Fourier transforms and convolution

(without the agonizing pain)

CS/CME/Biophys/BMI 279

Oct. 27 and 29, 2015

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

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)
  - Electron microscopy (particularly for single-particle reconstruction)
  - X-ray crystallography
- The computational aspects of each of these methods involve Fourier transforms and convolution
- 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)

Fourier transforms

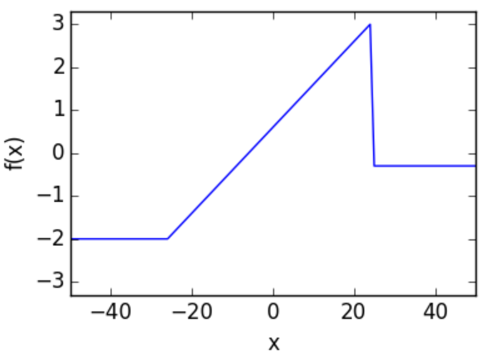
Fourier transforms

Writing functions as sums of sinusoids

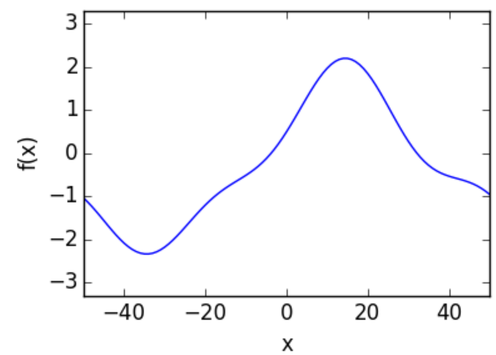
Writing functions as sums of sinusoids

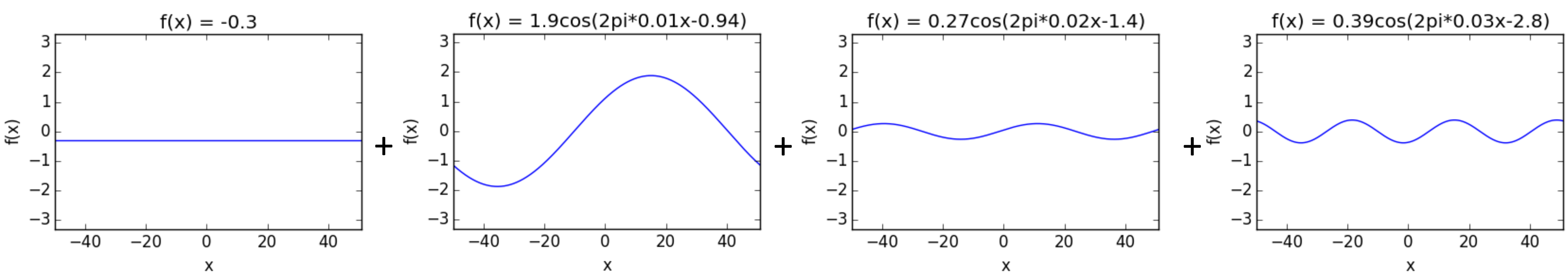
- Given a function defined on an interval of length *L*, we can write it as a sum of sinusoids whose periods are *L*, *L/2*, *L/3*, *L/4*, **…** (plus a constant term)

Original function



Sum of sinusoids below





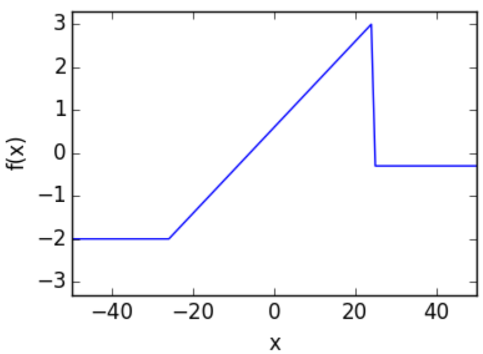
Decreasing period

Increasing frequency

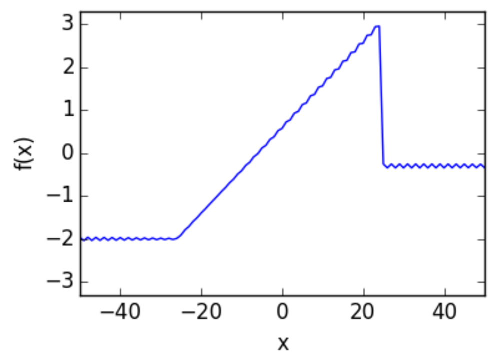
Writing functions as sums of sinusoids

- 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*, **…** (plus a constant term)

