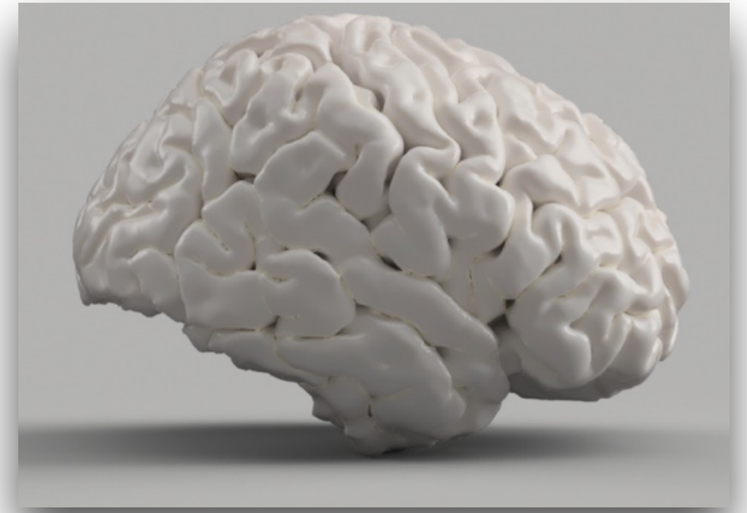
COGS109: Lecture 5



Filtering II,

July 13, 2023

Modeling and Data Analysis

Summer Session 1, 2023

C. Alex Simpkins Jr., Ph.D.

RDPRobotics LLC | Dept. of CogSci, UCSD

Plan for today

- Announcements
- Review of last time
- Lecture 1
  - Review - Sampling, discretization and filtering
  - Loading files examples, some details about making it work
    - ASCII
    - Binary
  - Visualizing and confirming the data, basic slicing in Pandas
  - Some more on filtering and filtering issues
- Lecture 2
  - Perceptually aware visualization

Announcements

- Discussion of groups, repos and timing
  - github repos
  - groups
- A1, D3
- Checking in on paper review

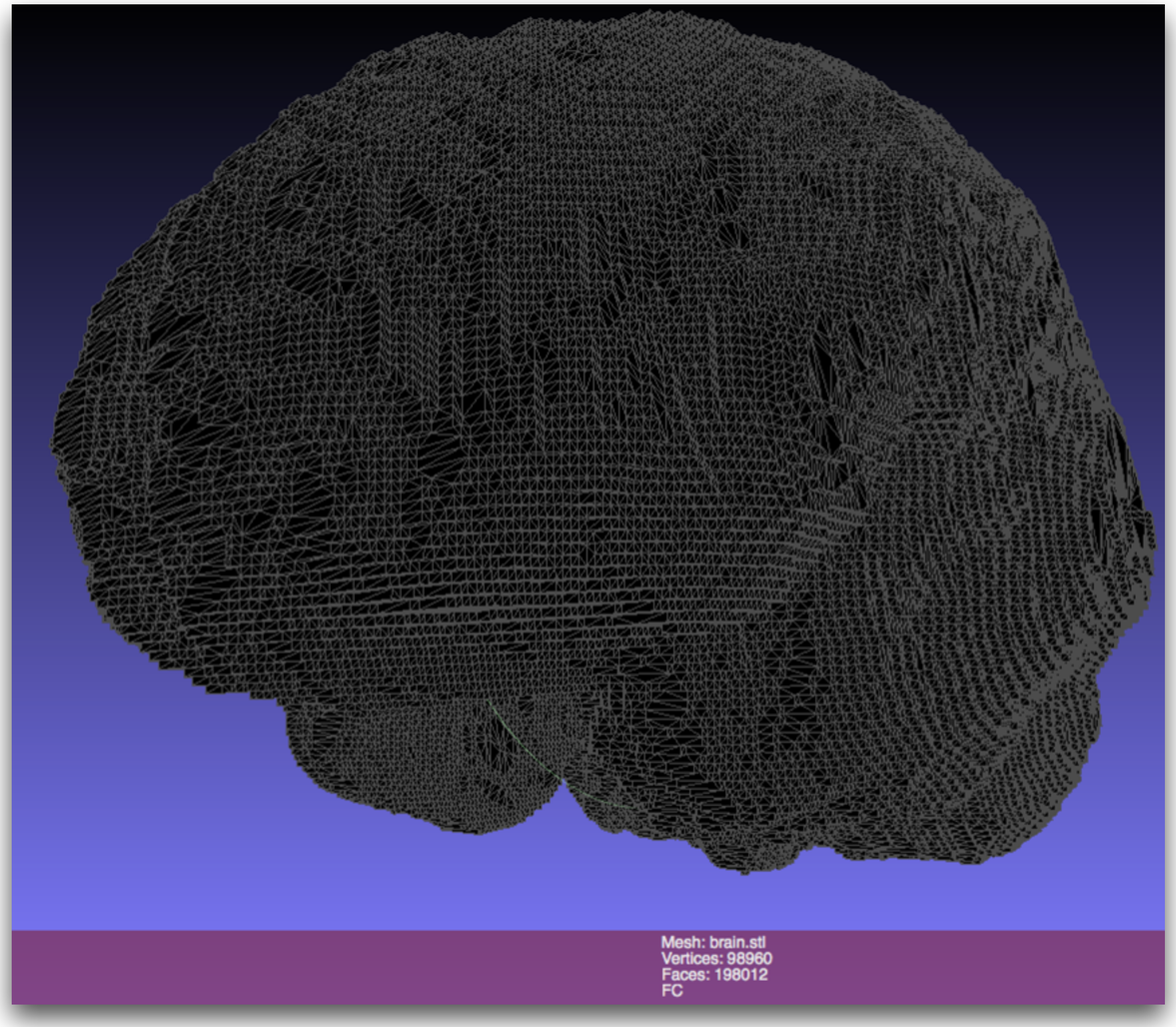
Review and discussion of last time

Quick review- Data structures

- Structured data
- Semi-structured data
- Unstructured data

Types of data files (low level format)

