CS168: The Modern Algorithmic Toolbox
Lecture #15 and #16: The Fourier Transform and Convolution

Tim Roughgarden & Gregory Valiant*

May 21, 2024

1 Intro

Thus far, we have seen a number of different approaches to extracting information from data. This week, we will discuss the Fourier transform, and other related transformations that map data into a rather different space, while preserving the information. While the Fourier transform is certainly not modern, extremely impactful novel applications and adaptations of it continue to arise every year. We view it as an essential part of the modern algorithmic toolbox.

The Fourier transformation is usually presented as an operation that transforms data from the "time" or "space" domain, into "frequency" domain. That is, given a vector of data, with the ith entry representing some value at time i (or some value at location i), the Fourier transform will map the vector to a different vector, w, where the ith entry of w represents the "amplitude" of frequency i. Here, the amplitude will be some complex number.

We will revisit the above interpretation of the Fourier transformation. First, a quick disclaimer: one of the main reasons people get confused by Fourier transforms is because there are a number of seemingly different interpretations of the Fourier transformation. In general, it is helpful to keep all of the interpretations in mind, since in different settings, certain interpretations will be more natural than others.

2 The Fourier Transformation

We begin by defining the (discrete) Fourier transformation of a length n vector v. As will become apparent shortly, it will prove more convenient to think of vectors and matrices as

being 0-indexed. Hence $v = v_0, v_1, \ldots, v_{n-1}$.

Definition 2.1 Given a length n vector v, its Fourier transform $\mathcal{F}(v)$ is a complex-valued vector of length n defined as the product

$$\mathcal{F}(v) = M_n v,$$

where the $n \times n$ matrix M_n is defined as follows: Let $w_n = e^{-\frac{2\pi i}{n}} = \cos(\frac{2\pi}{n}) - i\sin(\frac{2\pi}{n})$ be an nth root of unit (i.e. $w_n^n = 1$). The j,kth entry of M_n is defined to be w_n^{jk}. [Note that M_n is 0-indexed.] Note that for any integer j, w_n^j is also an nth root of unity, since $(w_n^j)^n = ((w_n)^n)^j = 1^j = 1$.

Throughout, we will drop the subscripts on M_n and w_n, and just refer to matrix M and complex number w, though keep in mind that both are functions of the dimension of the vector in question: if we are referring to the Fourier transform of a length n vector, then M is $n \times n$, and $w = e^{-2\pi i/n}$.

Figure 1 plots M_{20}—the matrix corresponding to the Fourier transformation of length 20 vectors. Each entry of the matrix is represented by an arrow, corresponding to the associated complex number, plotted on the complex plain. The upper left entry of the matrix corresponds to $M_{20}(0,0) = w^0 = 1$, which is the horizontal unit vector in the complex plain. The entry $M_{20}(1,1) = w^1$ corresponds to the complex number that is 1/20th of the way around the complex unit circle. Please look at the Figure 1 until you understand exactly why each of the rows/columns looks the way it does. In the remainder of these notes, we will omit the subscript M_n and just write M, in settings where the dimensionality of the transform is clear from the context.

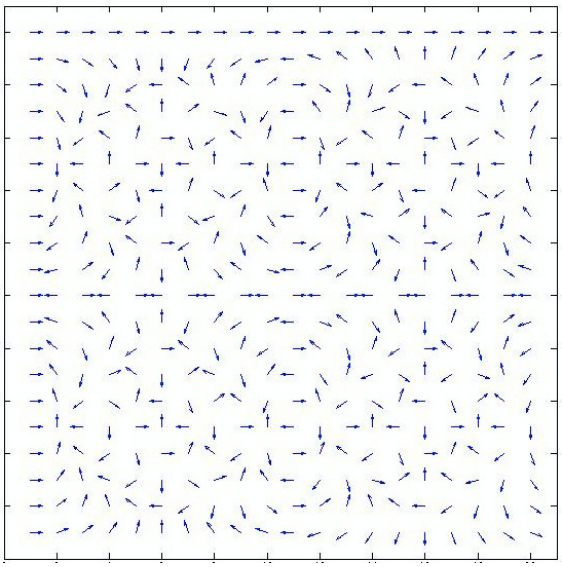

Figure 1: The matrix M_{20} corresponding to the Fourier transform of length 20 vector. Each arrow depicts the complex number corresponding to that location: for example the top row (entries $(0,0),(0,1),\ldots,(0,19)$ corresponds to the value 1, and the entry in location $(1,1)$ corresponds to the value $e^{-2\pi i/20}$.

