Fourier transforms and convolution
(without the agonizing pain)

CS/CME/BioE/Biophys/BMI 279
Oct. 22, 2019
Ron Dror
Outline

• Why do we care?

• Fourier transforms
  – Writing functions as sums of sinusoids
  – The Fast Fourier Transform (FFT)
  – Multi-dimensional Fourier transforms

• Convolution
  – Moving averages
  – Mathematical definition
  – Performing convolution using Fourier transforms

FFT and convolution is everywhere! For our purposes, these methods are used frequently in image analysis, a fundamental component of the experimental pipeline.

FFTs along multiple variables - an image for instance, would encode information along two dimensions, x and y.
Why do we care?
Why study Fourier transforms and convolution?

- In the remainder of the course, we’ll study several methods that depend on analysis of images or reconstruction of structure from images:
  - Light microscopy (particularly fluorescence microscopy) 2D imaging
  - Electron microscopy (particularly for single-particle reconstruction) Atomic-level structure resolution
  - X-ray crystallography Traditionally how we determine protein structure.
- The computational aspects of each of these methods involve Fourier transforms and convolution Latter two methods require reconstruction of 3D structures from numerous images taken
- These concepts are also important for:
  - Some approaches to ligand docking (and protein-protein docking)
  - Fast evaluation of electrostatic interactions in molecular dynamics
  - (You’re not responsible for these additional applications)

^ Protein-protein interactions can be analyzed by using convolution to analyze shape complementation, FFTs are used to perform convolutions quickly
Fourier transforms
Fourier transforms

Writing functions as sums of sinusoids

Motivation: rather than writing a function as an array of n values, we can approximate the function using amplitude and phase information for a number of sinusoids.
Writing functions as sums of sinusoids

• Given a function defined on an interval of length $L$, we can write it as a sum of sinusoids whose periods are $L, L/2, L/3, L/4, \ldots$ (plus a constant term)

  Each sinusoid is centered at 0

  v This term determines the average or offset value
Writing functions as sums of sinusoids

- Given a function defined on an interval of length $L$, we can write it as a sum of sinusoids whose periods are $L, L/2, L/3, L/4, \ldots$ (plus a constant term)

Original function

sum of 49 sinusoids (plus constant term)

With more sinusoids, and thus sinusoids with higher frequency, we get closer and closer to the original function

sum of 50 sinusoids (plus constant term)
Writing functions as sums of sinusoids

- Each of these sinusoidal terms has a magnitude (scale factor) and a phase (shift).

Original function

![Graph of the original function](image1)

Sum of sinusoids below

![Graph of the sum of sinusoids](image2)

You need n/2 sinusoids to exactly capture a n value dataset

Sinusoids with higher frequencies can resolve high-frequency features like sharp edges

Adjusting magnitude and phase allow each sinusoid to be fine-tuned to best reconstruct the desired wave

<table>
<thead>
<tr>
<th>Function</th>
<th>Magnitude</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>f(x) = -0.3</td>
<td>-0.3</td>
<td>0 (arbitrary)</td>
</tr>
<tr>
<td>f(x) = 1.9cos(2pi*0.01x-0.94)</td>
<td>1.9</td>
<td>-0.94</td>
</tr>
<tr>
<td>f(x) = 0.27cos(2pi*0.02x-1.4)</td>
<td>0.27</td>
<td>-1.4</td>
</tr>
<tr>
<td>f(x) = 0.39cos(2pi*0.03x-2.8)</td>
<td>0.39</td>
<td>-2.8</td>
</tr>
</tbody>
</table>
Expressing a function as a set of sinusoidal term coefficients

- We can thus express the original function as a series of magnitude and phase coefficients
  - If the original function is defined at \( N \) equally spaced points, we’ll need a total of \( N \) coefficients
  - If the original function is defined at an infinite set of inputs, we’ll need an infinite series of magnitude and phase coefficients—but we can approximate the function with just the first few

\(^\text{N/2 magnitude coefficients, N/2 phase coefficients}\)

\(^\text{This is because very high frequency sinusoids tend to contribute less to the overall function structure.}\)

