{M} y_n e^{-j\frac{2\pi}{M}nk} = \sum_{n=0}^{M-1} y_n e^{-j\frac{2\pi}{M}nk} = Y_k \quad \square$$

$$\text{because } e^{-j\frac{2\pi}{M}Mk} = e^0 = 1 \text{ and } y_M = y_0$$

So when y_n are redundant this way, the Y_k are too, in a very mirror way. We can now use the facts $Y_k = Y_{M-k}$ and $N = \frac{M}{2}$ in the FFT interpolation (Equation 6):

$$y(\theta) = \frac{1}{M}\left(Y_0 + \sum_{0<k<\frac{M}{2}} \left(Y_k e^{jk\theta} + Y_{M-k}e^{-jk\theta}\right) + Y_{M/2}\cos(\frac{M}{2}\theta)\right)$$

$$= \frac{1}{M}\left(Y_0 + 2\sum_{k=1}^{N-1}\left(Y_k \underbrace{\frac{e^{jk\theta} + e^{-jk\theta}}{2}}_{\cos(k\theta)}\right) + Y_N\cos(N\theta)\right) \tag{10}$$

At samples $\theta_n = \frac{2\pi}{M}n = \frac{\pi}{N}n$, this becomes:

$$y(\theta_n) = \frac{1}{M}\left(Y_0 + Y_N\underbrace{\cos(\pi n)}_{(-1)^n} + 2\sum_{k=1}^{N-1} Y_k\cos(\frac{\pi nk}{N})\right) \tag{DCT-I^{-1}}$$

This is exactly the DCT-I^{-1}, which, except for the $\frac{1}{M}$ term and a flip of Y and y, is the same as the forward DCT-I! But the DCT and DCT^{-1} operate on the shorter set of $\vec{Y} = [Y_0, ...Y_N]$, without redundant information. Thus

$$\text{FFT}^{-1}(\underbrace{[Y_0, ...Y_N, Y_{N-1}, ...Y_1]}_{\vec{Y}_{\text{ext}}})[:N+1] = \text{DCT-I}^{-1}([Y_0, ...Y_N]) \tag{11}$$

Where $[:N+1]$ truncates to only the first $N+1$ elements ($\{0, ...N\}$). Given the equality above, we can line up everything we now know in a diagram:
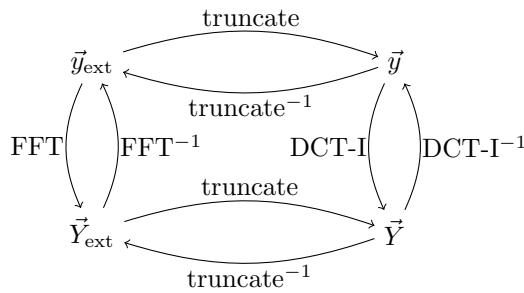
where truncate^{-1} does periodic extension, stacking back in redundant information

We can now easily see that in addition to the inverse relationship (Equation 11), we also have the forward relationship:

$$\text{FFT}([y_0, ...y_N, y_{N-1}, ...y_1])[:N+1] = \text{DCT-I}([y_0, ...y_N])$$

Notice that the 0th and N^{th} terms appear *outside the sum*, and that the sum *is multiplied by 2*. In our original conception of the cosine series for $y(\theta)$ (Equation 9), all the cosines appear equally within the sum, so our Y_k are subtly different from the a_k in that formulation (some scaled by a factor of $\frac{1}{2}$ and all scaled by M). Both are valid, but it's more computationally convenient to use the DCT-based formulation.

5.2 Even and Odd Derivatives and the Discrete Sine Transform

The DCT can get us in to the frequency domain, but we'll need the help of another transform to get back out. We again start with the DCT-I formulation for simplicity.

If we look at the full $\vec{Y}_{\text{ext}}$ (Equation 11), we have a palindromic structure around N, but also around 0, because of the repetitions[16], which ensure we can read the values of Y_k at negative k by wrapping around to the end of the vector. This is describing an *even* function, $f(-x) = f(x)$, which makes sense, because $y(\theta)$ is entirely composed of cosines, which are even functions, and because the forward transform is symmetrical with the inverse transform, the interpolation $Y(\omega)$ between Y_k is also ultimately a bunch of cosines.

The derivative of an even function is an *odd* function, $f(-x) = -f(x)$, which in principle should be constructable from purely sines, which are odd. And the derivative of an odd function is an even function again.

To see this more granularly, let's look in more detail at the multiplication by $(jk)^\nu$ that produces all the $Y_k^{(\nu)}$ (Equation 7), for $k \in \{0, ...M-1\}$:

$$\vec{Y}_{\text{ext}}^{(\nu)} = [0, j^\nu, ...(j(N-1))^\nu, \underbrace{(0 \text{ or } (jN)^\nu)}_{\substack{\text{depending on} \\ \nu \text{ odd or even}}}, (-j(N-1))^\nu, ...(-j)^\nu] \odot \vec{Y}_{\text{ext}}$$

$$= j^\nu \cdot \underbrace{[0, 1, ...1, (0 \text{ or } 1), -1, ..., -1]^\nu}_{\tilde{\mathbb{1}}} \odot \underbrace{[0, 1, ...N-1, N, N-1, ...1]^\nu}_{\text{palindromic}} \odot \vec{Y}_{\text{ext}}$$
$$\underset{\text{constant}}{\uparrow}$$

where $\odot$ is a Hadamard, or element-wise, product, and raising a vector to a power is also element-wise. We can see

$$\tilde{\mathbb{1}}^\nu = \begin{cases} [0, 1, ...1, 0, -1, ..., -1] & \text{if } \nu \text{ is odd} \\ [0, 1, ...1, 1, 1, ...1] & \text{if } \nu \text{ is even} \end{cases}$$

$[0, 1, ...1, 0, -1, ..., -1]$ is odd around entries 0 and N, and $[0, 1, ...1, 1, 1, ...1]$ is even around entry 0.

Let's now use this to reconstruct samples in the θ domain, $y_n^{(\nu)}$, for odd and even derivatives:

$$y_n^{(\text{odd } \nu)} = \frac{1}{M} \sum_{0 < k < \frac{M}{2}} (jk)^\nu (Y_k e^{jk\theta_n} \underset{\substack{\text{from odd-} \\ \text{ness of } \tilde{\mathbb{1}}^\nu}}{-} \underbrace{Y_{M-k}}_{=Y_k} e^{-jk\theta_n}) = \frac{1}{M} \sum_{k=1}^{N-1} (jk)^\nu Y_k \underbrace{(e^{jk\theta_n} - e^{-jk\theta_n})}_{2j \sin(k\theta_n)}$$

$$= \frac{1}{M} 2 \underbrace{\sum_{k=1}^{N-1} (jk)^\nu Y_k j \sin(\frac{\pi nk}{N})}_{= \text{ a DST-I of } \vec{Y}^{(\nu)} \cdot j!} \tag{12}$$

$$y_n^{(\text{even } \nu)} = \frac{1}{M} \Big(\sum_{0 < k < \frac{M}{2}} (jk)^\nu (Y_k e^{jk\theta_n} \underset{\substack{\text{from even-} \\ \text{ness of } \tilde{\mathbb{1}}^\nu}}{+} \underbrace{Y_{M-k}}_{=Y_k} e^{-jk\theta_n}) + (j\frac{M}{2})^\nu Y_{M/2} \cos(\frac{M}{2}\theta_n) \Big)$$

$$= \frac{1}{M} \Big((jN)^\nu Y_N \cos(\pi n) + \sum_{k=1}^{N-1} (jk)^\nu Y_k \underbrace{(e^{jk\theta_n} + e^{-jk\theta_n})}_{2\cos(k\theta_n)} \Big)$$

$$= \frac{1}{M} \underbrace{\Big((j0)^\nu Y_0 + (jN)^\nu Y_N (-1)^n + 2 \sum_{k=1}^{N-1} (jk)^\nu Y_k \cos(\frac{\pi nk}{N}) \Big)}_{= \text{ a DCT-I of } \vec{Y}^{(\nu)}!} \tag{13}$$

Brilliant! So we can use only the non-redundant $Y_k^{(\nu)}$ with a DST-I or DCT-I to convert odd and even functions, respectively, back to the θ domain!

