Spectral Derivatives

Pavel Komarov

January 8, 2025

One of the happiest accidents in all math is the ease of 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` 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 **2**
1.1 The Fourier Basis . . . 2

2 Transforms **3**
2.1 The Fourier Transform . . . 3
2.2 A Whole Family . . . 4

3 Taking Derivatives in the Fourier Domain **5**
3.1 Taking Derivatives in the Discrete Case . . . 5
3.1.1 The DFT Pair . . . 5
3.1.2 Interpolation . . . 5
3.1.3 Taking Derivatives of the Interpolant . . . 8
3.2 Limitations . . . 9

4 The Chebyshev Basis **9**
4.1 The Advantage of Chebyshev . . . 11

5 An Algorithm **11**
5.1 The Discrete Cosine Transform . . . 12
5.2 Even and Odd Derivatives and the Discrete Sine Transform . . . 13
5.3 Transforming Back to the Chebyshev Domain . . . 14
5.3.1 Higher Derivatives . . . 14
5.3.2 Handling Domain Endpoints . . . 15
5.3.3 Endpoints for Higher Derivatives . . . 16

6 Multidimensionality **17**
6.1 Dimensions Together versus In Series . . . 17
6.2 Splintering . . . 18

7 Arbitrary Domains **19**
7.1 Fourier . . . 19
7.2 Chebyshev . . . 20
7.3 Accounting for Smoosh . . . 20

1 Bases

A *basis* is a set of "orthogonal" functions, call them $\{\xi_k\}$, that can be summed together in various quantities to produce other functions. Othogonal means that if we take the "inner product" of one funtion from the set with itself, we get back 1, and if we take the inner product of a function with a different member of the set, we get back 0. In this sense the members of the basis set are independent of one another, just 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 $\bar{\cdot}$ 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:

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

where I'm using j to represent the imaginary unit, 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:

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

So

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

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

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

Notice all of 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[4], 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 $e^{j\omega}$ as oscillations at a particular frequency, ω.

If we inner product mismatched wiggles, they misalign and integrate to 0, but if we inner product matched wiggles, they align, multiply to 1 because of the complex conjugate, and integrate to 2π over a period.

2 Transforms

I can use a basis to "transform" a function, meaning I take the function's inner product with each of the basis functions to produce numbers:

$$\langle f, \xi_k \rangle = \int_a^b f(x)\overline{\xi_k(x)}dx = \text{a constant coefficient}, c_k$$

These numbers descibe *how much* of each basis function ξ_k is present in the signal f on a domain between a and b. If I do this for all the $\{\xi_k\}$, I essentially get a *recipe*, which says "I need this much of ξ_0 and that much of ξ_1 and howevermuch of ξ_2 ... added together to reproduce the original signal."

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

where M is the number of basis functions I'm using in my reconstruction.

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. We've *transformed* the function to a set of numbers which now live in a different domain.

Beware that the terminology "transform" and "domain" is not always used to describe transforming a continuous function to a discrete set of c_k like this, because "domain" technically refers to a "connected" set, not just a collection of k things. 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, having infinitely many members which are infinitesimally close together. This is the case for the Fourier basis, where we choose $\omega \in \mathbb{R}$, and hence ω really can become a new domain.

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 k^{th} 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 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.

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

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[8]. That is, we can choose *complex* c_k such that when multiplied by *complex* exponentials, 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 k^{th} frequency, thereby accomplishing a phase shift[9]. 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 of the entire set of $\mathbb{R}$, then

*It's worth considering how weird it is this is true. In fact, it's so weird that Joseph Lagrange publicly declared Fourier was wrong at a meeting of the Paris Academy in 1807![5] It's valuable to ask why this works[6] and sift through some analysis.[7]

it makes more sense to express the transformed coefficients as a function in ω and to make both our inner product and reconstruction expression integrals from $-\infty$ to $+\infty$:

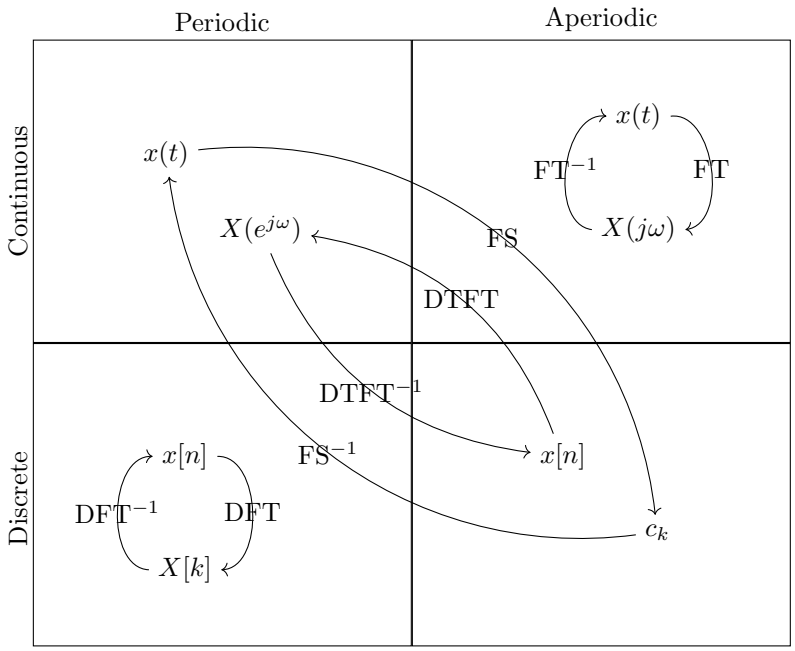
$$\begin{aligned} \hat{f}(\omega) &= \int_{-\infty}^{\infty} f(x) e^{-j\omega x} dx = \mathcal{F}\{f(x)\} \\ f(x) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{j\omega x} d\omega = \mathcal{F}^{-1}\{\hat{f}(\omega)\} \end{aligned} \tag{2}$$

where the hat $\hat{\cdot}$ represents a function in the Fourier domain, and the $\frac{1}{2\pi}$ is a scaling factor that corrects for the fact the inner product of a Fourier basis function with itself integrates to 2π over a period instead of to 1 as we need for orthonormality.

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.[10]. 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?



Note that, following a more signal-processing-y convention[11], 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 2. 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 capital-T 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 [11].

3 Taking Derivatives in the Fourier Domain

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

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

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

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

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 accompish the DFT (subsection 2.2, Equation 3) 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

$$\begin{aligned} \text{DFT:} \quad & Y_k = \sum_{n=0}^{M-1} y_n e^{-j\frac{2\pi}{M}nk} \\ \text{DFT}^{-1}\text{:} \quad & y_n = \frac{1}{M}\sum_{k=0}^{M-1} Y_k e^{j\frac{2\pi}{M}nk} \end{aligned} \tag{3}$$

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

To see where this comes from, see [16] or [11].