We will see a few other interpretations of the Fourier transform, though if you ever get confused, you should always go back to the above definition: the Fourier transform of a vector v is simply the product Mv, for the matrix M as defined above.

2.1 Basic Properties

Perhaps the two most important properties of the Fourier transform are the following:

- The Fourier transformation of a length n vector can be computed in time $O(n \log n)$, using the "Fast Fourier Transformation" algorithm. This is a basic recursive divide/conquer algorithm, and is described in the last section of these notes. It might seem miraculous that we can actually multiply a vector by the $n \times n$ matrix M in much less time than it would take to even write M—the intuition behind why this is possible can be gleaned from Figure 1: M is a highly structured matrix, with lots of repeated structure (e.g. the top half looks rather similar to the bottom half).

- The Fourier transform is *almost* its own inverse. Letting M denote the transformation matrix as defined above,

$$MM = \begin{pmatrix} n & 0 & \ldots & 0 & 0 \\ 0 & \ldots & 0 & 0 & n \\ 0 & \ldots & 0 & n & 0 \\ \vdots & & \iddots & \vdots & \\ 0 & n & 0 & \ldots & 0 \end{pmatrix}.$$

Specifically, $w = M(Mv)$ is the same as v if one scales the entries by $1/n$, and then reverses the order of the last $n-1$ entries. In general, it is convenient to think of the inverse Fourier transform as being essentially the same as the Fourier transform; their properties are essentially identical.

Explicitly, the inverse Fourier transform is multiplication by the matrix M^{-1}, whose j,kth entry is $(M^{-1})_{j,k} = \frac{1}{n}w^{-jk} = \frac{1}{n}e^{2jk\pi i/n}$. From our definition, it is clear that $M^{-1}Mv = v$, and hence the inverse transform maps the transformed vector Mv back to v.

2.2 An alternate perspective

Given the form of the inverse transformation explained above, a different interpretation of the Fourier transformation of a vector v, is the vector of (complex-valued) coefficients $c_0, \ldots, c_{n-1}$ in the unique representation of v in the "Fourier basis". Namely, the coefficients satisfy

$$v = \sum_{j=0}^{n-1} c_j b_j,$$

where b_j is a length n vector defined by the jth column of M^{-1}, namely $b_j(k) = \frac{1}{n}e^{2\pi i \frac{jk}{n}}$. (The lack of a minus sign in the exponent is *not* a typo!!!) The uniqueness of this representation can be observed from the fact that the columns of M^{-1}, which are the vectors (b_j), are linearly independent (and, in fact, are orthogonal to each other).

What does this all mean? The jth column of M^{-1} is a vector whose entries go around the complex unit circle j times—you can think of this vector as representing the *frequency* j, whose real component consists of a cosine that cycles j times, and the complex portion consists of a sine that cycles j times. The jth coordinate of the Fourier transform Mv tells you how much of the jth frequency the signal, v has. In this sense, the Fourier transform gives the representation of v in the frequency domain, as a sum of different frequencies.

2.3 The Fourier Transform as Polynomial Evaluation and Interpolation

One of the more illuminating properties of the Fourier transform is that it can be regarded as *both* polynomial evaluation, and polynomial interpolation.

Given a length n vector v, define

$$P_v(x) = v_0 + v_1 x + v_2 x^2 + \ldots + v_{n-1}x^{n-1}$$

to be the polynomial associated to vector v. The kth entry of the transform Mv, is

$$v_0 + v_1 w^k + v_2 w^{2k} + v_3 w^{3k} + \ldots v_{n-1} w^{(n-1)k} = P_v(w^k).$$

Hence the Fourier transform of a vector v is simply the vector of evaluations of the associated polynomial $P_v(x)$, evaluated at the n complex roots of unity.

The fact that the transform can be computed in time $O(n \log n)$ should be a bit striking: it takes $O(n)$ operations to evaluate $P_v(x)$ at a single value—we are claiming that we can evaluate $P_v(x)$ at all n roots of unity in an amount of time that is only a logarithmic (as opposed to linear) factor more than the amount of time needed to make a single evaluation.