- But how do we encode files in 1’s and 0’s?
- Files can typically be classified into two different formats
  - ASCII (“Text”)
  - Binary
- STL example
  - Brain.STL (ASCII - 52MB)
  - Brain.STL (BINARY - 9MB)



Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	“	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	‘	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

source: https://bournetocode.com/projects/GCSE_Computing_Fundamentals/pages/3-3-5-ascii.html

How to load text files in Python

- Depends on the file type (generic or specific)
- https://www.geeksforgeeks.org/reading-writing-text-files-python/
- Python can load either generic text or binary files
- **`open()`** function
- No special module needed
- very similar to C
- Add that ‘r’ if the file is not in the same folder as the script/current directory in order to make the string raw and avoid processing special characters

```
File_object = open(r"File_Name","Access_Mode")
```

Loads a file into ‘primary memory’ or RAM

- Secondary memory is your nonvolatile storage
- If successful, returns a file ‘handle’ that allows you to then access that memory
- Pay attention to the mode it is opened in (r, r+, w, w+, a, a+)
- other operations:
  - file_handle.close()
  - file_handle.write()
  - file_handle.read()
  - more…(e.g. https://www.geeksforgeeks.org/reading-writing-text-files-python/)

```
# Open function to open the
file "MyFile1.txt"
# (same directory) in append
mode and
file1 =
open("MyFile1.txt","a")

# store its reference in the
variable file1
# and "MyFile2.txt" in D:\Text
in file2
file2 = open(r"D:\Text
\MyFile2.txt","w+")
```

File access modes using open()

1. **Read Only (‘r’) :** Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises the I/O error. This is also the default mode in which a file is opened.
2. **Read and Write (‘r+’):** Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exist.
3. **Write Only (‘w’) :** Open the file for writing. For the existing files, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exist.
4. **Write and Read (‘w+’)** : Open the file for reading and writing. For an existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.
5. **Append Only (‘a’)**: Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.
6. **Append and Read (‘a+’) :** Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

Source: https://www.geeksforgeeks.org/reading-writing-text-files-python/

Load various files using Pandas

- https://pandas.pydata.org/pandas-docs/stable/reference/io.html
- **pd.read_csv('data.csv')**
- **pd.read_table(‘data.csv’)**
- A bit simpler?

How to load text files in Matlab/Octave

- *Import wizard* menu (also works for matlab binary files)
  - Demo
- *M=xlsread('filename')*
  - Reads an excel spreadsheet file and stores it into a matrix of your choosing (here it's M)
- *Load filename .ext*
  - loads the data in the ASCII text file *filename.ext (*where *.ext* is the extension of the filename, such as .txt)
- Octave documentation: https://docs.octave.org/latest/Simple-File-I_002fO.html

Binary files and .mat files

- A more efficient way to store files generally is binary format
  - Smaller
  - But…Less platform independent - ie need to know exactly what the format is to read the file
  - Can’t load these files into just any text editor like you can with ASCII
  - Image files are examples of binary
  - Matlab stores a binary format with the extension .mat
  - Python and Pandas can read/write binary files fairly simply as well
    - Have to choose carefully what techniques you use - with large files the slower approaches might not work due to being too slow or memory intensive

Tidy data == rectangular data

A

	A	B	C	D	E
1	id	sex	glucose	insulin	triglyc
2	101	Male	134.1	0.60	273.4
3	102	Female	120.0	1.18	243.6
4	103	Male	124.8	1.23	297.6
5	104	Male	83.1	1.16	142.4
6	105	Male	105.2	0.73	215.7

Broman KW, Woo KH. (2017) Data organization in spreadsheets. *PeerJ Preprints* 5:e3183v1 https://doi.org/10.7287/peerj.preprints.3183v1

Data wrangling vs. data cleaning

- Data wrangling focuses on transforming the data from a ‘raw’ format into a format suitable for computational use
- Data cleaning focuses on, as discussed, fixing/removing incorrect, corrupted, incorrectly formatted, duplicate, incomplete, data within a dataset

Loading binary files in Python

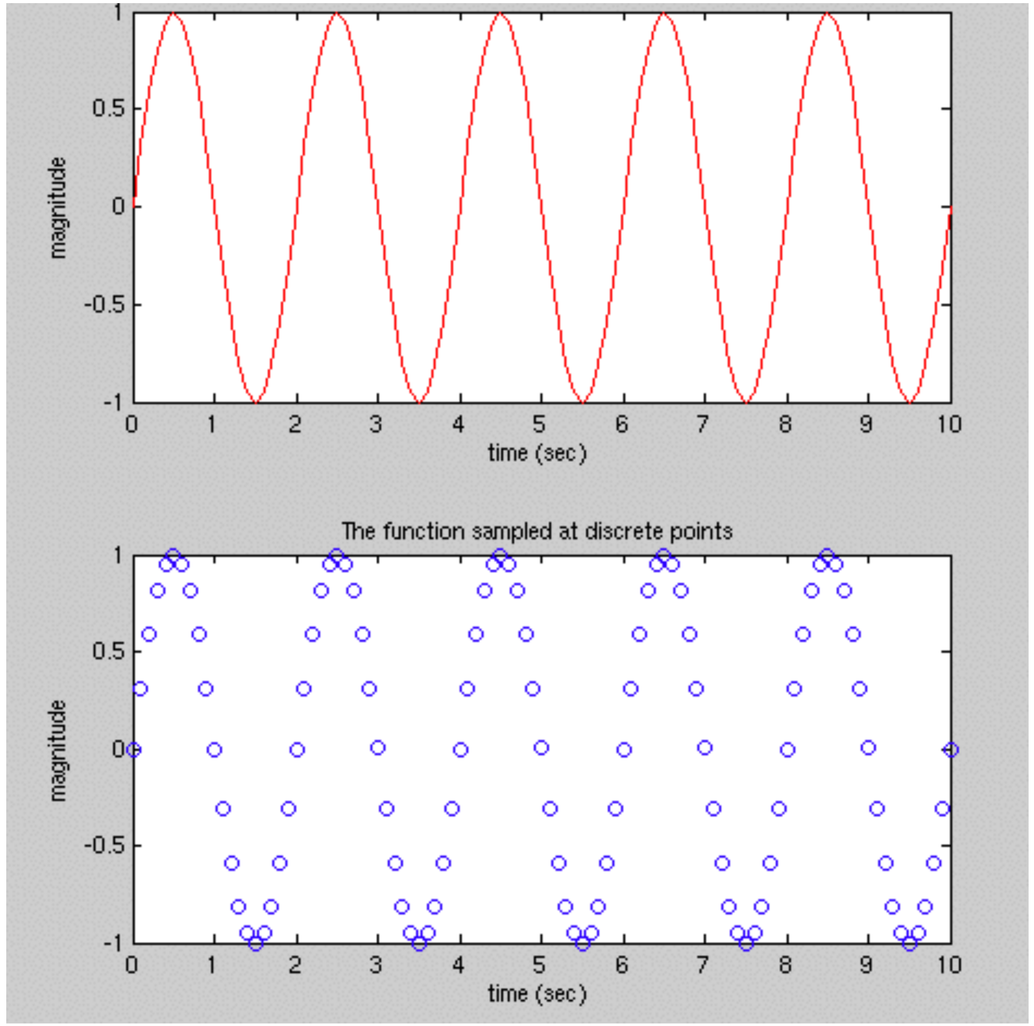
- One approach (https://stackoverflow.com/questions/16573089/reading-binary-data-into-pandas)
- There are many ways to do this
  - Often you will work with standardized formats or formats that provide tools if from a commercial system
  - Not always
  - Knowing how binary files and text files work as well as both simplifying functions and low level python functions allows you to work with anything