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[17] (hence periodicity and aliasing) and are being sampled at equally-spaced angles/angular velocities.

3.1.2 Interpolation

I now quote Steven Johnson[18], 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*"[†]

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

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:

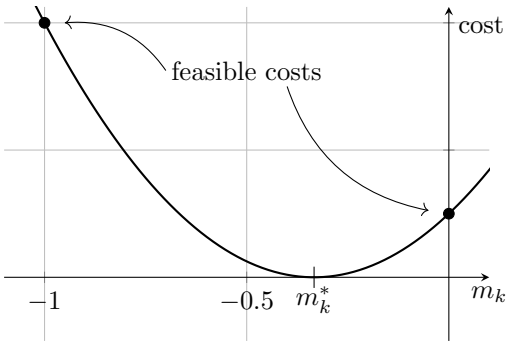
$$\begin{aligned}
\underset{m_k}{\text{minimize}} \quad & (k+m_k M)^2 \\
\text{s.t.} \quad & 0 \leq k < M \\
& m_k \in \mathbb{Z}
\end{aligned}$$

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.

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



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 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 $Y_{M/2}(ue^{j\frac{M}{2}\frac{2\pi}{M}n} + (1-u)e^{-j\frac{M}{2}\frac{2\pi}{M}n}) = Y_{M/2}(u\overbrace{e^{j\pi n}}^{(-1)^n} + (1-u)\overbrace{e^{-j\pi n}}^{(-1)^n}) = Y_{M/2}(-1)^n$, which at $n = \frac{M}{2}$ will be $Y_{M/2}(-1)^{\frac{M}{2}} = Y_{M/2}\cos(\frac{M}{2}\pi)$, "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:

$$\begin{aligned}
&Y_{M/2}\overline{Y_{M/2}}\frac{1}{2\pi}\int_0^{2\pi}(ue^{j\frac{M}{2}\theta} + (1-u)e^{-j\frac{M}{2}\theta})\overline{(ue^{j\frac{M}{2}\theta} + (1-u)e^{-j\frac{M}{2}\theta})}d\theta \\
&= |Y_{M/2}|^2\frac{1}{2\pi}\Big(u\overline{u}\int_0^{2\pi}\underbrace{e^{j\frac{M}{2}\theta}e^{-j\frac{M}{2}\theta}}_{=1}d\theta + u\overline{(1-u)}\int_0^{2\pi}\underbrace{e^{j\frac{M}{2}\theta}e^{j\frac{M}{2}\theta}}_{\text{periodic!}}d\theta + (1-u)\overline{u}\int_0^{2\pi}\underbrace{e^{-j\frac{M}{2}\theta}e^{-j\frac{M}{2}\theta}}_{\text{periodic!}}d\theta \\
&\quad + (1-u)\overline{(1-u)}\int_0^{2\pi}\underbrace{e^{-j\frac{M}{2}\theta}e^{j\frac{M}{2}\theta}}_{=1}d\theta\Big) \\
&= |Y_{M/2}|^2\frac{1}{2\pi}(|u|^2 2\pi + |1-u|^2 2\pi) = |Y_{M/2}|^2(|u|^2 + |1-u|^2)
\end{aligned}$$

because integrating something periodic over a multiple of its period yields 0.

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[19] (DCT), 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":

$$y(\theta) = \frac{1}{M}\Big(Y_0 + \sum_{0<k<\frac{M}{2}}\left(Y_k e^{jk\theta} + Y_{M-k}e^{-jk\theta}\right) + Y_{M/2}\cos(\frac{M}{2}\theta)\Big) \tag{4}$$

"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:

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

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

$$y'_n = \frac{1}{M}\Big(\sum_{0<k<\frac{M}{2}} jk(Y_k e^{jk\frac{2\pi}{M}n} - Y_{M-k}e^{-jk\frac{2\pi}{M}n}) - \cancelto{0}{\frac{M}{2}Y_{M/2}\sin(\pi n)}\Big) = \frac{1}{M}\sum_{k=0}^{M-1} Y'_k e^{j\frac{2\pi}{M}kn}$$

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

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

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

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

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."[18]

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:

$$Y_k^{(\nu)} = \begin{cases} (jk)^\nu \cdot Y_k & k < \frac{M}{2} \\ (j\frac{M}{2})^\nu \cdot Y_k & k = \frac{M}{2} \text{ and } \nu \text{ even} \\ 0 & k = \frac{M}{2} \text{ and } \nu \text{ odd} \\ (j(k-M))^\nu \cdot Y_k & k > \frac{M}{2} \end{cases} \tag{5}$$

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.

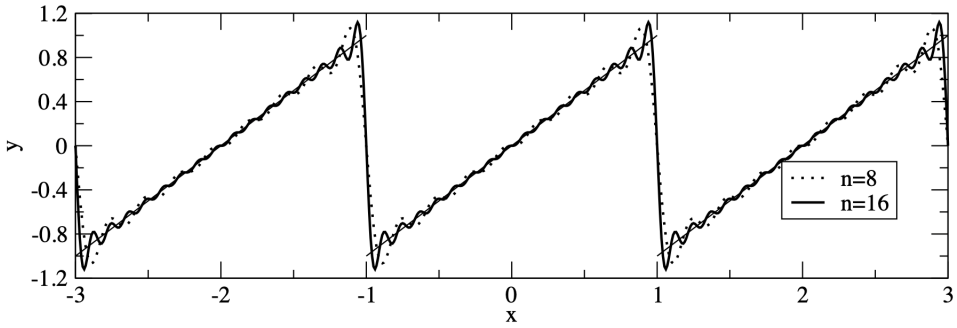
[‡]For *real* signals it is common (e.g. in the kind of code ChatGPT might generate to do this) not to worry about zeroing out the Nyquist term and *instead* throw away the imaginary part of the inverse transform. This works because $Y_{M/2} = \sum_{n=0}^{M-1} y_n e^{-j\frac{2\pi}{M}n\frac{M}{2}} = \sum_{n=0}^{M-1} y_n(-1)^n$, which will be purely real for real y_n, and when we mulitply by $(jk)^\nu$ for odd ν, then $Y^{(\nu)}_{M/2}$ 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 $+Y^{(\nu)}_{M/2}e^{j\frac{2\pi}{M}n\frac{M}{2}} = +Y^{(\nu)}_{M/2}(-1)^n$, 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.

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], we in fact need *all* the infinite modes to reconstruct the true signal[11], and 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.[11]

If there are discontinuities or corners, we get what's called the Gibbs Phenomenon[11], 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!