Now for the real magic: if the Fourier transform takes a vector representation of a polynomial, and returns the evaluation at the roots of unity, then the inverse transform takes a vector representing the values of P_v evaluated at the roots of unity, and returns the coefficients of the polynomial P_v. This is simply *polynomial interpolation*!!!

Recall from grade school that interpolating polynomials is often a headache; evaluating a polynomial at some number is easy—you just plug it in, and you are done. Polynomial interpolation usually involves writing out a big system of equations, and getting terms to cancel, etc—usually polynomial interpolation is much harder than polynomial evaluation. The fact that the Fourier transform is essentially its own inverse (and both the transform and its inverse can be computed in time $O(n \log n)$ means that interpolating a polynomial from its evaluation at the n roots of unity is easy (and is no harder than simply evaluating the polynomial at the roots of unity)!

This duality between polynomial evaluation and interpolation (at the roots of unity) provides the main connection between convolution and the Fourier transform, which we explore in Section 4.

3 The Fast Fourier Transformation

The Fast Fourier Transformation was first discovered by Gauss, though was largely forgotten, and was independently discovered by Cooley and Tukey in the 1950's. All we want to do is compute the product Mv, where M is the $n \times n$ Fourier transformation matrix whose j,kth entry is w^{jk} for $w = e^{-2\pi i/n}$ is an nth root of unity. This high level intuition why it might be possible to compute this product in much less time than it would take to even write out the matrix M, is because matrix M has tons of structure. In fact, M has so much structure that we will be able to compute Mv by essentially reusing portions of the computation.

The Fast Fourier transform (FFT) algorithm is a recursive algorithm. We will describe it in the case that $n = 2^k$ for some integer k. In the case that n is not a power of 2 similar ideas and some messy tricks still allow for fast computation, though we will not discuss these modifications.

To define the FFT, all we will do is rummage around inside the matrix M_n, and try to find some reoccurring copies of submatrices that resemble $M_{n/2}$—the matrix corresponding to the Fourier transformation of a vector of length $n/2$. Figure 2 depicts M_{16}, with the entries color coded for ease of reference.

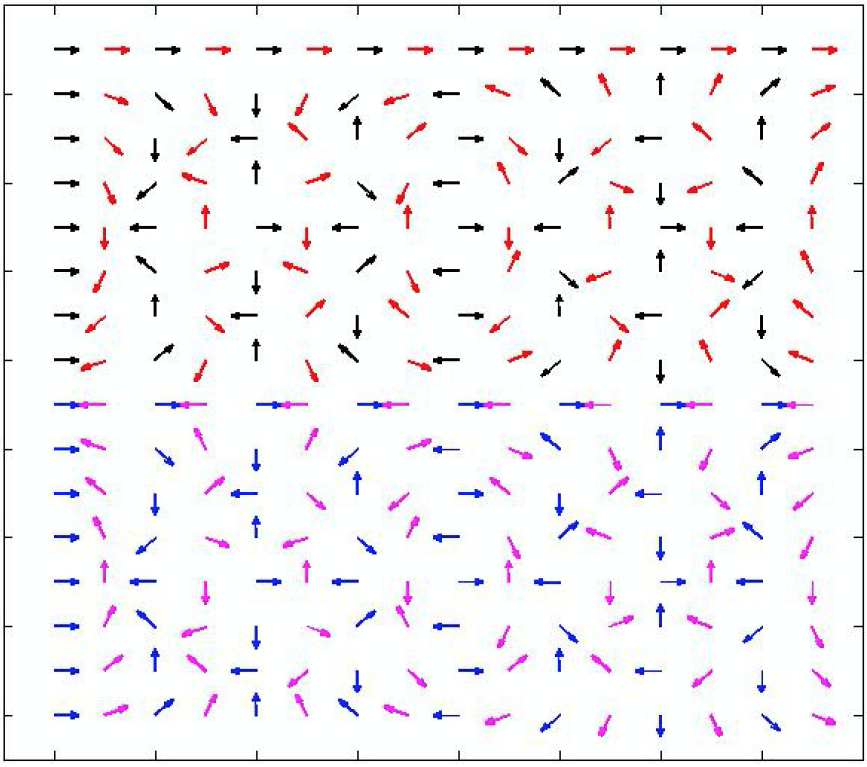

Figure 2: A plot depicting the matrix M_{16} corresponding to the Fourier transform of length 16 vectors. The entries have been color coded to illustrate that M_{16} contains 4 submatrices (one in each color) that can be related to M_8).

