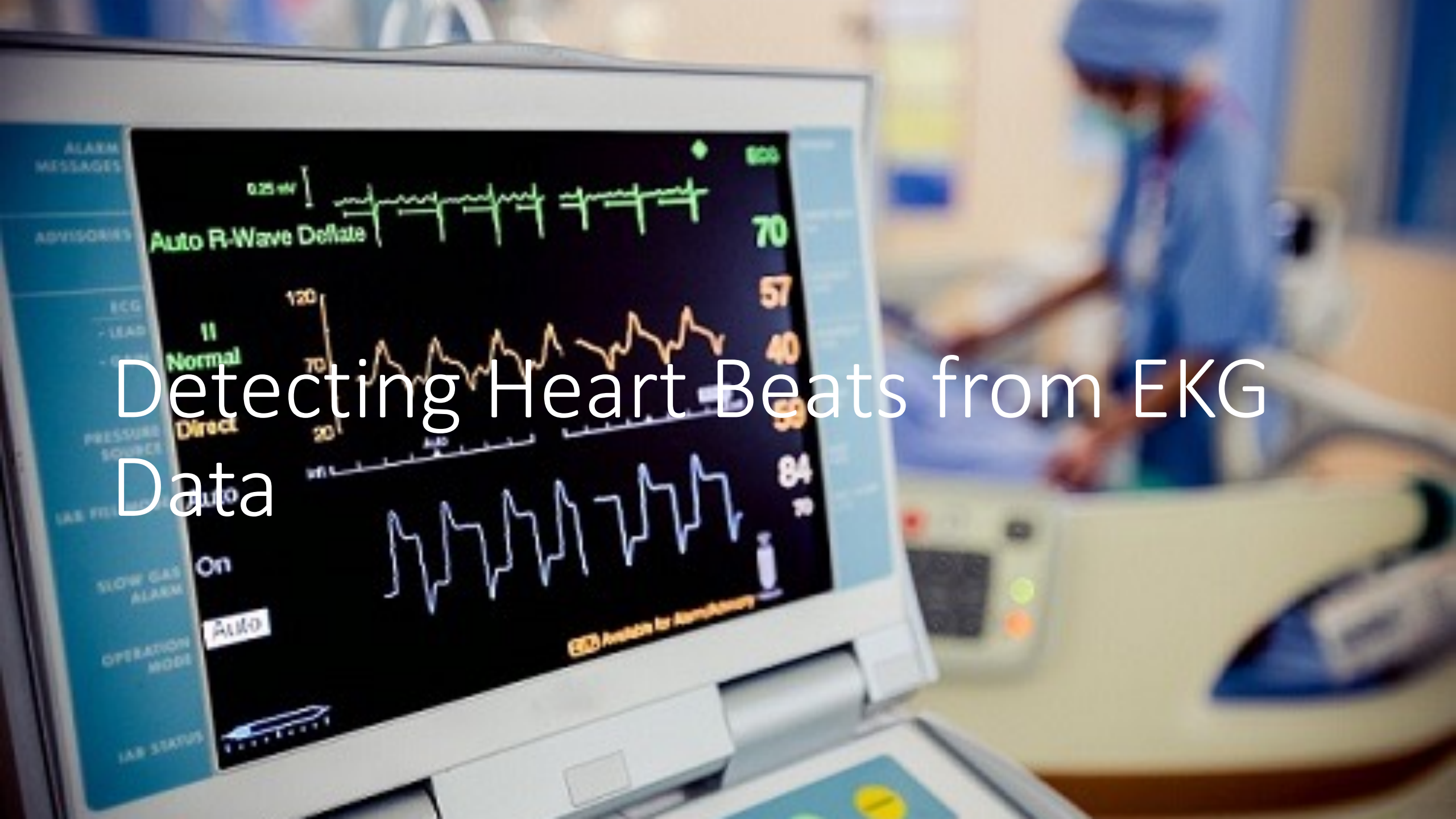
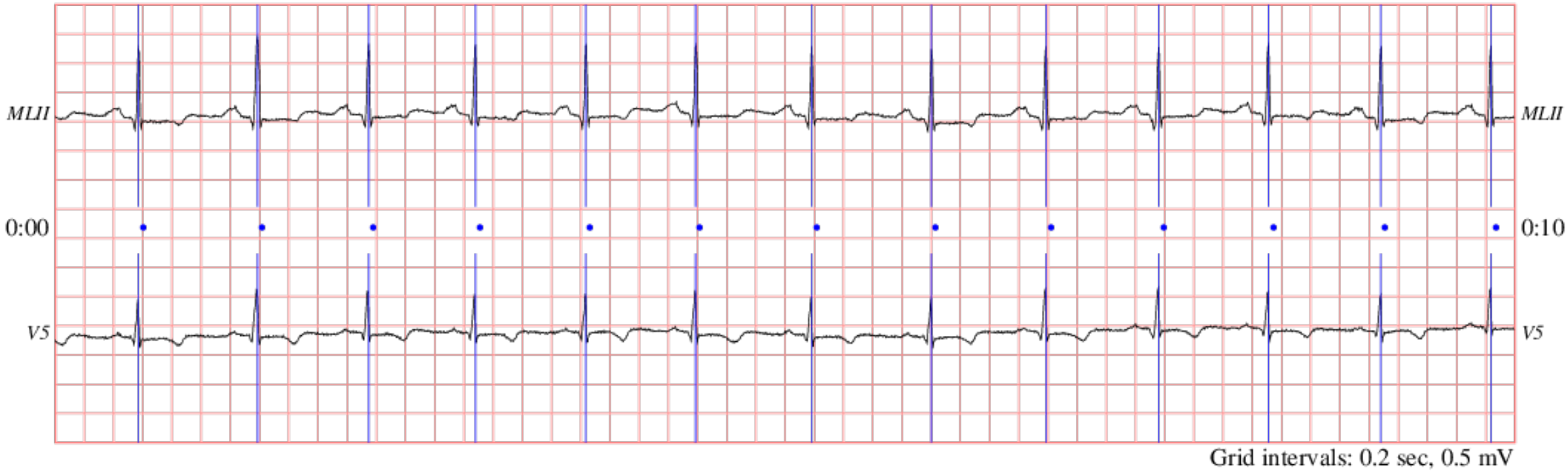


Detecting Heart Beats from EKG Data

PhysioNet Database



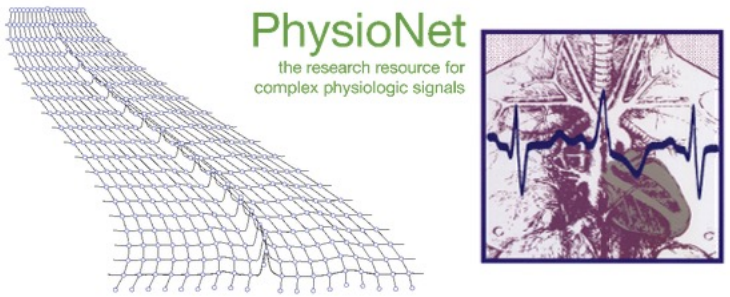
- Database with heartbeat, chest x-ray, sleep data (weight, posture), MRI, limb movement…. Basically anything physiological that you'd like to know.