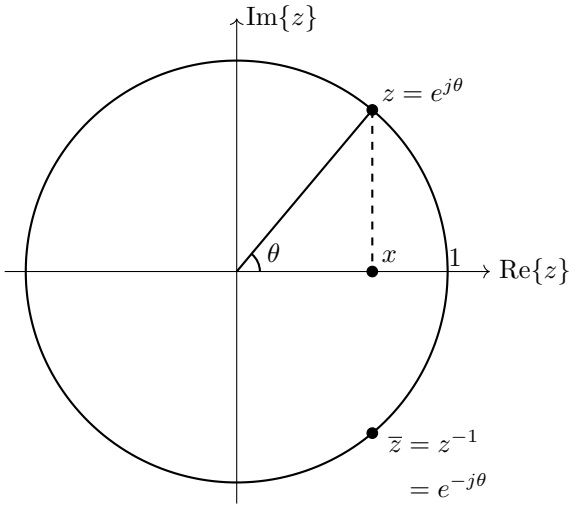
An example of the Gibbs phenomenon, from [14]

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). 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.

4 The Chebyshev Basis

There is another basis which we can use to represent arbitrary functions, called the Chebyshev polynomials[21], which has a really neat relationship to the Fourier basis.



$$\begin{aligned} \text{Let} \quad x \in [-1,1] \qquad & \text{Chebyshev} \\ = \cos(\theta),\ \theta \in \mathbb{R} \qquad & \text{Fourier} \\ = \frac{1}{2}(z + z^{-1}),\ |z| = 1 \qquad & \text{Laurent} \end{aligned} \tag{6}$$

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

$$T_0(x) = Re\{z^0\} = 1$$

$$T_1(x) = Re\{z^1\} = \frac{1}{2}(e^{j\theta} + e^{-j\theta}) = \cos(\theta) = x$$

$$T_2(x) = \frac{1}{2}(e^{j2\theta} + e^{-j2\theta}) = \cos(2\theta)$$

$$\text{but also } = \frac{1}{2}\underbrace{(z^2 + 2 + z^{-2})}_{\text{perfect square}} - 1 = \left(\sqrt{\frac{1}{2}}(z + z^{-1})\right)^2 - 1 = \underbrace{\left(\frac{2}{\sqrt{2}}\right)^2}_{2}\underbrace{\frac{z + z^{-1}}{2}}_{\cos(\theta)} - 1 = 2x^2 - 1$$

$$T_3(x) = \frac{1}{2}(e^{j3\theta} + e^{-j3\theta}) = \cos(3\theta)$$

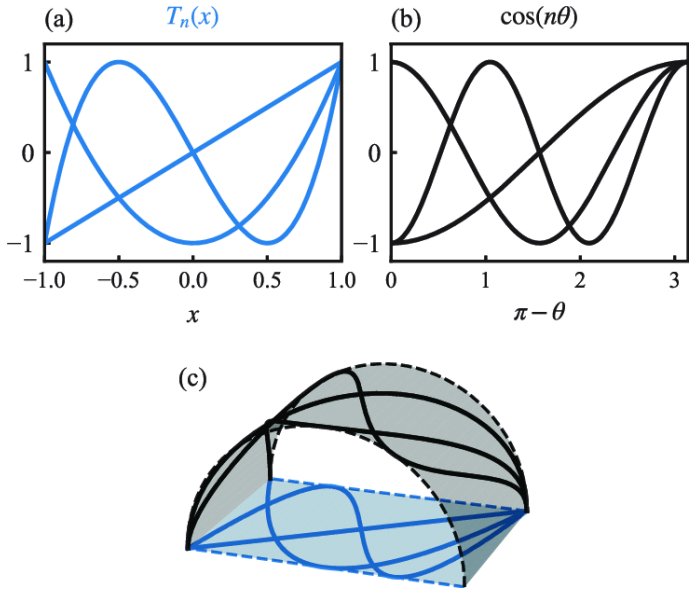
$$\text{but also } = \frac{1}{2}(z + z^{-1})^3 - \frac{3}{2}(z + z^{-1}) = 4x^3 - 3x$$

...

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."[21]



Relationship of Chebyshev domain and Fourier Domain, from [22]. Notice the cosines are horizontally flipped. The authors use n instead of k, which is common for Chebyshev polynomials (e.g. [21]), 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-increasing $x \in [-1, 1]$, 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.

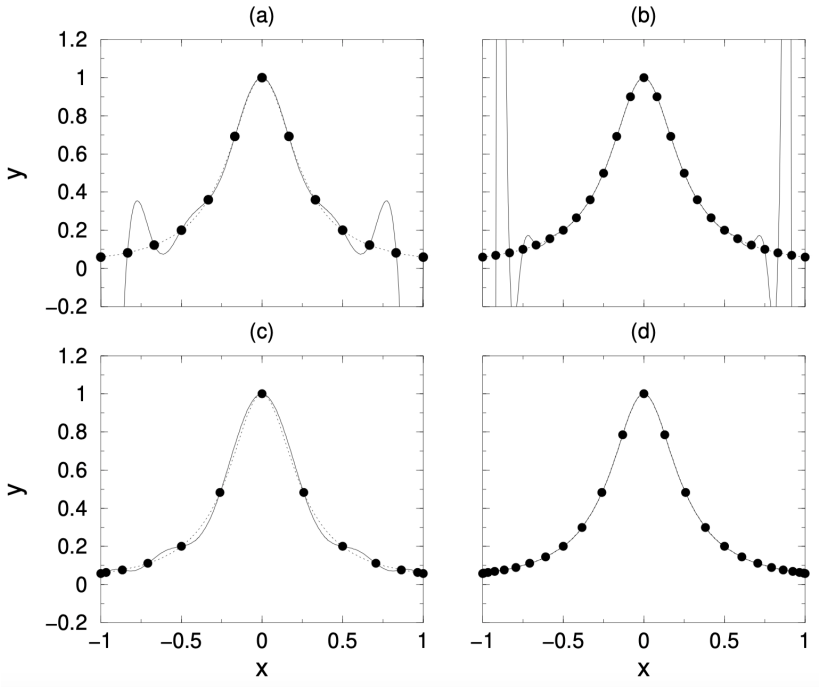
We can reconstruct a function using the different variables/basis formulations, and as long as our variables are related as in Equation 6, these reconstructions are *equivalent*:

$$y(x) = \sum_{k=0}^{N} a_k T_k(x) \; ; \quad y(z) = \sum_{k=0}^{N} a_k \frac{1}{2}(z^k + z^{-k}) \; ; \quad y(\theta) = \sum_{k=0}^{N} a_k \cos(k\theta) \tag{7}$$

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 $x_n = \cos(\theta_n = \pi n/N)$, 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![19][20]

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 An Algorithm

This all suggests a solution procedure:

Chebyshev Derivatives

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 taking derivatives.
4: Inverse transform using the DST if odd function, DCT if even function.
5: Change variables back, taking care that the derivative in the Chebyshev domain entails an extra chain rule.

