Fourier transforms and convolution

(without the agonizing pain)

CS/CME/BioE/Biophys/BMI 279

Oct. 26, 2023

Ron Dror

Outline

- Why do we care?
- Convolution
  - Moving averages
  - Mathematical definition
- Fourier transforms
  - Writing functions as sums of sinusoids
  - The Fast Fourier Transform (FFT)
  - Multi-dimensional Fourier transforms
- Performing convolution using Fourier transforms

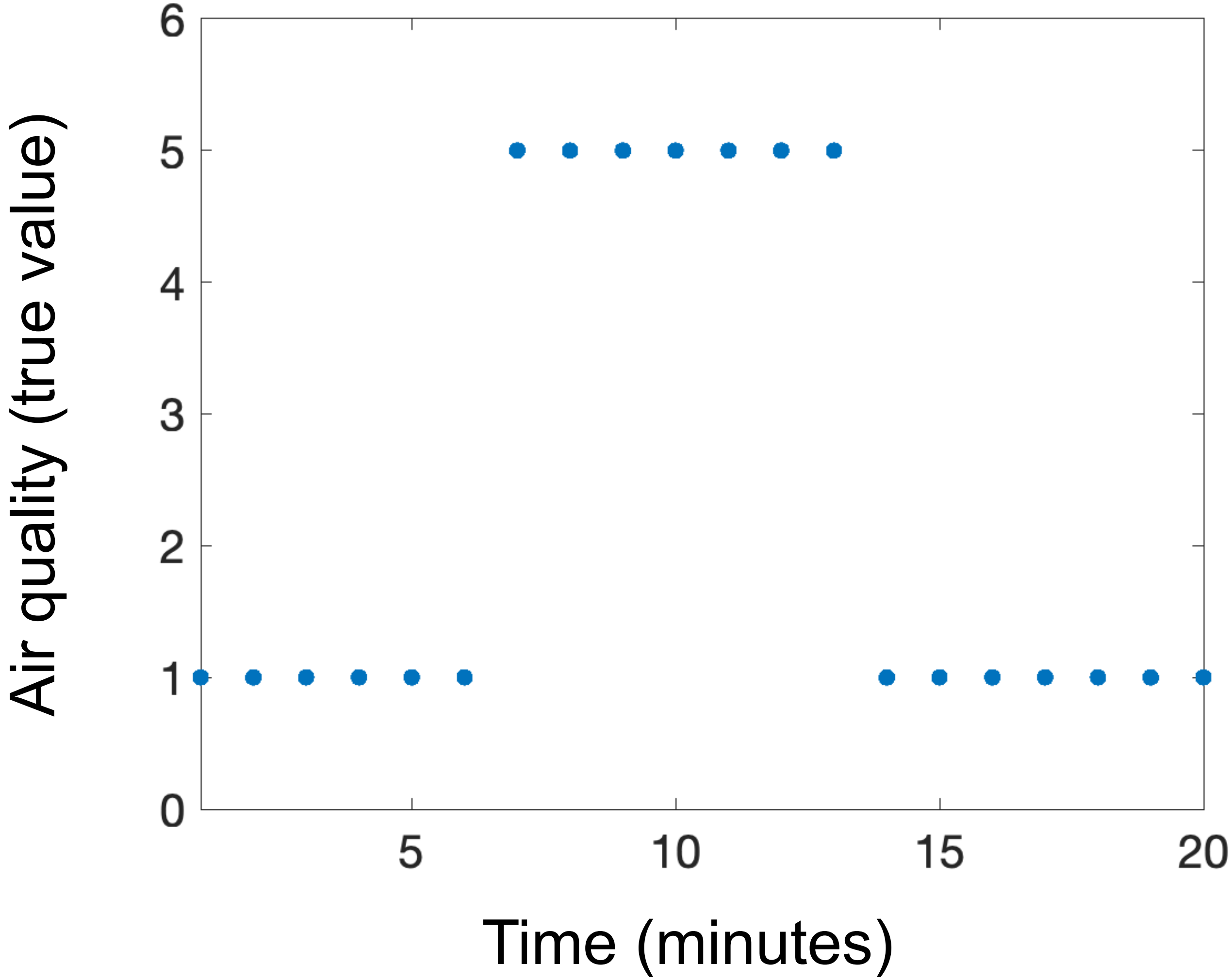
Why do we care?

Why study Fourier transforms and convolution?