We will now "rummage" around in M_n and identify four submatrices that all resemble $M_{n/2}$. If $w_n = e^{-2\pi i/n}$, then $w_n^2 = w_{n/2}$. Hence it follows that the even indexed columns of the first half of the rows of M_n *exactly* form the matrix $M_{n/2}$. [For example, the black entries of Figure 2 exactly correspond to M_8.] Additionally, the even columns of the second half of the rows (i.e. the blue elements) are identical to the even columns of the first half of the rows (i.e. the black elements), and hence are also a copy of $M_{n/2}$. Now for the odd columns: observe that the jth odd column of M_n is equal to the jth even column, just with the ith entry scaled by w_n^i.

Based on the above, we have shown the following: the first $n/2$ indices of the Fourier transform of a length n vector v can be computed as $\mathcal{F}(v(evens)) + \mathcal{F}(v(odds)) \times s$, where s is the vector whose ith entry is w_n^i, and "$\times$" denotes element-wise multiplication. Similarly, the second half of the indices of the Fourier transform $\mathcal{F}(v)$ can be computed as $\mathcal{F}(v(evens)) - \mathcal{F}(v(odds)) \times s$.

The crucial observation is that the above decomposition only involves the computation of two Fourier transforms of length $n/2$ vectors, and two element-wise multiplications and additions. Letting $T(n)$ denote the time it will take to compute the Fourier transform of a length n vector, we have shown that the above recursive procedure will result in an algorithm

whose runtime satisfies

$$T(n) = 2T(n/2) + O(n).$$

Solving this recurrence relation yields that $T(n) = O(n \log n)$.

For clarity, we restate the recursive algorithm derived from the above decomposition:

Algorithm 1
FFT

Given a length $n = 2^k$ vector v, we output its Fourier transformation.

- Define $v_{even} = (v(0), v(2), v(4), \ldots, v(n-2))$ and $v_{odd} = (v(1), v(3), \ldots, v(n-1))$.
- Recursively compute $q_1 = \mathcal{F}(v_{even})$ and $q_2 = \mathcal{F}(v_{odd})$, both Fourier transformations of length $n/2$ vectors.
- Define the length $n/2$ vector s whose jth index is $e^{-2\pi ij/n}$.
- Output the concatenation of $q_1 + (q_2 \times s)$ and $q_1 - (q_2 \times s)$, where $q \times s$ is the element-wise product of the two vectors.

4 Convolution

Convolution is an incredibly useful tool that is closely intertwined with the Fourier transform. There are a number of interpretations/definitions, though perhaps the most insightful is via polynomial multiplication.

Definition 4.1 The convolution of two vectors, v, w of respective lengths n and m is denoted $v * w = w * v$, and is defined to be the vector of coefficients of the product of the polynomials associated to v and w, $P_v(x)P_w(x)$.

Example 4.2 Let $v = [1, 2, 3], w = [4, 5]$, then their convolution $v * w = [4, 13, 22, 15]$ representing the fact that $P_v(x) = 1 + 2x + 3x^2$, $P_w(x) = 4 + 5x$, and their product $(1 + 2x + 3x^2)(4 + 5x) = 4 + 13x + 22x^2 + 15x^3$.

The following fact—that convolutions can be computed incredibly quickly via the Fast Fourier transformation—is the main reason that convolutions are as useful and pervasive as they are.

Fact 4.3 *The convolution $v * w$ can be expressed as the inverse Fourier transformation of the component-wise product of the Fourier transformations of v and w:*

$$v * w = \mathcal{F}^{-1}\left(\mathcal{F}(v) \times \mathcal{F}(w)\right),$$

where "$\times$" denotes the element-wise product of the two vectors.

This fact follows immediately from the polynomial interpolation/evaluation view of the Fourier transform: The product of the evaluations of P_v and P_w will be the evaluations of the polynomial corresponding to the product of P_v and P_w, hence the interpolation of these evaluations will give the coefficients of this product, namely the convolution $v * w$. Of course, in order for this to make sense, we need to ensure that the number of evaluation points in larger than the degree of $P_v(x)P_w(x)$: i.e. if v and w are length n vectors, then we should pad them with n zeros, and compute their Fourier transforms as length $2n$ vectors.

