# Digital Signal Processing

## Lecture 10 – Filtering and Convolution

상명대학교
컴퓨터과학과
강상욱 교수

# Smoothing I

- Removing short-term variations from a signal in order to reveal long-term trends.
  - A common smoothing algorithm is a moving average.
  - It computes the mean of the previous n values.

The gray line is the raw data. The darker line shows the 30-day moving average.

Smoothing removes the most extreme changes.

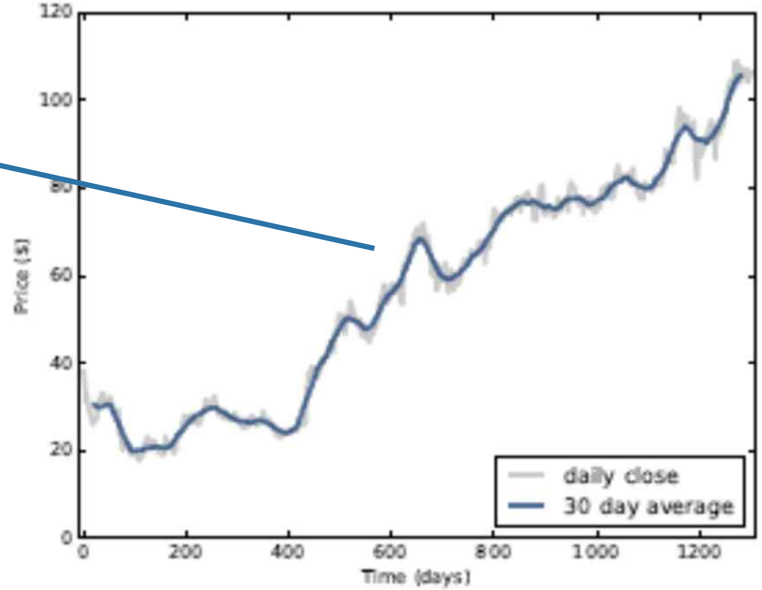



Figure 8.1: Daily closing price of Facebook stock and a 30-day moving average.

# Smoothing 2

```
signal = thinkdsp.SquareSignal(freq=440)
wave = signal.make_wave(duration=1, framerate=44100)
segment = wave.segment(duration=0.01)
```

Create a window with 11 elements and normalize them so that the elements add up to 1.

```
window = np.ones(11)
window /= sum(window)
```

The window is added to the end (to N) with zero values.

```
ys = segment.ys
N = len(ys)
padded = thinkdsp.zero_pad(window, N)
```

The sum of the elementwise products is the average of the first 11 elements.

```
prod = padded * ys
sum(prod)
```

Here, elements are all -1 so the average is also -1.

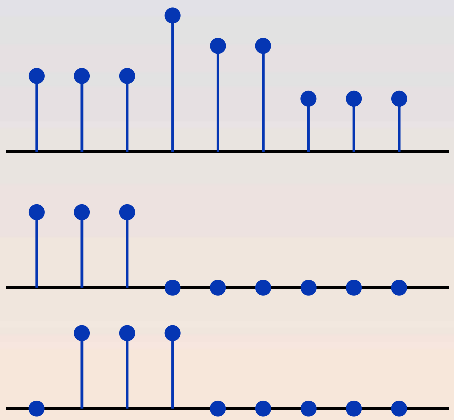
# Smoothing the square waveform

The average of the NEXT 11 elements of the wave array.

```
rolled = np.roll(rolled, 1)
prod = rolled * ys
sum(prod)
```

Ex> r0 = [0, 1, 2, 3, 4, 5]
r1 = np.roll (r0, 1)
print(r1)