Non-standardized binary data

- So some file you know the structure
  - Data acquisition, image file (there would be a module normally though), other arbitrary type
  - **Need documentation for how the bytes encode the data**
  - Typically either just a sequence of numbers and you have to know the order or…
  - A file with a header then body, header specifies the rest
  - A series of records consisting of a header (identifying info) and the record one after the other
  - https://towardsdatascience.com/loading-binary-data-to-numpy-pandas-9caa03eb0672

Continuous vs. Discrete quantities

- Information storage
  - **Continuous** signals have information at every point in time
  - **Discrete** signals have info only at specified intervals (fixed or variable)



Examples of continuous and discrete systems

- Continuous or discrete?
  - –# of people in this class •Discrete
  - –# of Time zones •Discrete
  - –Time •Continuous
  - –Answers on multiple choice tests •Discrete
  - –A Sound •Continuous
  - –Body temperature •Continuous

Analog vs. Digital quantities

- Information storage
  - **Analog** contains infinite information
  - **Digital** contains limited information, depending on the number of bits of information the digital value can store
    - 0 or 1 in each bit means each bit multiplies the possible combinations of numbers by 2
    - 2^4 = 0-15 (a 4-bit number, 16 different values)
    - 2^8 = 0-255 (an 8-bit number, 256 different values)
    - 2^16 = 0-65535 (a 16-bit number, 65536 different values)

More on digital quantities

- Measuring an EEG boils down to recording a sequence of numbers into computer memory, stored in values of a specific size, such as 8 bit numbers.
  - i.e. signal is 0-5V, digitized with 8 bit ***precision*** would yield a ***resolution*** of 5V/256 = 0.020V, or 20mV (mV = 'milli-Volts')
  - **Resolution -** defined as the smallest quantity which can be reliably measured
  - **Digital Precision -** The number of bits of information contained in a digital quantity
- Also important for computations
  - Round off errors can accumulate
    - Example
      - 2.245+3.432+1.234 = 6.911
      - 2+3+1 = 6, and that's only 3 samples! Imagine 1000/sec (1kHz) !
  - More on this later

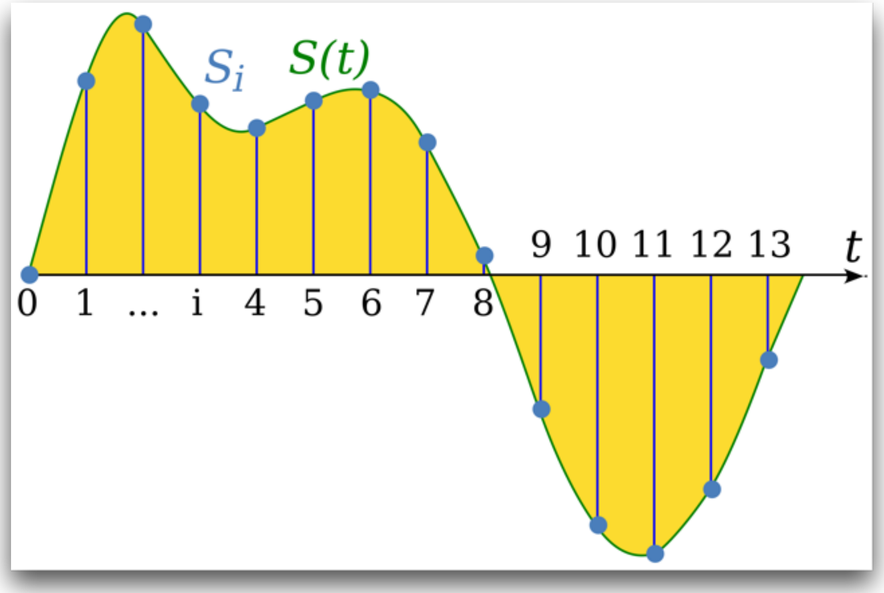
Discretization

- Measuring a continuous (analog) signal means capturing information at specified (fixed or variable) intervals
  - **Sampling frequency -** the frequency at which data is recorded from a signal (Typically in Hz, ie 5kHz)
- When capturing data, or when manipulating data which has been discretized, there are several issues to consider
  - Aliasing (not the TV show:)
  - Sampling rates
  - Post-processing – filtering data to remove unwanted information while retaining desired information