Note that the DCT-I and DST-I definitions given in `scipy`[20, 21] use slightly different indexing than in my definitions here, which can be a point of confusion. I consistently take N to be the *index* of the last element of the

non-redundant y_n, not its length, following [18]. Note too that I consistently use n to index samples and k to index basis domain, whereas `scipy` uses n for the domain being transformed *from* and k for the domain being transformed *to*, which means these symbols are consistent with mine for forward transforms but flipped for inverse transforms.

Even more confusing, the DST-I only takes the $k \in \{1, ... N-1\}$ elements, since sines will result in zero crossings at $k = 0$ and N (no informational content), whereas the DCT-I takes all $k \in \{0, ... N\}$ elements!

5.3 Transforming Back to the Chebyshev Domain

At this point we've accomplished all but the last step of the algorithm, but we've been operating with $y_n = y(\theta_n)$ and $y_n^{(\nu)} = y^{(\nu)}(\theta_n)$, which are really samples from the θ domain, when what we really need to do is take derivatives in the $x = \cos(\theta)$ domain.

When we do this, we have to employ a chain rule, which introduces a new factor: the derivative of one of our variables w.r.t. the other. For the 1st derivative it looks like:

$$\frac{d}{dx}y(\theta) = \frac{d}{d\theta}y(\theta) \cdot \frac{d\theta}{dx} = y'(\theta) \cdot \frac{d}{dx}\cos^{-1}(x) = y'(\theta) \cdot \frac{-1}{\sqrt{1-x^2}}$$

The $y'(\theta)$ term is actually pretty easy to handle, because we know its value (and for higher orders too) at discretized θ_n from earlier! (Equations 12 & 13) If we use the sampled $x_n = \cos(\theta_n)$ from step 1 of the algorithm, then our $\{x_n\}$ and $\{\theta_n\}$ align, and we can find samples of the derivative w.r.t. x by plugging $\{x_n\}$ in to the new factor(s) and multiplying appropriately (pointwise):

$$[\frac{d}{dx}y(\theta)]_n = \frac{-1}{\sqrt{1-x_n^2}} \odot y_n'$$

5.3.1 Higher Derivatives and Splintering Terms

Let's see it for the second derivative:

$$\frac{d^2}{dx^2}y(\theta) = \frac{d}{dx}\frac{y'(\theta)}{-\sqrt{1-x^2}} = \frac{-\sqrt{1-x^2}\frac{d}{dx}y'(\theta) - y'(\theta)\frac{d}{dx}(-\sqrt{1-x^2})}{1-x^2}$$

$$= \frac{-\frac{d}{d\theta}y'(\theta)\cdot\frac{d\theta}{dx}}{\sqrt{1-x^2}} - \frac{y'(\theta)\frac{x}{\sqrt{1-x^2}}}{1-x^2} = \frac{y''(\theta)}{1-x^2} - \frac{xy'(\theta)}{(1-x^2)^{3/2}}$$

$$\longrightarrow [\frac{d^2}{dx^2}y(\theta)]_n = \frac{1}{1-x_n^2}\odot y_n'' - \frac{x_n}{(1-x_n^2)^{3/2}}\odot y_n'$$

Notice that the 2nd derivative in x requires *both* the 1st and 2nd derivatives in θ! This *splintering* phenomenon will be a considerable source of pain as we take higher derivatives: For the νth derivative in x we require all derivatives up to order ν in θ.

Let's see a few more:

$$\frac{d^3}{dx^3}y(\theta) = \frac{-1}{(1-x^2)^{3/2}}y'''(\theta) + \frac{3x}{(1-x^2)^2}y''(\theta) + \frac{-2x^2-1}{(1-x^2)^{5/2}}y'(\theta)$$

$$\frac{d^4}{dx^4}y(\theta) = \frac{1}{(1-x^2)^2}y^{IV}(\theta) + \frac{-6x}{(1-x^2)^{5/2}}y'''(\theta) + \frac{11x^2+4}{(1-x^2)^3}y''(\theta) + \frac{-6x^3-9x}{(1-x^2)^{7/2}}y'(\theta) \qquad (14)$$

$$\frac{d^5}{dx^5}y(\theta) = \frac{-1}{(1-x^2)^{5/2}}y^{V}(\theta) + \frac{10x}{(1-x^2)^3}y^{IV}(\theta) + \frac{-35x^2-10}{(1-x^2)^{7/2}}y'''(\theta) + \frac{50x^2+55x}{(1-x^2)^4}y''(\theta) + \frac{-24x^4-72x^2-9}{(1-x^2)^{9/2}}y'(\theta)$$

We can see a bit of a pattern here, though. In particular, the function of x multiplying each derivative of y in θ comes from at most two terms in the preceding derivative, which have a predictable form:

$$\frac{d}{dx}\left(\frac{p(x)}{(1-x^2)^{c-1}}y^{(\mu)}(\theta) + \frac{q(x)}{(1-x^2)^{c-\frac{1}{2}}}y^{(\mu-1)}(\theta)\right) =$$

$$\frac{p(x)}{(1-x^2)^{c-1}}y^{(\mu+1)}(\theta)\cdot\frac{d\theta}{dx} + \frac{(1-x^2)^{c-1}\frac{d}{dx}p(x) - p(x)(c-1)(1-x^2)^{c-2}(-2x)}{(1-x^2)^{2c-2}}y^{(\mu)}(\theta) +$$

$$\frac{q(x)}{(1-x^2)^{c-\frac{1}{2}}}y^{(\mu)}\cdot\frac{d\theta}{dx} + \frac{(1-x^2)^{c-\frac{1}{2}}\frac{d}{dx}q(x) - q(x)(c-\frac{1}{2})(1-x^2)^{c-\frac{3}{2}}(-2x)}{(1-x^2)^{2c-1}}y^{(\mu-1)}(\theta)$$

15

If we now gather the $y^{(\mu)}(\theta)$ terms and use the fact $\frac{d\theta}{dx} = \frac{-1}{\sqrt{1-x^2}}$, we can find its new multiplying factor is equal to:

$$\frac{(1-x^2)p'(x) + 2(c-1)xp(x) - q(x)}{(1-x^2)^c}$$

This relationship holds no matter which, μ, c, p, q we're addressing, which allows us to build up a kind of pyramid of terms:
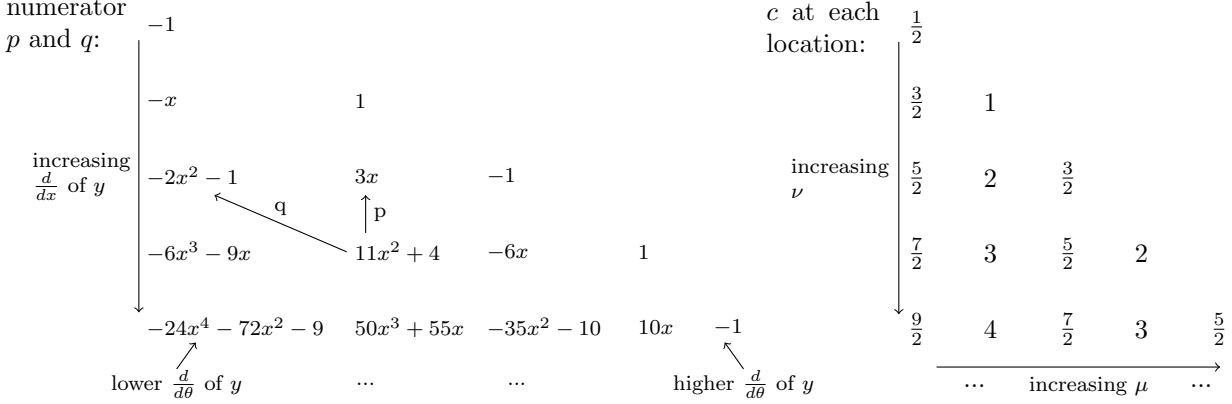
q always refers to the element up and to the left, and p always refers to the element above. If the arrows roll out of the pyramid, the corresponding p or q is 0. I've done the above programmatically in code, such that we can find and apply the factors—and thereby accomplish the variable transformation back to the Chebyshev domain—for arbitrarily high derivatives.