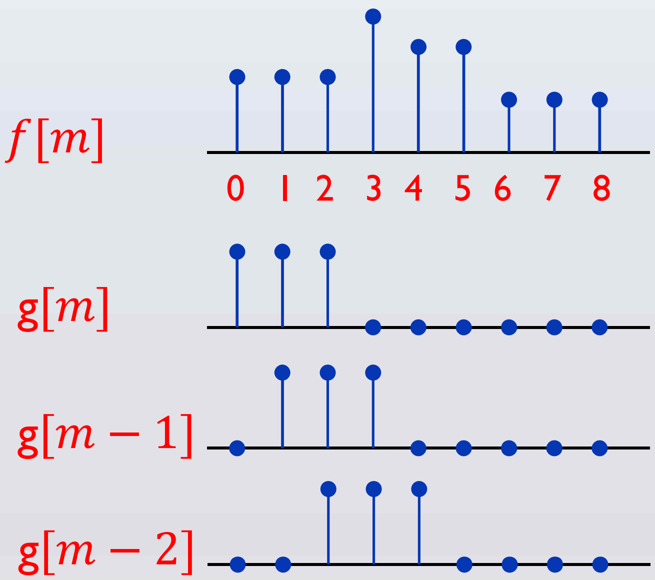
r1 = [5, 0, 1, 2, 3, 4]



```
def smooth(ys, window):
    N = len(ys)
    smoothed = np.zeros(N)
    padded = thinkdsp.zero_pad(window, N)
    rolled = padded

    for i in range(N):
        smoothed[i] = sum(rolled * ys)
        rolled = np.roll(rolled, 1)
    return smoothed
```

# Smoothing the square waveform

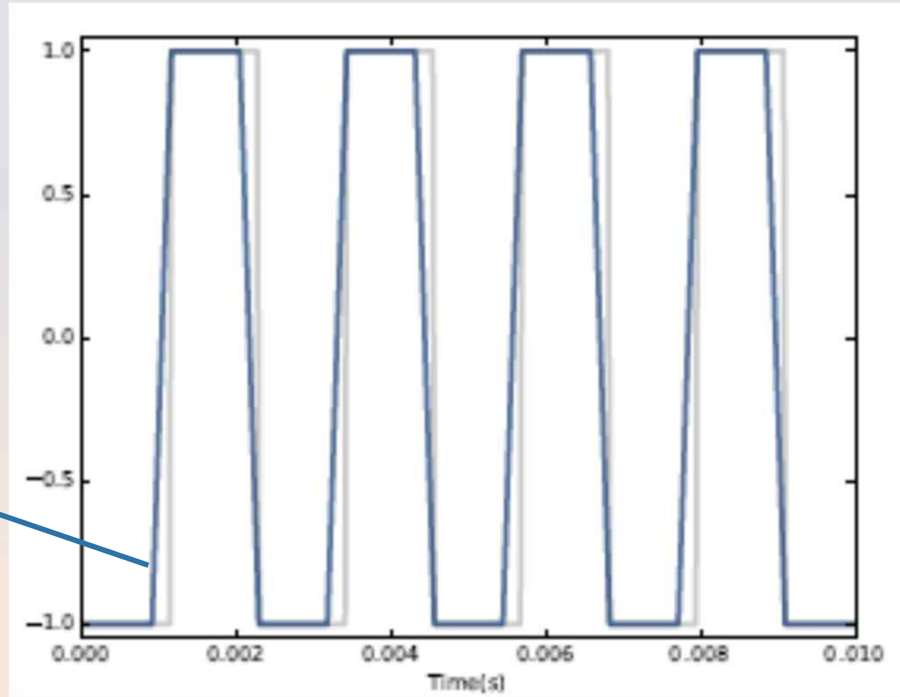



*Cross correlation*

$$(f \star g)[n] = \sum_{m=0}^{N-1} f[m]g[m-n]$$

, where f is a wave, g is the window.

The smoothed signal starts to ramp up when the leading edge of the window reaches the first transition, and levels off when the window crosses the transition.