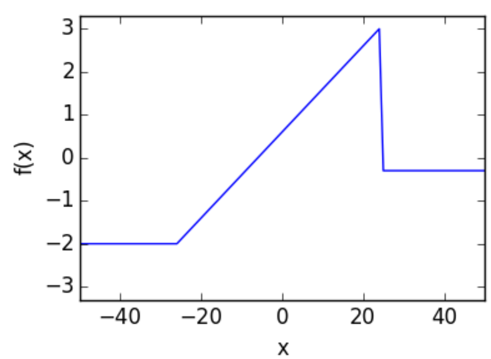
Original function



sum of 49 sinusoids (plus constant term)



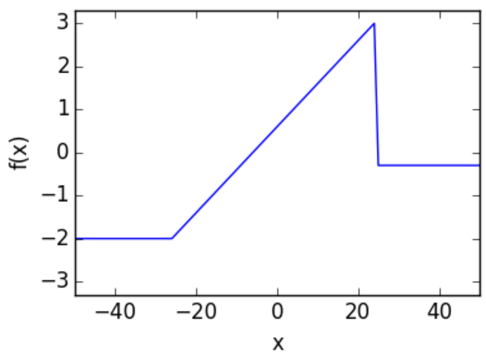
sum of 50 sinusoids (plus constant term)



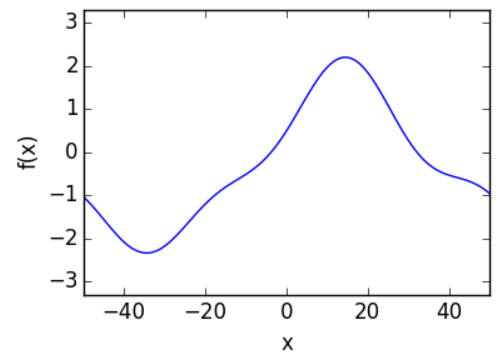
Writing functions as sums of sinusoids

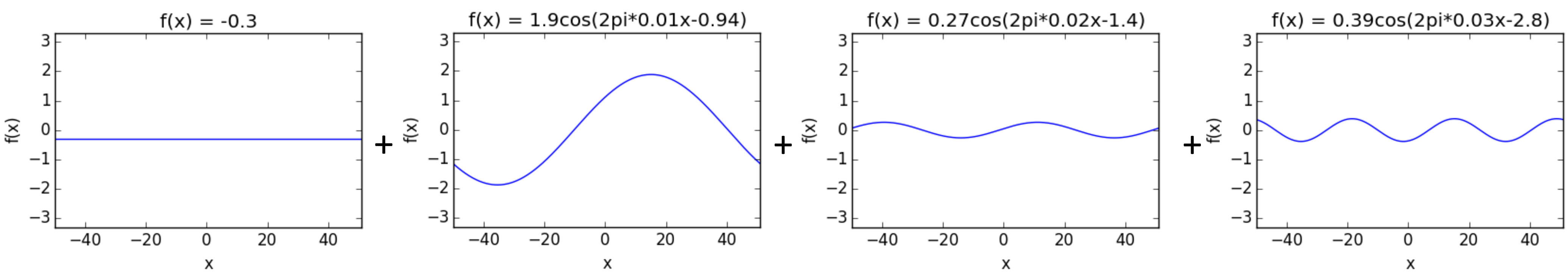
- Each of these sinusoidal terms has a magnitude (scale factor) and a phase (shift).