There are 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 7) only contains cosines, doing a full FFT, which tries to fit sines as well as cosines, would be doing double work. Instead we can use the DCT. There are actually several possible definitions of the DCT[19], but I use the DCT-I, because it is in some sense closest to the interpolant (Equation 4) we've already derived.

Say $$\vec{y} = \underbrace{[\underbrace{y_0, y_1, ...y_{N-1}, y_N}_{\text{original vector, length N+1}}, \underbrace{y_{N-1}, ...y_1}_{\text{redundant information}}]}_{\text{length M = 2N, necessarily even!}}, \text{ that is: } y_n = y_{M-n},\ 0 \le n \le N$$

Then using $M-k$ for k in the DFT equation (Equation 3), we get:

$$\begin{aligned} Y_{M-k} &= \sum_{n=0}^{M-1} y_n e^{-j\frac{2\pi}{M}n(M-k)} = \sum_{n=0}^{M-1} y_n \underbrace{e^{-j2\pi n}}_{1} e^{j\frac{2\pi}{M}nk} \overset{\text{let } n_{new} = M - n_{old}}{=} \sum_{n=M}^{1} y_{M-n} e^{j\frac{2\pi}{M}(M-n)k} \\ &= \sum_{n=1}^{M} \underbrace{y_{M-n}}_{=y_n} \underbrace{e^{j2\pi k}}_{1} e^{-j\frac{2\pi}{M}nk} = \sum_{n=1}^{M} y_n e^{-j\frac{2\pi}{M}nk} \overset{\text{because } e^{-j\frac{2\pi}{M}Mk} = e^0 = 1 \text{ and } y_M = y_0}{=} \sum_{n=0}^{M-1} y_n e^{-j\frac{2\pi}{M}nk} = Y_k \quad \square \end{aligned}$$

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 4):

$$\begin{aligned} y(\theta) &= \frac{1}{M}\Big(Y_0 + \sum_{0<k<\frac{M}{2}} \big(Y_k e^{jk\theta} + Y_{M-k} e^{-jk\theta}\big) + Y_{M/2}\cos(\frac{M}{2}\theta)\Big) \\ &= \frac{1}{M}\Big(Y_0 + 2\sum_{k=1}^{N-1} \big(Y_k \underbrace{\frac{e^{jk\theta} + e^{-jk\theta}}{2}}_{\cos(k\theta)}\big) + Y_N \cos(N\theta)\Big) \end{aligned} \tag{8}$$

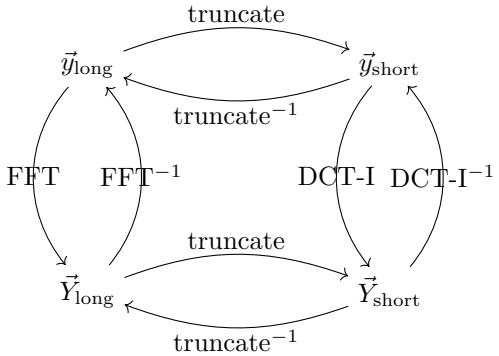
At samples $\theta_n = \frac{2\pi}{M}n = \frac{\pi}{N}n$, this becomes:

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

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

$$\text{FFT}^{-1}([Y_0, ...Y_N, Y_{N-1}, ...Y_1])[:N+1] = \text{DCT-I}^{-1}([Y_0, ...Y_N]) \tag{9}$$

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} stacks back in redundant information

We can now easily see that in addition to the inverse relationship (Equation 9), we also have the forward relationship:

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

Before we wrap up discussion of the DCT-I[19], note that the 0^{th} 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 7), 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). This will be important when we later transform back to the Chebyshev domain, because we will use *the DCT-I version* of $y(\theta)$, not the original conception.

5.2 Even and Odd Derivatives and the Discrete Sine Transform

The DCT-I can get us in to the frequency domain, but we'll need the help of another transform to get back out.

If we look at the full Y_{long} with redundant information, we have a palindromic structure around N, but also around 0, because of the repetitions[17], which ensure we can read the values of Y 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 the forward transform is symmetrical with the inverse transform, also ultimately a bunch of cosines if we were to interpolate $Y(\omega)$ from Y_k.

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 5), for $k \in \{0, ...M-1\}$:

$$\begin{aligned}\vec{Y}^{(\nu)}_{\text{long}} &= [0, j^\nu, ...(j(N-1))^\nu, \underbrace{(0 \text{ or } (jN)^\nu)}_{\substack{\text{depending on} \\ \nu \text{ odd or even}}}, (-j(N-1))^\nu, ...(-j)^\nu] \odot \vec{Y}_{\text{long}} \\ &= \underbrace{j^\nu}_{\text{constant}} \cdot \underbrace{[0, 1, ...1, (0 \text{ or } 1), -1, ..., -1]^\nu}_{\tilde{\mathbb{1}}} \odot \underbrace{[0, 1, ...N-1, N, N-1, ...1]^\nu}_{\text{palindromic}} \odot \vec{Y}_{\text{long}}\end{aligned}$$

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:

$$\begin{aligned} y_n^{(\text{odd } \nu)} &= \frac{1}{M} \sum_{0<k<\frac{M}{2}} (jk)^\nu (Y_k e^{jk\theta_n} - \underbrace{Y_{M-k}}_{=Y_k} e^{-jk\theta_n}) = \frac{1}{M} \sum_{k=1}^{N-1} (jk)^\nu Y_k \underbrace{(e^{jk\theta_n} - e^{-jk\theta_n})}_{2j\sin(k\theta_n)} \\ &\quad \text{(from oddness of } \tilde{\mathbb{1}}^\nu\text{)} \\ &= \underbrace{\frac{1}{M} 2 \sum_{k=1}^{N-1} (jk)^\nu Y_k j \sin(\frac{\pi nk}{N})}_{= \text{a DST-I of } \vec{Y}^{(\nu)}_{\text{short}} \cdot j!} \qquad (10) \\ y_n^{(\text{even } \nu)} &= \frac{1}{M} \Big(\sum_{0<k<\frac{M}{2}} (jk)^\nu (Y_k e^{jk\theta_n} + \underbrace{Y_{M-k}}_{=Y_k} e^{-jk\theta_n}) + (j\frac{M}{2})^\nu Y_{M/2} \cos(\frac{M}{2}\theta_n) \Big) \\ &\quad \text{(from evenness of } \tilde{\mathbb{1}}^\nu\text{)} \\ &= \frac{1}{M} \Big((jN)^\nu Y_N \cos(\pi n) + \sum_{k=1}^{N-1} (jk)^\nu Y_k \underbrace{(e^{jk\theta_n} + e^{-jk\theta_n})}_{2\cos(k\theta_n)} \Big) \\ &= \underbrace{\frac{1}{M} \Big((j0)^\nu Y_0 + (jN)^\nu Y_N (-1)^n + 2 \sum_{k=1}^{N-1} (jk)^\nu Y_k \cos(\frac{\pi nk}{N}) \Big)}_{= \text{a DCT-I of } \vec{Y}^{(\nu)}_{\text{short}}!} \qquad (11) \end{aligned}$$