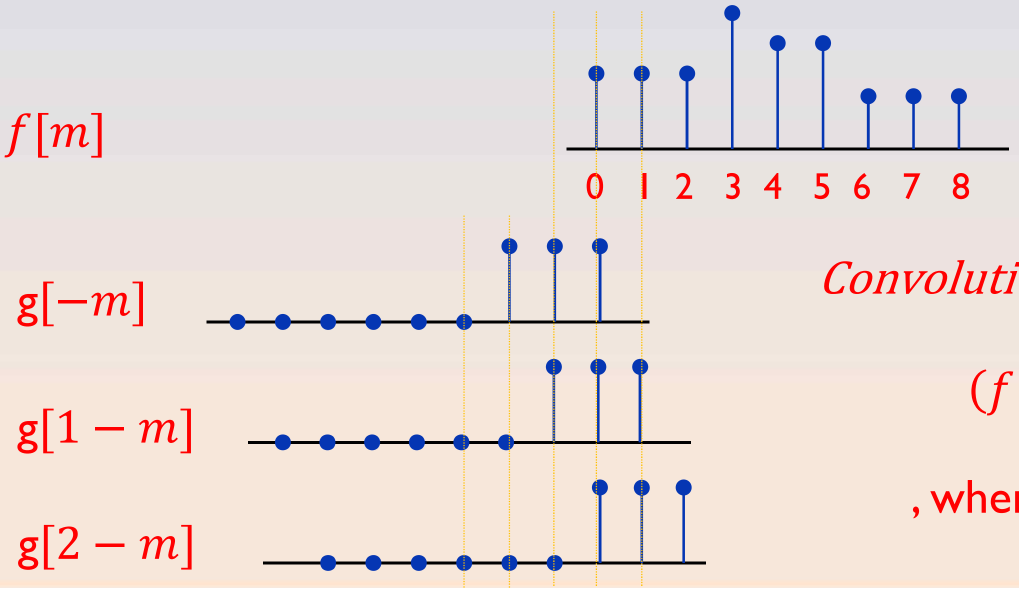


# Convolution

- Definition
  - Applying a window function to each overlapping segment of a wave.
  - NumPy provides a simpler and faster version.

Mode 'valid' : only computes values when the window and the wave array overlap completely, so it stops when the right edge of the window reaches the end of the wave array.

```
convolved = np.convolve(ys, window, mode='valid')
smooth2 = thinkdsp.Wave(convolved, framerate=wave.framerate)
```