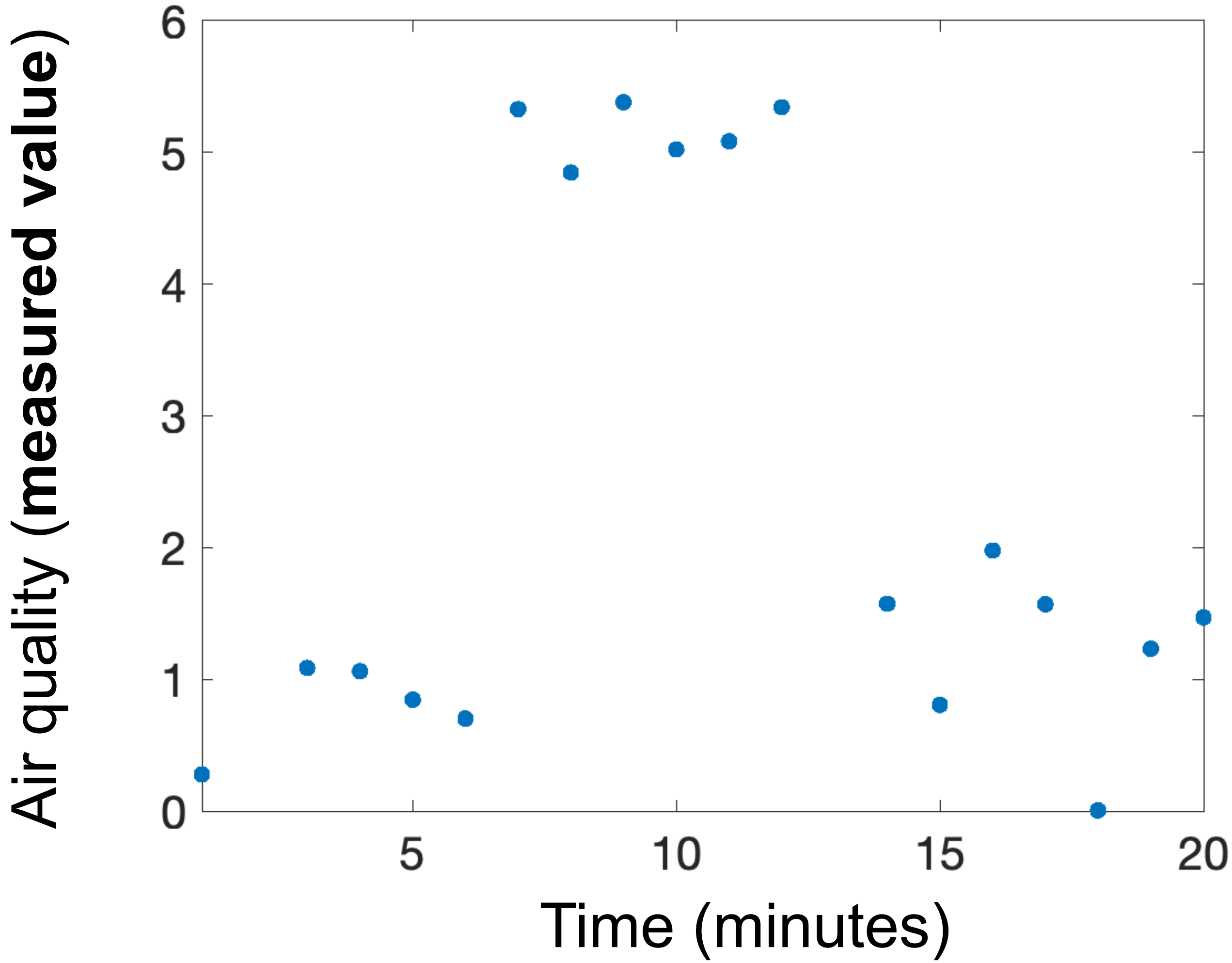
- In the remainder of the course, we'll study several methods that depend on analysis of images or reconstruction of structure from images:
  - Light microscopy (particularly fluorescence microscopy)
  - Cryoelectron microscopy
  - X-ray crystallography
- The computational aspects of each of these methods involve Fourier transforms and convolution
- Fourier transforms are also used in methods for
  - Ligand docking and virtual screening
  - Molecular dynamics simulations

Convolution

A function, as stored in a computer



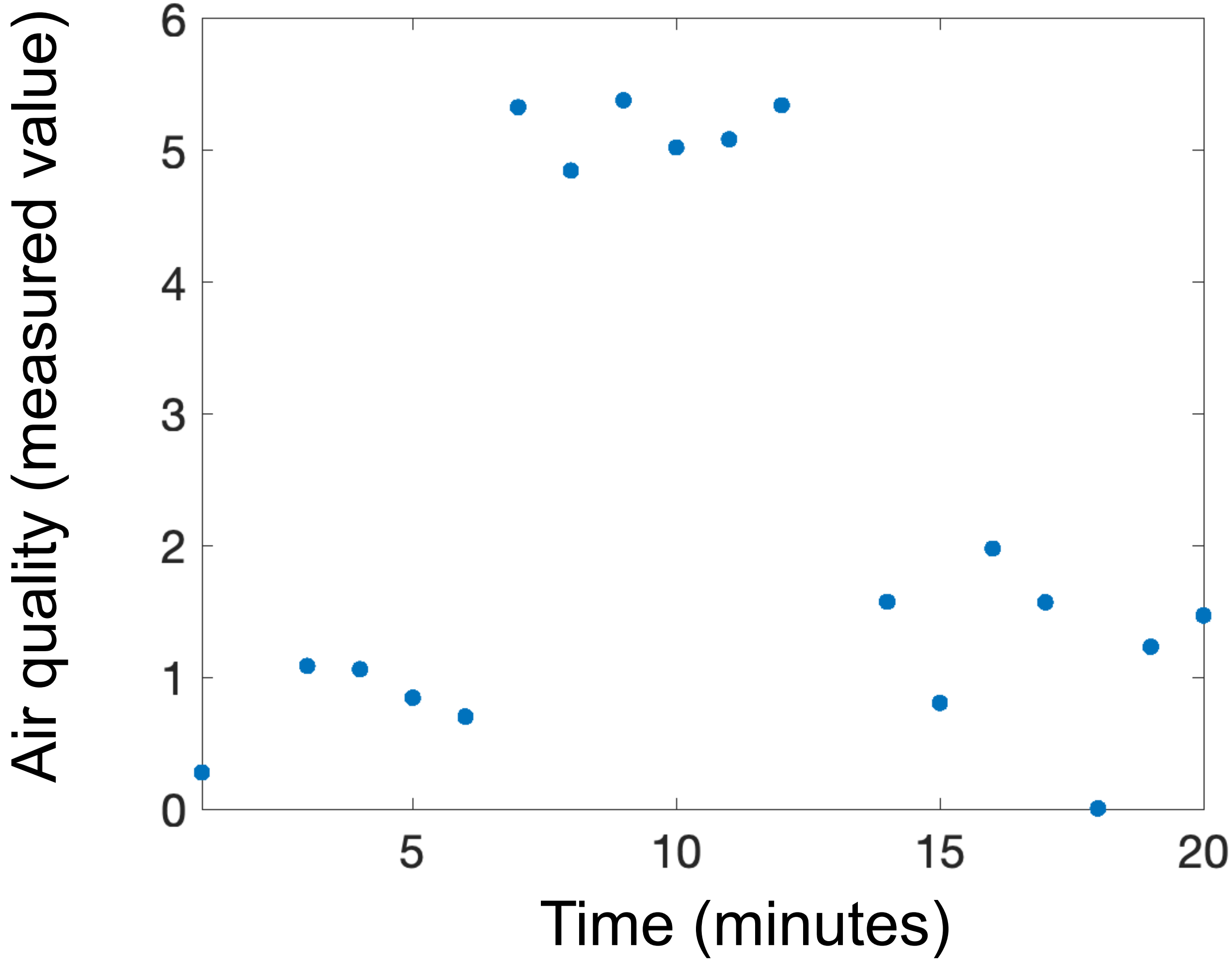
In practice, measurements are imperfect —there's always some noise