Sampling

- **Sample -** We record data at specific points in time
- **Period -** The time between samples, ***T*** **[sec]**
- **Sample frequency -** The frequency of sampling, *f* [Hz]

$$f = \frac{1}{\Delta T}$$



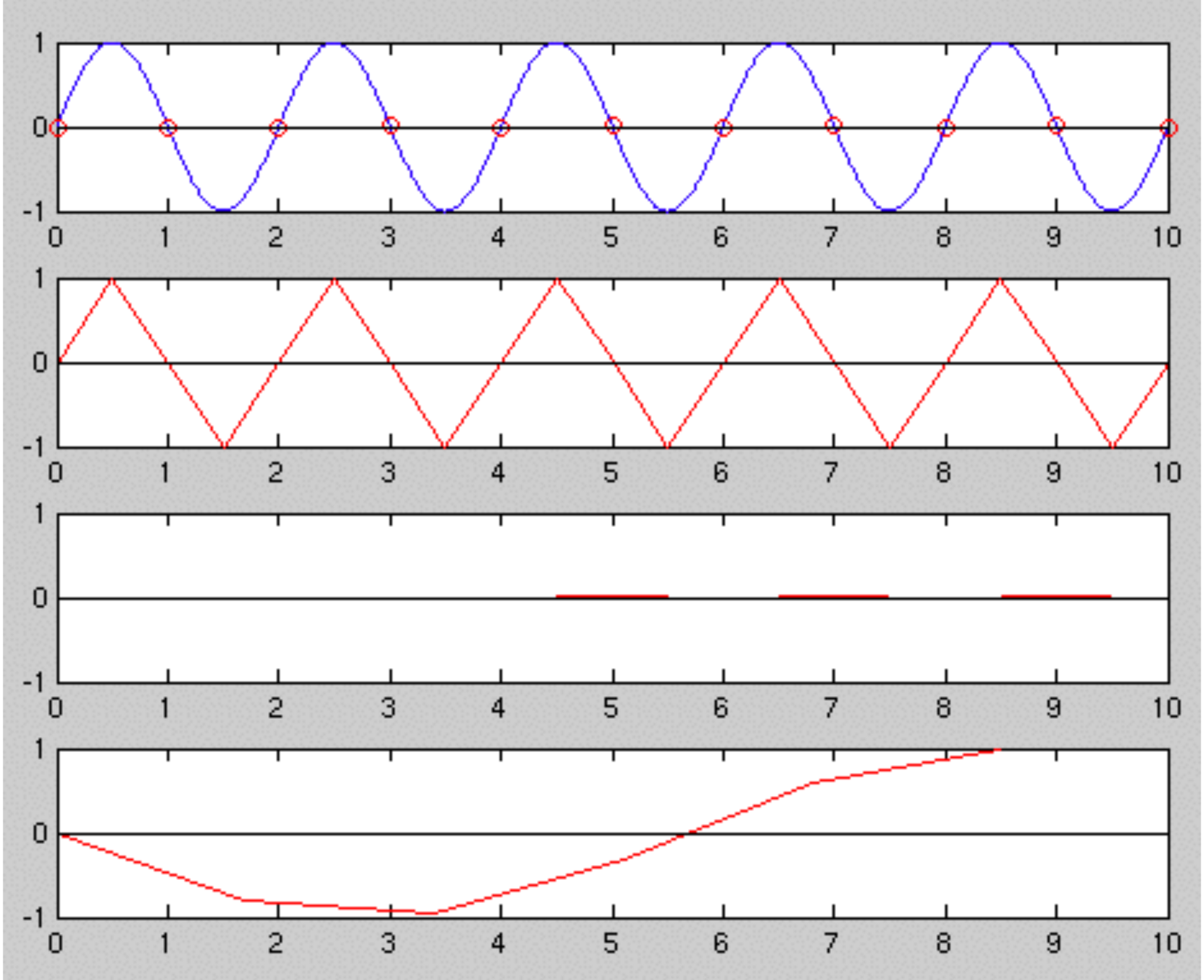
https://en.wikipedia.org/wiki/Sampling_%28signal_processing%29

Nyquist and Sampling

- Stories
  - Running in the dark with periodic lights on the ground, with sharp turns
  - Ping pong (no sound, periodic view of the system)
- As a rule of thumb, you must sample AT LEAST twice as fast as the highest frequency you want to measure
  - **Nyquist frequency -** max freq. that can be measured [Hz]
  - **Nyquist rate -** sampling frequency (which is 2x the nyquist frequency) required to sample at the nyquist frequency
  - 20 times as fast is better
  - Filter out higher frequency components