Original function



Sum of sinusoids below





Magnitude: -0.3	Magnitude: 1.9	Magnitude: 0.27	Magnitude: 0.39
Phase: 0	Phase: -.94	Phase: -1.4	Phase: -2.8

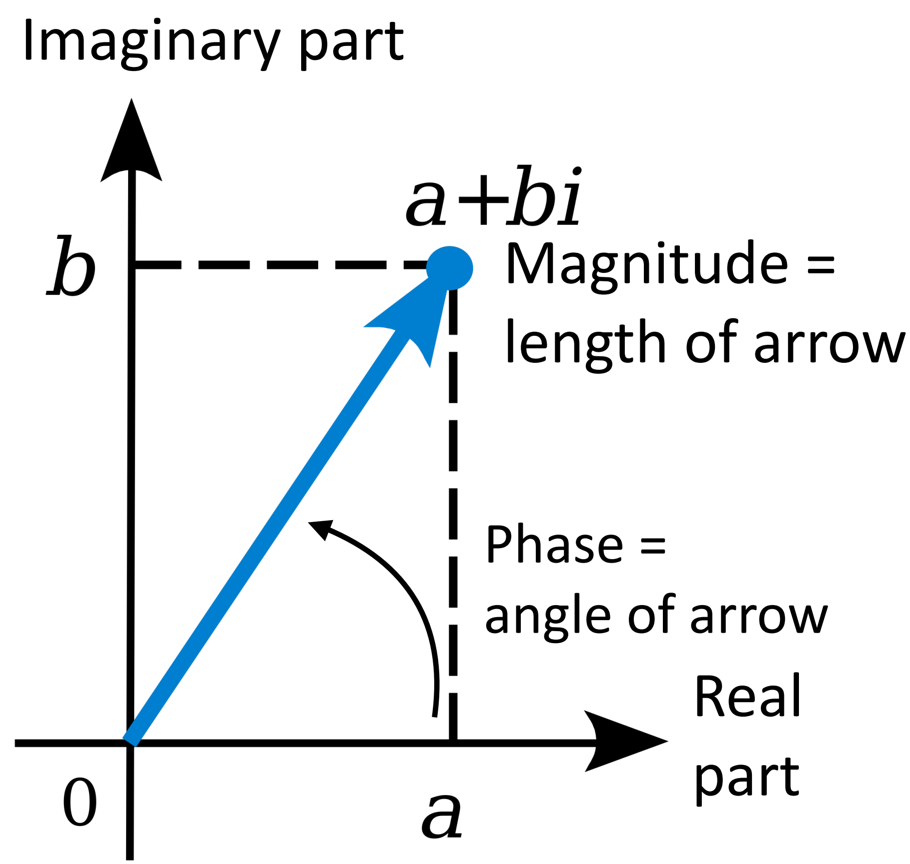
Expressing a function as a set of sinusoidal term coefficients

- We can thus express the original function as a series of magnitude and phase coefficients
  - We treat the constant term as having phase 0
  - If the original function is defined at *N* equally spaced points, we'll need a total of *N* coefficients
  - If the original function is continuous, we'll need an infinite series of magnitude and shift coefficients—but we can approximate the function with just the first few

Constant term (frequency 0)	Sinusoid 1 (period *L*, frequency 1/*L*)	Sinusoid 2 (period *L*/2, frequency 2/*L*)	Sinusoid 3 (period *L*/3, frequency 3/*L*)
Magnitude: -0.3	Magnitude: 1.9	Magnitude: 0.27	Magnitude: 0.39
Phase: 0	Phase: -.94	Phase: -1.4	Phase: -2.8

Using complex numbers to represent magnitude plus phase

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



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
  - 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
  - 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.