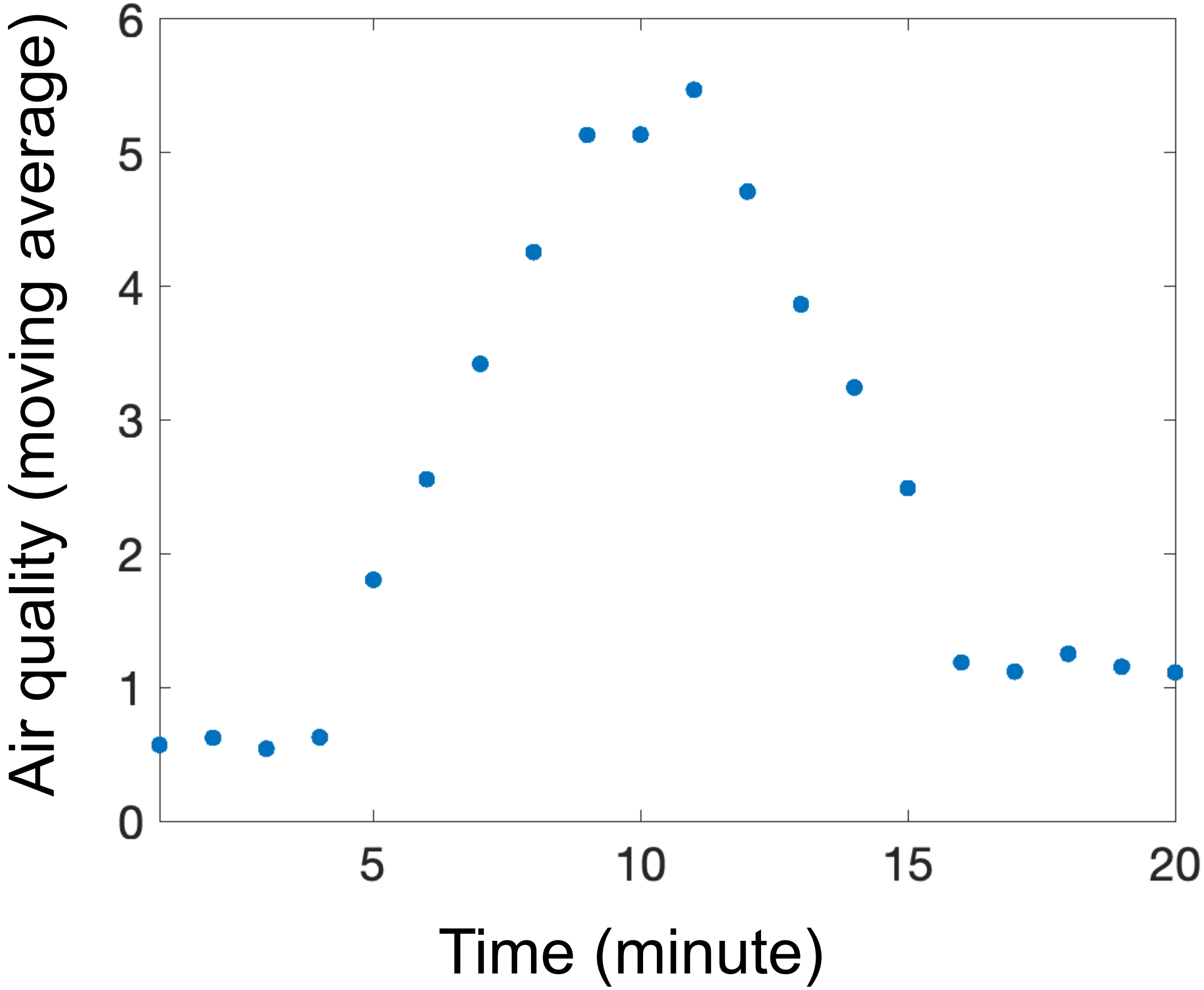
Convolution

Moving averages

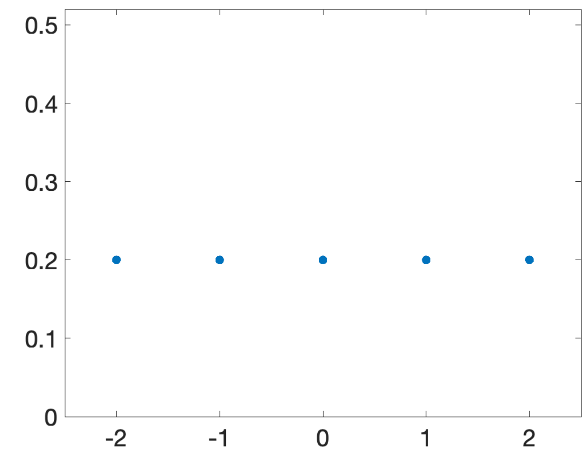
Original data (measurements)