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`[19][20] 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 [21]. Note too that the DST-I only takes the $k = \{1, ...N-1\}$ elements, since sines will result in zero crossings at $k = 0$ and N (no informational content), whereas the DCT-I takes all $k = \{0, ...N\}$ 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 1^{st} derivative it looks like:

$$\frac{d}{dx}y(\theta) = \frac{d}{d\theta}y(\theta)\cdot\frac{d\theta}{dx} = \frac{\frac{d}{d\theta}y(\theta)}{\frac{dx}{d\theta}}, \quad \text{where } \frac{dx}{d\theta} = \frac{d}{d\theta}\cos(\theta) = -\sin(\theta)$$

$$\sin^2(\theta)+\cos^2(\theta) = 1 \longrightarrow \sin(\theta) = \pm\sqrt{1-\cos^2(\theta)} = \sqrt{1-x^2} \text{ using upper semicircle}$$
$$\longrightarrow -\sin(\theta) = -\sqrt{1-x^2} \longrightarrow \frac{d\theta}{dx} = \frac{1}{\frac{dx}{d\theta}} = \frac{-1}{\sqrt{1-x^2}} \quad \Big(\text{and also } \frac{d}{dx}\cos^{-1}(x) = \frac{-1}{\sqrt{1-x^2}}\Big)$$

The $\frac{d}{d\theta}y(\theta)$ a.k.a $y'(\theta)$ term is actually pretty easy to handle, because we know its value (and for higher orders too) at discretized θ_n from earlier (Equation 10 and Equation 11)! 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

Let's see it for the second derivative:

$$\begin{aligned}\frac{d^2}{dx^2}y(\theta) &= \frac{d}{dx}\frac{y'(\theta)}{-\sqrt{1-x^2}} = \frac{-\sqrt{1-x^2}\frac{d}{dx}y'(\theta) - y'(\theta)\frac{d}{dx}(-\sqrt{1-x^2})}{1-x^2} \\ &= \frac{-\frac{d}{d\theta}y'(\theta)\cdot\frac{d\theta}{dx}}{\sqrt{1-x^2}} - \frac{y'(\theta)\frac{x}{\sqrt{1-x^2}}}{1-x^2} = \frac{y''(\theta)}{1-x^2} - \frac{xy'(\theta)}{(1-x^2)^{3/2}} \\ \longrightarrow [\frac{d^2}{dx^2}y(\theta)]_n &= \frac{1}{1-x_n^2}\odot y''_n - \frac{x_n}{(1-x_n^2)^{3/2}}\odot y'_n\end{aligned} \tag{12}$$

Notice that the 2^{nd} derivative in x requires *both* the 1^{st} and 2^{nd} derivatives in θ! This pattern will turn out to hold for higher derivatives as well: For the ν^{th} derivative in x we require all derivatives up to order ν in θ.

Let's see a few more:

$$\begin{aligned}\frac{d^3}{dx^3}y(\theta) &= \frac{-1}{(1-x^2)^{3/2}}y'''(\theta) + \frac{3x}{(1-x^2)^2}y''(\theta) + \frac{-2x^2-1}{(1-x^2)^{5/2}}y'(\theta) \\ \frac{d^4}{dx^4}y(\theta) &= \frac{1}{(1-x^2)^2}y^{IV}(\theta) + \frac{-6x}{(1-x^2)^{5/2}}y'''(\theta) + \frac{11x^2+4}{(1-x^2)^3}y''(\theta) + \frac{-6x^3-9x}{(1-x^2)^{7/2}}y'(\theta) \\ \frac{d^5}{dx^5}y(\theta) &= \frac{-1}{(1-x^2)^{5/2}}y^{V}(\theta) + \frac{10x}{(1-x^2)^3}y^{IV}(\theta) + \frac{-35x^2-10}{(1-x^2)^{7/2}}y'''(\theta) + \frac{50x^2+55x}{(1-x^2)^4}y''(\theta) + \frac{-24x^4-72x^2-9}{(1-x^2)^{9/2}}y'(\theta)\end{aligned} \tag{13}$$

As we take higher derivatives, we can see a bit of a pattern. 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:

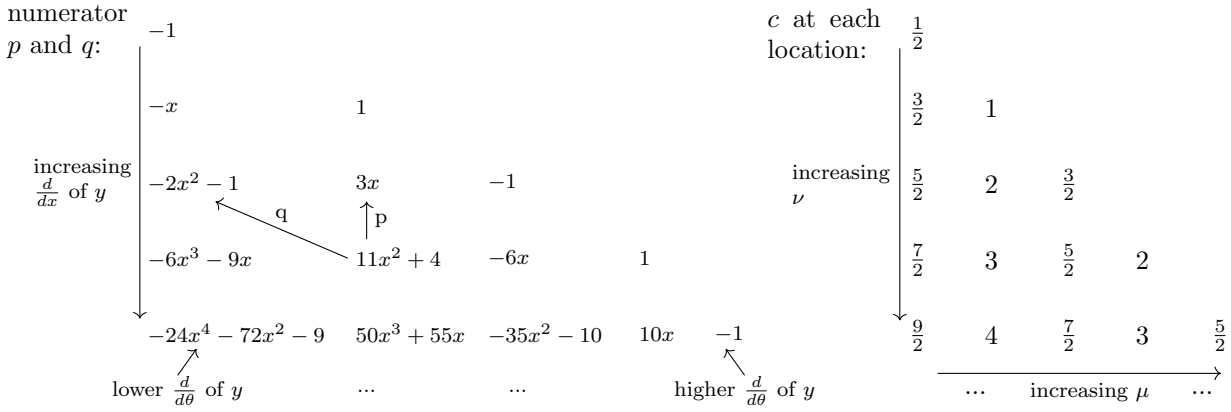
$$\frac{d}{dx}\Big(\frac{p(x)}{(1-x^2)^{c-1}}y^{(\mu)}(\theta) + \frac{q(x)}{(1-x^2)^{c-\frac{1}{2}}}y^{(\mu-1)}(\theta)\Big) =$$

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

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 the 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. You're welcome.