5.4 Handling Domain Endpoints

There's a problem in the above at the edges of the domain: If $x = \pm 1$, the denominators of all the factors, which are powers of $\sqrt{1-x^2} = 0$!

However, this doesn't mean $\frac{d^\nu}{dx^\nu}y$ can't have a valid *limit value* at those points. First remember our reconstruction $y(\theta)$ is composed of cosines, so notice if we take odd derivatives in θ, we get sines, and at the edges of the domain where $x \to \pm 1$, $\theta \to 0, \pi$, sine will be 0! However, if we take even derivatives, then $\cos(0, \pi) \to 1, -1$. Then, if we look closely at the derivatives in x (Equation 14), we can see that *even* derivatives in θ of y are divided by *even* powers of $\sqrt{1-x^2}$, and the *highest* power in a denominator is an *odd* power of $\sqrt{1-x^2}$. If we multiply through so everything is over the highest-power denominator and then combine the expression into a single fraction, we get a situation where the odd-derivative terms are 0 because sines, and the even-derivative terms are 0 because they're multiplied by at least one $\sqrt{1-x^2}$.

This means the *numerator* as well as the denominator is 0 at the domain endpoints. $\frac{0}{0}$ is an indeterminate form, so we can use L'Hôpital's rule!

Let's see it in fine detail for the 1st derivative. The below uses the DCT-I reconstruction (Equation 10) for $y(\theta)$.

$$\lim_{\substack{x \to \pm 1 \\ \theta \to 0, \pi}} \frac{d}{d\theta} \frac{1}{M}\left(Y_0 + Y_N \cos(N\theta) + 2\sum_{k=1}^{N-1} Y_k \cos(k\theta)\right) \cdot \frac{d\theta}{dx} = \lim_{\substack{x \to \pm 1 \\ \theta \to 0, \pi}} \frac{\frac{1}{M}\left(-NY_N \sin(N\theta) - 2\sum_{k=1}^{N-1} kY_k \sin(k\theta)\right)}{-\sqrt{1-x^2}}$$

$$\xrightarrow{\frac{d}{dx}} = \lim_{\substack{x \to \pm 1 \\ \theta \to 0, \pi}} \frac{\frac{1}{M}\left(-N^2 Y_N \cos(N\theta) - 2\sum_{k=1}^{N-1} k^2 Y_k \cos(k\theta)\right) \cdot \frac{-1}{\sqrt{1-x^2}}}{\frac{x}{\sqrt{1-x^2}}} = \lim_{\substack{x \to \pm 1 \\ \theta \to 0, \pi}} \frac{\frac{1}{M}\left(N^2 Y_N \cos(N\theta) + 2\sum_{k=1}^{N-1} k^2 Y_k \cos(k\theta)\right)}{x}$$

$$= \begin{cases} \frac{1}{M}\left(N^2 Y_N + 2\sum_{k=1}^{N-1} k^2 Y_k\right) & \text{at } x = 1, \theta = 0 \\ -\frac{1}{M}\left(N^2(-1)^N Y_N + 2\sum_{k=1}^{N-1} k^2(-1)^k Y_k\right) & \text{at } x = -1, \theta = \pi \end{cases} \qquad (1^{st} \text{ endpoints})$$

And now let's do it for the 2nd derivative, with some slightly more compact notation, where we can be agnostic about $y(\theta)$'s exact structure until the end:

$$\lim_{\substack{x\to\pm1\\\theta\to0,\pi}} \frac{\sqrt{1-x^2}\,y''(\theta)-xy'(\theta)}{(1-x^2)^{3/2}} \xrightarrow{\frac{0}{0}\;\frac{\frac{d}{dx}}{\frac{d}{dx}}} \frac{\sqrt{1-x^2}\,y'''(\theta)\frac{-1}{\sqrt{1-x^2}} + \frac{-x}{\sqrt{1-x^2}}y''(\theta) - \left(xy''(\theta)\frac{-1}{\sqrt{1-x^2}} + y'(\theta)\right)}{-3x\sqrt{1-x^2}}$$

$$\xrightarrow{\frac{0}{0}\;\frac{\frac{d}{dx}}{\frac{d}{dx}}} \frac{y^{IV}(\theta)\frac{-1}{\sqrt{1-x^2}} - y''(\theta)\frac{-1}{\sqrt{1-x^2}}}{\frac{6x^2-3}{\sqrt{1-x^2}}} = \frac{1}{6x^2-3}\left(y^{IV}(\theta)+y''(\theta)\right)$$

We already know $y''(\theta) = \frac{1}{M}\left(-N^2Y_N\cos(N\theta) - 2\sum_{k=1}^{N-1}k^2Y_k\cos(k\theta)\right)$, assuming type I reconstruction. We can easily find

$$y^{IV}(\theta) = \frac{1}{M}\left(N^4Y_N\cos(N\theta) + 2\sum_{k=1}^{N-1}k^4Y_k\cos(k\theta)\right)$$

Now we can evaluate these and the factor $\frac{1}{6x^2-3}$ at the limit values and put it all together to find:

$$\begin{cases} \frac{1}{3M}\left((N^4-N^2)Y_N + 2\sum_{k=1}^{N-1}(k^4-k^2)Y_k\right) & \text{at } x=1, \theta=0 \\ \frac{1}{3M}\left((N^4-N^2)(-1)^NY_N + 2\sum_{k=1}^{N-1}(k^4-k^2)(-1)^kY_k\right) & \text{at } x=-1, \theta=\pi \end{cases} \qquad (2^{nd}\text{ endpoints})$$

5.4.1 Endpoints for Higher Derivatives

We can do the above for higher derivatives too. However, in general finding the endpoints for the νth derivative involves ν applications of L'Hôpital's rule, slowly cancelling one power of $\sqrt{1-x^2}$ at a time after each. The algebra gets to be pretty gnarly.

But there is some hope: We can see a pattern like the pyramid scheme from earlier, because the functions multiplying each $y^{(\mu)}(\theta)$ in the numerator of the limit argument depend only on one or two terms from before the latest L'Hôpital. We can additionally use the relationship between variables $x=\cos(\theta)$ to recognize $\sqrt{1-x^2}=\sin(\theta)$ and substitute to put everything in terms of a single variable, and then just as well perform L'Hôpital's derivatives more simply w.r.t. θ rather than x and cancel a $\sin(\theta)$ rather than a $\sqrt{1-x^2}$.

When we proceed, the denominator eventually acquires a single standalone term of the form $D\cos^\nu(\theta)$, which at the domain endpoints 0 and π will be something nonzero, $D_0 = \pm D_\pi$, thereby ending our *journey*. At the same iteration, the numerator reduces to a set of constants, C, multiplying even-order θ-derivatives of $y(\theta)$ up to the $2\nu^{th}$. Putting it all together, the endpoint formulas can be found as:

$$\begin{cases} \frac{1}{D_0M}\left((\ldots - C_3N^6 + C_2N^4 - C_1N^2)Y_N + 2\sum_{k=1}^{N-1}(\ldots - C_3k^6 + C_2k^4 - C_1k^2)Y_k\right) & \text{at } x=1, \theta=0 \\ \frac{1}{D_\pi M}\left((\ldots - C_3N^6 + C_2N^4 - C_1N^2)(-1)^NY_N + 2\sum_{k=1}^{N-1}(\ldots - C_3k^6 + C_2k^4 - C_1k^2)(-1)^kY_k\right) & \text{at } x=-1, \theta=\pi \end{cases}$$

where the alternating plus and minus in the k and N terms comes from the fact the 2nd derivative contains $-$cosines, the 4th $+$cosines, the 6th $-$cosines again, and so on.