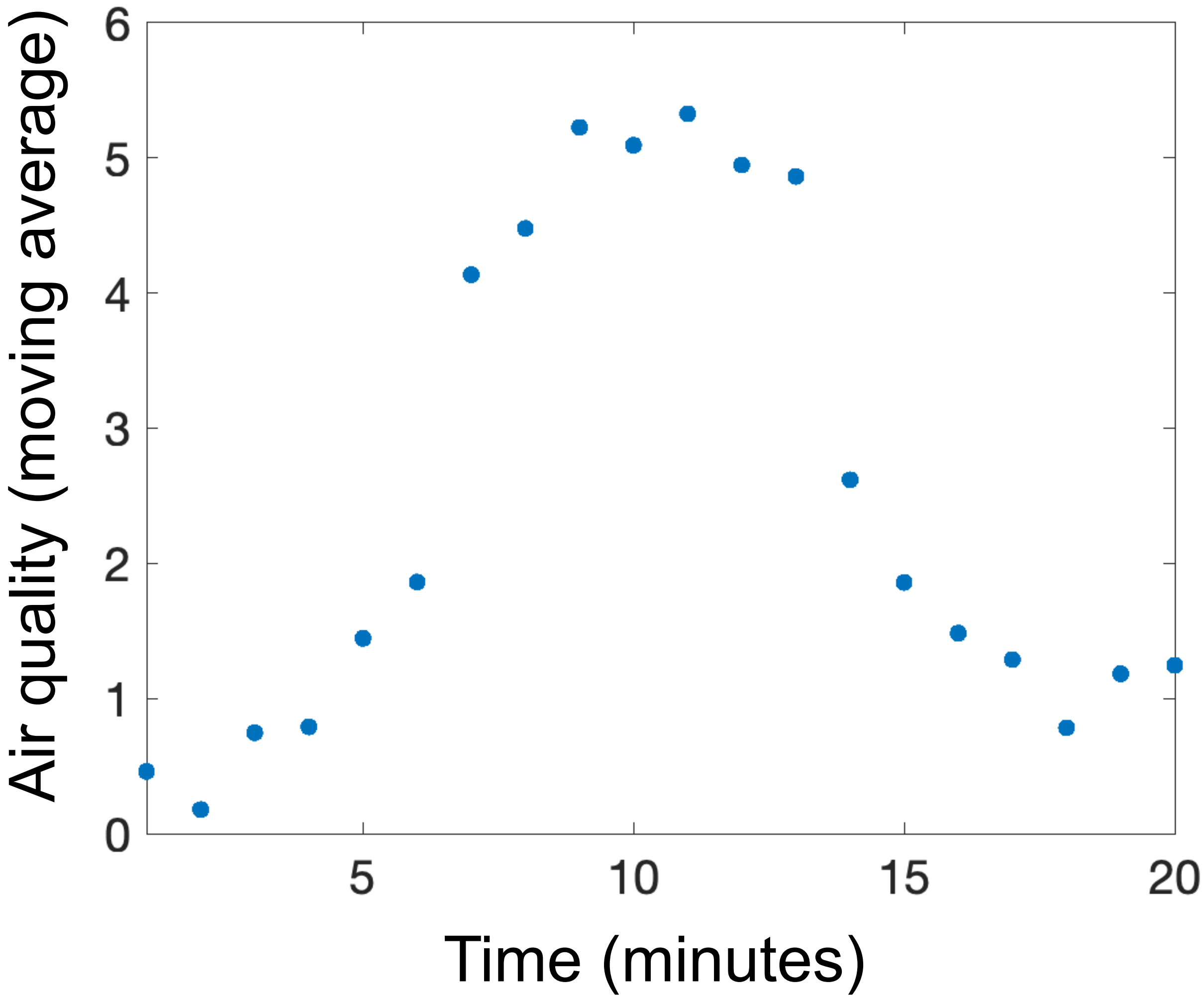
Moving average



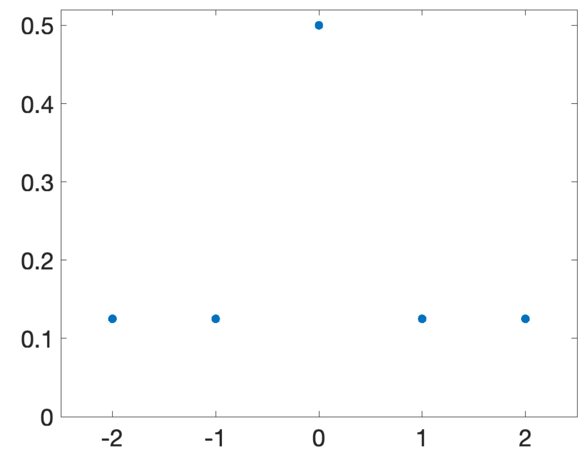
Weighting function (equal weights)



Weighted moving average



Weighting function (unequal weights)

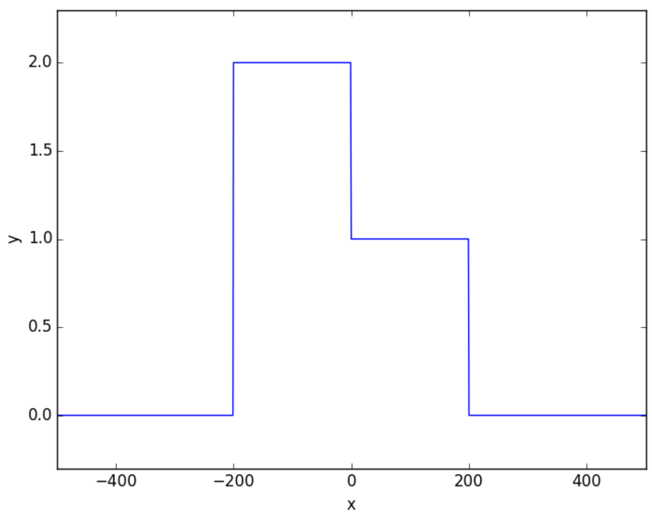


A convolution is basically a *weighted moving average*

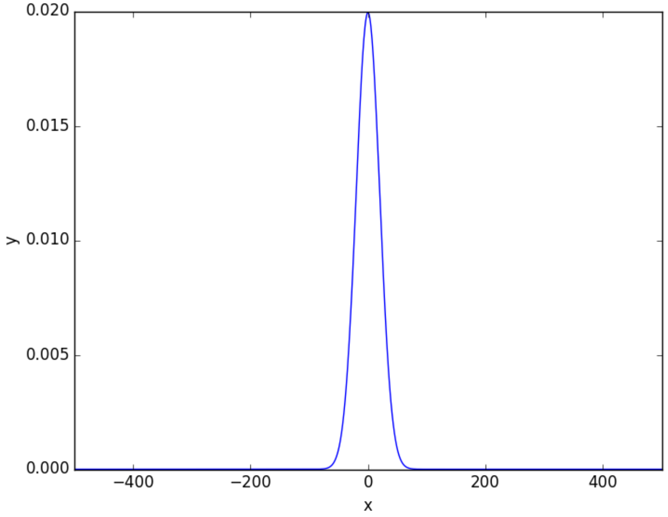
- We're given an array of numerical values
  - We can think of this array as specifying values of a function at regularly spaced intervals
- To compute a moving average, we replace each value in the array with the average of several values that precede and follow it (i.e., the values within a *window*)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