- Data is annotated by algorithm or human. Can compare approaches against the "gold" standards

Open databases

- Abdominal and Direct Fetal ECG Database: Multichannel fetal electrocardiogram recordings obtained from 5 different women in labor, between 38 and 41 weeks of gestation.
- AF Termination Challenge Database: ECG recordings created for the Computers in Cardiology Challenge 2004, which focused on predicting spontaneous termination of atrial fibrillation.
- AHA Database Sample Excluded Record: Two ECG signals that were excluded from the 1980 American Heart Association database.
- A multi-camera and multimodal dataset for posture and gait analysis: Multimodal dataset with 166k samples for vision-based applications with a smart walker used in gait and posture rehabilitation. It is equipped with a pair of Depth cameras with data synchronized with an inertial MoCap system worn by the participant.
- ANSI/AAMI EC13 Test Waveforms: The files in this set can be used for testing a variety of devices that monitor the electrocardiogram. The recordings include both synthetic and real waveforms.
- Apnea-ECG Database: Seventy ECG signals with expert-labelled apnea annotations and machine-generated QRS annotations.
- A Pressure Map Dataset for In-bed Posture Classification: Pressure sensor data captured from 13 participants in various sleeping postures.
- Auditory evoked potential EEG-Biometric dataset: Recording of electroencephalogram (EEG) signals with the aim to develop an EEG-based Biometric. The Data includes resting-state and auditory stimuli experiments.
- Autonomic Aging: A dataset to quantify changes of cardiovascular autonomic function during healthy aging: This database contains resting recordings of ECG and continuous noninvasive blood pressure of 1,104 healthy volunteers

George B. Moody PhysioNet Challenge



Quick links for this year's Challenge:

- Registration form
- Example MATLAB and Python classifier and scoring code
- Submission instructions and form
- Leaderboard
- Public discussion forum
- Rules and deadlines
- Current and General FAQs
- About

George B. Moody PhysioNet Challenges

For the past 23 years, PhysioNet and Computing in Cardiology have co-hosted a series of annual challenges, now called the George B. Moody PhysioNet Challenges, to tackle clinically interesting questions that are either unsolved or not well-solved.

The George B. Moody PhysioNet Challenge 2022 invites participants to identify murmurs in heart sound recordings collected from multiple auscultation locations.

We ask participants to design and implement a working, open-source algorithm that, based only on the provided recordings and routine demographic data, can determine whether any murmurs are audible from a patient's recordings. We have designed a scoring function that reflects the burden of algorithmic pre-screening, expert screening, treatment, and missed diagnoses. The team with the lowest score wins the Challenge.

Please check the below links for information about current and past Challenges, including important details about scoring and test data for previous Challenges.

Challenge: Implement a Heart Beat Detection Algorithm. Assess Accuracy on Various Datasets.

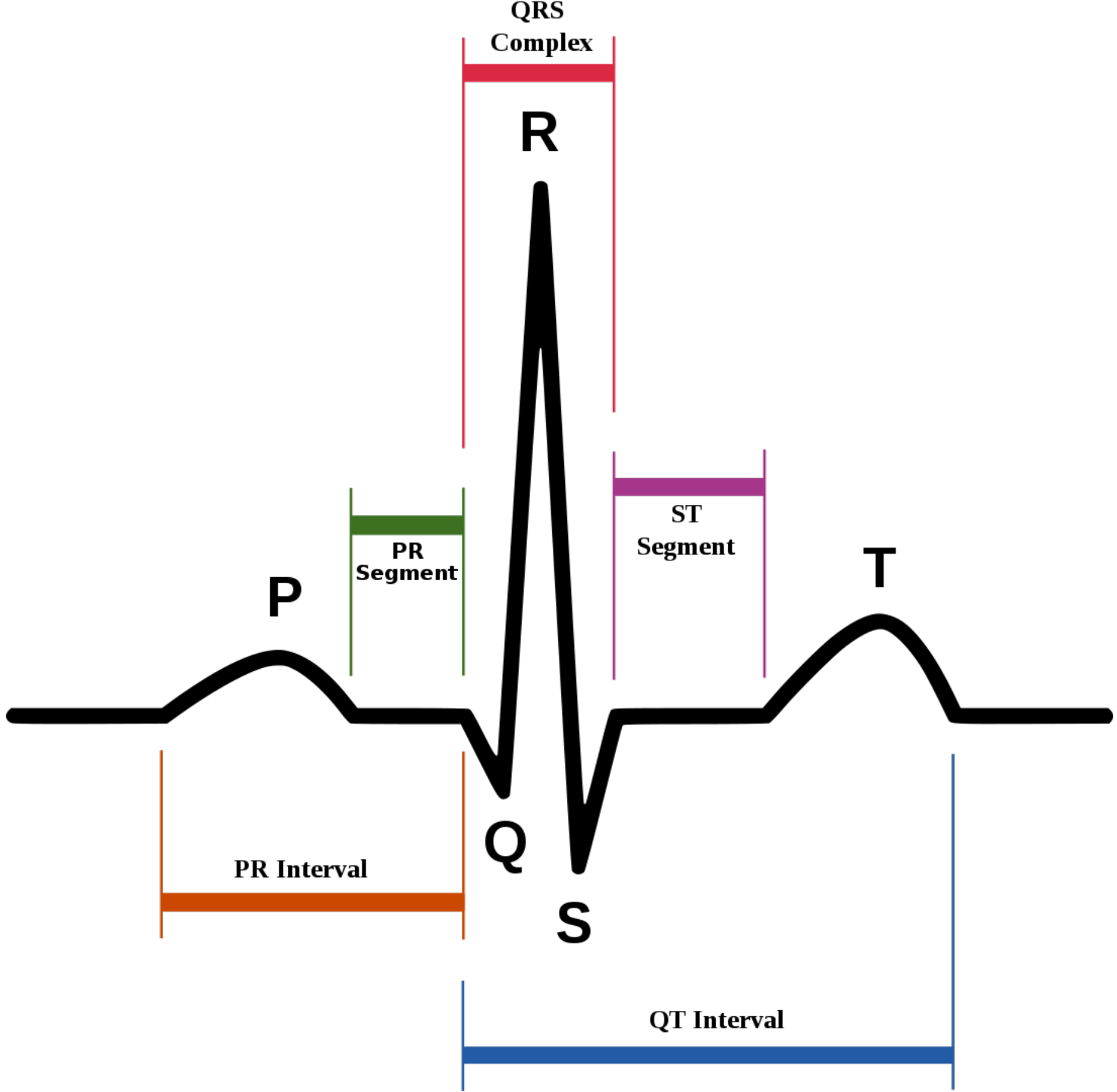
A Sample from the QT Database

```
'Elapsed time','MLII','V5'
'hh:mm:ss.mmm','mV','mV'
'0:00.000',4.725,4.775
'0:00.004',4.735,4.745
'0:00.008',4.725,4.735
'0:00.012',4.715,4.745
'0:00.016',4.720,4.735
'0:00.020',4.705,4.725
'0:00.024',4.700,4.710
'0:00.028',4.690,4.715
```