Because the act of cancellation and the functions containing powers of $\sqrt{1-x^2}=\sin(\theta)$ can't be easily represented in numpy, computing C and D requires a symbolic solver like sympy. I've devised an implementation to construct expressions for the endpoints, up to arbitrary order.

6 Chebyshev Derivatives via Series Recurrence

The above algorithm is a generalization of the method suggested by Trefethen[18], but it actually overcomplicates matters considerably. It is possible to sidestep all the ballooningly-difficult variable mappings and higher derivative limit-evaluations by exploiting a direct relationship[24, 25, 26] between a function's Chebyshev series and its derivative's Chebyshev series. This is analogous to how we can represent a function as a power series (a sum of integer powers of the independent variable) and find the coefficients of its derivative with Power Rule[27].

If we sandwich this key rule with the realization that the DCT is not only getting Fourier series coefficients of the warped function, but also (scaled) Chebyshev series coefficients of the original function (because these are exactly the same modulo constant factors!), and thus the DCT^{-1} can go from the Chebyshev series representation to a cosine-spaced sampling of a function, then we have the ingredients to craft an algorithm that stays entirely in the x domain:

Chebyshev Derivative via Series Rule

1: Sample y at $\{x_n = \cos(\theta_n)\}$ for equispaced $\{\theta_n\}$.
2: Use the DCT to get Chebyshev basis coefficients.
3: Use the Chebyshev series derivative rule to calculate the derivative's coefficients in $O(N)$.
4: Inverse transform with the DCT to resample the derivative function at $\{x_n\}$.

This algorithm turns out to be essentially numerically identical to the first, and due to its relative simplicity (no splinters!) and ease of extension to support non-cosine-spaced samples (albeit with considerably greater computational cost, $O(N^3)$ rather than $O(N \log N)$), it is the method of choice implemented in the main library code.

6.1 Chebyshev Series Derivative Rule

To most easily explain the rule, we'll need a little extra machinery: There are actually *two different kinds* of Chebyshev polynomials, and so far we've only been working with the first kind, denoted $T_k(x) = \cos(k\theta)$. The *second kind* is given as:

$$U_k(x) = \frac{\sin((k+1)\theta)}{\sin(\theta)}$$

Notice that:

$$\frac{d}{dx}T_k(x) = \frac{d}{d\theta}\cos(k\theta) \cdot \frac{d\theta}{dx} = k(\diagup\sin(k\theta)) \cdot \frac{\diagup 1}{\sqrt{1-x^2}} = \frac{k\sin(k\theta)}{\sin(\theta)} = k \cdot U_{k-1}(x)$$

But we really want the derivative in terms of T, not U. Lucky for us, there is a relationship between the two, based on trigonometric identities, making particular use of $\cos(\alpha)\sin(\beta) = \frac{1}{2}(\sin(\alpha+\beta) - \sin(\alpha-\beta))$:

$$2 \sum_{\substack{\text{odd } \kappa > 0}}^{\text{odd } k-1} \cos(\kappa\theta)\sin(\theta) = \diagup 2 \sum_{\substack{\text{odd } \kappa > 0}}^{\text{odd } k-1} \frac{\diagup 1}{\diagup 2}(\sin((\kappa+1)\theta) - \sin((\kappa-1)\theta)$$

$$= \sin(k\theta) - \overcancel{\sin((k-2)\theta)} + \overcancel{\sin((k-2)\theta)} - ... - \overcancel{\sin(2\theta)} + \overcancel{\sin(2\theta)} - \overset{0}{\overcancel{\sin(0\theta)}} = \sin(k\theta)$$

$$\longrightarrow \frac{\sin(k\theta)}{\sin(\theta)} = 2 \sum_{\substack{\text{odd } \kappa > 0}}^{k-1} \cos(\kappa\theta) \quad \text{for even } k$$

Similarly:

$$2 \sum_{\substack{\text{even } \kappa \geq 0}}^{\text{even } k-1} \cos(\kappa\theta)\sin(\theta) = \diagup 2 \sum_{\substack{\text{even } \kappa \geq 0}}^{\text{even } k-1} \frac{\diagup 1}{\diagup 2}(\sin((\kappa+1)\theta) - \sin((\kappa-1)\theta)$$

$$= \sin(k\theta) - \overcancel{\sin((k-2)\theta)} + \overcancel{\sin((k-2)\theta)} - ... - \overcancel{\sin(\theta)} + \overcancel{\sin(\theta)} - \underbrace{\sin(-\theta)}_{=+\sin(\theta)} = \sin(k\theta) + \sin(\theta)$$

$$\longrightarrow \frac{\sin(k\theta)}{\sin(\theta)} = -1 + 2 \sum_{\substack{\text{odd } \kappa > 0}}^{k-1} \cos(\kappa\theta) \quad \text{for odd } k$$

Thus:

$$k \cdot U_{k-1}(x) = \frac{d}{dx}T_k(x) = k \cdot \begin{cases} 2\sum_{\text{odd } \kappa > 0}^{k-1} T_\kappa(x) & \text{for even } k \\ -1 + 2\sum_{\text{even } \kappa \geq 0}^{k-1} T_\kappa(x) & \text{for odd } k \end{cases}$$

Let's see this on a couple examples to get a better intuition: In practice we represent a function with N Chebyshev series coefficients, stored low to high. If $N = 5$, then $T_3(x)$ would be $[0, 0, 0, 1, 0]$, and $T_4(x)$ would be $[0, 0, 0, 0, 1]$. If we differentiate these two, we should get $\frac{d}{dx}T_3(x) = 3 \cdot (-1 + 2 \cdot [1, 0, 1, 0, 0]) = [3, 0, 6, 0, 0]$, and $\frac{d}{dx}T_4(x) = 4 \cdot 2 \cdot [0, 1, 0, 1, 0] = [0, 8, 0, 8, 0]$. Notice the extra constant from the -1 in the first example is factored in to the coefficient of $T_0(x) = 1$.

Because differentiation is linear, we can scale and stack these particular results, e.g. $\frac{d}{dx}(2T_3(x) + T_4(x)) = 6T_0(x) + 8T_1(x) + 12T_2(x) + 8T_3(x)$. And because each term's derivative only affects every-other term of lower order, and these

effects are cumulative, it's possible to calculate the new sequence by starting at the higher-order end and working downwards, modifying only two numbers at each step[26]. An implementation of this procedure called chebder[24] lives in numpy.

7 Multidimensionality

We are now fully equipped to find derivatives for 1-dimensional data. This is technically all we need, because, due to linearity of the derivative operator, we can find the derivative along a particular dimension of a multidimensional space by using our 1D solution along each constituent vector running in that direction, and we can find derivatives along multiple dimensions by applying the above in series along each dimension:

$$\frac{\partial^2}{\partial x_1 \partial x_2} y(x_1, x_2) = \texttt{Algo}(\texttt{Algo}(y_i, 1^{st}, x_1)_j, 1^{st}, x_2) \quad \forall\, i, j$$

$$\nabla^2 y = (\frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2}) y = \texttt{Algo}(y_i, 2^{nd}, x_1) + \texttt{Algo}(y_j, 2^{nd}, x_2) \quad \forall\, i, j$$

where i, j are indexers as in the computing sense and have nothing to do with the imaginary unit, Algo applies the algorithm to each vector along the dimension given by the third argument, and the 1st and 2nd in the second argument refer to the derivative order.

Each application to a vector incurs $O(N \log N)$ cost, and fundamentally applying the method to higher-dimensional data must involve a loop, so the full cost of applying along any given direction is (assuming length N in all dimensions) $O(N^D \log N)$, where D is the dimension of the data. Aside from pushing this loop lower down into numpy to take advantage of vectorized compute, there can be no cost savings for a derivative in a particular dimension.