<table>
<thead>
<tr>
<th>Constant term</th>
<th>Sinusoid 1</th>
<th>Sinusoid 2</th>
<th>Sinusoid 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(frequency 0)</td>
<td>(period L, frequency 1/L)</td>
<td>(period L/2, frequency 2/L)</td>
<td>(period L/3, frequency 3/L)</td>
</tr>
<tr>
<td>Magnitude: -0.3</td>
<td>Magnitude: 1.9</td>
<td>Magnitude: 0.27</td>
<td>Magnitude: 0.39</td>
</tr>
<tr>
<td>Phase: 0 (arbitrary)</td>
<td>Phase: -.94</td>
<td>Phase: -1.4</td>
<td>Phase: -2.8</td>
</tr>
</tbody>
</table>
Using complex numbers to represent magnitude plus phase

- We can express the magnitude and phase of each sinusoidal component using a complex number.

![Diagram showing complex number notation: $a + bi$.](image)

- **Magnitude:** length of blue arrow
- **Phase:** angle of blue arrow
Using complex numbers to represent magnitude plus phase

- We can express the magnitude and phase of each sinusoidal component using a complex number.
- Thus we can express our original function as a series of complex numbers representing the sinusoidal components.
- This turns out to be more convenient (mathematically and computationally) than storing magnitudes and phases.
The Fourier transform

• The Fourier transform maps a function to a set of complex numbers representing sinusoidal coefficients. Generally we take the function, represented in time or position, and then convert the function into values over a frequency. In the frequency domain, we plot the magnitude of each sine wave by the frequency.
  – We also say it maps the function from “real space” to “Fourier space” (or “frequency space”)
  – Note that in a computer, we can represent a function as an array of numbers giving the values of that function at equally spaced points.

• The inverse Fourier transform maps in the other direction. A common use for an inverse FFT would be to convert a function back into the time-domain after performing a convolution using FFT.
  – It turns out that the Fourier transform and inverse Fourier transform are almost identical. A program that computes one can easily be used to compute the other.
Why do we want to express our function using *sinusoids*?

- Sinusoids crop up all over the place in nature
  - For example, sound is usually described in terms of different frequencies
  - By chance, x-ray crystallography images happen to be FFTs of the 3D structure

- Sinusoids have the unique property that if you sum two sinusoids of the same frequency (of any phase or magnitude), you always get another sinusoid of the same frequency
  - This leads to some very convenient computational properties that we’ll come to later

**Practical uses of sinusoid representation**
- Data compression
- Filtering frequency bands
- Identify key frequencies

**Sinusoids are eigenfunctions of linear system**
- ie, you can express the system as a sum of the eigenfunctions, each multiplied by some scalar
- Every eigenfunction is orthogonal to each other
Fourier transforms

The Fast Fourier Transform (FFT)
The Fast Fourier Transform (FFT)

- The number of arithmetic operations required to compute the Fourier transform of \( N \) numbers (i.e., of a function defined at \( N \) points) in a straightforward manner is proportional to \( N^2 \).
- Surprisingly, it is possible to reduce this \( N^2 \) to \( N \log N \) using a clever algorithm.
  - This algorithm is the Fast Fourier Transform (FFT).
  - It is arguably the most important algorithm of the past century.
  - You do not need to know how it works—only that it exists.

JPEG images store images using an FFT. Specifically, to save space, JPEGs store lower frequency components more accurately than higher frequency components. This works well generally, but can be poor for images with high frequencies, including images with text and many borders/hard-edges.
Fourier transforms

Multidimensional Fourier Transforms
Images as functions of two variables

- Many of the applications we’ll consider involve images
- A grayscale image can be thought of as a function of two variables
  - The position of each pixel corresponds to some value of \( x \) and \( y \)
  - The brightness of that pixel is proportional to \( f(x,y) \)

ex. We store the pixel value at each \( x,y \) position in the image
For rgb, we simply store three values (RGB) at each pixel location
Two-dimensional Fourier transform

• We can express functions of two variables as sums of sinusoids
• Each sinusoid has a frequency in the $x$-direction and a frequency in the $y$-direction
• We need to specify a magnitude and a phase for each sinusoid
• Thus the 2D Fourier transform maps the original function to a complex-valued function of two frequencies

$$f(x, y) = \sin(2\pi \cdot 0.02x + 2\pi \cdot 0.01y)$$

To examine frequency in one direction, you can fix the value of the other direction