Demo

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
- 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

Fourier transforms

The Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT)

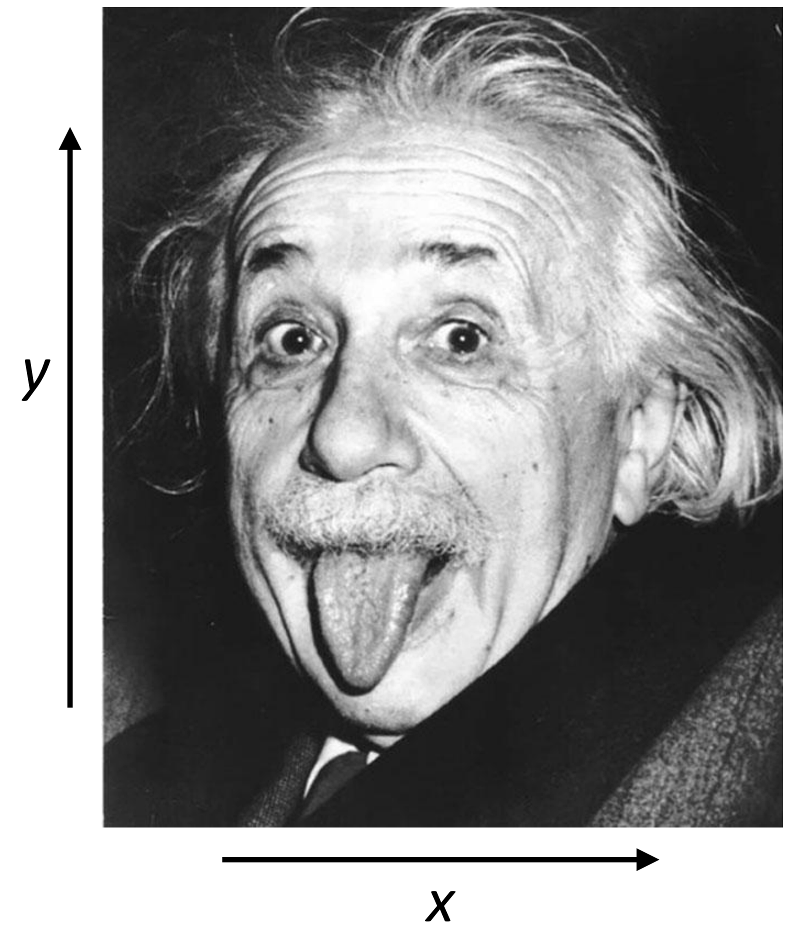
- The number of arithmetic operations required to compute the Fourier transform of N numbers (i.e., of a function defined at N points) in a straightforward manner is proportional to N^2
- Surprisingly, it is possible to reduce this N^2 to $N\log N$ using a clever algorithm
  - This algorithm is the Fast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

Fourier transforms

Multidimensional Fourier Transforms

Images as functions of two variables

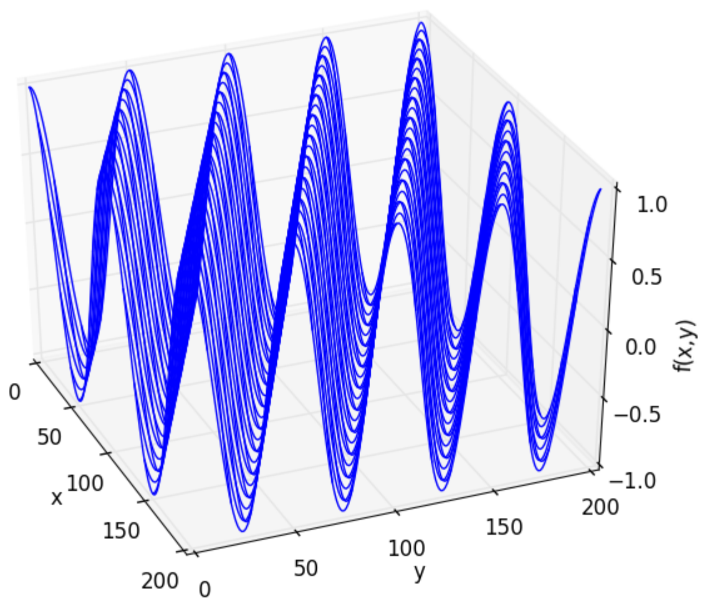
- Many of the applications we'll consider involve images
- A grayscale image can be thought of as a function of two variables
  - The position of each pixel corresponds to some value of *x* and *y*
  - The brightness of that pixel is proportional to *f*(*x*,*y*)