Another nice property of Fact 4.3 is that it implies that convolutions can be computed in time $O(n \log n)$, and that they can also be inverted (provided the inverse exists) in time $O(n \log n)$. To clarify, if we are given a vector $q = v * w$ for some known vector w, we can invert this operation and find v. In particular, we have

$$v = \mathcal{F}^{-1}\left(\mathcal{F}(q)./\mathcal{F}(w)\right),$$

where "./" denotes component-wise division. We will see some applications of this in the next section.

4.1 Examples

Example 4.4 Consider Figure 3 representing some vector v, and let w be a Gaussian (see Figure 4). The Fourier transform of a Gaussian is a Gaussian—the transform of a skinny Gaussian is a fat Gaussian, and vice versa. Figure 5 depicts the convolution $v * w$. Look at this figure, and the definition of convolution, until it makes sense to you.

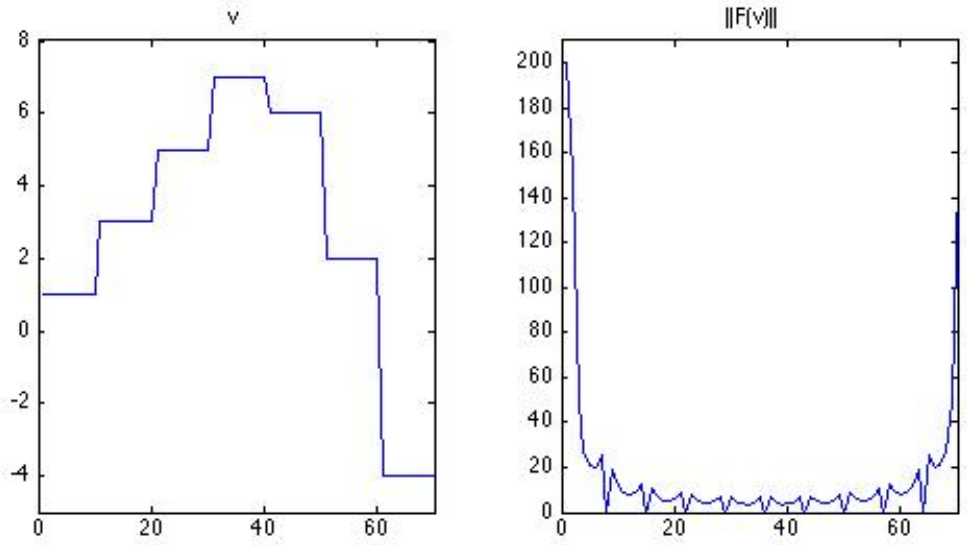

Figure 3: The vector v and the magnitude of its Fourier transform, $||\mathcal{F}(v)||$.

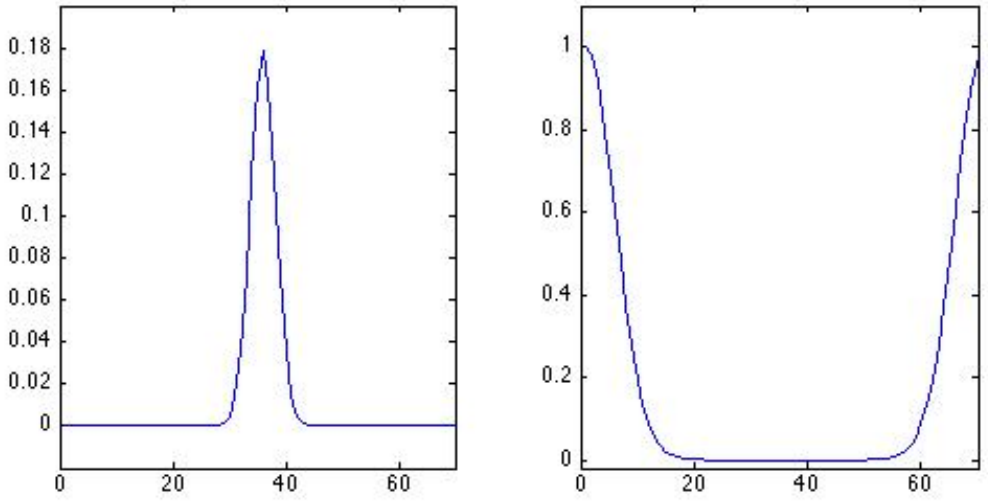

Figure 4: A plot of the Gaussian filter w and the magnitude of its Fourier transform. Note that the Fourier transform of a Gaussian is Gaussian (though, in this case, you need to put the two sides of the plot together to get the Gaussian).