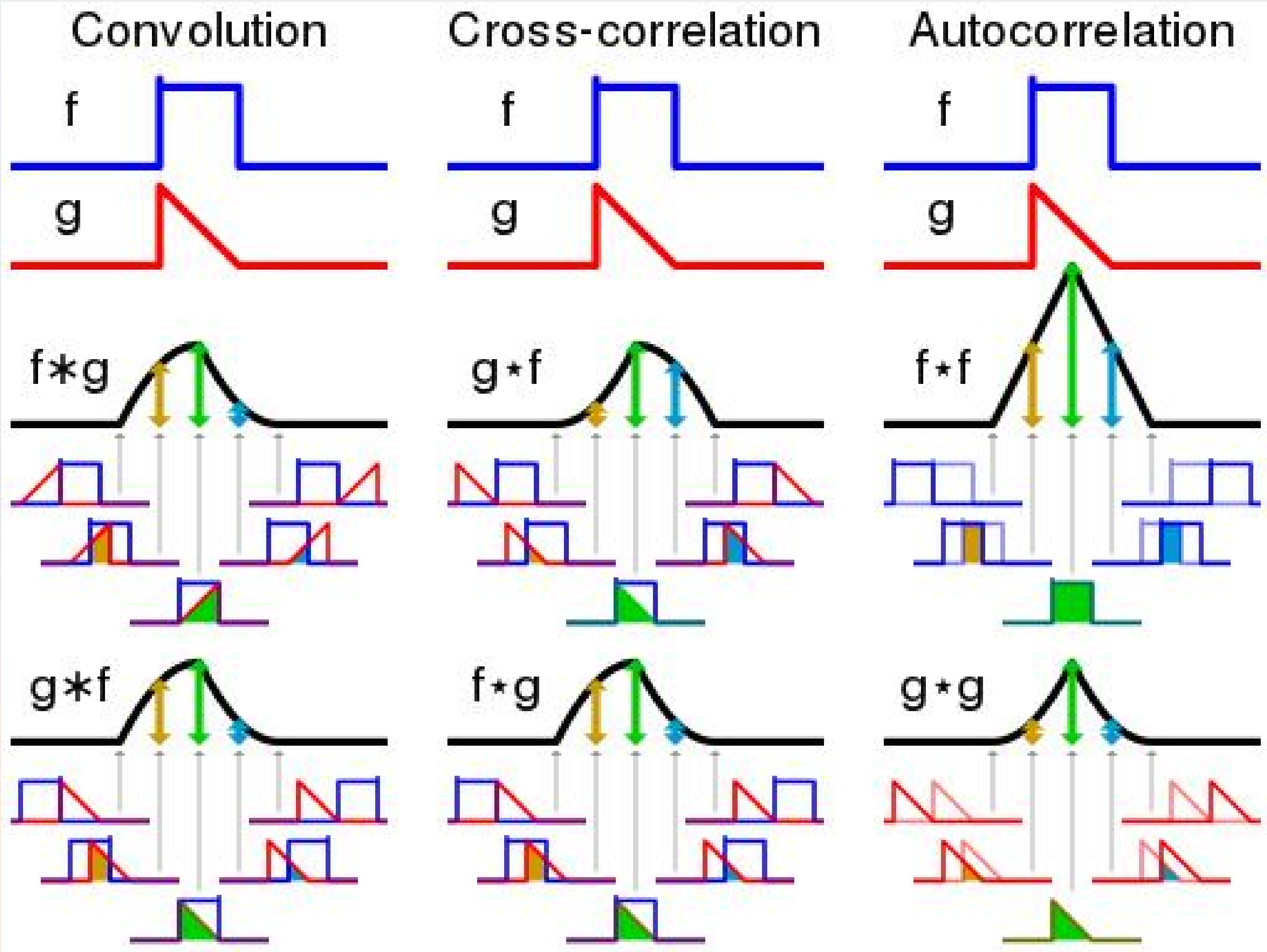



*Convolution*

$$(f * g)[n] = \sum_{m=0}^{N-1} f[m]g[n-m]$$

, where f is a wave, g is the window.

# Convolution, Crosscorelation, Autocorrelation



# The frequency domain I

‘same’ : the result should have the same length as the input

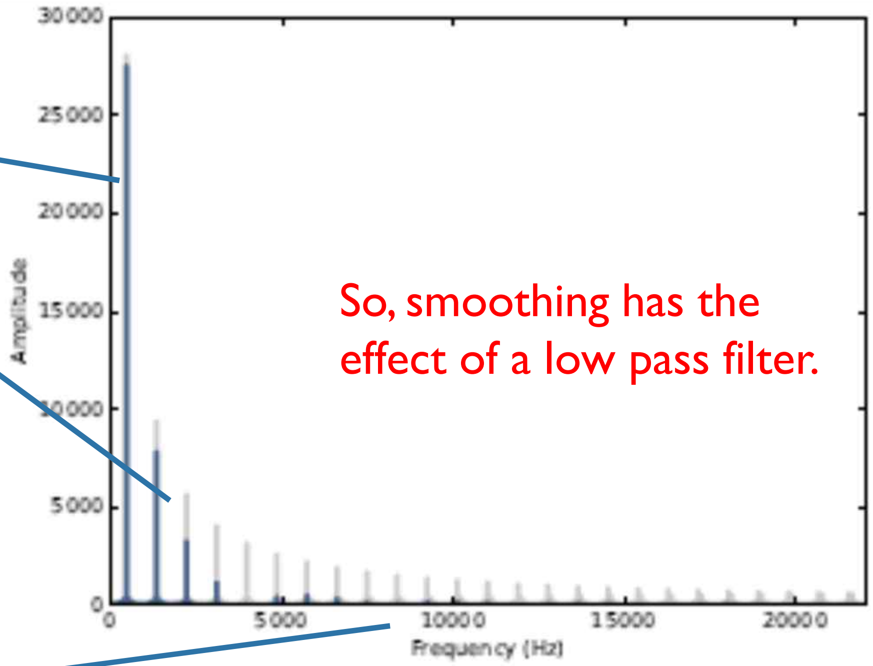
```
convolved = np.convolve(wave.ys, window, mode='same')
smooth = thinkdsp.Wave(convolved, framerate=wave.framerate)
spectrum2 = smooth.make_spectrum()
spectrum2.plot()
```

This example includes a few values that wrap around

The fundamental freq. is almost unchanged.