5.3.2 Handling Domain Endpoints

There's a problem in the above at the edges of the domain: If $x=\pm1$, 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 $x\to\pm1$, $\theta\to0,\pi$, sine will be 0! However, if we take even derivatives, then $\cos(0,\pi)\to1,-1$. Then, if we look closely at the derivatives in x (Equation 13), 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 for the 1^{st} derivative. Plugging in the DCT-I reconstruction (Equation 8) for $y(\theta)$:

$$\lim_{\substack{x\to\pm1\\\theta\to0,\pi}}\frac{d}{d\theta}\frac{1}{M}\Big(Y_0+Y_N\cos(N\theta)+2\sum_{k=1}^{N-1}Y_k\cos(k\theta)\Big)\cdot\frac{d\theta}{dx}=\lim_{\substack{x\to\pm1\\\theta\to0,\pi}}\frac{\frac{1}{M}\Big(-NY_N\sin(N\theta)-2\sum_{k=1}^{N-1}kY_k\sin(k\theta)\Big)}{-\sqrt{1-x^2}}$$

$$\xrightarrow[\frac{d}{dx}]{\frac{d}{dx}}=\lim_{\substack{x\to\pm1\\\theta\to0,\pi}}\frac{\frac{1}{M}\Big(-N^2Y_N\cos(N\theta)-2\sum_{k=1}^{N-1}k^2Y_k\cos(k\theta)\Big)\cdot\frac{-1}{\cancel{\sqrt{1-x^2}}}}{\frac{x}{\cancel{\sqrt{1-x^2}}}}=\lim_{\substack{x\to\pm1\\\theta\to0,\pi}}\frac{\frac{1}{M}\Big(N^2Y_N\cos(N\theta)+2\sum_{k=1}^{N-1}k^2Y_k\cos(k\theta)\Big)}{x}$$

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

And now let's do it for the 2^{nd} derivative, with some slightly more compact notation:

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

$$\to \frac{0}{0} \xrightarrow[\frac{d}{dx}]{\frac{d}{dx}} \frac{\cancel{y^{IV}(\theta)\frac{1}{\sqrt{1-x^2}}}\cancel{y''(\theta)\frac{x}{\sqrt{1-x^2}}}}{\frac{6x^2-3}{\cancel{\sqrt{1-x^2}}}} = \frac{1}{6x^2-3}(y^{IV}(\theta) + y''(\theta))$$

We already know $y''(\theta) = \frac{1}{M}\Big(-N^2Y_N\cos(N\theta) - 2\sum_{k=1}^{N-1}k^2Y_k\cos(k\theta)\Big)$. We can easily find

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

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

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

5.3.3 Endpoints for Higher Derivatives

We can do the above for higher derivatives too. However, in general finding the endpoints for the ν^{th} 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:

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

where the alternating plus and minus in the k and N terms comes from the fact the 2^{nd} derivative contains $-$cosines, the 4^{th} $+$cosines, the 6^{th} $-$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. But due to its complexity, I've opted to keep this full process out of the main package code. Instead, I've hard-coded the solutions up to 4^{th} order and raise a warning with reference to the notebook if users try to go higher.

It is possible to sidestep all these ballooningly-complicated variable mappings and higher derivative limit-evaluations by building up to higher derivatives from lower ones in stages (being careful to only use even derivatives if our end goal is even, so we preserve the Nyquist term). This would entail more forward transforms but actually no more inverse ones, since we have to remember the results of all the inverse transforms up to the ν^{th} order anyway.

6 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:

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

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

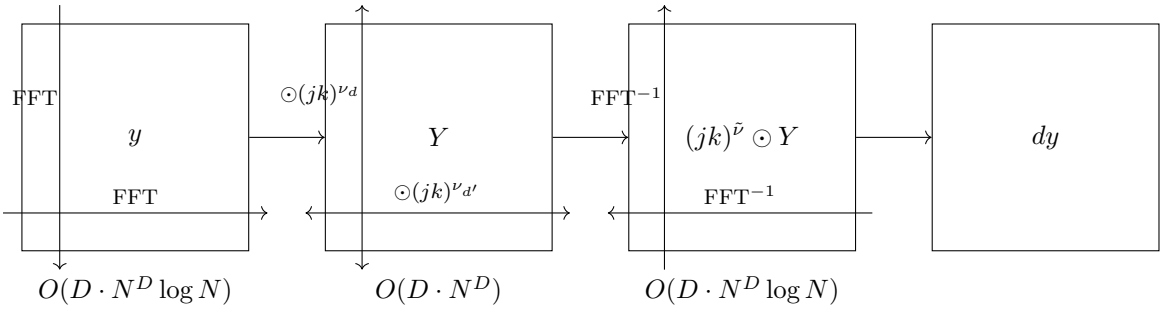
where i, j are indexers as in the computing sense and have nothing to do with the imaginary unit, `Algo` applies the algorithm to each vector along the dimension given by the third argument, and the 1^{st} and 2^{nd} in the second argument refer to the derivative order.

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.

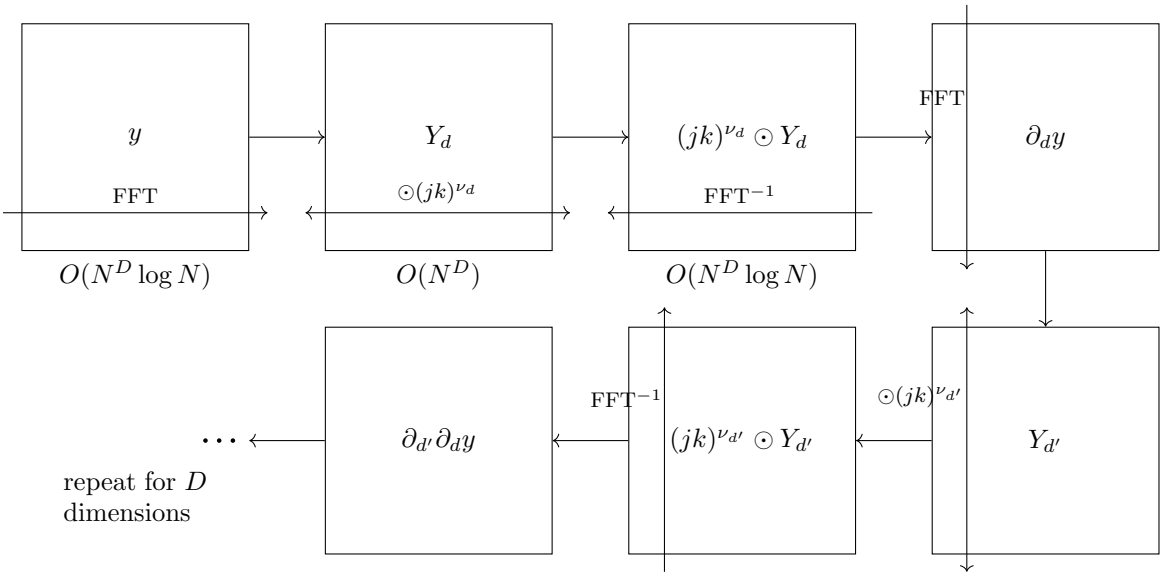
6.1 Dimensions Together versus In Series

But can we simplify the situation at all?