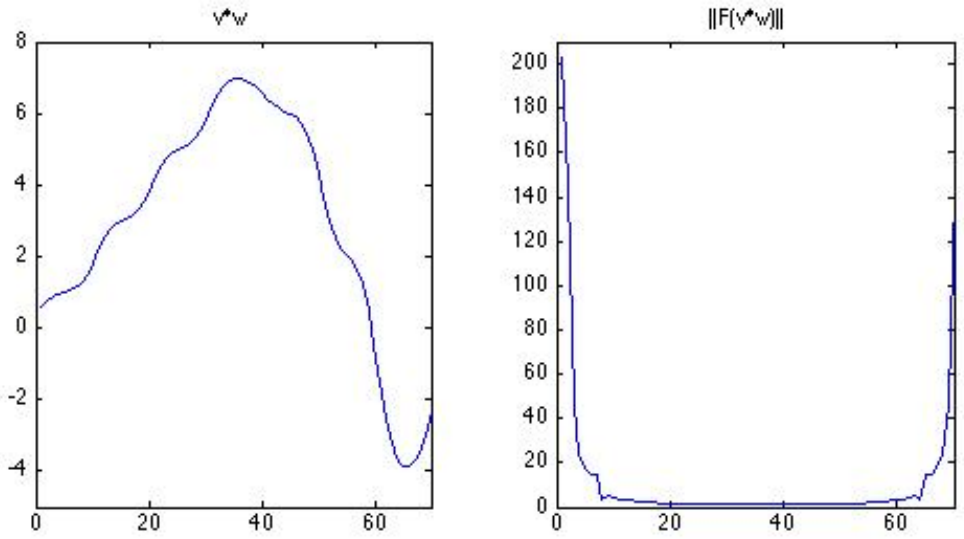

Figure 5: The convolution $v * w$, and the magnitude of its Fourier transform $||\mathcal{F}(v * w)|| = \mathcal{F}(v) \times \mathcal{F}(w)$, where "$\times$" denotes the component-wise product.

Example 4.5 There are many setting where we might want to "invert" a convolution. In class, we saw a few examples where we are given y—a motion-blurred version of an image

v, and wish to recover the un-blurred v. If we know that the motion-blur results from convolution with some w, then we can accomplish this de-blurring by $v = \mathcal{F}^{-1}\left(\mathcal{F}(q)./\mathcal{F}(w)\right)$. As we saw in class, for some types of blurring operations, such as convolution by a Gaussian, $\mathcal{F}(w)$ will have some extremely small entries, in which case this recovery will be numerically unstable, and will not be effective if there is some extra noise that has been added to the image. For other blurring operations such as motion blurs, $\mathcal{F}(w)$ will be well-behaved, and this de-convolution will be fairly robust to noise.

4.2 Convolutions Everywhere!

Given a vector a, consider the transformation that takes a vector v, and returns the convolution $v * a$. For every vector a, this transformation has two crucial properties:

- Linearity: For any vectors v and w,

  $$(v + w) * a = (v * a) + (w * a),$$

  and for any constant c, $(cv) * a = c(v * a)$.

- Translational invariance: For any vector v, if one "shifts" it to the right by padding it with k zeros, then the convolution of the shifted v with a will be the same as if one shifted the convolution of v and a. For example, for $v = [1, 2, 3]$, $a = [4, 5]$, $v * a = [4, 13, 22, 15]$, and if we shift v to get $v_s = [0, 0, 0, 0, 1, 2, 3]$, then $v_s * a = [0, 0, 0, 0, 4, 13, 22, 15]$, namely the shifted vector $v * a$.

While the above two properties are easily seen to hold for convolution, it turns out that they *define* convolution:

Fact 4.6 *Any transformation of a vector that is linear, and translation invariant is a convolution!*

To appreciate the above fact, we will discuss two examples of natural transformations for which we can fruitfully leverage the fact that they are convolutions.

