Time Series and Signal Processing Primer

4 September 2018
Overview

• Basics of time series
• Signal representations
  • Time vs. frequency vs. time/frequency
  • Sampling, quantization
• The Fourier transform
• Filtering
  • Convolution
  • Fast convolution
• Resampling
Time series

- An ordered collection of numbers
  - E.g. Temperatures throughout the year
  - Height/weight throughout a lifetime
  - Stock prices across time
  - Sound or video

- If order is important, then what you have is a time series
  - Doesn’t have to be throughout time!
  - Images are seen as time-series because left-to-right and up-down order matters

<table>
<thead>
<tr>
<th>Average Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
</tr>
<tr>
<td>28.8</td>
</tr>
</tbody>
</table>

![Average Temperature Chart]

![Apple Stock Chart]

![Image as Time Series]
Representing signals

• There are multiple ways to represent time series
  • Some are good, some are bad, some are convenient

• A useful representation is the frequency domain
  • Provides an alternative representation
  • Is very mathematically convenient!
Sound as a time series

- Sounds are a typical case of time series
- We’ll start with sounds as an illustrative example
  - But everything generalizes to all 1-D series
The time domain

- Sound is one-dimensional
  - “Waveform” format
  - Series of numbers denoting instantaneous air pressure

- Not very intuitive for complex sounds like speech and music
  - Can display some information about the energy and the length of the signal
  - But not what it sounds like!
Building blocks for sound

- Sinusoids are special
  - Simplest waveform – a single frequency

- A sinusoid has three parameters
  - Frequency, amplitude & phase
    - $s(t) = a(t) \sin(ft + \phi)$
  - Its simplicity makes sinusoids excellent building blocks for many time series

Making a square wave with sines
Frequency domain representation

- Time series can be decomposed in terms of sinusoidal building blocks
  - That is called the spectrum
- There is no temporal information in the spectrum, only frequencies
- Not that great of a representation for dynamically changing sounds
Time/frequency representation

• Many names/varieties
  • Spectrogram, sonogram, periodogram, ...

• A time-ordered sequence of frequency compositions
  • Can help show how things change in both time and frequency

• Most useful representation so far!
  • Reveals information about both the time and frequency content without loss
A real example

• **Time domain**
  - We see the events
    - How do they sound like though?

• **Frequency domain**
  - We see bass and mids
    - Where are they though?

• **Spectrogram**
  - We “see” all the sounds
    - And have a sense of how they sound
What about other kinds of data?

- Works with all types of time series
- E.g. Energy consumption

Residential energy consumption in IL
How about astronomy?

- **Sunspot count data**
  - Fourier reveals sunspot period
  - Spectrogram reveals more info
The Discrete Fourier Transform (DFT)

- So how do we get from the time domain to the frequency domain?
  - It is a linear transform $\rightarrow$ a matrix multiplication

- The Fourier matrix is square, orthogonal and has complex-valued elements

$$F_{j,k} = \frac{1}{\sqrt{N}} e^{\frac{ijk2\pi}{N}} = \frac{1}{\sqrt{N}} \left( \cos \frac{jk2\pi}{N} + i \sin \frac{jk2\pi}{N} \right)$$

- Just multiply a vectorized time-series with the Fourier matrix and voila!

![Real part](image1.png)

![Imaginary part](image2.png)
And for bigger sizes
How does the DFT work?

- Multiplying with the Fourier matrix
  - Dot-product each Fourier matrix row with input

- Each Fourier row focuses on a single frequency from the signal
  - Since all the Fourier sinusoids are orthogonal there is no overlap

- The resulting vector describes how much of each sinusoid the original vector has
The DFT in a little more detail

- The DFT returns complex numbers

- For real signals DFT is conjugate symmetric
  - The middle value represents the highest frequency
  - The two halves are mirrored complex conjugates

- The interesting parts of the DFT are the magnitude and the phase
  - $\text{Abs}(F) = \|F\|$
  - $\text{Arg}(F) = \angle F$

- To go back to the time domain we apply the DFT again (with some scaling)
Size of a DFT

- The bigger the DFT input the more frequency resolution
  - But the more data we need!

- Zero padding
  - Append a lot of zeros at the end of the input to make up for small inputs
  - But we don’t really infuse any more information we just make prettier plots
From the DFT to a time/frequency

- The spectrogram is a series of DFTs
  - They are applied on successive input segments
    - Which can overlap if desired

- We most often only show the magnitude of the result
  - But we also need the phase for reconstruction

- The parameters to use are
  - The DFT size, the overlap, the window
Why window?

- **The DFT is periodic**
  - The Fourier sinusoids extend infinitely
  - If ends do not match there are discontinuities
  - These imply more sinusoids to explain them

- **Windowing**
  - Tapers the sharp edges of the input and removes the unwanted discontinuities
  - At the expense of changing the input though!