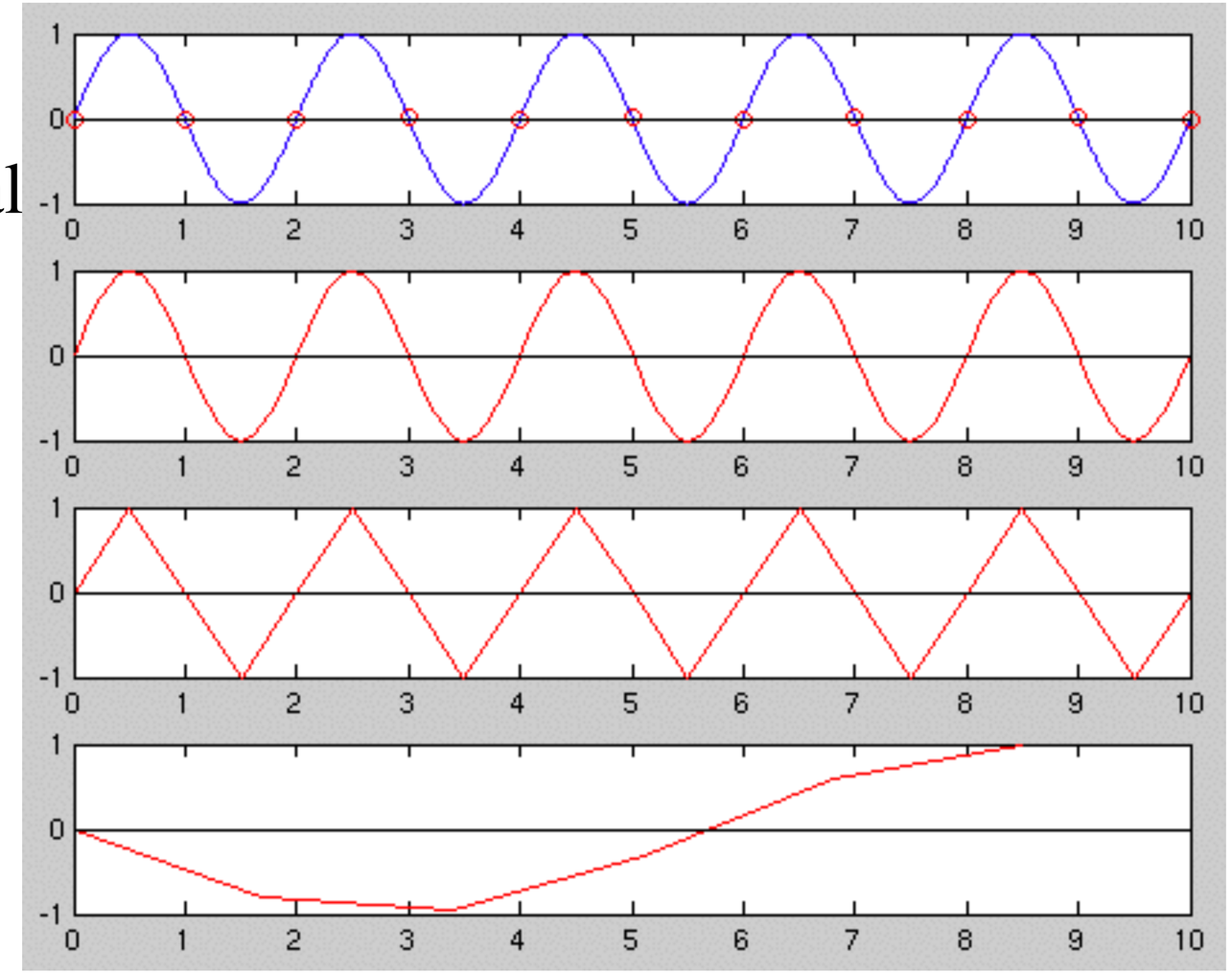
Nyquist frequency

$$f_n = \frac{1}{2\Delta T}$$


1
0
-1
0 1 2 3 4 5 6 7 8 9 10
1
0
-1
0 1 2 3 4 5 6 7 8 9 10
1
0
-1
0 1 2 3 4 5 6 7 8 9 10
1
0
-1
0 1 2 3 4 5 6 7 8 9 10

What do we see in this picture?

- **Aliasing -** the corrupting of a signal by components of higher frequencies overlapping into the lower frequency



How do we solve this?

- Filter out the frequencies we don’t want
  - Low pass filter
  - High pass filter

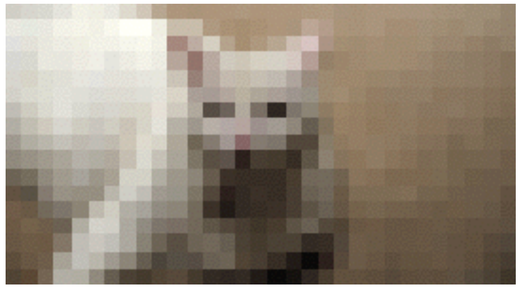
Examples: Visual discretization

- Color shading

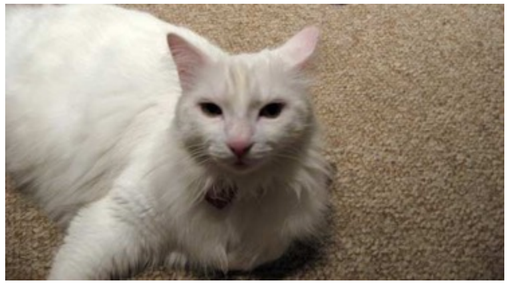
- Color and visual boundaries:

Few colors and low spatial resolution

Low spatial resolution only

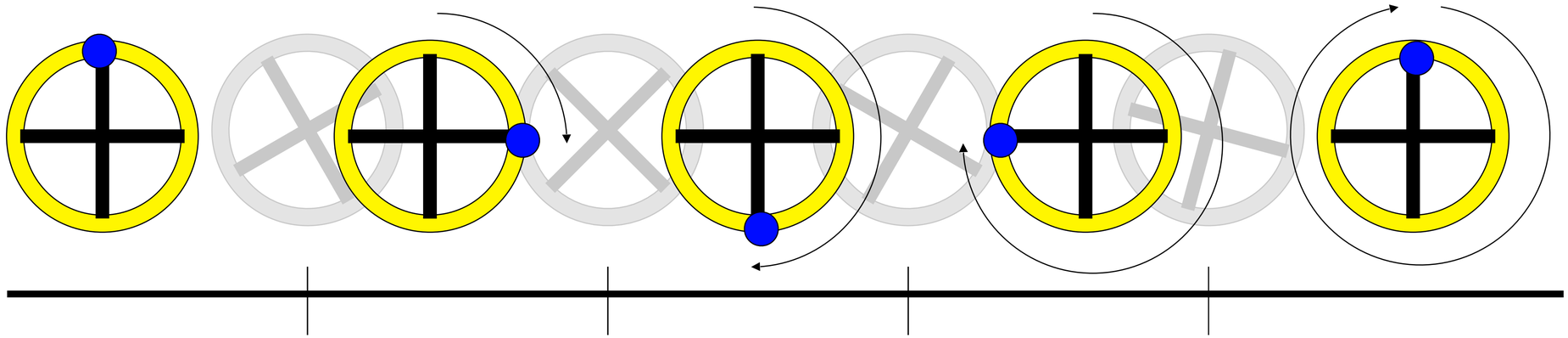


High spatial resolution and colors

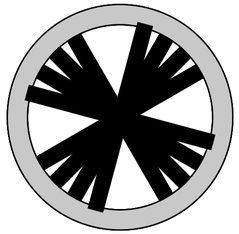


Example: Sampling and Aliasing

- The wheel spokes example…<Live demo>



- We're sampling at too slow a rate to accurately see the spokes rotate, and at a particular rotational velocity of the wheel, we see an 'aliased' reverse rotation!