Due to the linearity of the Fourier transform, transforming along all dimensions, multiplying by appropriate $(jk)^\nu$ along corresponding dimensions of the transformed data, and then inverse transforming along all dimensions is equivalent to transforming, multiplying, and inverse transforming each dimension in series[18]:



That's really neat, but does it save us anything, really? Let's see it in series:



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.

6.2 Splintering

The above is good news, because converting back to the Chebyshev domain turns out to be terribly knotty and difficult in the simultaneous-multidimensional derivatives case, for several reasons.

First, the fact we need all inverse transforms in θ up to order ν to compute the derivative of order ν in x leads to *far* more terms in multiple dimensions, due to interactions. I'll demonstrate with the simplest possible case, 2^{nd} order in one dimension and 1^{st} order in another:

$$\frac{\partial^3}{\partial x_1^2 \partial x_2} y(\theta_1, \theta_2) = \frac{\partial}{\partial x_2}\Big[\frac{\partial^2}{\partial x_1^2} y(\theta_1, \theta_2)\Big] \tag{14}$$

Let's break this up and apply the multivariable chain rule[23] and product rule to just evaluate that inner portion first:

$$\begin{aligned}
\frac{\partial^2}{\partial x_1^2} y(\theta_1, \theta_2) &= \frac{\partial}{\partial x_1}\Big(\frac{\partial}{\partial \theta_1} y \cdot \frac{d\theta_1}{dx_1} + \frac{\partial}{\partial \theta_2} y \cdot \cancelto{0}{\frac{d\theta_2}{dx_1}}\Big) && \text{multivariable chain rule} \\
&= \frac{\partial}{\partial \theta_1} y \cdot \frac{d^2\theta_1}{dx_1^2} + \frac{\partial}{\partial x_1}\Big(\frac{\partial}{\partial \theta_1} y\Big) \cdot \frac{d\theta_1}{dx_1} && \text{product rule} \\
&= \frac{\partial}{\partial \theta_1} y \cdot \frac{d^2\theta_1}{dx_1^2} + \Big(\frac{\partial^2}{\partial \theta_1^2} y \cdot \frac{d\theta_1}{dx_1} + \frac{\partial^2}{\partial \theta_2 \partial \theta_1} y \cancelto{0}{\frac{d\theta_2}{dx_1}}\Big) \cdot \frac{d\theta_1}{dx_1} && \text{multivariable chain rule} \\
&= \frac{\partial}{\partial \theta_1} y \cdot \frac{d^2\theta_1}{dx_1^2} + \frac{\partial^2}{\partial \theta_1^2} y \cdot \Big(\frac{d\theta_1}{dx_1}\Big)^2
\end{aligned}$$

Now using this to evaluate Equation 14:

$$\begin{aligned}
\frac{\partial^3}{\partial x_1^2 \partial x_2} y(\theta_1, \theta_2) =& \frac{\partial}{\partial x_2}\Big[\frac{\partial}{\partial \theta_1} y \cdot \frac{d^2\theta_1}{dx_1^2} + \frac{\partial^2}{\partial \theta_1^2} y \cdot \Big(\frac{d\theta_1}{dx_1}\Big)^2\Big] \\
=& \frac{\partial}{\partial \theta_1} y \cdot \cancelto{0}{\frac{\partial}{\partial x_2}\frac{d^2\theta_1}{dx_1^2}} + \Big(\frac{\partial^2}{\partial \theta_1^2} y \cdot \cancelto{0}{\frac{d\theta_1}{dx_2}} + \frac{\partial^2}{\partial \theta_1 \partial \theta_2} y \cdot \frac{d\theta_2}{dx_2}\Big) \cdot \frac{d^2\theta_1}{dx_1^2} \\
&+ \frac{\partial^2}{\partial \theta_1^2} y \cancelto{0}{\frac{\partial}{\partial x_2}\Big(\frac{d\theta_1}{dx_1}\Big)^2} + \Big(\frac{\partial^3}{\partial \theta_1^3} y \cdot \cancelto{0}{\frac{d\theta_1}{dx_2}} + \frac{\partial^3}{\partial \theta_1^2 \partial \theta_2} y \cdot \frac{d\theta_2}{dx_2}\Big) \cdot \Big(\frac{d\theta_1}{dx_1}\Big)^2 \\
=& \frac{\partial^2}{\partial \theta_1 \partial \theta_2} y \cdot \frac{d\theta_2}{dx_2} \cdot \frac{d^2\theta_1}{dx_1^2} + \frac{\partial^3}{\partial \theta_1^2 \partial \theta_2} y \cdot \frac{d\theta_2}{dx_2} \cdot \Big(\frac{d\theta_1}{dx_1}\Big)^2
\end{aligned}$$

The double derivative in x_1 has *splintered* the expression into two, and then the single derivative in x_2 has interacted with *both* those terms.

All the terms that involve derivatives of a θ w.r.t. an x are ultimately just functions of x. In fact, $\frac{d\theta_2}{dx_2}$ is just our old friend $\frac{-1}{\sqrt{1-x_2^2}}$, and you can pick out $\left(\frac{d\theta_1}{dx_1}\right)^2$ and $\frac{d^2\theta_1}{dx_1^2}$ in Equation 12. Together they are *a Cartesian product* of the the 1D case!

Meanwhile, the ∂y terms have different orders, which means that to find them we need to multiply Y, the all-dimensions transform of y, by *different* orders of (jk). If we do this carefully, the best-case scenario is that we incur the same amount of work as the in-series case, but it takes some extra bookkeeping and data copying.