7.1 Dimensions Together versus In Series

Can we simplify the situation at all?

Due to the linearity of the Fourier transform (and DCT), transforming along all dimensions, multiplying by appropriate $(jk)^\nu$ (or calling chebder) along corresponding dimensions of the transformed data, and then inverse transforming along all dimensions is equivalent to transforming, differentiating, and inverse transforming each dimension in series[17]:
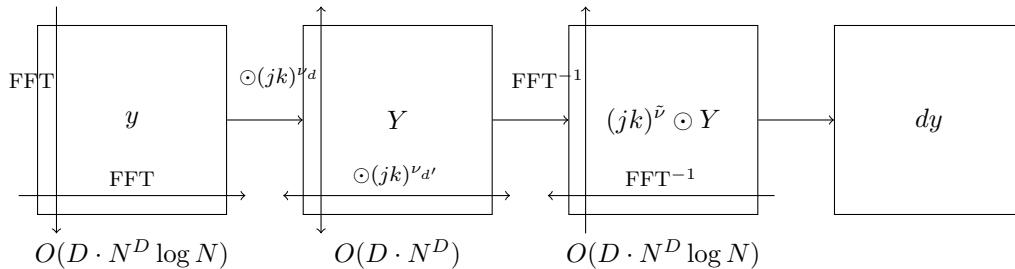
That's neat, but does it save us anything, really? Let's see it in series:

19
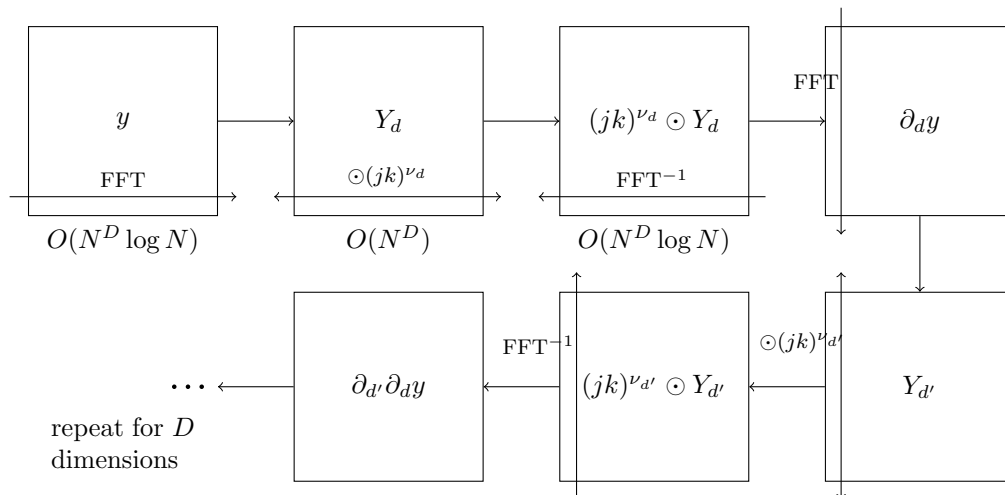
If we add up the costs, we can see that it's actually no more or less efficient to differentiate along all dimensions at once versus in series.

From a user-friendliness perspective, I judge it to be somewhat more confusing to specify multiple derivative dimensions at once (although generalizing the `order` and `axis` parameters to vectors is possible), so I have chosen to limit the package to differentation along a single dimension at a time, which also agrees with the interface of `chebder`.

Multidimensional *data* can still be handled, however, via clever indexing and use of `fft`, `dct`, and `chebder`'s `axis` parameter.

8 Arbitrary Domains

So far we've only used the domain $[0, 2\pi)$ in the Fourier case, because this is the domain assumed by the DFT, and the domain $[-1, 1]$ in the Chebyshev case, because this is the where a cosine wrapped around a cylinder casts a shadow. As you may have guessed, this hasn't curtailed the generality of the methods at all, because we can *map* any domain from a to b onto a canonical domain.

8.1 Fourier on [a, b)

Say we have $t \in [a, b)$ that we need to map to $\theta \in [0, 2\pi)$. We can accomplish this with:

$$\theta \in [0, 2\pi) \leftrightarrow t \in [a, b) = \underbrace{[0, 2\pi)}_{\theta} \cdot \frac{b - a}{2\pi} + a$$

To get a sense this is true, let's see an example:
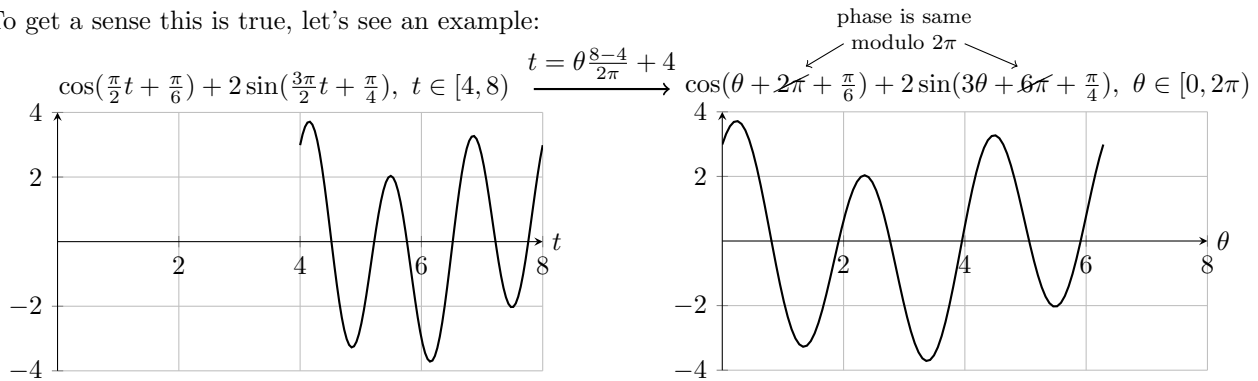
In the discrete case, where we have M samples on $[a, b)$, then we can map t_n with:

$$\theta_n \in \frac{\{0, ...M - 1\} \cdot 2\pi}{M} \leftrightarrow t_n \in \frac{\{0, ...M - 1\} \cdot \cancel{2\pi}}{M} \cdot \frac{b - a}{\cancel{2\pi}} + a$$

In simple code terms, if we want to take a spectral derivative of a function that's periodic on $[a, b)$, then we need to sample it at `t_n = np.arange(M)/M * (b - a) + a = np.linspace(a, b, M, endpoint=False)`.

8.2 Chebyshev on [a, b]

Here both ends are *inclusive*, so we have $t \in [a, b]$ that we need to map to $x \in [-1, 1]$. We can accomplish this with:

$$x \in [-1, 1] \leftrightarrow t \in [a, b] = \underbrace{[-1, 1]}_{x} \cdot \frac{b - a}{2} + \frac{b + a}{2}$$
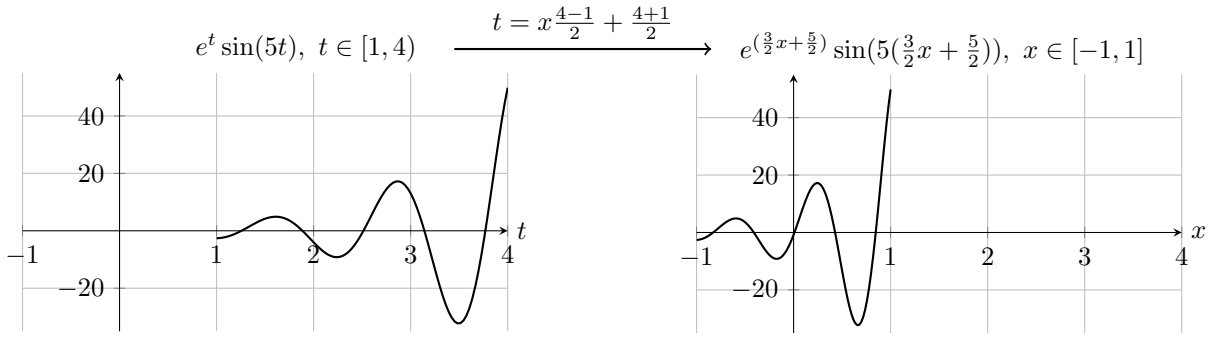
To get a sense this is true, let's see an example:

In the discrete case, where we have $N + 1$ samples on $[a, b]$, then we can map t_n with:

$$x_n \in \cos\left(\frac{\pi\{0, ...N\}}{N}\right) \leftrightarrow t_n \in \cos\left(\frac{\pi\{0, ...N\}}{N}\right) \cdot \frac{b - a}{2} + \frac{b + a}{2}$$

In code this is `t_n = np.cos(np.arange(N+1)*np.pi/N) * (b - a)/2 + (b + a)/2`.