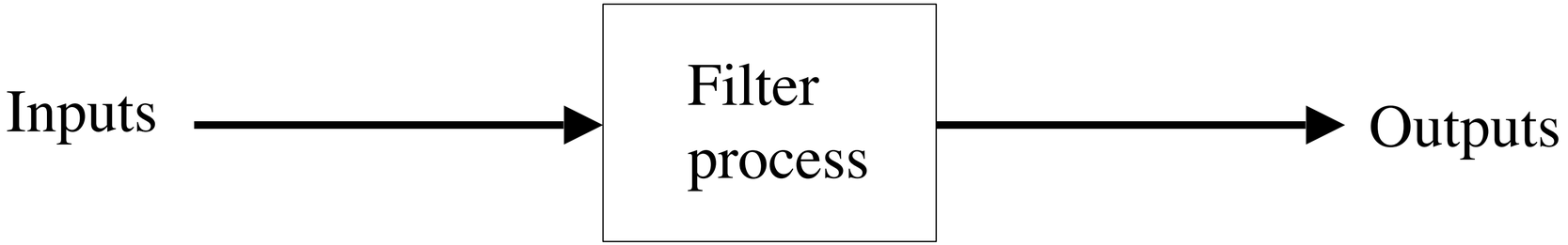


Obviously aliasing can be bad…

- Aliasing can lead to improper interpretations of data
  - **So what do we do about it?**
    - We must first sample at twice the rate of the fastest signal we care about
    - Filter our data (humans do this, and so do cognitive scientists!)

Thus we filter our data…

- **Filter -** an operation or process which alters input data according to some mathematical relationship or heuristic rule to produce output data which is more desirable



Computational filtering

- *Noisy auditory data can be filtered to remove undesired signals*
- *EEG signals can be filtered to remove 60Hz noise from AC lines nearby*
- *Other sensor signals can be filtered to improve results*

Frequency Response

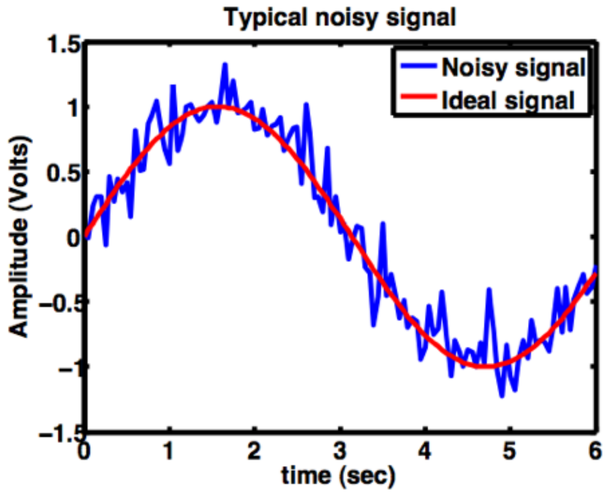
- Linearity of systems vs. nonlinearity
- The response of a linear system to a sinusoidal input is a sinusoidal output with the amplitude and phase shifted in some way
  - <demo volunteer needed>
- This is useful for characterizing the behavior of some signal over a range of possible input frequencies
- Example with the chalk

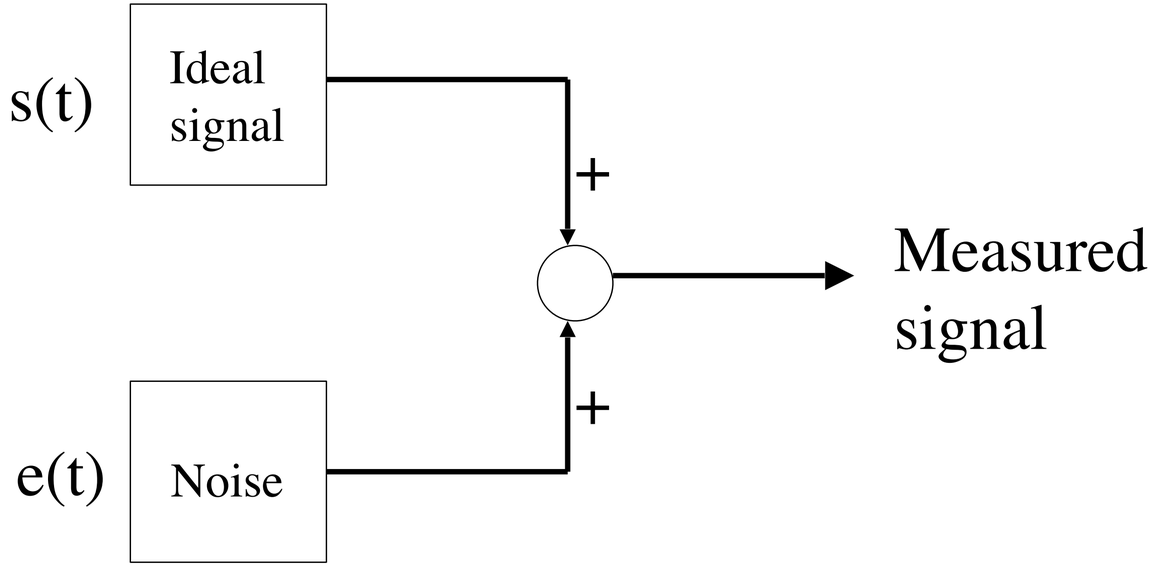
Common filter types in signal processing

- **Low-pass filter** - (ideal) attenuates high frequency data, while allowing low frequency data to pass unchanged
- **High-pass filter** - (ideal) attenuates low frequency data, while allowing high frequency data to pass unchanged
- **Band-pass filter** - (ideal) attenuates all frequencies except a particular frequency band (or bands)
- **Band-stop filter** - (ideal) attenuates one or a selection of frequency ranges of data, allowing all the rest to pass unchanged
- Actual filters are not exactly ideal…which we will discuss