Example 4.7 Nearly every physical force is linear, and translation invariant. For example, consider gravity. Gravity is linear (i.e. one can just add up the effects of different gravitational fields), and it is translation invariant (i.e. if I move 10 feet to the left, the gravitational field that my mass exerts gets translated 10 feet to the left, but is otherwise invariant). Hence the above fact shows that gravity is a convolution. We just need to figure out what to convolve by.

For the purposes of this example, we will work in 2 dimensions. Recall that the gravitational force exerted on a point at location x, y by a unit of mass at the origin has magnitude $\frac{1}{x^2+y^2}$, and is in the direction $\left(\frac{-x}{\sqrt{x^2+y^2}}, \frac{-y}{\sqrt{x^2+y^2}}\right)$. Hence, given a description of the density of some (2-d) asteroid, we can simply convolve this density with the function

$f_X(x,y) = \frac{-x}{(x^2+y^2)^{1.5}}$ to compute the X-coordinate of the force of gravity everywhere that results from the given density plot. Similarly for the Y-coordinate of the gravitational force. The Figure 6 depicts the density of an asteroid in space. Figure 7 depicts the force of gravity, whose X component is the convolution of the asteroid by the function $f_X(x,y) = \frac{-x}{(x^2+y^2)^{1.5}}$ and the Y component is the convolution of the asteroid by the function $f_Y(x,y) = \frac{-y}{(x^2+y^2)^{1.5}}$.

Note that since the computations are convolutions, they can be computed in time $O(n \log n)$, where n is the number of points in our grid. Naively, one might have thought that we would have needed to spend time $O(n^2)$ to compute the gravitational effect on every point caused by every other point. This illustrates one reason why convolutions and fast fourier transformations are so important to the efficient simulation of many physical systems.

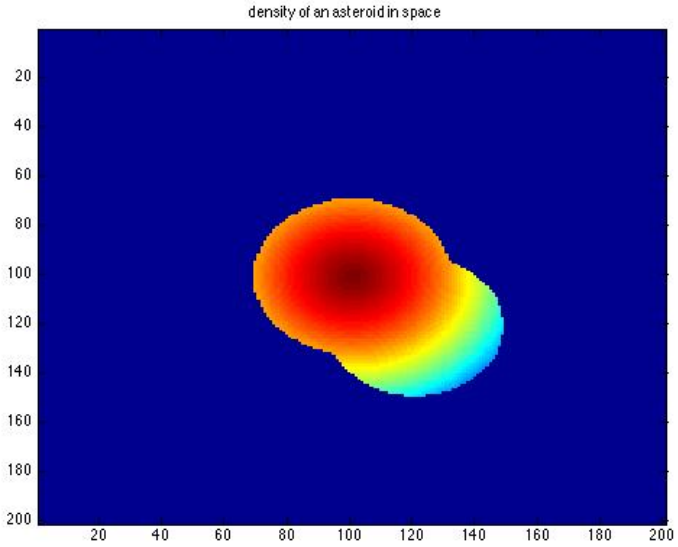

Figure 6: A plot depicting the density of an imaginary asteroid in space (blue denotes 0 density, dark red denotes highest density).

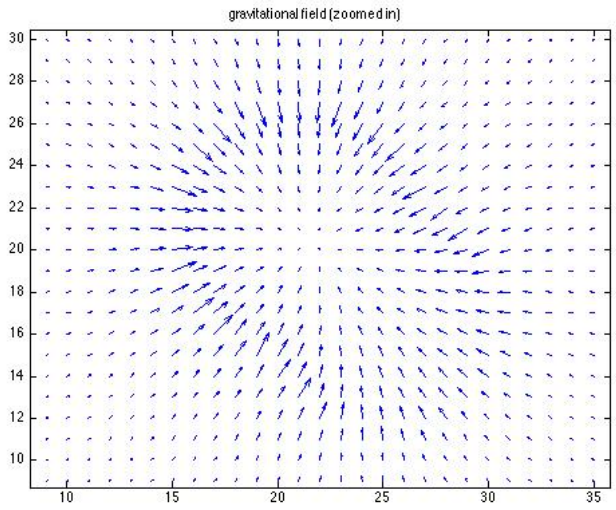

Figure 7: A zoomed-in depiction of the force of gravity due to the density plot of Figure 6. This can be computed as the convolution of the density function with the X and Y components of the force of gravity experienced at a point (x, y) due to a point mass at the origin, given by $f_X(x, y) = \frac{-x}{(x^2+y^2)^{1.5}}$ and $f_Y(x, y) = \frac{-y}{(x^2+y^2)^{1.5}}$. Recall that convolutions can be computed in near-linear time, via the fast fourier transform.