Example

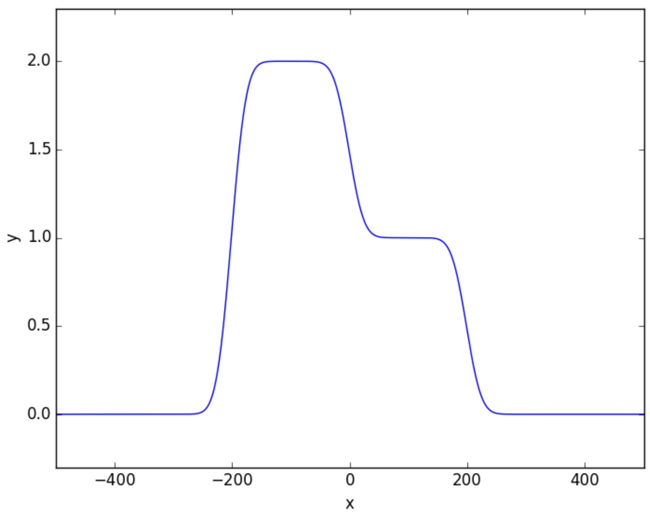
f



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:

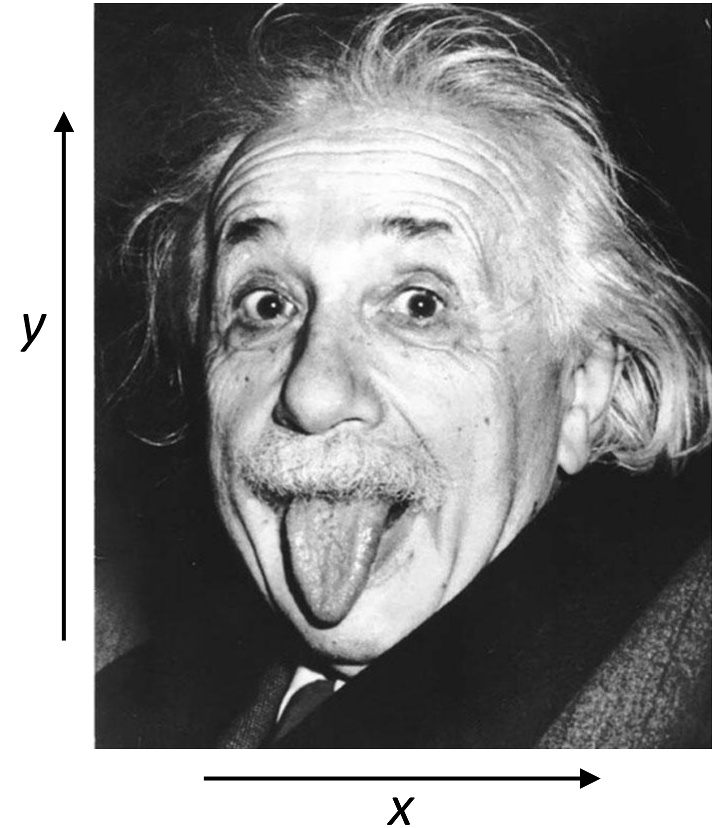
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

Convolution

Multidimensional convolution

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*)

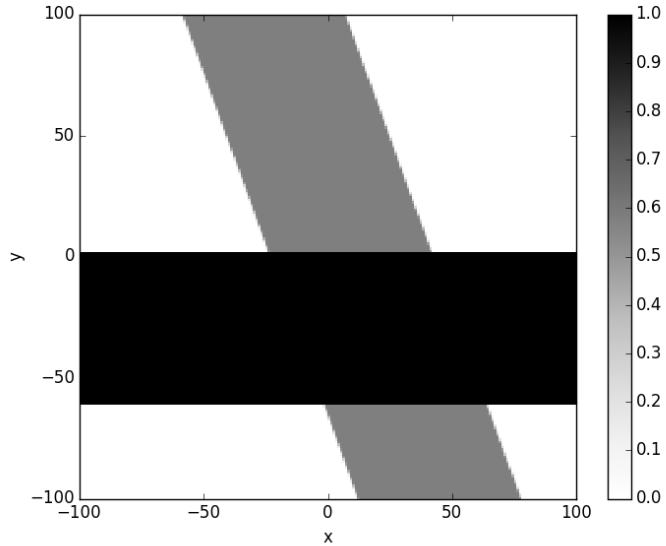


Two-dimensional convolution

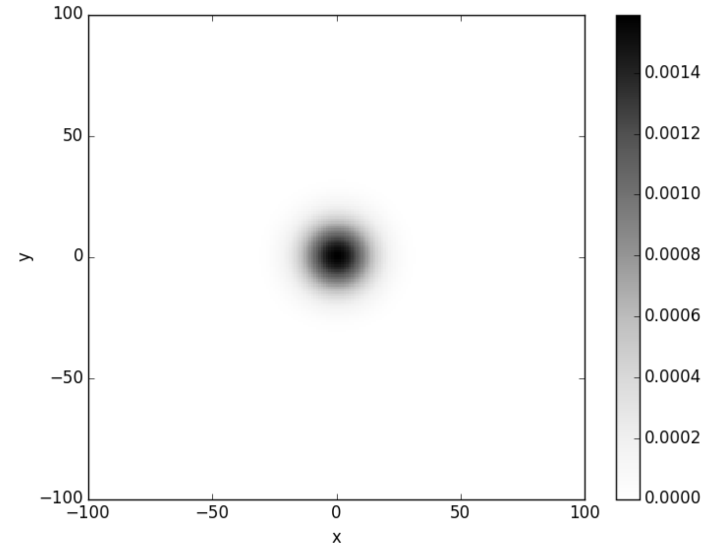
- In two-dimensional convolution, we replace each value in a two-dimensional array with a weighted average of the values surrounding it in two dimensions
  – We can represent two-dimensional arrays as functions of two variables, or as matrices, or as images