Signals and noise…

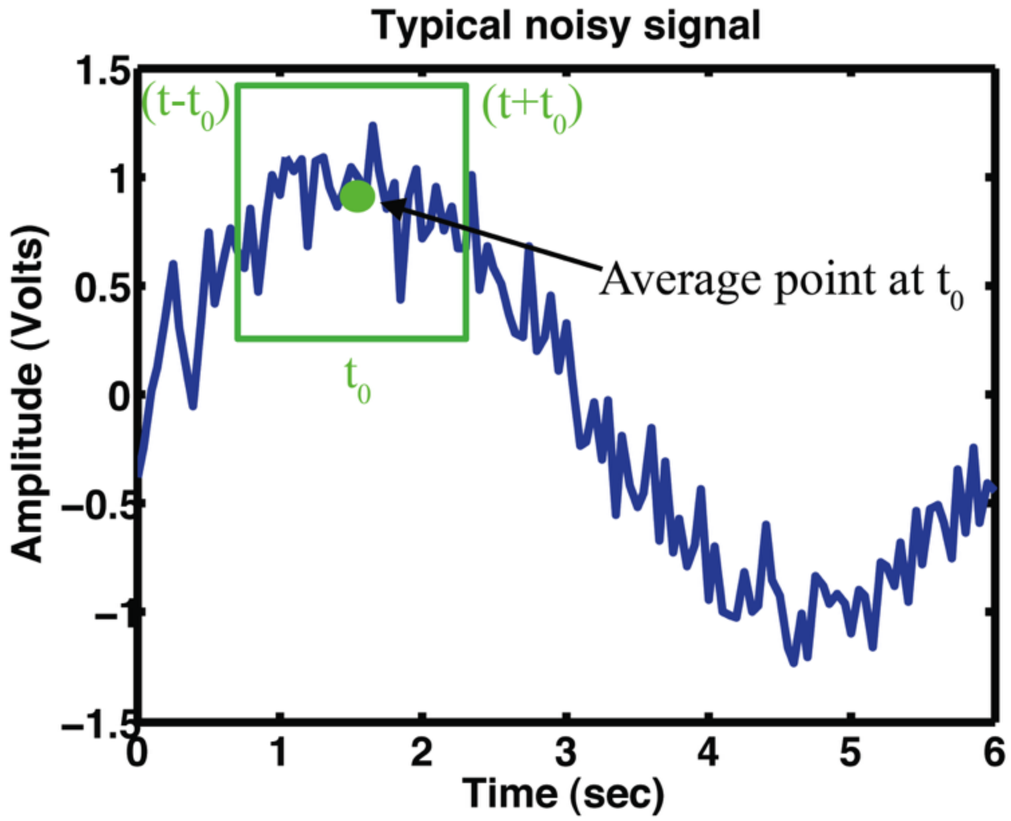
- By making assumptions about the properties of the unwanted ‘noise’ *e(t),* we can reconstruct an appropriate *estimate* of the original signal *s(t)*
  - **Noise - any unwanted portion of a signal, lumped together. It may come from multiple sources but tends toward some statistically predictable properties**





Low-pass filtering

- So the effect is this



More on linearity vs. nonlinearity

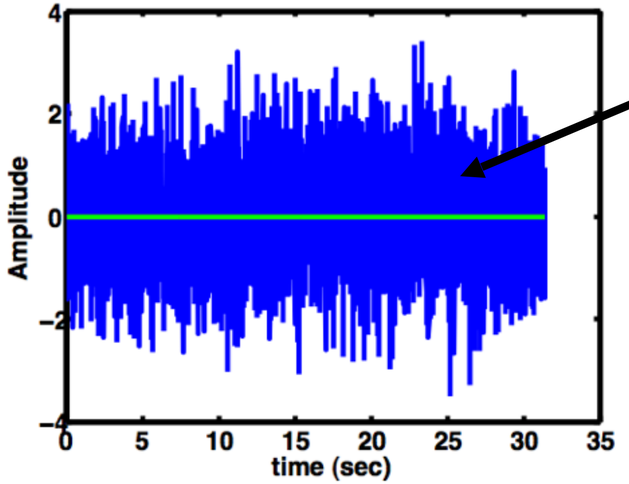
- Power
  - **A linear system is a system whose dependent variables are related to its independent variables by a power of one**
- Linear systems have these particular properties (and they are very favorable)
  - **Additive** $$T[x_1(n) + x_2(n)] = T[x_1(n)] + T[x_2(n)]$$
  - **Homogeneous** $$T[cx(n)] = cT[x(n)]$$
- Linear differential equations are more well-understood than nonlinear differential equations

Fourier transforms

- Frequency domain example : Musical note vs. the sound
  - **More parsimonious to describe a song in terms of its notes than time domain signal (when creating a ‘model’ for a song which can be communicated)**

We return to noisy data which we want to ‘clean up’

- We do this by removing undesired components of the signal
- One way to do this is *averaging* out the noise
- If it’s Gaussian and additive…

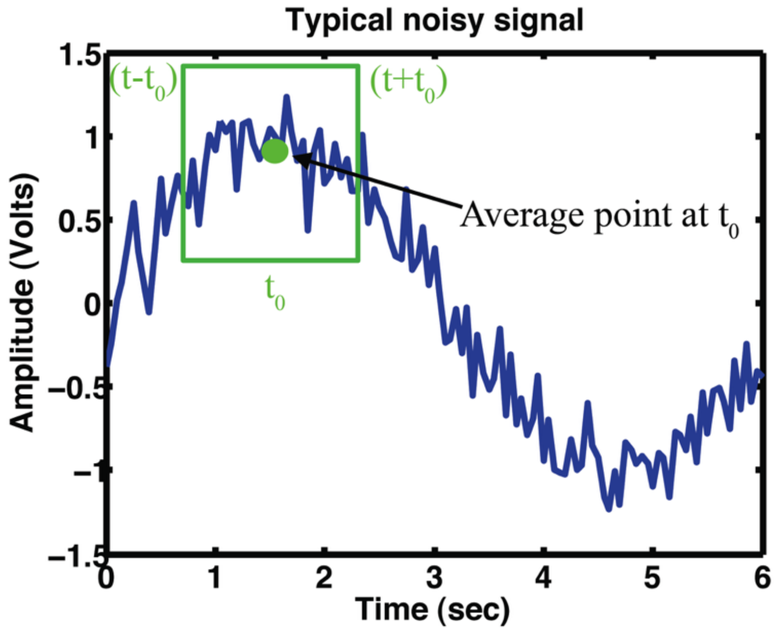


This is gaussian noise, and the average of this is approximately the green line, 0