The first few harmonics are attenuated.

The higher harmonics are almost eliminated.




So, smoothing has the effect of a low pass filter.

Figure 8.3: Spectrum of the square wave before and after smoothing.

# The frequency domain 2

To see how much each component has been attenuated.

The ratio is high for low frequencies and drops off at a cutoff freq. near 4400Hz.

```
amps = spectrum.amps
amps2 = spectrum2.amps
ratio = amps2 / amps
ratio[amps<560] = 0
thinkplot.plot(ratio)
```

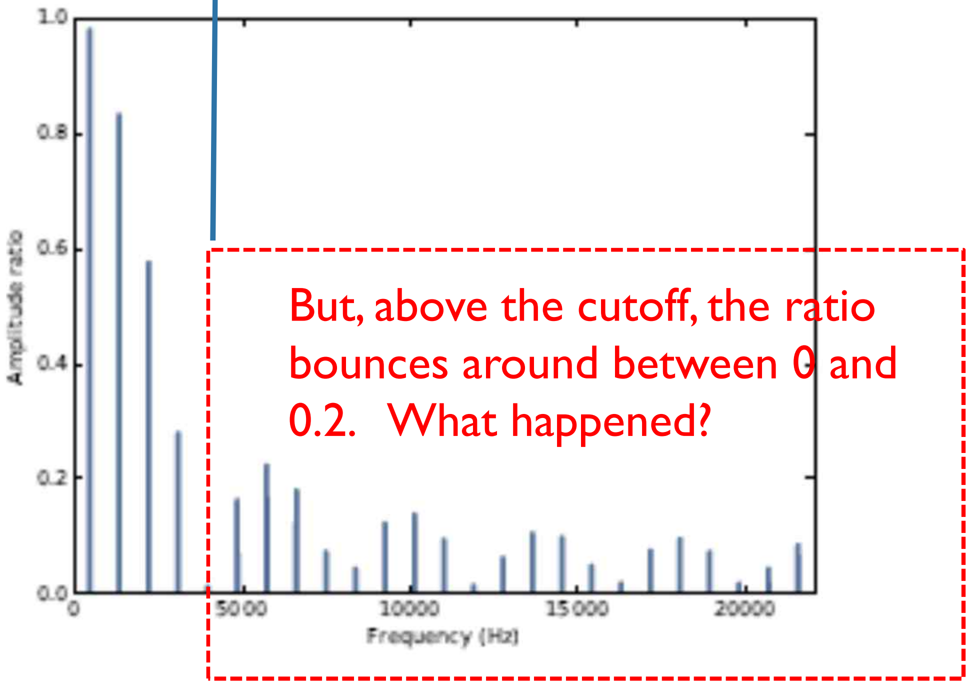



Figure 8.4: Ratio of spectrums for the square wave, before and after smoothing.

# The convolution theorem I

- The answer of the question raised up in the previous slide.
  - Convolution theorem
  - $DFT(f * g) = DFT(f) \cdot DFT(g)$
- Convolution in the time domain corresponds to multiplication in the frequency domain.