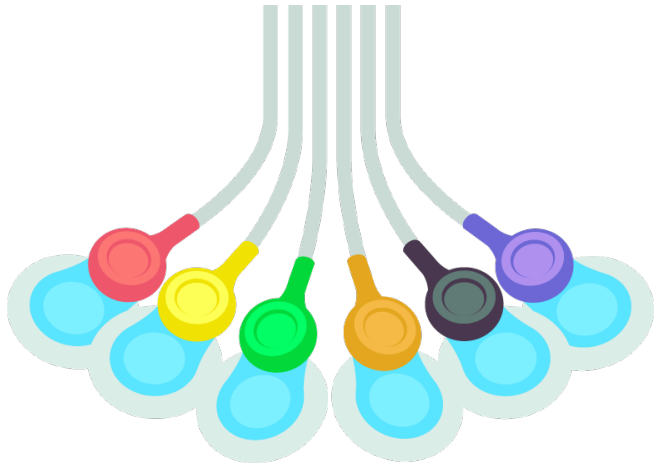
Sample EKG Reading

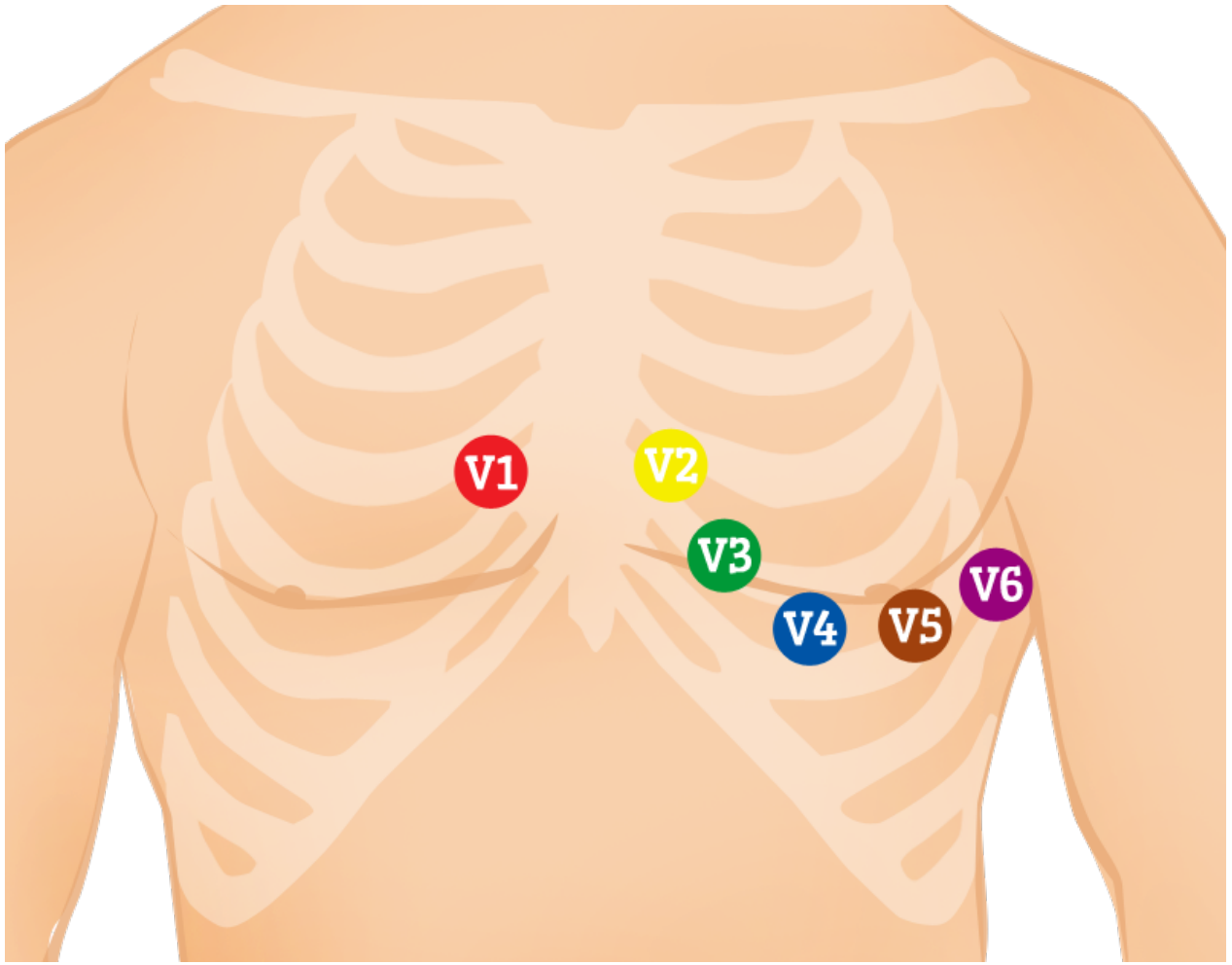
Elapsed time	Sample #	Type	Sub	Chan	Num
0:00.572	143	N	0	0	0
0:01.388	347	N	0	0	0
0:02.148	537	N	0	0	0
0:02.880	720	N	0	0	0
0:03.636	909	N	0	0	0
0:04.388	1097	N	0	0	0
0:05.184	1296	N	0	0	0
0:06.004	1501	N	0	0	0

Annotation


QRS
Complex
R
PR
Segment
ST
Segment
P
T
Q
S
PR Interval
QT Interval

Time (seconds)	MLII Lead (Volts)	V5 Lead (Volts)
Time (seconds)	MLII	V5
0	4.725	4.775
0.004	4.735	4.745
0.008	4.725	4.735
0.012	4.715	4.745
0.016	4.72	4.735
0.02	4.705	4.725
0.024	4.7	4.71

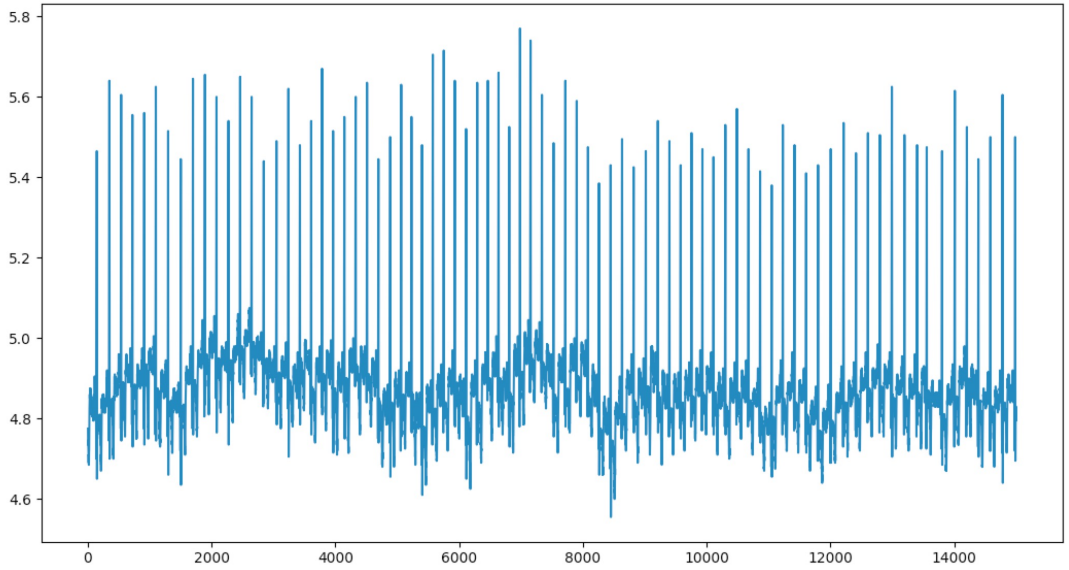




Manipulating and Analyzing Data

- A basic cardiac measurement is determining Beats Per Minute. From there, can diagnose arrythmia and heart rate variability issues.

- Would be easy if we find each “peak” in the QRS wave. But the data is noisy, how can be make the determination easier?