Two-dimensional convolution: example

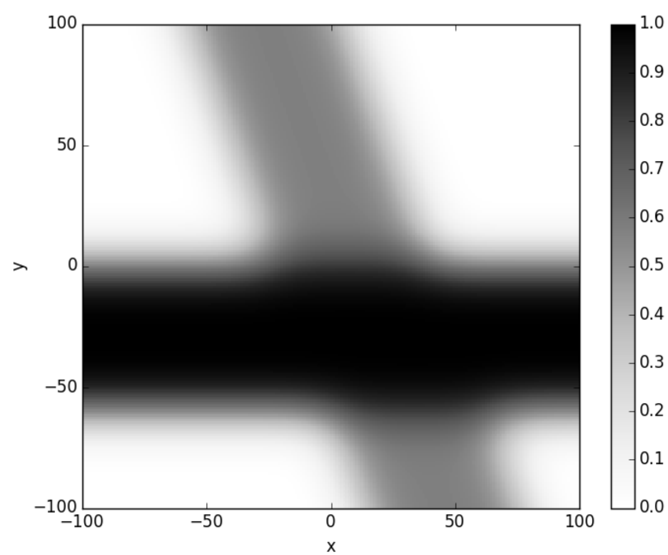
f



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

Fourier transforms

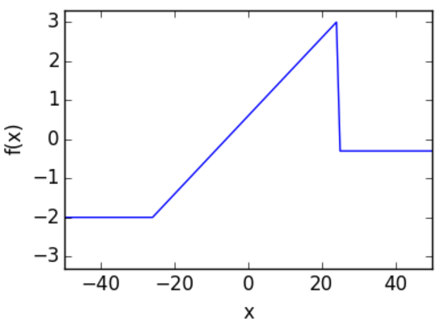
Fourier transforms

Writing functions as sums of sinusoids

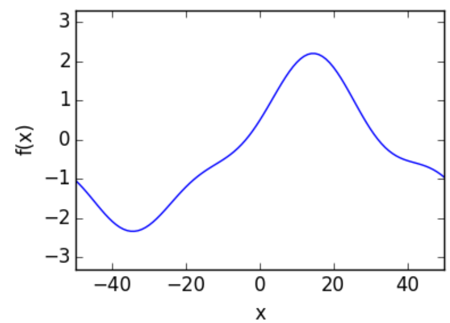
Writing functions as sums of sinusoids

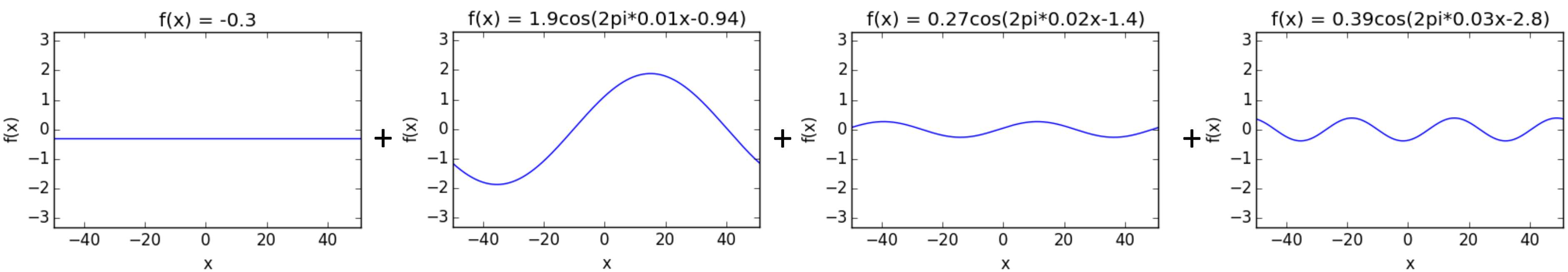
- Given a function defined on an interval of length *L*, we can express it as a sum of sinusoids whose periods are *L, L/2, L/3, L/4, …* (plus a constant term)

Original function



Sum of sinusoids below





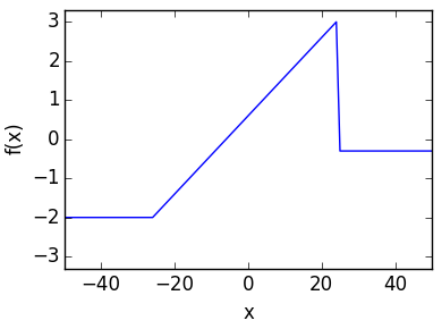
Decreasing period

Increasing frequency

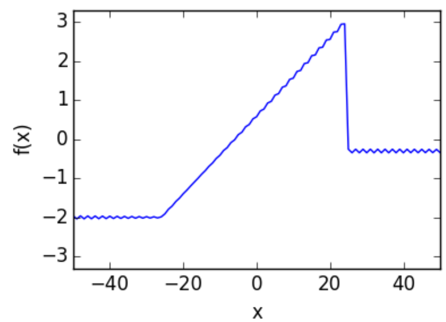
Writing functions as sums of sinusoids

- 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)

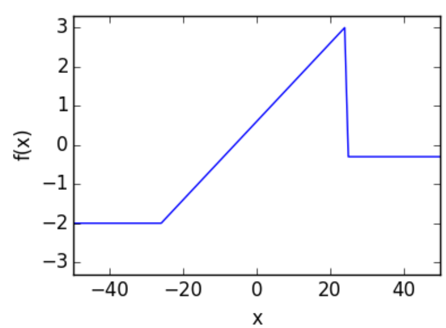
Original function



sum of 49 sinusoids (plus constant term)