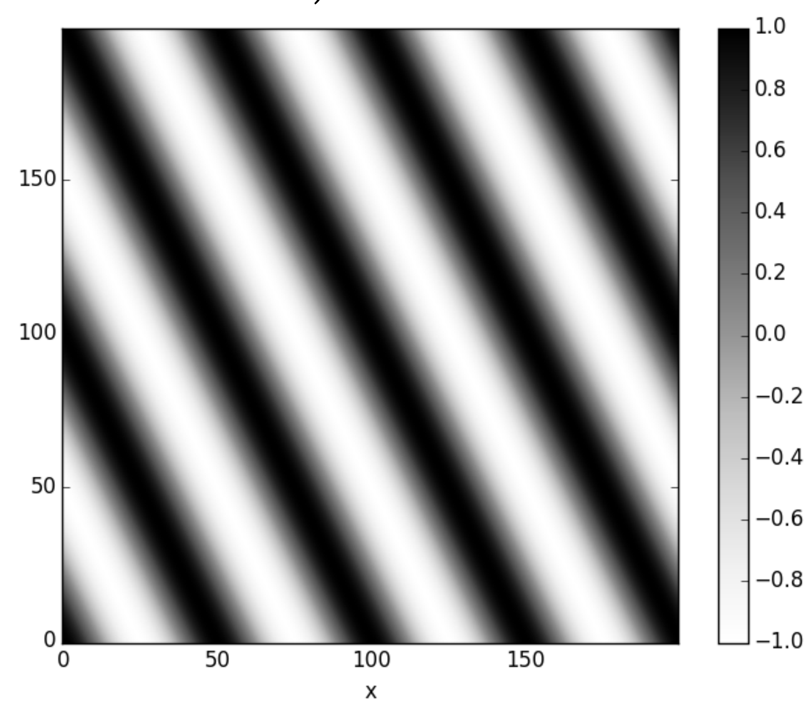


Two-dimensional Fourier transform

- 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

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





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
- Multidimensional Fourier transforms can also be computed efficiently using the FFT algorithm