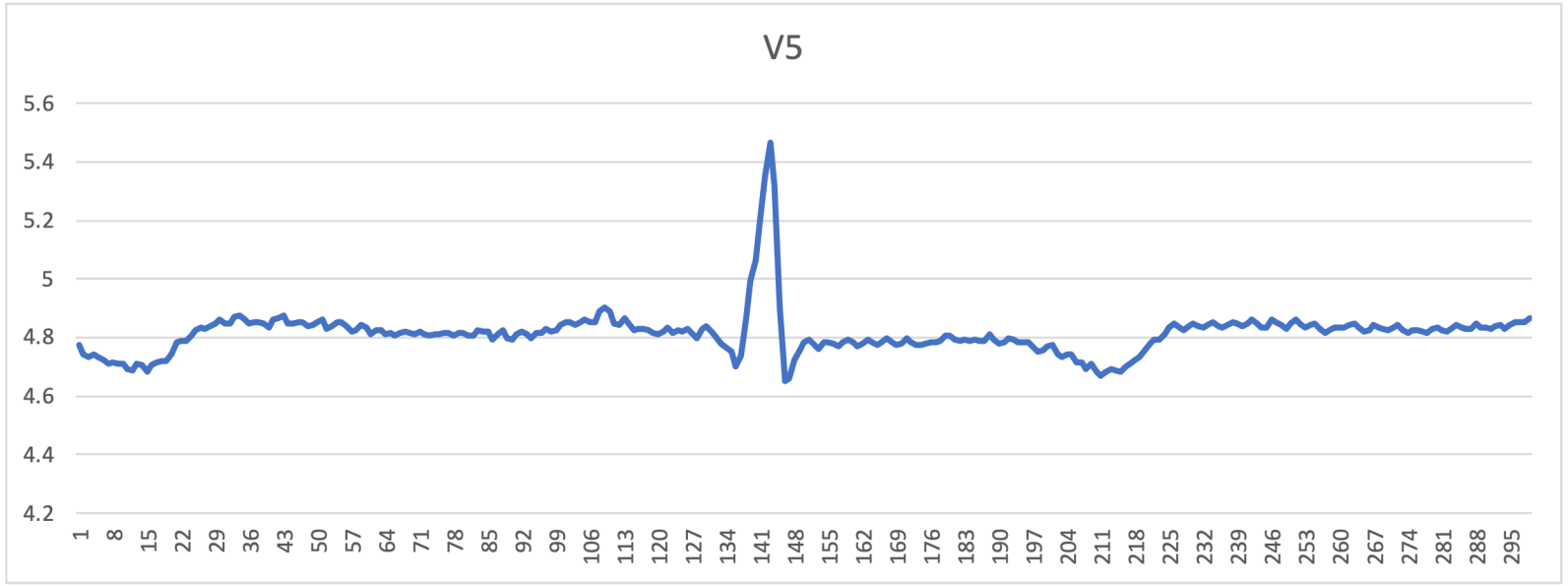


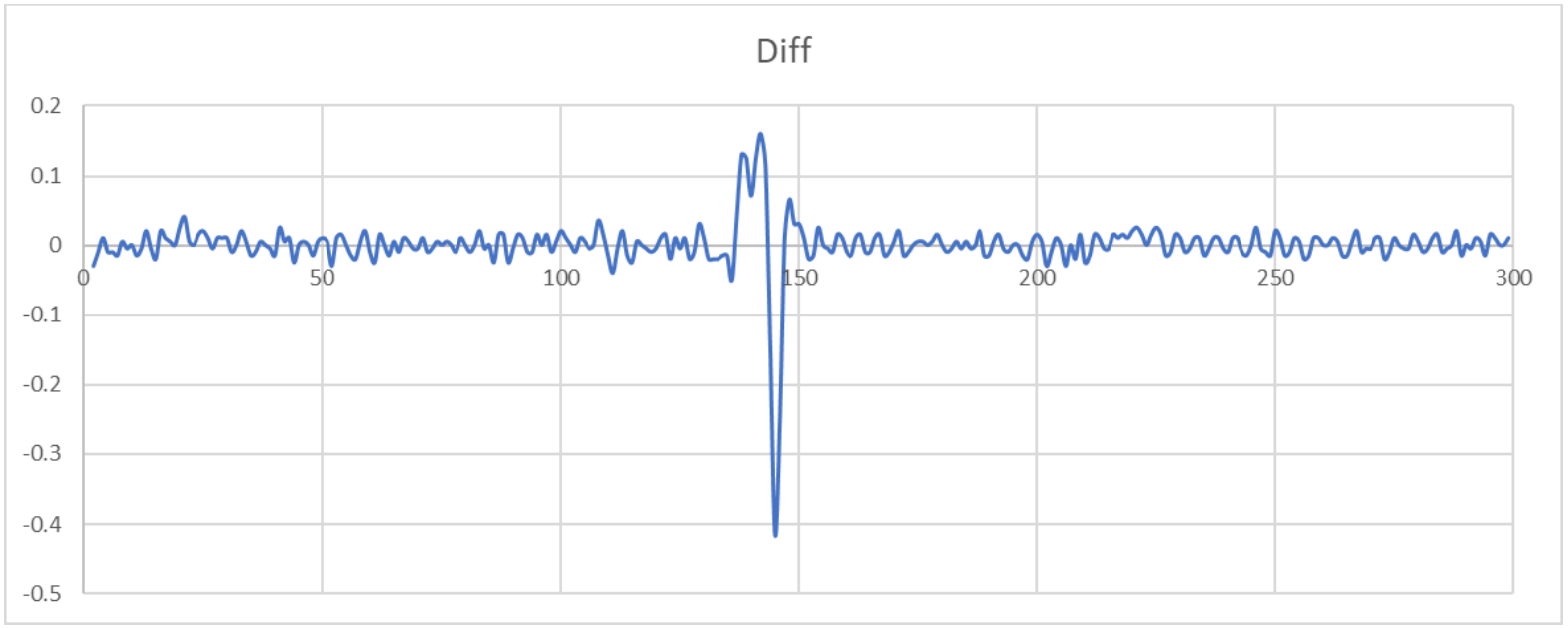
Pan-Tompkins Algorithm for QRS Detection

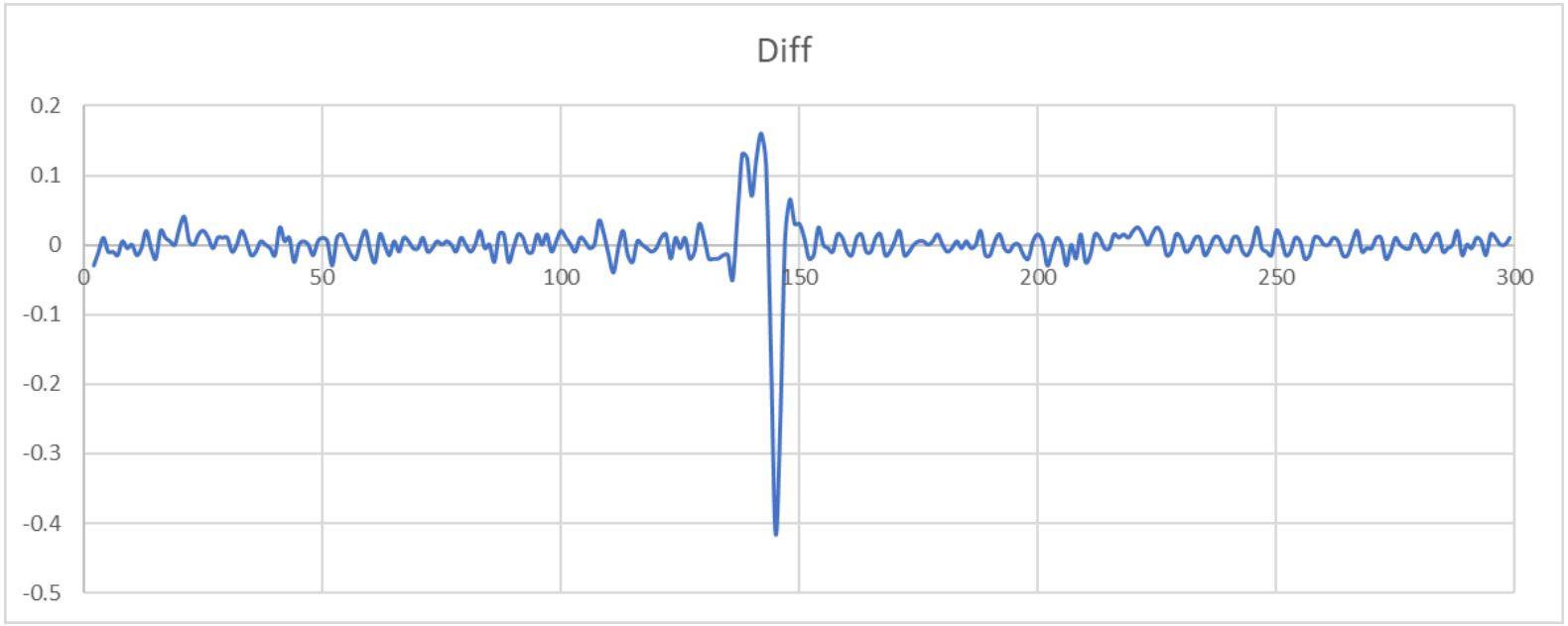
- To detect heart beats, we will follow a standard algorithm to isolate the “energy” from each heart beat. Will create a “pipeline” for processing the data.