- **Overlap**
  - We need to take overlapping segments to make up for the attenuated parts of the input

---

- **Non-existent broadband content**
  - Windowed vs. Not windowed

- **Nasty sharp edges**
  - Windowed vs. Not windowed

Time/Frequency tradeoff

• Heisenberg’s uncertainty principle
  • We can’t accurately know both the frequency and time location of a wave

• Spectrogram problems
  • Big DFTs sacrifice temporal resolution
  • Small DFTs have bad frequency resolution

• We can use a denser overlap to compensate for time resolution
  • Ok solution, not great
The Fast Fourier Transform (FFT)

- The Fourier matrix is special
  - Unique repeating structure with strong symmetries

- We can decompose a Fourier transform to a function of two Fourier transforms of half the size
  - Two Fourier smaller transforms are faster than one big one
  - We keep decomposing it until we have lots of 2-point DFT
  - Power-of-2 sizes work best

- If you can do something with an FFT, do it
  - It will always result in a computational gain
Image DFTs

- Images are 2d so we need to generalize
- Fourier bases are vertical and horizontal 2d sinusoids
  - Sinusoidal on one axis, constant on the other
  - e.g:

- Images get decomposed using this basis set
Some example DFTs

- $\sin(3x)$
- $\sin(5y)$
- $\sin(3(x+y))$
- $\sin(3(x+y)) + \sin(6x + 8y)$
And some real image examples

Input

2D DFT

Input

2D DFT
How do we compute that?

- DFT both sides of the data matrix
  \[ Y = F \cdot X \cdot F \]
  \[ X \in \mathbb{C}^{N,N}, F_{j,k} = \frac{1}{\sqrt{N}} e^{\frac{2\pi i j k}{N}} \]

- What about tensors?
  - Same thing ...
  
  \[ \text{vec}(Y) = (F \otimes F \otimes \cdots \otimes F) \cdot \text{vec}(X) \]
Sampling Signals

- How do we obtain a signal?
  - Almost always we convert from analog

- Not a straightforward operation!
Signal representation - Quantization

- Converting a real input to a digital number requires quantization.

- We need adequate resolution to represent the original measurements:
  - Precision is measured in bits.
  - Possible noise with small inputs.
  - Possible distortion with large inputs.

- Bit precision (headroom):
  - 8-bit = 48dB, poor.
  - 12-bits = 72dB, maybe ok.
  - 16-bits = 96dB, good.
  - 24-bits = 144dB, little too much.
  - Our ears deal with about 110dB, eyes 90dB.
Quantization in practice

- 11 bits
- 8 bits
- 7 bits
- 5 bits

- 12 bits
- 8 bits
- 4 bits
- 3 bits
Why dynamic range matters

https://www.youtube.com/watch?v=gp_D8r-2hwk
Sampling

• How often we sample measurements is also important!!
  • 1d time-series sampling is usually measured in Hz

• Must sample at least at $2 \times$ the highest frequency we want to represent
  • Because you need at least two samples to represent a period

• Perceptual limits
  • Hearing: we hear up to 20kHz, hence need a sample rate of 40kHz and up
  • Seeing: we cognitively perceive up to 60Hz, suggesting a sample rate of 120Hz and up

• Common sample rates
  • Speech 8kHz to 16kHz, Music: 32kHz to 44.1kHz, Pro-audio: 96kHz
  • Movies: 24fps (recently 48fps), TV: 30fps, HDTV: 60fps (high end up to 600fps!)
Aliasing

- Low sample rates result in *aliasing*
- High frequencies are misrepresented
- Frequency $f_1$ will become (sample rate – $f_1$)
In real-life

- Electricity consumption sampled poorly
- Different sampling makes for different conclusions
Aliasing examples

**Sinusoid sweeping from 0Hz to 20kHz**

- **44kHz SR, is ok**
- **22kHz SR, aliasing!**
- **11kHz SR, double aliasing!**

**Music example**

- Original
- 11 kHz
- 5 kHz
- 4 kHz
- 3 kHz

**In images**

- Original image
- Resampled at 11 kHz
- Resampled at 5 kHz
- Resampled at 4 kHz
- Resampled at 3 kHz

**In video**

- Original video
- Resampled at 11 kHz

---

**Note:**

- Aliasing occurs when the sampling rate (SR) is not high enough to capture the highest frequency present in the signal.
- For example, at 22kHz SR, aliasing occurs because the highest frequency is 20kHz, which is higher than the SR.
- At 11kHz SR, double aliasing occurs because the highest frequency is again higher than the SR, leading to a second aliasing effect.
Extreme video aliasing!

https://www.youtube.com/watch?v=R-lVw8OKjvQ
Convolution

- Operation between two time series
  - Shift and scale one of the time series according to the other one
- Formal definition:
  \[ z = y \ast x \]
  \[ z(t) = y(0) \cdot x(t) + y(1) \cdot x(t - 1) + \cdots \]
  \[ z(t) = \sum_{i} y(i) \cdot x(t - i) \]
- Commutative operation
  - \( x \ast y = y \ast x \)
- Resulting output is sum of lengths minus one
Convolution Examples

Source

Impulse

Result
Image convolution

Input

Convolved with

Results in

\[\text{Input} * \text{Convolved with} = \text{Results in}\]
One more convolution thing

• Example contains a sine and noise
  • The sine is low frequency, the noise is high

• Averaging
  • Convolving with \([\frac{1}{2}, \frac{1}{2}]\)
  • Smoothes input, removes high frequencies

• Differentiating
  • Convolving with \([1, -1]\)
  • Exaggerates noise, removes low frequencies

• Can we generalize?
  • Indeed we can, using filters
Filter types by function

• **Lowpass**
  • Allows only low frequencies through

• **Highpass**
  • Allows only high frequencies through

• **Bandpass**
  • Allows only a band of frequencies through

• **Band-reject/stop**
  • Allows everything but a band of frequencies through

• **Custom**
  • Arbitrary boosting/suppression
How do filters look like?