Most gnarly, at the *edges* of the domain we still need to use L'Hôpital's rule on an analytic expression to evaluate the limits of

$$\frac{\partial^{\sum \nu_i}}{\partial x_1^{\nu_1} ... \partial x_D^{\nu_D}} y(\theta_1, ...\theta_D)$$

This is made worse by the fact our analytic expression is based on the DCT-I, which has terms outside the central sum, so as we substitute the DCT-I in to the DCT-I, we get ever more terms (3^D of them), e.g. in 2D:

$$y(\theta_1, \theta_2) = \frac{1}{M^2}\Big[Y_{00} + Y_{N0}\cos(N\theta_1) + Y_{0N}\cos(N\theta_2) + Y_{NN}\cos(N\theta_1)\cos(N\theta_2)$$

$$+ 2\sum_{k_1=1}^{N-1} Y_{k_1 0}\cos(k_1\theta_1) + 2\sum_{k_2=1}^{N-1} Y_{0k_2}\cos(k_2\theta_2) + 2\cos(N\theta_2)\sum_{k_1=1}^{N-1} Y_{k_1 N}\cos(k_1\theta_1)$$
$$+ 2\cos(N\theta_1)\sum_{k_2=1}^{N-1} Y_{Nk_2}\cos(k_2\theta_2) + 4\sum_{k_1=1}^{N-1}\sum_{k_2=1}^{N-1} Y_{k_1 k_2}\cos(k_1\theta_1)\cos(k_2\theta_2)\Big]$$

We could generalize the original conception (Equation 7), which has a single sum with $a_0 = \frac{Y_0}{M}, a_k = \frac{2Y_k}{M}$ for $k \in [1, N-1], a_N = \frac{Y_N}{M}$, to get a $y(\vec{\theta})$ with only a single term, but this involves still more extra bookkeeping.

Worst of all, we then still have to take limits as different *combinations* of dimensions reach the edges, which becomes a combinatorial nightmare. This was already hard enough in 1D!

So although `numpy` does provide the `fftn` function for transforming in multiple dimensions at once, and `scipy` provides similar `dctn` and `dstn` functions, they wouldn't confer a computational-complexity benefit and would require the math and code to get massively more complicated, so I have chosen not to use them. From a user-friendliness perspective, I also judge it to be somewhat more confusing to specify multiple derivatives at once (although generalizing the `nu` and `axis` parameters to vectors is possible). For all these reasons, I have chosen to limit the package to taking derivatives along a single dimension at once.

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

7 Arbitrary Domains

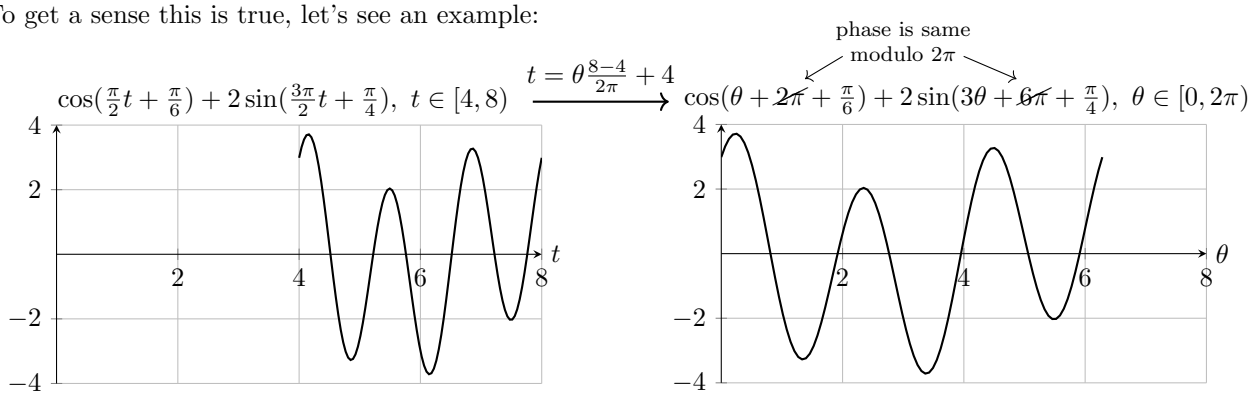
So far we've only used the domain $[0, 2\pi)$ in the Fourier case, because this is the domain assumed by the DFT, and the domain $[-1, 1]$ in the Chebyshev case, because this is the where a cosine wrapped around a cylinder casts a shadow. As you may have guessed, this hasn't curtailed the generality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

7.1 Fourier

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:

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

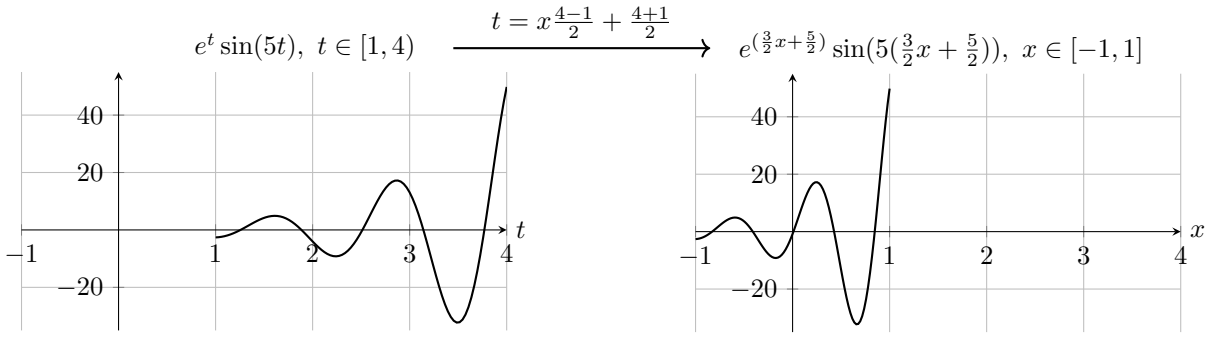
In other words, if we want to take a spectral derivative of a function that's periodic on $[a, b)$, then we need to sample it at `t_n = np.arange(M)/M * (b - a) + a = np.linspace(a, b, M, endpoint=False)`.

7.2 Chebyshev

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:

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

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.

In simple code terms, if we want to take a spectral derivative of a function on $[a, b]$, then we need to sample it at `t_n = np.cos(np.arange(N+1) * np.pi / N) * (b - a)/2 + (b + a)/2`.

7.3 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), the functions take the sample locations as a parameter.

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 into 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!"

In that spirit, with Patron Saint Hamming watching over us, let's continue: 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] Disintegration By Parts, Why do Fourier transforms use complex numbers?, https://math.stackexchange.com/a/1293127/278341

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

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

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

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

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

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

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

[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] The Fourier spectral method, https://atmos.washington.edu/~breth/classes/AM585/lect/DFT_FS_585_notes.pdf

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

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

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

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

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

[22] 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

[23] The Chain Rule for Multivariable Functions, https://math.libretexts.org/Bookshelves/Calculus/Calculus_(OpenStax)/14%3A_Differentiation_of_Functions_of_Several_Variables/14.05%3A_The_Chain_Rule_for_Multivariable_Functions