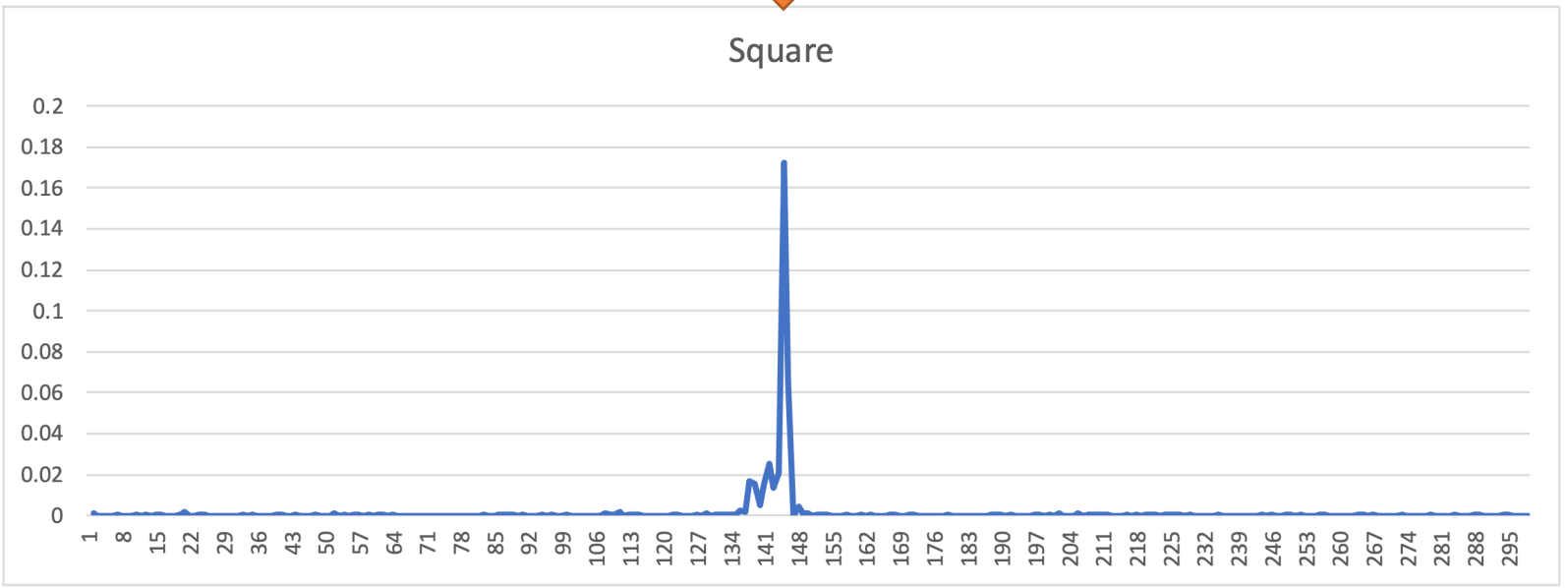
- Each signal is an array/vector of sequential values from [1-n]: $x_1, x_2, \ldots, x_n$. Each operation will result in a new array/vector.

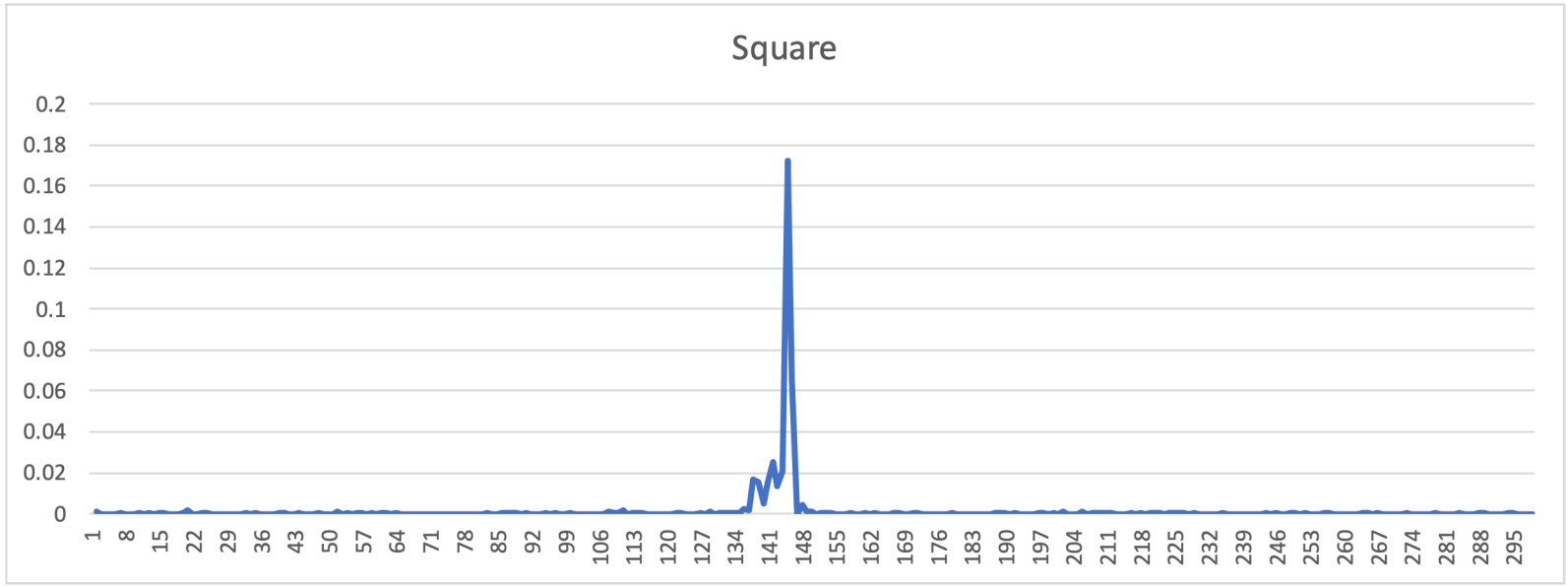
- Three/Four major operations to be performed:
  - Bandpass filter: remove noise outside of heartbeat bounds (optional)
  - Diff: take the difference between sequential points $y_1 = x_1 - x_2$
  - Square: take the result of Diff and perform ^2 operation $z_1 = y_1 * y_1$
  - Moving average: take the result from Diff and average some M values $Q = \sum_1^M z_i$