$$-5 + 5 = 0$$

How to do it

- **Decide on a ‘window’ of data to average over, which is narrower than the fastest component to your changing signal**
- **Sum up over that window of points and divide by the number of points (average)**



Continuous form

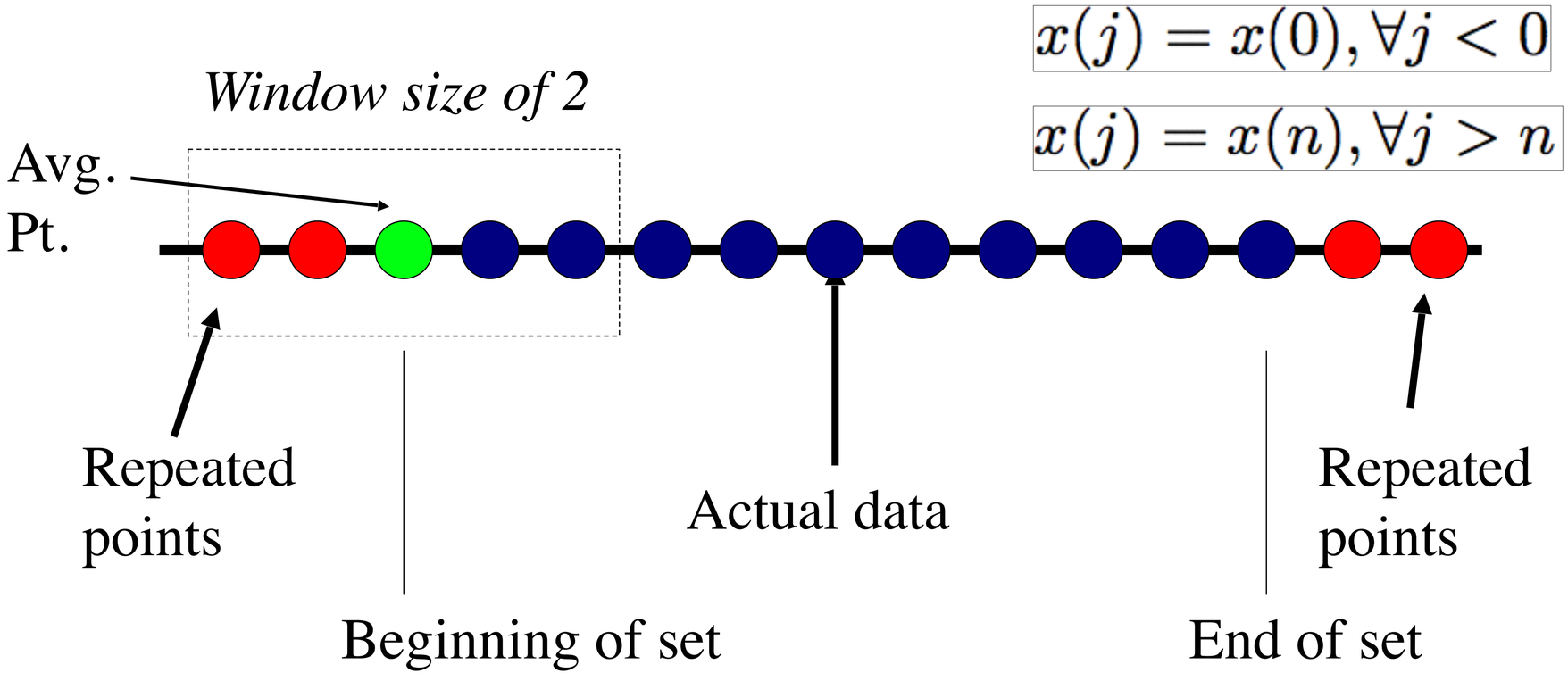
$$x_f(t) = \int_{t-t_0}^{t+t_0} x(\tau)d\tau$$

Discrete form

$$x_f(i) = \frac{1}{2k+1} \sum_{j=i-k}^{i+k} x(j)$$

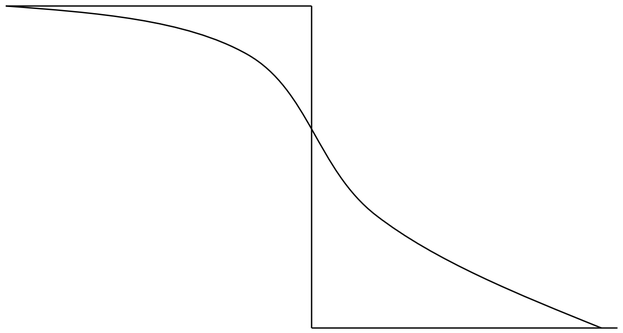
A few details

- What about at the ends of the data where we don't have information before (at the beginning of the data set) or after (at the end of the data set)?
  - **Copy the first or last point and repeat as necessary**



Disadvantages…

- Need to have all data in memory already, so it isn't an 'online' filter
- Causality
  - **If we care about an exact event timing, this is a poor filter to use:**



Signal anticipates changes!

Solution

- Recursive filter
  - Solves causality issue
  - Easy to implement as we saw last time

Before we get into writing the filters, let’s practice getting data

The data

- Let’s import the dataset
- Then let’s look it over - how many points do we have? What are our variables? Is there a problem with the signal? Is there anything we can do about it?
- <to the workbook>