Notice the order has flipped here, that counting *up* in n means we traverse x from $+1 \to -1$. This is actually what we want; it corresponds to the horizontal flip necessary to make cosine shadows equate with Chebyshev polynomails.

At the top of the figure:

$$e^t \sin(5t), \ t \in [1,4] \qquad \xrightarrow{\ t = x\frac{4-1}{2} + \frac{4+1}{2}\ } \qquad e^{(\frac{3}{2}x + \frac{5}{2})} \sin(5(\tfrac{3}{2}x + \tfrac{5}{2})), \ x \in [-1,1]$$

8.3 Accounting for Smoosh

When a function is sampled at one of the t_n above, then it is *as if* the function lives on the canonical domain. The actual mapping is purely notional, and the spectral differentation procedure proceeds completely agnostic to where the data really came from.

This means the result will actually be the derivative of the smooshed or stretched version of the function on the canonical domain. As the examples hopefully clarified, the *height* of this smooshed function is exactly as it was before, but the *width* is compressed or expanded by a factor of:

$$\text{smoosh} = \frac{\text{length of new interval}}{\text{length of old interval}} = \begin{cases} \frac{2\pi}{b-a} & \text{for Fourier} \\ \frac{2}{b-a} & \text{for Chebyshev} \end{cases}$$

Because a derivative is calculating slope, and slope is rise over run, the answer is effectively now

$$\frac{dy}{dx \cdot \frac{2 \text{ or } 2\pi}{b-a}} = \frac{dy}{dx} \cdot \underbrace{\frac{b-a}{2 \text{ or } 2\pi}}_{\text{scale}}$$

In other words, the overall derivative is scaled by the inverse of the width-smoosh. So to recover the true derivative we want, $\frac{dy}{dx}$, we have to divide by this scale, which is a familiar term from our variable transformations $t \leftrightarrow \theta$ or x.

For higher derivatives:

$$\frac{d^\nu y}{(dx \cdot \text{smoosh})^\nu} = \frac{d^\nu y}{dx^\nu} \cdot \text{scale}^\nu$$

So we can always correct the derivative by dividing by scale^ν.

To enable calculation of the `scale`, and to double check the user sampled their function at a correct `t_n` (especially in the Chebyshev case, since cosine-spacing is easy to flub and especially confusing with the DCT-II), the functions take the sample locations as a parameter and raise error messages with correct examples if the sampling is invalid.

9 Differentiating in the Presence of Noise

Finding the true derivative of a noisy signal is ill-posed, because random variations cause unknown (and often dramamtic) local deviations of function slope. This problem only gets worse for calculations of curvature and higher order derivatives, so the only solution is to try to remove the noise *before* differentating.

9.1 White Noise

Absent further information about a noise-generating process, we can't assume anything about the noise's structure, which makes it extra challenging to remove. We instead can only rely on the aid of a central observation from the field of signal processing:

> "Every spectrum of real noise falls off reasonably rapidly as you go to infinite frequencies, or else it would have infinite energy. But the sampling process aliases higher frequencies in lower ones, and the folding ... tends to produce a flat spectrum. ... *white noise*. The signal, usually, is mainly in the lower frequencies." –Richard Hamming, *The Art of Doing Science and Engineering*[2], Digital Filters III

By use of the term "frequencies", Hamming is implying use of the *Fourier* basis. He's saying there is *band-separation* of signal and noise in the frequency domain, yet another reason for its popularity. This general principle extends to other noise reduction techniques in spirit: At bottom, all accomplish some kind of *smoothing*, be it moving average, FIR filtering, Savitzky-Golay, Kalman filtering, total variation penalty, etc.

9.2   Filtering with the Fourier Basis

It's helpful to see a picture:
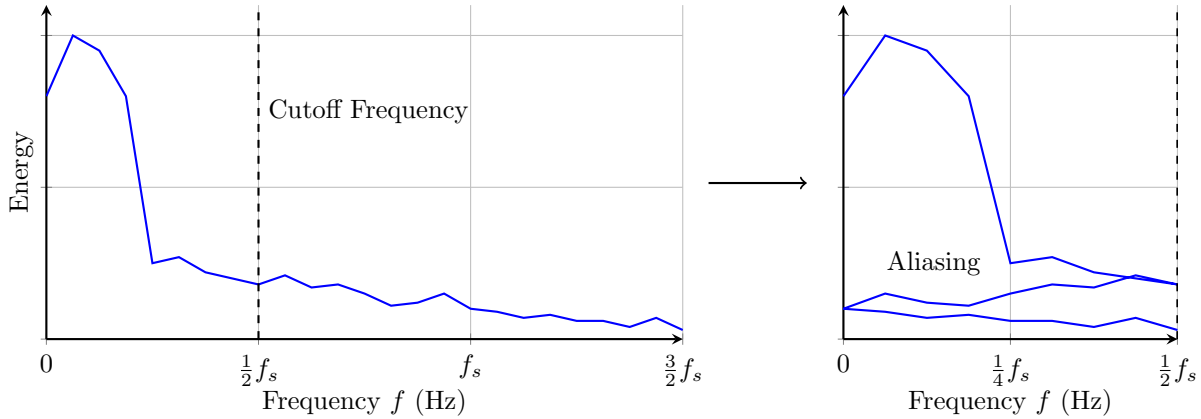
Typical energy spectrum of a noisy signal, before and after sampling. f_s is a sampling rate of our choosing.

In practice we expect to sample a signal frequently enough to capture fluctuations of interest in the data, so signal energy should be concentrated in frequencies below some cutoff. Noise energy, decaying up to higher frequencies, is added in linearly.

To reach the frequency domain, we use the FFT, which requires *equispaced* samples and a *periodic* signal, because a discontinuity or corner causes artefacts at higher frequency, which become impossible to model in finitely many coefficients and impossible to distinguish from noise. Nyquist theorem[4] tells us that we need > 2 equispaced samples per cycle to unambiguously reconstruct a frequency, so this process creates a natural cutoff at the Nyquist rate, $f_s/2$. Equispaced samples taken from frequencies slightly over this limit are best matched by frequencies slightly under the limit, just as unitary complex numbers alias: $e^{j(\pi+\epsilon)} = e^{-j(\pi-\epsilon)}$. This creates a *folding* pattern in the bandlimited FFT spectrum, which distributes decaying noise energy somewhat evenly.

Notice the more we sample, the more we can concentrate the legitimate signal's energy in low frequencies, and the more we can distribute noise energy across higher frequencies. We can then *zero out* or otherwise dampen the upper part of the spectrum to chop down the noise that hasn't aliased on to the periodic signal's band! The `filter parameter` is designed exactly for this. If we then inverse transform, we get a smoother signal, or we can multiply by $(jk)^\nu$ and then inverse transform to sample the derivative of that smoother signal.

9.3   The Advantage of Spectral Smoothing

Most alternative noise-quelling techniques can only take advantage of local information, but a spectral representation builds a function out of basis functions that span the entire domain, so every point's value takes holistic fit under consideration. This makes the reconstruction much more *robust* to perturbations than one that uses only a few neighboring points. I.e. it's much harder to *corrupt* the signal so thoroughly that it can't be successfully recovered.