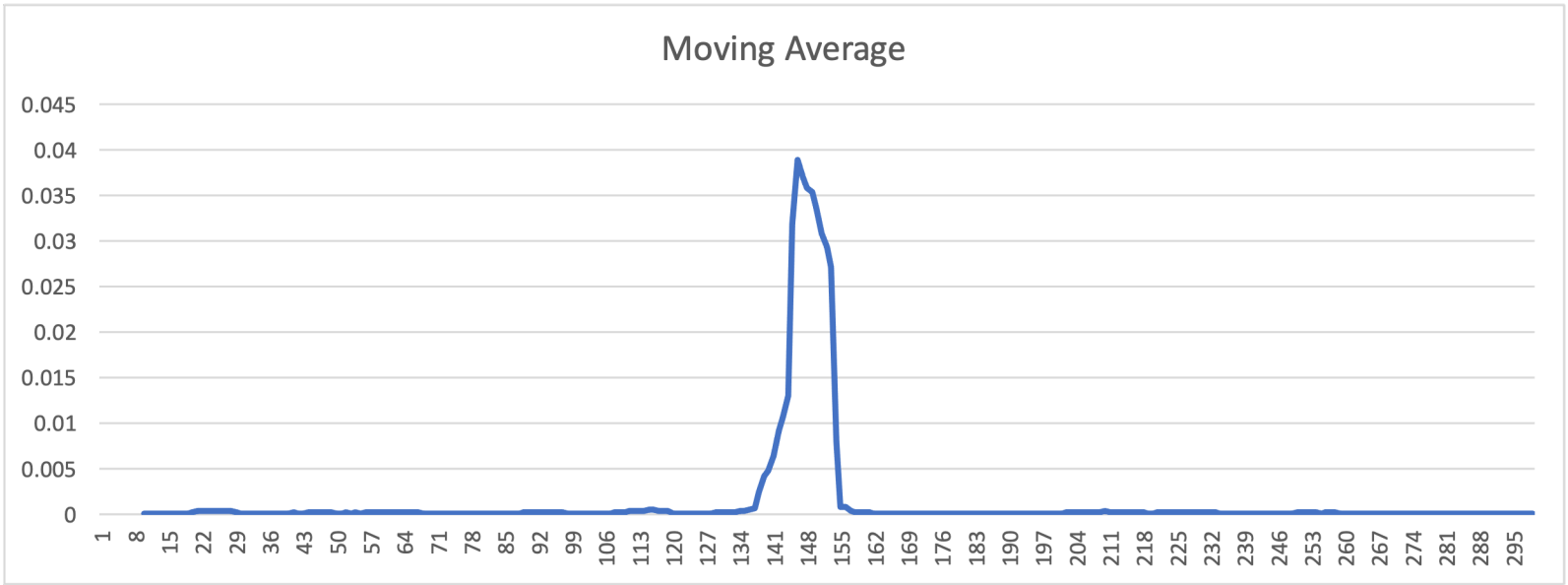

V5
5.6
5.4
5.2
5
4.8
4.6
4.4
4.2
1 8 15 22 29 36 43 50 57 64 71 78 85 92 99 106 113 120 127 134 141 148 155 162 169 176 183 190 197 204 211 218 225 232 239 246 253 260 267 274 281 288 295


Diff
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0 50 100 150 200 250 300


Diff
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0
50
100
150
200
250
300

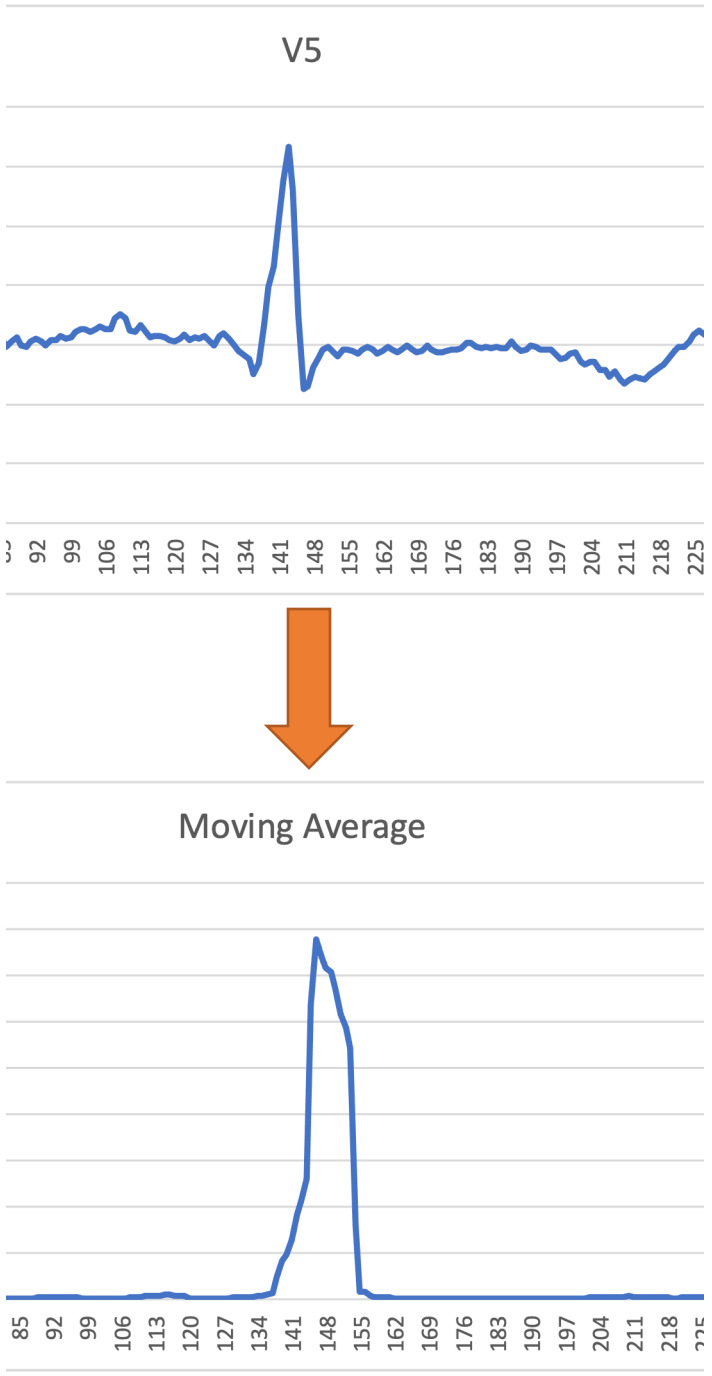

Square
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
1
8
15
22
29
36
43
50
57
64
71
78
85
92
99
106
113
120
127
134
141
148
155
162
169
176
183
190
197
204
211
218
225
232
239
246
253
260
267
274
281
288
295


Square
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
1 8 15 22 29 36 43 50 57 64 71 78 85 92 99 106 113 120 127 134 141 148 155 162 169 176 183 190 197 204 211 218 225 232 239 246 253 260 267 274 281 288 295


Moving Average
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
1 8 15 22 29 36 43 50 57 64 71 78 85 92 99 106 113 120 127 134 141 148 155 162 169 176 183 190 197 204 211 218 225 232 239 246 253 260 267 274 281 288 295

Some important items to notice

- The “peak” is now more prominent, relative to the signal “floor”, so it is easier to “see”.
- Signal is generally “0” when there is no peak nearby. This makes detection easier.
- Minor noise in signal is removed. However, signal is time delayed due to processing (average filter)