Convolution

Convolution

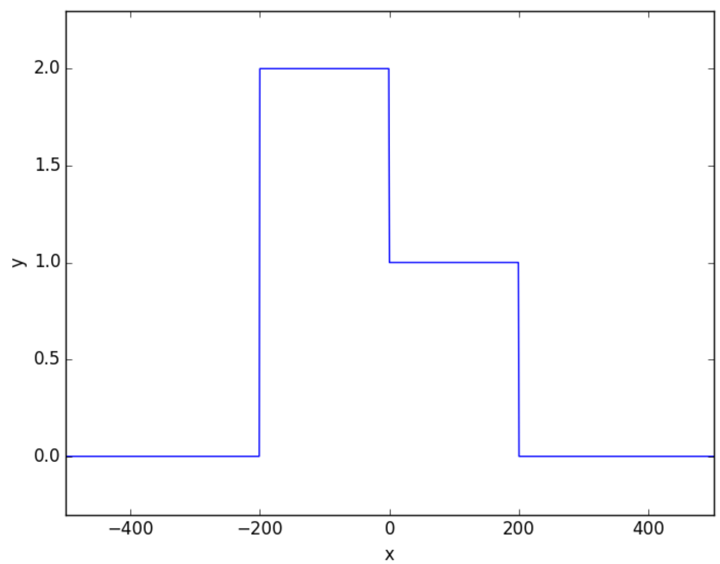
Moving averages

Convolution generalizes the notion of a *moving average*

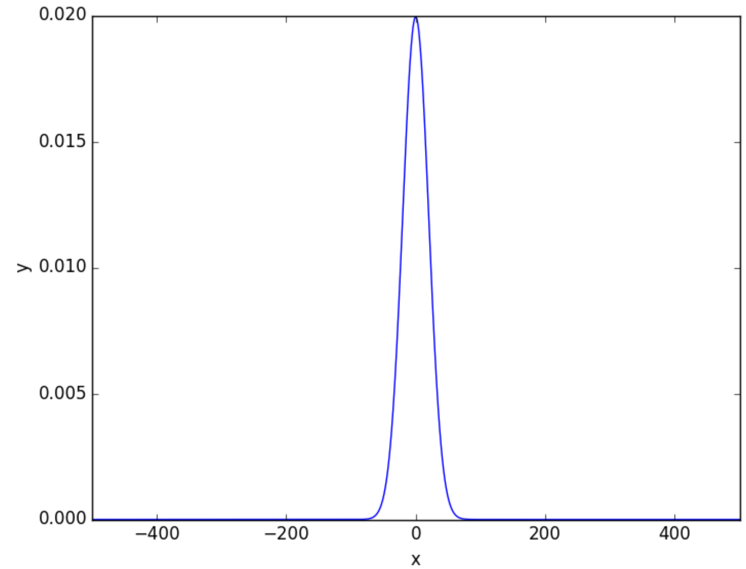
- We're given an array of numerical values
  - We can think of this array as specifying values of a function at regularly spaced intervals
- To compute a moving average, we replace each value in the array with the average of several values that precede and follow it (i.e., the values within a *window*)
- We might choose instead to calculate a *weighted moving average*, were 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

Example

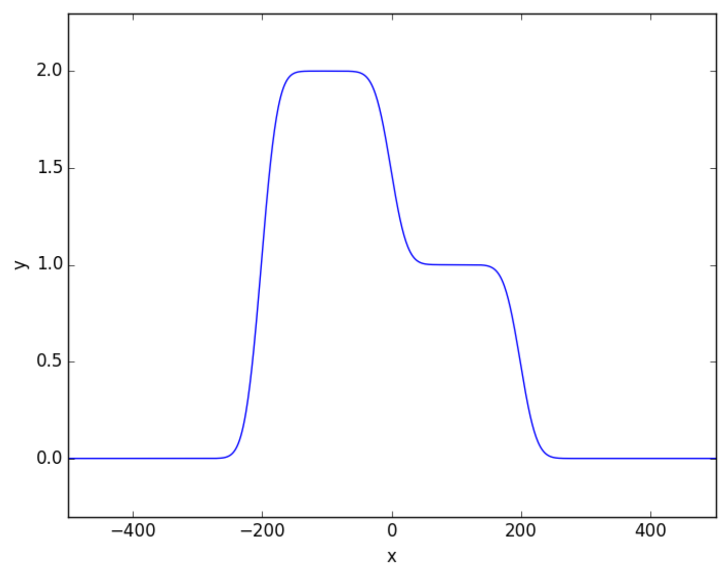
f



g



f convolved with g (written f∗g)