In the Fourier case, this has an analogy with error-correcting codes[28, 2]; notice the locations corresponding to Hamming code parity checks constitute selections of different frequency:

check #1   check #2   check #3   check #4

22

Of course there are distinctions: In the context of continuous signals, "corruption" means (discrete representations of) continuous numbers have slightly inaccurate values, whereas in error correcting codes each datum is a single discrete bit which can simply be flipped. Likewise, a Fourier coefficient indicates "We need this much of the kth basis function", whereas the analogous parity check indicates "There is/isn't a parity error among my bits."

But in both cases a spot-corruption will stand out, because it appears in a particular combination of parity checks or introduces a value that can't as easily be represented with a finite combination of smooth basis functions. In this sense, the Fourier basis is *the ideal* basis for filtering high frequency noise, so long as the signal of interest is periodic, and other methods are merely trying to approximate a lowpass filter.

9.4 Measurement versus Process Uncertainty

Noise can have a couple different sources. If noise is due to imperfect sensing, then each sample is drawn from a random variable. By contrast, if there is uncertainty in the underlying dynamics, then we might say there is noise in the process itself. If we sample our signal uniformly, as we do when using the Fourier basis and most other filtering methods, since they tend to be more computationally expensive to generalize to non-uniform samplings[29], then we don't really need to make a distinction between these noise sources, because they look the same. But if we cosine-sample our data for expediency when using the Chebyshev basis, there will be a difference, because samples taken closer together will have greater correlation.

| cosine-spaced noise | equispaced noise | cosine-sampling of equispaced noise |
| measurement noise in x domain | measurement noise in θ domain or process noise in x domain | process noise in θ domain |
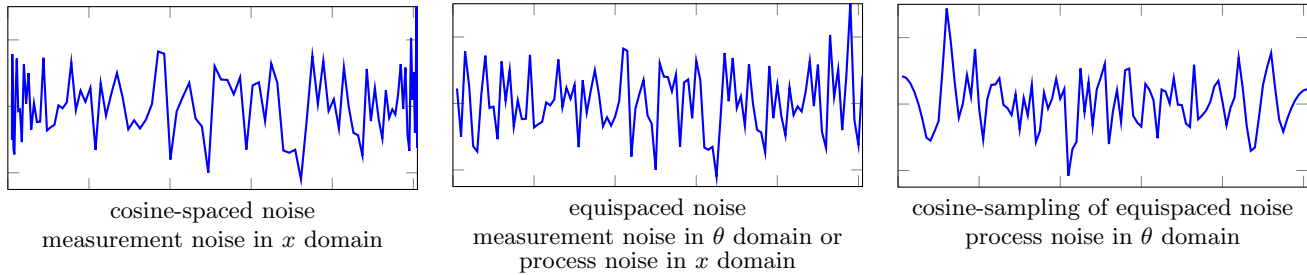
When we sample at cosine-space points, Δx decreases near the edges. In the case of measurement noise, effects on Δy are undiminished in these regions, so the error in $\frac{\Delta y}{\Delta x}$ increases dramatically. In the case of process noise the situation is a little more hopeful, because its effects on Δy naturally get smaller as Δx shrinks.

9.5 Filtering with Polynomial Bases (is Terrible)

To avert Gibbs phenomenon in the case of aperiodic data or to achieve better compression (more energy represented in fewer modes) in case we know something about how our data should look, we may prefer to use another basis, like Chebyshev polynomials or PCA modes for representation. All bases have "lower frequency" and "higher frequency" elements, meaning some modes have fewer transitions between low and high values, and some have more. Similar to Fourier basis representations, signal energy empirically tends to cluster in lower modes, and noise tends to be scattered across modes, similar to the typical spectrum above.

However, smoothing aperiodic data is fundamentally more difficult than the periodic case at the edges of the domain, because in the periodic case we can smooth across the domain boundary, whereas with an aperiodic function we only get one-sided information about those regions. And unlike Fourier basis functions, which have uniform frequency throughout their domain, other bases are typically nonuniform. This may be desirable if the data or noise is known to fit a particular pattern, but in the naive case, where we don't know the noise shape or expect white noise, nonuniform basis functions' variable expressive power across the domain can mismatch the situation. Chebyshev polynomials exhibit this characteristic, getting steeper and therefore higher frequency near the boundaries, which makes them worse at filtering and better at *fitting* high frequency noise in these regions, i.e. more *sensitive* to disruptions there.

This problem is perhaps best shown mathematically in the Chebyshev-via-Fourier method, where we not only implicitly warp the function to be $y(\cos(\theta))$ by taking a DCT (thereby treating measurement noise as if it's evenly distributed in the θ domain), but we later also explicitly *unwarp* to get back to the x domain, involving division by powers of $\sqrt{1-x^2}$[25, 18], which $\to 0$ as $x \to \pm 1$. Numerically equivalent, in the Chebyshev series-based method, "Tight coupling between coefficents enables propagation of errors from high frequency to low frequency modes."[25] We see that higher-order coefficients have an impact on ever-other lower-order coefficient[26], and this effect is cumulative, so a slight error in coefficient values, especially higher-order ones, compounds in a very nasty way when we differentiate. Thus "while the representation of a function by a Chebyshev series may be most accurate near $x = \pm 1$, these results indicate that the derivatives computed from a Chebyshev series are *least accurate* at the edges of the domain."[25]
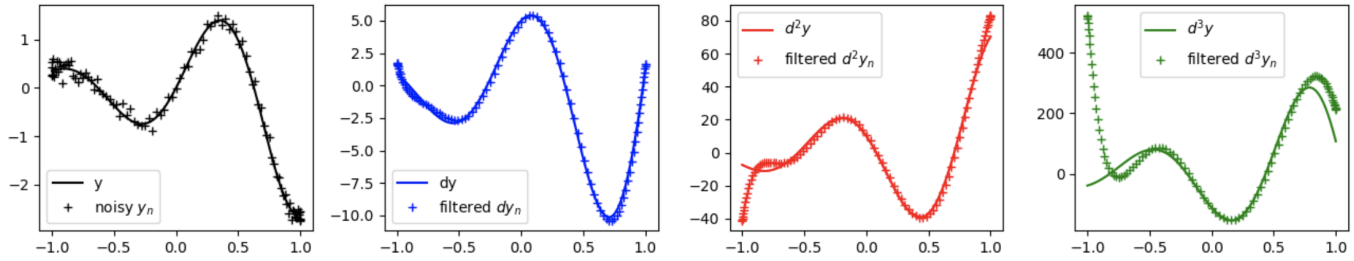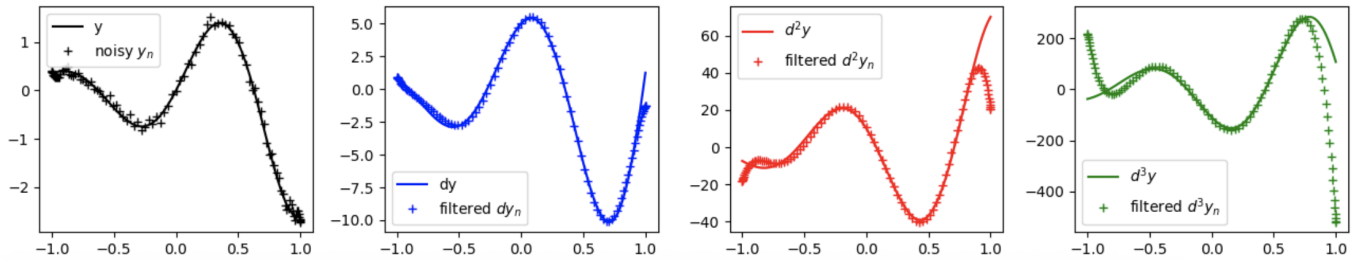
An example of systematic edge blowup for Chebyshev derivatives in the presence of noise.