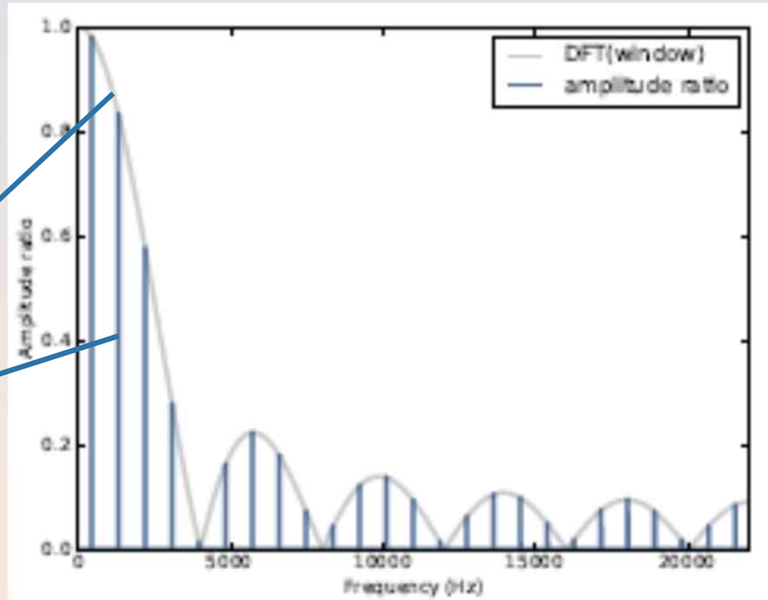
This explains Figure 8.4 because when we convolve a wave and a window, we multiply the spectrum of the wave with the spectrum of the window.

```
padded = zero_pad(window, N)
dft_window = np.fft.rfft(padded)
thinkplot.plot(abs(dft_window))
```

The DFT of the smoothing window

The ratio from the previous slide

$$abs\ (DFT(f * g))/abs\ (DFT(f)) = abs\ (DFT(g))$$



# Gaussian filter I

- The DFT of a window is called filter.
  - For any convolution window in the time domain, there is a corresponding filter in the frequency domain.
- The moving average window is a low-pass filter, but it is not a very good one.
  - The DFT drops off steeply at first, then it bounces around.
  - The bounces are called sidelobes.
  - The spectrum contains high freq. harmonics that drops off relatively slow.
- Better low freq. filter : Gaussian filter

```
gaussian = scipy.signal.gaussian(M=11, std=2)
gaussian /= sum(gaussian)
```

M : the number of elements in the window
std : the standard deviation of the Gaussian distribution

# Gaussian filter 2

The ratio of the spectrums before and after smoothing.

As a low-pass filter, Gaussian smoothing is better than a simple moving average.
After the ration drops off, it stays low