2 values of frequency, one along the $x$-axis, one along $y$-axis
Three-dimensional Fourier transform

• The 3D Fourier transform maps functions of three variables (i.e., a function defined on a volume) to a complex-valued function of three frequencies

• 2D and 3D Fourier transforms can also be computed efficiently using the FFT algorithm

Here, our sinusoid will be a 3D sinusoid along x,y,z

To compute an FFT along 3 dimensions, you compute FFT along one-dimension at a time.
Convolution
Convolution

Moving averages
A convolution is basically a *weighted moving average*

- We’re given an array of numerical values
  - We can think of this array as specifying values of a function at regularly spaced intervals
- To compute a moving average, we replace each value in the array with the average of several values that precede and follow it (i.e., the values within a *window*)
- We might choose instead to calculate a *weighted moving average*, where we again replace each value in the array with the average of several surrounding values, but we weight those values differently
- We can express this as a *convolution* of the original function (i.e., array) with another function (array) that specifies the weights on each value in the window
Convolve: apply a convolution function along some window/ subset of the original function

This convolution function specifies the weights that you use to compute your average

Ex.

Conv Function =
1/3 for x_i-1
1/3 for x_i
1/3 for x_i+1

Here, we slide our convolution function along 3-points along the original function. The output value of the convolution for a given point x_i in our original function will be an average of x_i and it’s two adjacent neighbors

Ex. 2

It might make more sense to apply a larger weight to the current point x_i, so we might have a convolution function as follows:

Conv Function =
1/4 for x_i-1
1/2 for x_i
1/4 for x_i+1

Weighting values are selected based on the application. Notice how we normalize the weightings so that the overall scaling of the original function does not change (ie the sum or average of the original function stays the same)
Example

\[f \ast g \text{ the same result as } g \ast f\]

To see why, think about how the output values will be calculated; you will be applying the same multiplication and summation of terms for \(f \ast g\) and \(g \ast f\), independent of which function comes first.
Convolution

Mathematical definition
Convolution: mathematical definition

- If $f$ and $g$ are functions defined at evenly spaced points, their convolution is given by:

$$ (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] $$
Convolution

Multidimensional convolution
Two-dimensional convolution

• In two-dimensional convolution, we replace each value in a two-dimensional array with a weighted average of the values surrounding it in two dimensions
  – We can represent two-dimensional arrays as functions of two variables, or as matrices, or as images

For a 2D convolution, rather than specifying a vector of weights, we specify a matrix of weights
Two-dimensional convolution: example

Here g applies an averaging window, in which the most heavily weighted value is the current pixel/location. The output image appears blurred, as neighboring pixels also contribute to the output pixel value, diluting the edges of the image.
Multidimensional convolution

- The concept generalizes to higher dimensions
- For example, in three-dimensional convolution, we replace each value in a three-dimensional array with a weighted average of the values surrounding it in three dimensions

For an N-dimension convolution, you would specify the weights for the convolution using an N-dimension array
Convolution

Performing convolution using Fourier transforms
Relationship between convolution and Fourier transforms

• It turns out that convolving two functions is equivalent to *multiplying* them in the frequency domain
  – One multiplies the complex numbers representing coefficients at each frequency

• In other words, we can perform a convolution by taking the Fourier transform of both functions, multiplying the results, and then performing an inverse Fourier transform

To calculate the output of the convolution, we calculate the FFT of f and g, and then multiply the corresponding coefficients at each frequency. We will then calculate an inverse FFT to revert our output from the frequency domain back into the time/location domain.
Why does this relationship matter?

• First, it allows us to perform convolution faster
  – If two functions are each defined at \( N \) points, the number of operations required to convolve them in the straightforward manner is proportional to \( N^2 \)
  – If we use Fourier transforms and take advantage of the FFT algorithm, the number of operations is proportional to \( N \log N \). N\log N\ operations are significantly faster than \( N^2 \) operations as \( N \) gets large

• Second, it allows us to characterize convolution operations in terms of changes to different frequencies
  – For example, convolution with a Gaussian will preserve low-frequency components while reducing high-frequency components
    This in turn allows tuning of which frequencies we want to keep/remove