- Simple time series with specific structure

- How do we design them?
  - Many, many ways
    - Go take a DSP course!
A better way to examine filters

- Filters alter frequency content
  - We can measure this effect by taking a filter’s Fourier transform!

- This is the filter frequency response
  - Allows us to judge the quality and effectiveness of a filter we convolve with

- For simple filters we have
  - The passband – what goes through
  - The stopband – what is filtered out
  - The transition band – the in-between
Filtering trade-offs, Length

- **Length vs. Strength**
  - The longer a filter is the better it can approximate the desired frequency response

- **Length vs. Efficiency**
  - The longer the filter is the more time it takes to compute it
Fast convolution

- Convolution can be quite slow
  - FFT to the rescue!

- Fourier domain multiplication is time domain convolution
  - And vice versa: \( F(x \ast y) = F(x) \odot F(y) \)
  \[
  F(x \odot y) = F(x) \ast F(y) \\
  x \ast y = F^{-1}(F(x) \odot F(y))
  \]

- Using the FFT we get immense speedups
  - The catch is that the latter half of the inputs has to be all zeros
Convolving with matrices

• Convolutions are linear transforms

• We can implement convolution as a matrix product

\[ a \ast x = \begin{bmatrix} a_1 \\ a_2 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \ast \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = C \cdot x = \begin{bmatrix} a_2 & a_1 & 0 & 0 & 0 \\ 0 & a_2 & a_1 & 0 & 0 \\ 0 & 0 & a_2 & a_1 & 0 \\ 0 & 0 & 0 & a_2 & a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} a_2 x_1 + a_1 x_2 \\ a_2 x_2 + a_1 x_3 \\ a_2 x_3 + a_1 x_4 \\ a_2 x_4 + a_1 x_5 \end{bmatrix} \Rightarrow x_i = \sum_{k} x_{i-k} a_k \]

• We can adjust \( C \) to deal with the necessary zero padding
Fun with convolution: Rooms

- Sound bounces on walls
  - Each bounce creates an echo which repeats the same sound delayed
  - Effectively this is a filter!
- We can emulate that with convolution!
  - Make an impulse response that has lots of random echoes at varying levels
  - Also make it decay a bit over time to make more realistic
- Cheap way to make “cathedral emulations”!

*Noise masquerading as a room impulse response*
Filters and images
Resampling

- Changing a signal’s sampling rate
  - Usually to accommodate different playback hardware
    - E.g. CD vs. DAT audio, or 4k vs. 720p videos
  - Also useful to reduce large data sets

- Multirate processing
  - Upsampling, downsampling

- Proper resampling bypasses aliasing
  - Remember that we need to sample correctly!
Downsampling

- Lowering the sampling rate
- Downsampling by 2
  - Picking every other sample will alias!
  - We must first remove the high frequencies first and then pick samples
- For downsampling by $M$
  - Lowpass filter to $1/M$
  - Pick one out of $M$ samples
    - Only works for integer $M$
- High frequencies will be lost!
Upsampling

- Increasing the sample rate
- For upsampling by $L$
  - Put $L-1$ zeros between each sample
  - Lowpass filter to $1/L$
  - Only works for integer $L$
- Without filtering we get noisy clicks
- No loss of information
- Upsampling does not recover high frequencies!
Ditto for images
What if the ratio is not integer?

- Describe it using two integer resampling operations
  - E.g. to go from 44.1kHz to 48kHz
    - Need to upsample by 1.0884
  - This is 480/441
  - We can upsample by 480 and downsample by 441
- Efficient algorithms exist
  - There are ways to avoid explicitly upsampling by 480
    - Reduces storage requirements
- That’s why CDs are 44.1kHz and DATs were 48kHz, to make resampling a pain
Recap

- Time-series
- Time/frequency domain
  - Good for visualizing periodic information
- Convolution
  - Various uses
- Filters
  - low/high/band-pass/stop
- Sampling
  - Aliasing, up/down-sampling, resampling
Further reading

- Wikipedia.org
  - Search any of the terms used so far to get a nice and easy to understand overview
- Mathworld.com
  - Search for cross-correlation/convolution to find out more
- Dspguru.com
  - Nice resource, code, books, tutorials
  - MATLAB’s signal processing toolbox manual page
  - Lots of detailed information about both SP and MATLAB
  - Nice and free overview book
  - Very detailed
Next Lecture

- No more background material!

- Fixed features for signals
  - How we perceive signals
  - How that helps us make machines that do so