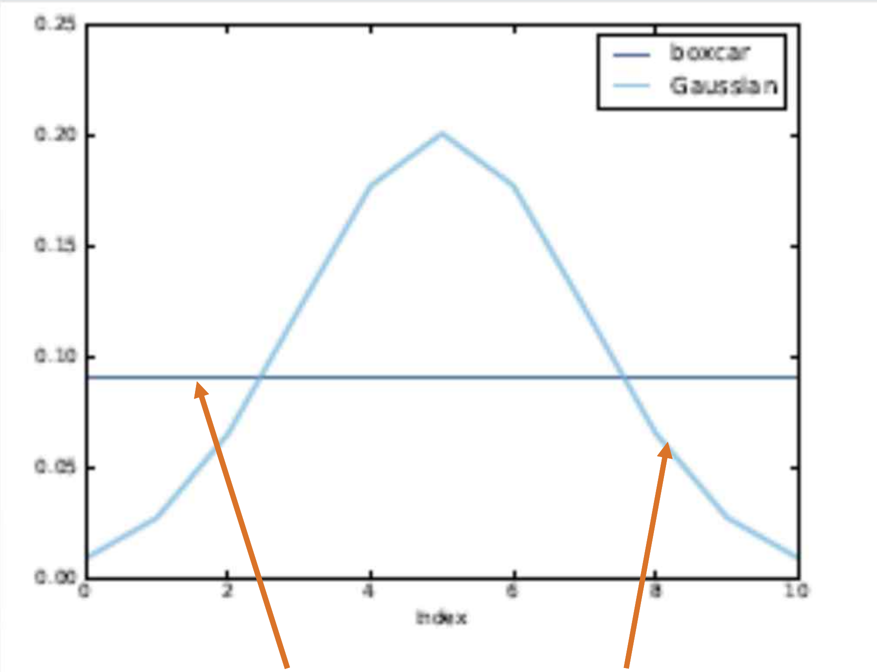
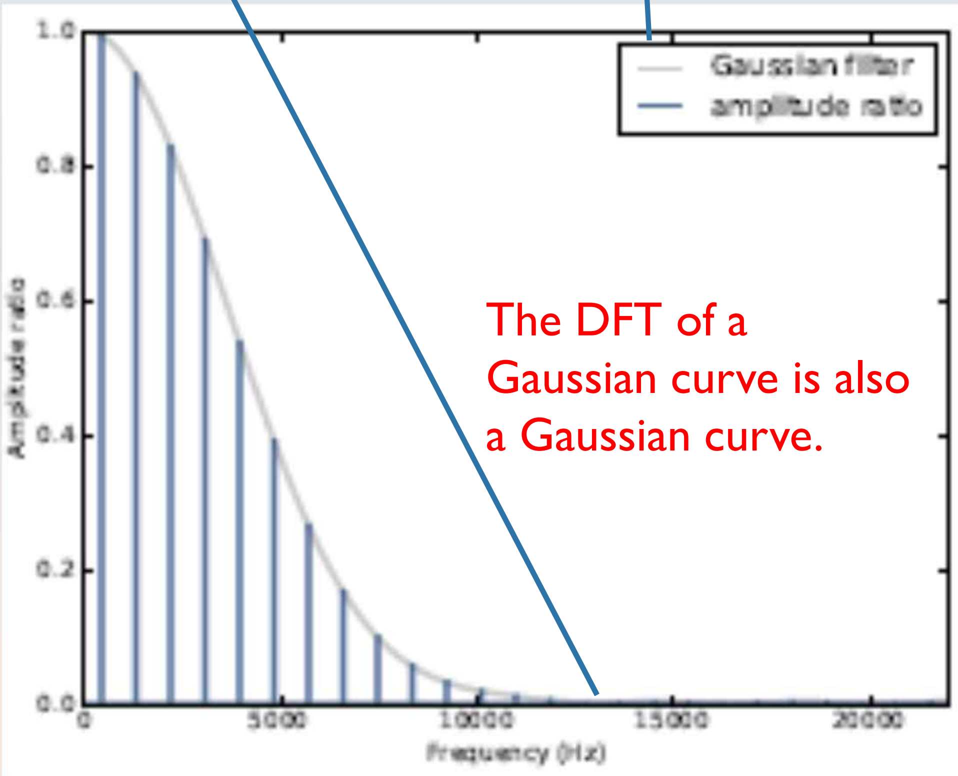



Figure 8.6: Boxcar and Gaussian windows.



# Efficient convolution I

- The FFT provides an efficient way to compute convolution, cross correlation and autocorrelation combined with the Convolution Theorem.
  - Convolution theorem : $DFT(f * g) = DFT(f) \cdot DFT(g)$
  - Convolution computation : $f * g = IDFT\ (DFT(f) \cdot DFT(g))$
  - Complexity change : $O(N^2) \rightarrow O(N \log N\ )$

Pandas is a software library that offers data structures and operations for manipulating numerical tables and time series

```
import pandas as pd

names = ['date', 'open', 'high', 'low', 'close', 'volume']
df = pd.read_csv('fb.csv', header=0, names=names)
close = df.close.values[::-1]
```

close is a NumPy array of daily closing prices

Let's compare two computing methods

df is a dataframe provided by pandas.