Practice in Excel then do it in Python

Signal Analysis in Excel

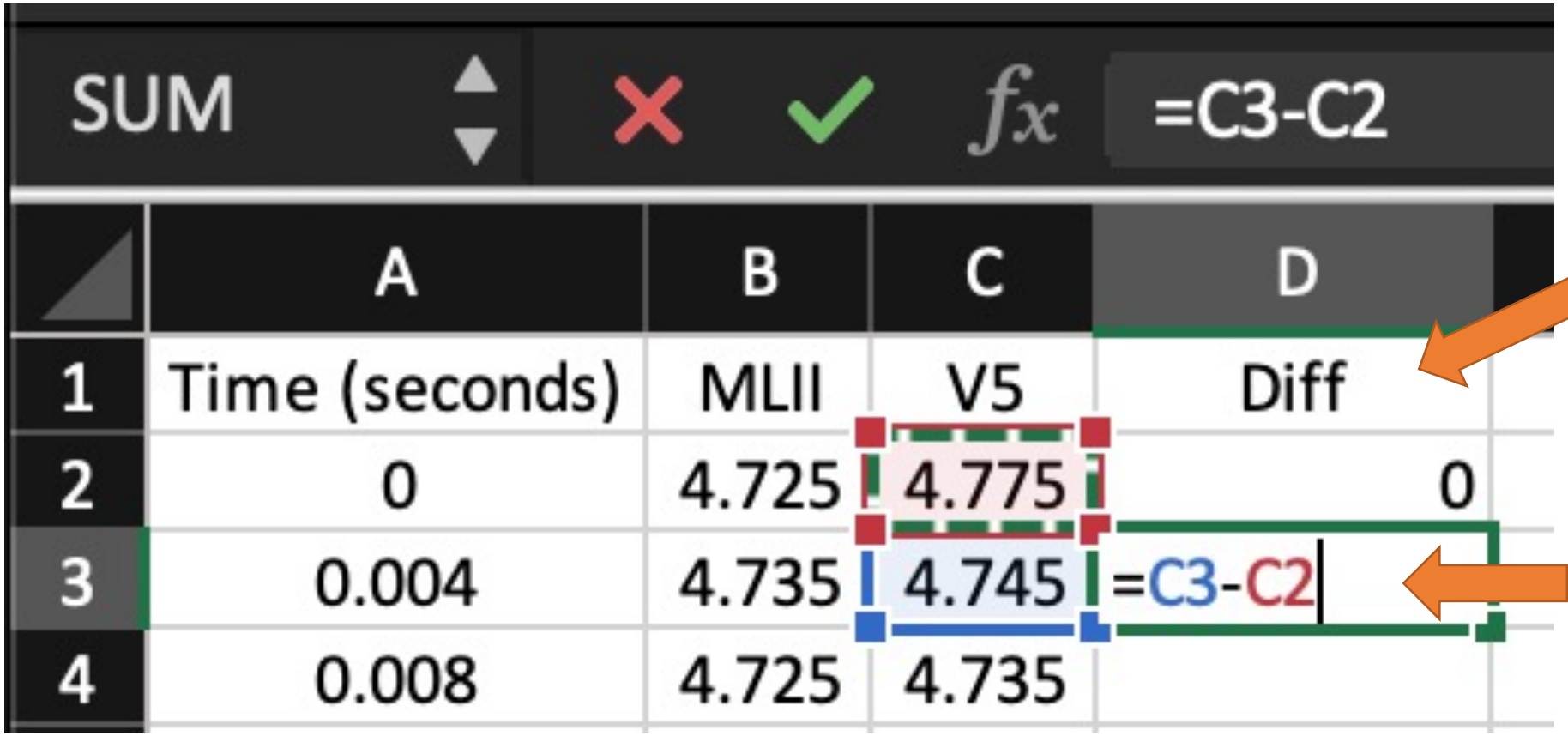
- Leave the original data alone and process each “signal” as a new column. Use the basic +/* operators in Excel and the average() function.

- Write the equation once and then copy it down the excel column. This will propagate the equation into each cell.

- Some initial values may be unknown or deleted. Can’t have a result from a 10 data point average after the second result.

Time (seconds)	MLII	V5	Diff	Square	Moving Average
0	4.725	4.775			
0.004	4.735	4.745	-0.03	0.0009	
0.008	4.725	4.735	-0.01	1E-04	
0.012	4.715	4.745	0.01	1E-04	
0.016	4.72	4.735	-0.01	1E-04	
0.02	4.705	4.725	-0.01	0.0001	
0.024	4.7	4.71	-0.015	0.000225	
0.028	4.69	4.715	0.005	2.5E-05	
0.032	4.71	4.71	-0.005	2.5E-05	
0.036	4.71	4.71	0	0	0.000175
0.04	4.695	4.695	-0.015	0.000225	1E-04
0.044	4.69	4.69	-0.005	2.5E-05	9.16667E-05
0.048	4.71	4.71	0.02	0.0004	0.000125
0.052	4.695	4.705	-0.005	2.5E-05	0.000116667
0.056	4.705	4.685	-0.02	0.0004	0.00015