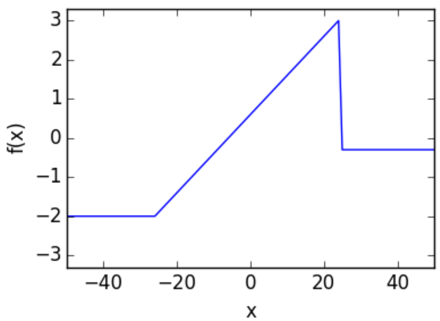
sum of 50 sinusoids (plus constant term)



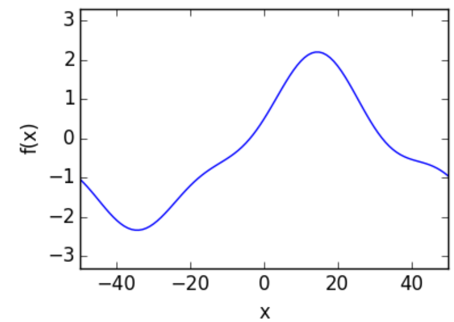
Writing functions as sums of sinusoids

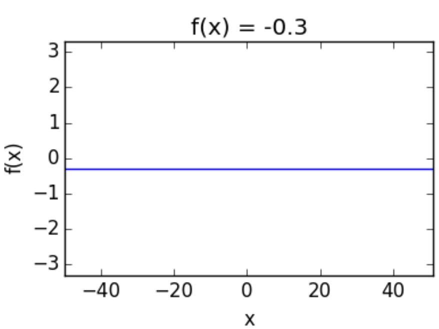
- Each of these sinusoidal terms has a magnitude (scale factor) and a phase (shift).

Original function

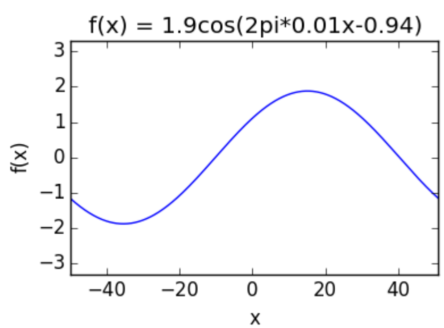


Sum of sinusoids below

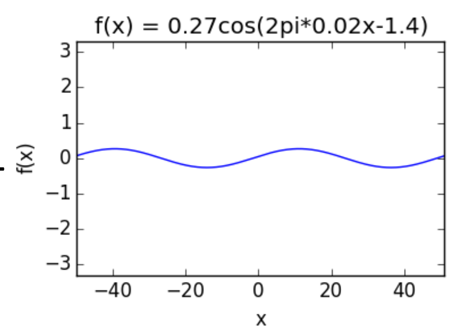




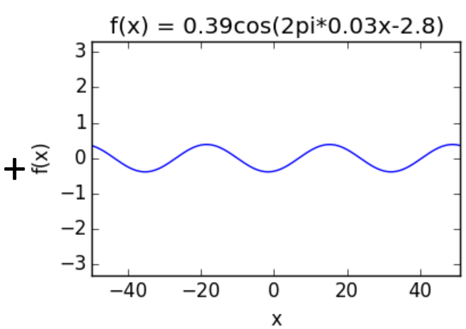
+



+



+



Magnitude: -0.3
Phase: 0 (arbitrary)

Magnitude: 1.9
Phase: -.94

Magnitude: 0.27
Phase: -1.4

Magnitude: 0.39
Phase: -2.8

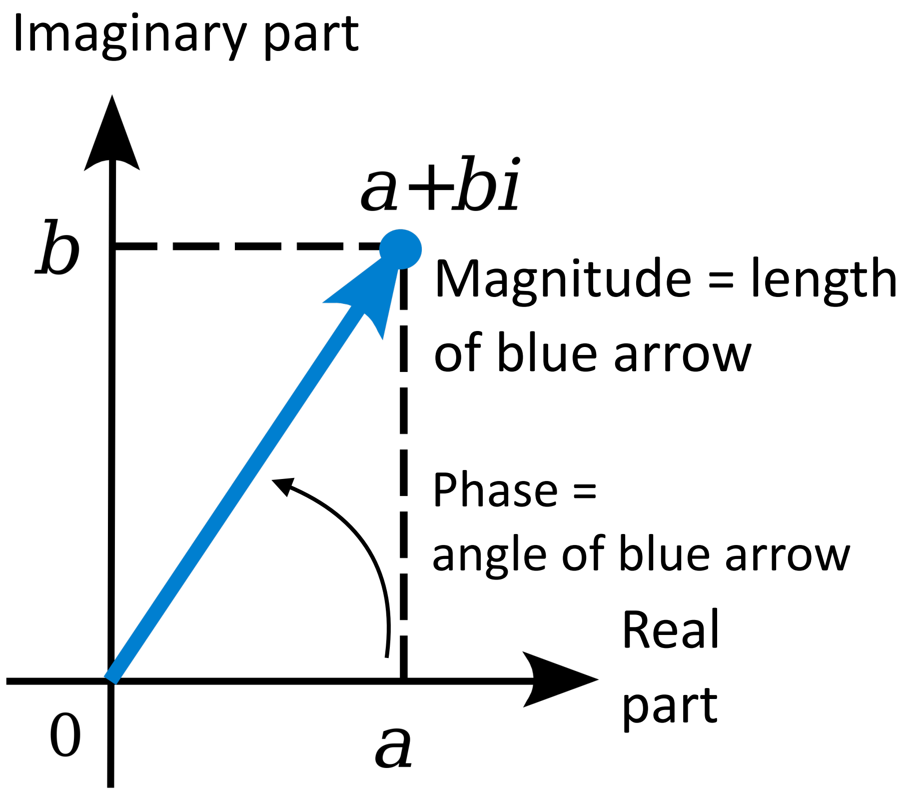
Expressing a function as a set of sinusoidal term coefficients

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

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

Using complex numbers to represent magnitude plus phase

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



Using complex numbers to represent magnitude plus phase

- We can express the magnitude and phase coefficients of each sinusoidal component using a single complex number
- Thus we can express our original function as a set 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.