# Loading CSV file

## Create dataframe (that we will be importing)

```
raw_data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'last_name': ['Miller', 'Jacobson', ".", 'Milner', 'Cooze'],
        'age': [42, 52, 36, 24, 73],
        'preTestScore': [4, 24, 31, ".", "."],
        'postTestScore': ["25,000", "94,000", 57, 62, 70]}
df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'preTestScore', 'postTestScore'])
df
```

| | first_name | last_name | age | preTestScore | postTestScore |
|---|---|---|---|---|---|
| 0 | Jason | Miller | 42 | 4 | 25,000 |
| 1 | Molly | Jacobson | 52 | 24 | 94,000 |
| 2 | Tina | . | 36 | 31 | 57 |
| 3 | Jake | Milner | 24 | . | 62 |
| 4 | Amy | Cooze | 73 | . | 70 |

# Efficient convolution 2

```
window = scipy.signal.gaussian(M=30, std=6)
window /= window.sum()
smoothed = np.convolve( close, window, mode='valid')
```

$f * g$

$IDFT\ (DFT(f) \cdot DFT(g))$

```
from np.fft import fft, ifft

def fft_convolve(signal, window):
    fft_signal = fft(signal)
    fft_window = fft(window)
    return ifft(fft_signal * fft_window)
```

```
padded = zero_pad(window, N)
smoothed2 = fft_convolve(ys, padded)
```

To remove bogus values at the beginning. Why remove?

```
M = len(window)
smoothed2 = smoothed2[M-1:]
```

Two versions are the same, within floating point error.

# Efficient autocorrelation I

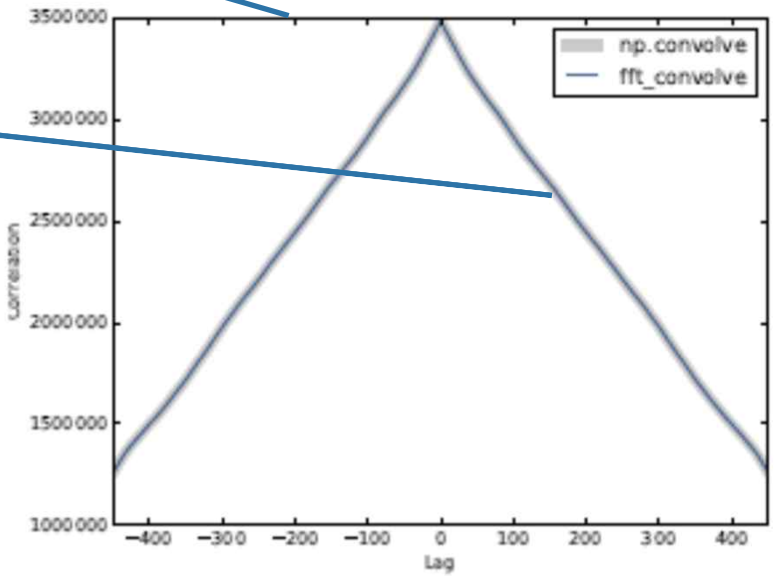
- The difference between cross correlation and convolution
  - The window is reversed.

```
corrs = np.correlate(close, close, mode='same')
```

There's no apparent periodic behavior.
Why?? Hint: find peak.

same : the result has the same length as close (-N/2 ~ N/2)

But, the autocorrelation function drops off slowly. → pink noise!!




To compute autocorrelation using convolution, we have to zero-pad the signal to double the length. This is necessary because the FFT assumes that the signal is periodic.
→ If the signal is not periodic, adding zeros and trimming the result will remove the bogus values

Figure 8.8: Autocorrelation functions computed by NumPy fft_correlate.

# Efficient autocorrelation 2

- Convolution reverses the direction of the window in order to cancel the effect in previous slide.
  - So, reverse the direction of the window before calling fft_convolve.

Make signal length twice by adding zero.

Flips a NumPy array.

roll(array, int) : circular shifting to the right

```
def fft_autocorr(signal):
    N = len(signal)
    signal = thinkdsp.zero_pad(signal, 2*N)
    window = np.flipud(signal)

    corrs = fft_convolve(signal, window)
    corrs = np.roll(corrs, N//2+1)[:N]
    return corrs
```

Questions.
Why need zero padding?
Why trim the corrs to take first and last N/2?
Only using libraries is good strategy?