In fact, the Chebyshev basis is not the only one suffer this weakness, because other polynomial bases, e.g. the Legendre polynomials (orthogonal without a weighting function) and Bernstein polynomials (linearly independent but not orthogonal), experimented with in the filtering noise notebook, are also more sensitive at the domain edges, even when sampled at cosine-spaced points to help dampen Runge phenomenon. This is a fundamental limitation of polynomial-based methods in the presence of noise. Occasionally differentiation can be made to work okay by filtering higher modes, but there is always systematic blowup in higher-order derivatives.

Notes

a. There's a great passage in Richard Hamming's book *The Art of Doing Science and Engineering*[2] where he wonders why we use the Fourier basis so much:

> "It soon became clear to me digital filter theory was dominated by Fourier series, about which theoretically I had learned in college, and actually I had had a lot of further education during the signal processing I had done for John Tukey, who was a professor from Princeton, a genius, and a one or two day a week employee of Bell Telephone Laboratories. For about ten years I was his computing arm much of the time.
>
> Being a mathematician I knew, as all of you do, that any complete set of functions will do about as good as any other set at representing arbitrary functions. Why, then, the exclusive use of the Fourier series? I asked various electrical engineers and got no satisfactory answers. One engineer said alternating currents were sinusoidal, hence we used sinusoids, to which I replied it made no sense to me. So much for the usual residual education of the typical electrical engineer after they have left school!
>
> So I had to think of basics, just as I told you I had done when using an error-detecting computer. What is really going on? I suppose many of you know what we want is a time-invariant representation of signals, since there is usually no natural origin of time. Hence we are led to the trigonometric functions (the eigenfunctions of translation), in the form of both Fourier series and Fourier integrals, as the tool for representing things.
>
> Second, linear systems, which is what we want at this stage, also have the same eigenfunctions—the complex exponentials which are equivalent to the real trigonometric functions. Hence a simple rule: if you have either a time-invariant system or a linear system, then you should use the complex exponentials.
>
> On further digging in to the matter I found yet a third reason for using them in the field of digital filters. There is a theorem, often called Nyquist's sampling theorem (though it was known long before and even published by Whittaker, in a form you can hardly realize what it is saying, even when you know Nyquist's theorem), which says that if you have a band-limited signal and sample at equal spaces at a rate of at least two in the highest frequency, then the original signal can be reconstructed from the samples. Hence the sampling process loses no information when we replace the continuous signal with the equally spaced samples, provided the samples cover the whole real line. The sampling rate is often known as the Nyquist rate after Harry Nyquist, also of servo stability fame, as well as other things [also reputed to have been just a really great guy who often had productive lunches with his colleagues, giving them feedback and asking questions that brought out the best in them]. If you sample a non-band-limited function, then the higher frequencies are "aliased" into lower ones, a word devised by Tukey to describe the fact that a single high frequency will appear later as a single low frequency in the Nyquist band. The same is not true for any other set of functions, say powers of t. Under equally spaced sampling and reconstruction a single high power of t will go into a polynomial (many terms) of lower powers of t.

Thus there are three good reasons for the Fourier functions: (1) time invariance, (2) linearity, and (3) the reconstruction of the original function from the equally spaced samples is simple and easy to understand.

Therefore we are going to analyze the signals in terms of the Fourier functions, and I need not discuss with electrical engineers why we usually use the complex exponents as the frequencies instead of the real trigonometric functions. [It's down to convenience, really.] We have a linear operation, and when we put a signal (a stream of numbers) into the filter, then out comes another stream of numbers. It is natural, if not from your linear algebra course then from other things such as a course in differential equations, to ask what functions go in and come out exactly the same except for scale. Well, as noted above, they are the complex exponentials; they are the eigenfunctions of linear, time-invariant, equally spaced sampled systems.

Lo and behold, the famous transfer function [contains] exactly the eigenvalues of the corresponding eigenfunctions! Upon asking various electrical engineers what the transfer function was, no one has ever told me that! Yes, when pointed out to them that it is the same idea they have to agree, but the fact it is the same idea never seemed to have crossed their minds! The same, simple idea, in two or more different disguises in their minds, and they knew of no connection between them! Get down to the basics every time!"

Back: subsection 1.1

References

[1] Lebesgue Integrable, https://mathworld.wolfram.com/LebesgueIntegrable.html

[2] Hamming, R., 1996, *The Art of Doing Science and Engineering*

[3] Pego, B., Simplest proof of Taylor's theorem, https://math.stackexchange.com/a/492165/278341

[4] Oppenheim, A. & Willsky, A., 1996, *Signals and Systems, 2nd Ed.*

[5] Disintegration By Parts, Why do Fourier transforms use complex numbers?, https://math.stackexchange.com/a/1293127/278341

[6] Yagle, A., 2005, https://web.eecs.umich.edu/~aey/eecs206/lectures/fourier2.pdf

[7] https://math.stackexchange.com/questions/1105265/why-do-fourier-series-work

[8] Xue, S., 2017, Convergence of Fourier Series, https://math.uchicago.edu/~may/REU2017/REUPapers/Xue.pdf

[9] Derivation of Fourier Series, http://lpsa.swarthmore.edu/Fourier/Series/DerFS.html

[10] Sego, D., Demystifying Fourier analysis, https://dsego.github.io/demystifying-fourier/

[11] Nakagome, S. Fourier Transform 101 — Part 4: Discrete Fourier Transform, https://medium.com/sho-jp/fourier-transform-101-part-4-discrete-fourier-transform-8fc3fbb763f3

[12] Brunton, S., The Fourier Transform and Derivatives, https://www.youtube.com/watch?v=d5d0ORQHNYs

[13] Trefethen, N., 2000, *Spectral Methods in Matlab*, Chapter 4, https://epubs.siam.org/doi/epdf/10.1137/1.9780898719598.ch4

[14] Kutz, J.N., 2023, *Data-Driven Modeling & Scientific Computation*, Ch. 11, https://faculty.washington.edu/kutz/kutz_book_v2.pdf

[15] Discrete Fourier Transform, https://numpy.org/doc/2.1/reference/routines.fft.html

[16] Bristow-Johnson, R., 2014, About Discrete Fourier Transform vs. Discrete Fourier Series, https://dsp.stackexchange.com/a/18931/40873

[17] Johnson, S., 2011, Notes on FFT-based differentiation, https://math.mit.edu/~stevenj/fft-deriv.pdf

[18] Trefethen, N., 2000, *Spectral Methods in Matlab*, Chapter 8, https://epubs.siam.org/doi/epdf/10.1137/1.9780898719598.ch8

[19] Burns, K., et al., 2020, Dedalus: A flexible framework for numerical simulations with spectral methods, https://www.researchgate.net/publication/340905766_Dedalus_A_flexible_framework_for_numerical_simulations_with_spectral_methods

[20] https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.dct.html

[21] https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.dst.html

[22] Giesen, F., DCT-II vs. KLT/PCA, https://www.farbrausch.de/%7Efg/articles/dct_klt.pdf

[23] Royi, 2025, Why does the DCT-II have better energy compaction than DCT-I?, https://dsp.stackexchange.com/a/96197/40873

[24] Harris, C., 2009, `chebder`, https://github.com/numpy/numpy/blob/v2.2.0/numpy/polynomial/chebyshev.py#L874-L961

[25] Breuer, K. & Everson, R., 1990, On the errors incurred calculating derivatives using Chebyshev polynomials, https://doi.org/10.1016/0021-9991(92)90274-3

[26] Komarov, P., 2025, Chebyshev Series Derivative in terms of Coefficients, https://scicomp.stackexchange.com/q/44939/48402

[27] Power Rule, https://mathworld.wolfram.com/PowerRule.html

[28] Sanderson, G., 2020, But what are Hamming codes? The origin of error correction, https://www.youtube.com/watch?v=X8jsijhllIA

[29] 2012, Savitzky-Golay smoothing filter for not equally spaced data, https://dsp.stackexchange.com/a/9494/40873