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

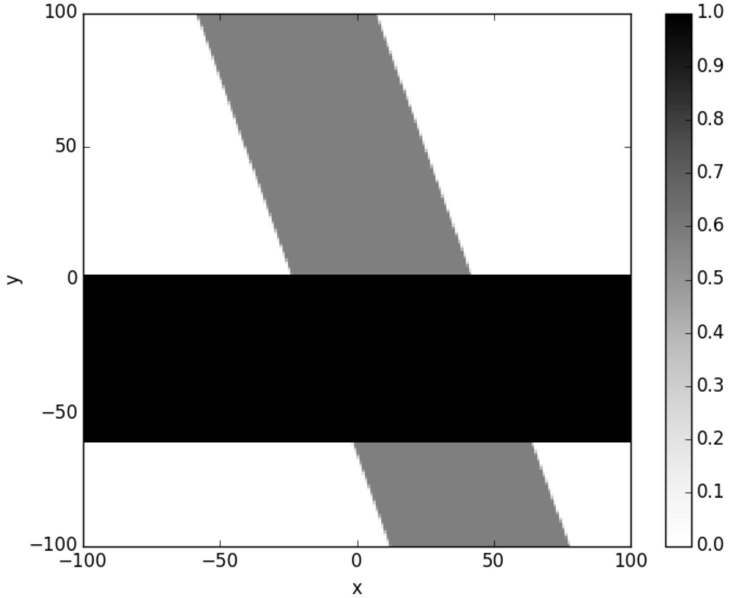
Multidimensional convolution

Two-dimensional convolution

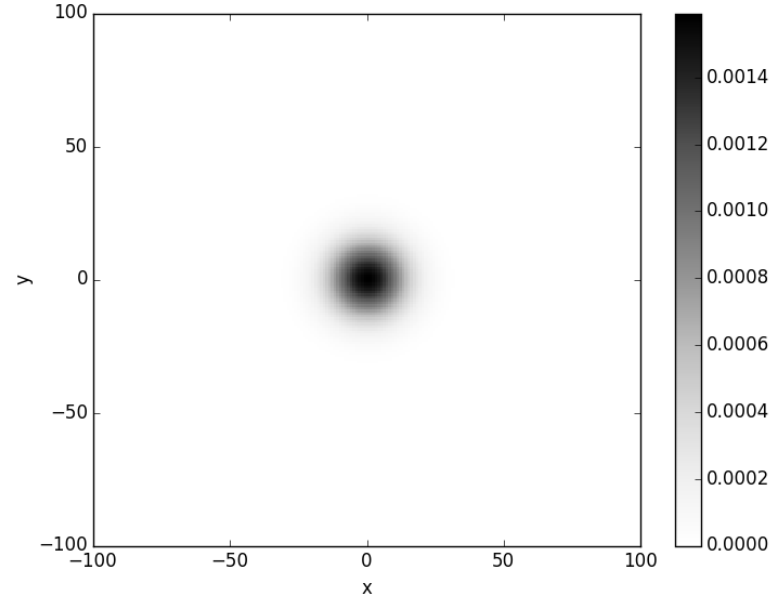
- In two-dimensional convolution, we replace each value in a two-dimensional array with a weighted average of the values surrounding it in two dimensions
  - We can represent two-dimensional arrays as functions of two variables, or as matrices, or as images

Two-dimensional convolution: example

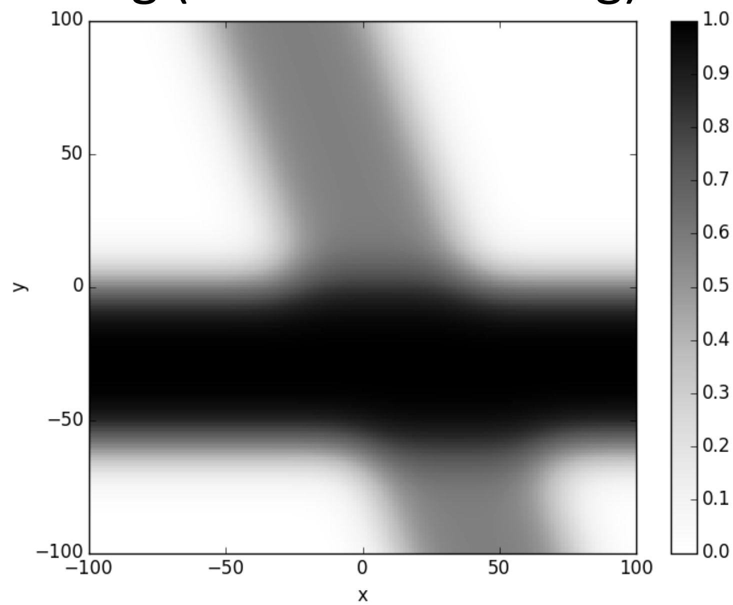
f



g



f∗g (f convolved with g)



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

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

Why does this relationship matter?

- 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 $N\log N$
- 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