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)

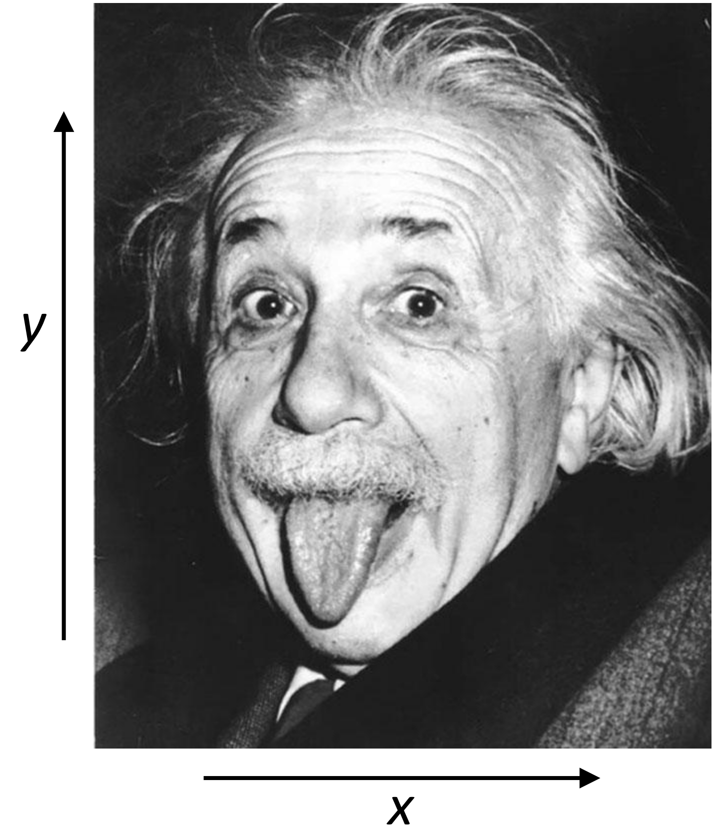
- The number of arithmetic operations required to compute the Fourier transform of N numbers (i.e., of a function defined at N points) in a straightforward manner is proportional to N^2
- Surprisingly, it is possible to reduce this N^2 to $N\log N$ using a clever algorithm
  - This algorithm is the Fast Fourier Transform (FFT)
  - It is arguably the most important algorithm of the past century
  - (For this class, you're not required to know just how this algorithm works, although it's really interesting!)

Fourier transforms

Multidimensional Fourier Transforms

Images as functions of two variables

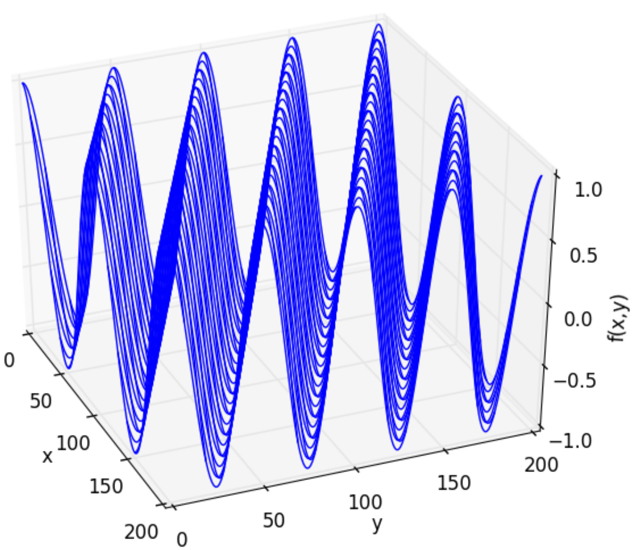
- Many of the applications we'll consider involve images
- A grayscale image can be thought of as a function of two variables
  - The position of each pixel corresponds to some value of *x* and *y*
  - The brightness of that pixel is proportional to *f*(*x*,*y*)

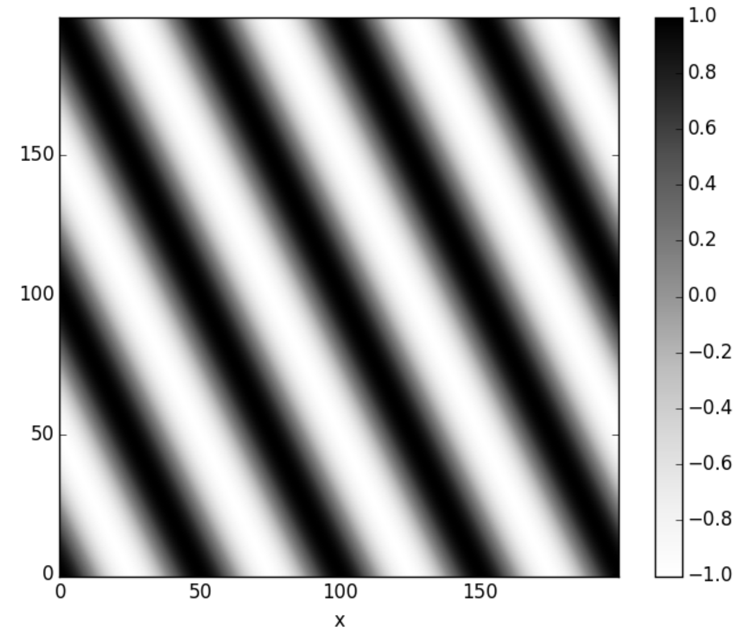


Two-dimensional Fourier transform

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

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?

- 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 $N \log N$
- Second, it allows us to characterize convolution operations in terms of modification to components of a function at each frequency
  - For example, convolution with a Gaussian will preserve low-frequency components while reducing magnitude of high-frequency components