If we were given the gravitational field depicted in Figure 7, we could invert the convolution to derive the density of the asteroid. This inversion would could also be computed just as efficiently using fast Fourier transformations because $v = \mathcal{F}^{-1}\left(\mathcal{F}(v * w)./\mathcal{F}w\right)$, where "./" denotes element-wise division.

Example 4.8 To see a second illustration of the power of Fact 4.6, note that differentiation is also both linear, and translation invariant (i.e. if you shift a function over to the right by 3 units, then you also shift its derivative over by 3 units). Since differentiation is linear and translation invariant, it is a convolution, we just need to figure out what it is a convolution of. Since differentiation only really applies to continuous functions, and we have been talking about discrete Fourier transformations, thus far, we will be somewhat informal. Consider a vector v with n components, and imagine it as a function $v(x)$, defined for $x = 0, 1, \ldots, n-1$. Recall that

$$v(x) = q(0) + q(1)e^{2\pi i \frac{x}{n}} + q(2)e^{2\pi i \frac{2x}{n}} + q(3)e^{2\pi i \frac{3x}{n}} + \ldots + q(n-1)e^{2\pi i \frac{2(n-1)x}{n}}.$$

If we differentiate, ignoring that x is supposed to only be defined on the integers, we get:

$$v'(x) = 0 \cdot q(0) + i \cdot \frac{2\pi}{n} q(1)e^{2\pi i \frac{x}{n}} + (2i) \cdot \frac{2\pi}{n} q(2)e^{2\pi i \frac{2x}{n}} + (3i) \cdot \frac{2\pi}{n} q(3)e^{2\pi i \frac{3x}{n}} + \ldots + (n-1)i \cdot \frac{2\pi}{n} q(n-1)e^{2\pi i \frac{(n-1)x}{n}}.$$

Namely, the jth entry of the Fourier transform of the derivative of v is obtained by multiplying the jth entry of the Fourier transform of v by ij, and then scaling by the constant $2\pi/n$.

If we want to take a second or third derivative, we simply convolve again (and again). Recall that we can also invert convolutions just as computationally efficiently as we can compute them. In this case, this will let us solve *differential equations* extremely efficiently. For example, suppose we have some 2-d image depicting some density of space, $f(x, y)$. We can very efficiently compute the sum of the second derivatives in the x and y direction: $g(x, y) = \frac{d^2 g(x,y)}{dx^2} + \frac{d^2 g(x,y)}{dy^2}$. Given $g(x, y)$, we can also just as easily invert this convolution to obtain $f(x, y)$ (up to an additive constant). This corresponds to solving the Poisson equation (a second-order differential equation) that arises in several places in physics.

5 Beyond the Fourier Transform: Wavelets and Other Bases

- Fourier transforms are just one possible change-of-basis, and it is natural to ask whether there are other sorts of transformations that have similar nice properties (such as being computed in near-linear time).

- One motivation for considering other transformations is motivated by compressing audio signals. As we saw in class, some signals (such as singing or whistling), are "dense" in the time domain, but sparse (and thus easily compressible) in the frequency domain (i.e. the Fourier transform is fairly sparse). Other signals, such as percussive noises (the clapping example from class) are sparse in the time domain, but dense in the frequency domain (the Fourier transform looked extremely complicated and was NOT sparse). This observation that signals that are sparse in one domain transform to dense signals in the other, is not a coincidence: the basis functions corresponding to the frequency domain are localized in the frequency domain, but not in time, and the basis functions in the standard time domain are localized in time, but not in the frequency domain. From a compression standpoint, this motivates the *wavelet basis*, in which each basis function is localized BOTH in the time and frequency domain; hence representing a signal in this basis decomposes the signal into a linear combination of 'pieces' (with each piece corresponding to one of the basis functions), where each piece is localized in time and frequency.

- This is, of course, just the tip of the iceberg....