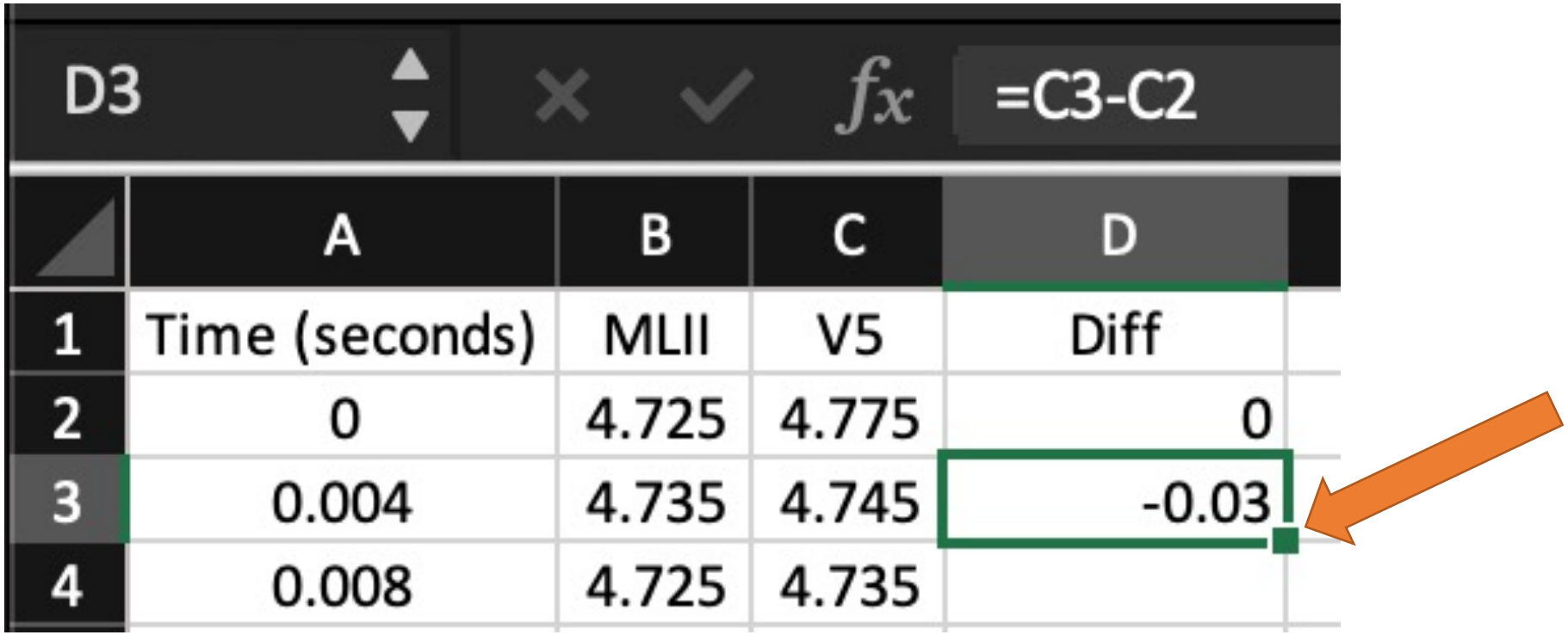
Creating the First Diff Entry



1. Give the column a name. Fill in the first value as 0

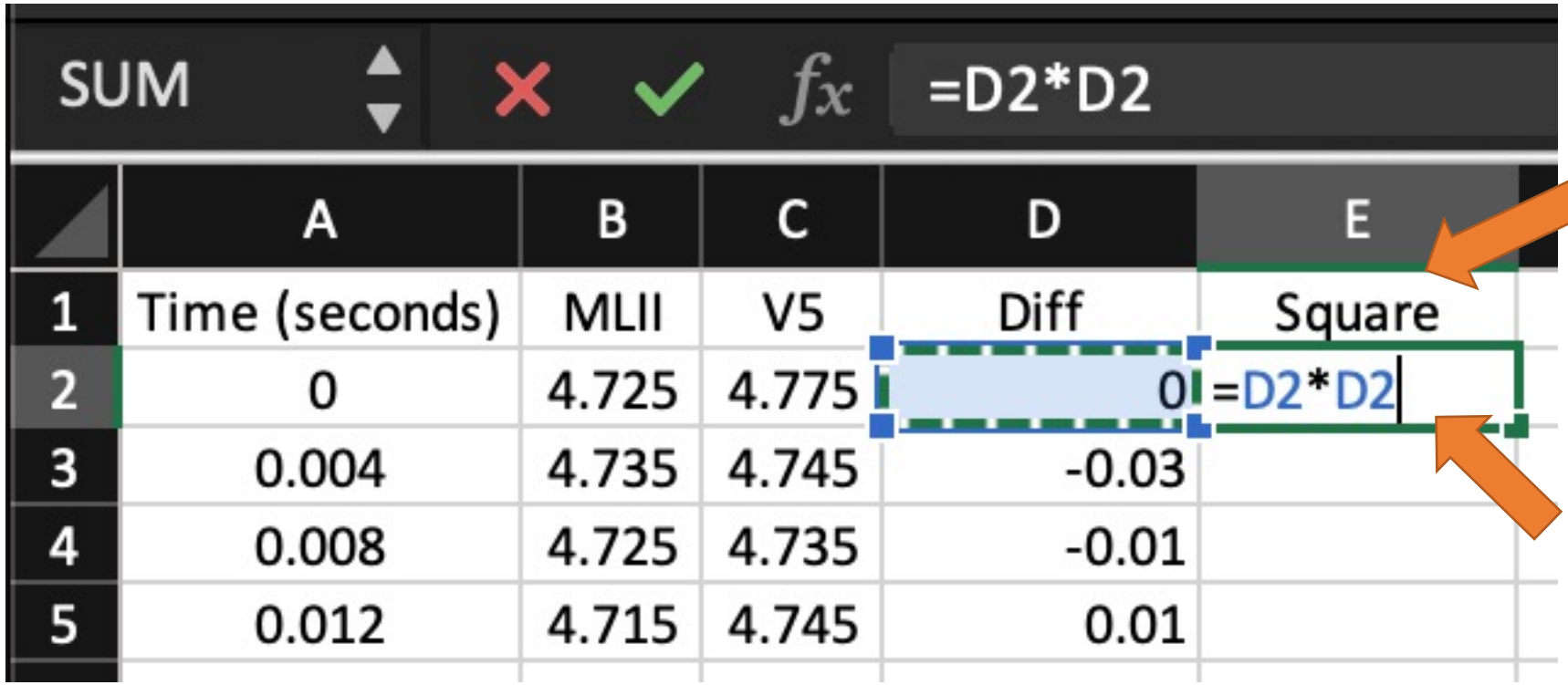
2. In the cell, type "=" and then click the cell in the V5 column. Type "-". Then click the next cell. Press enter when done.

When Done, double click the bottom right of cell to copy down the entire column.



	A	B	C	D
1	Time (seconds)	MLII	V5	Diff
2	0	4.725	4.775	0
3	0.004	4.735	4.745	-0.03
4	0.008	4.725	4.735	

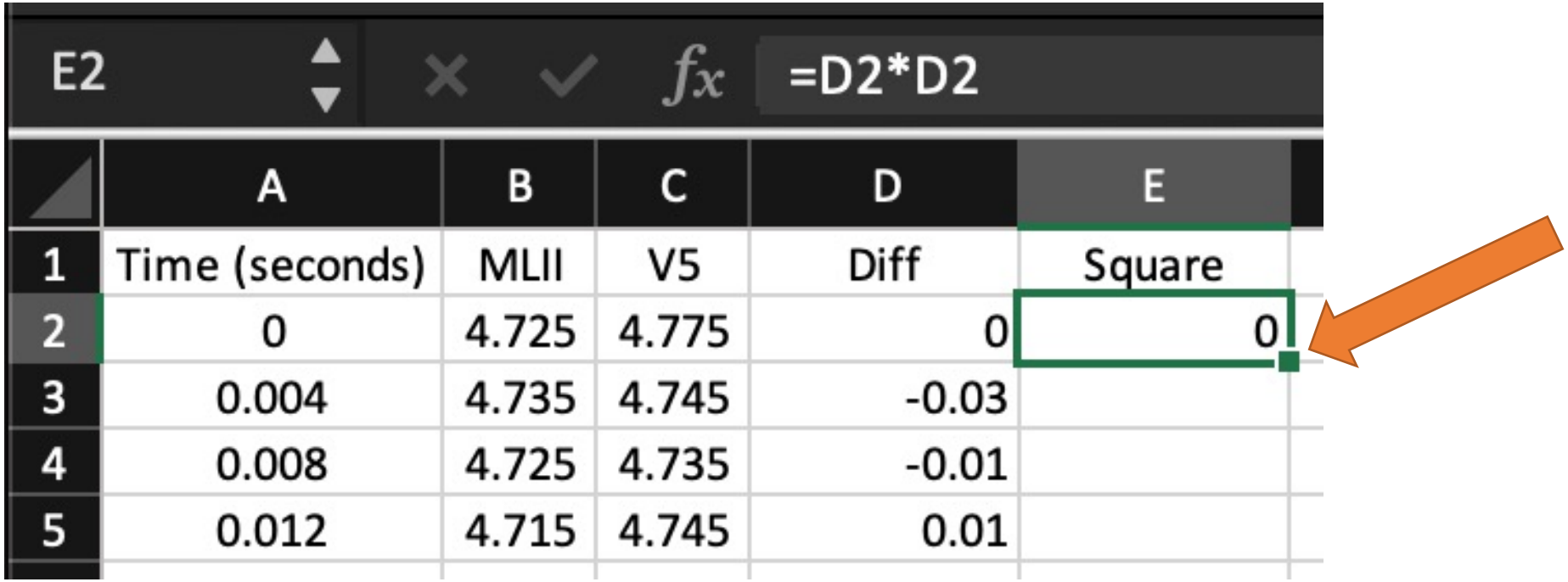
Creating the Square Column



1. Give the column a name.

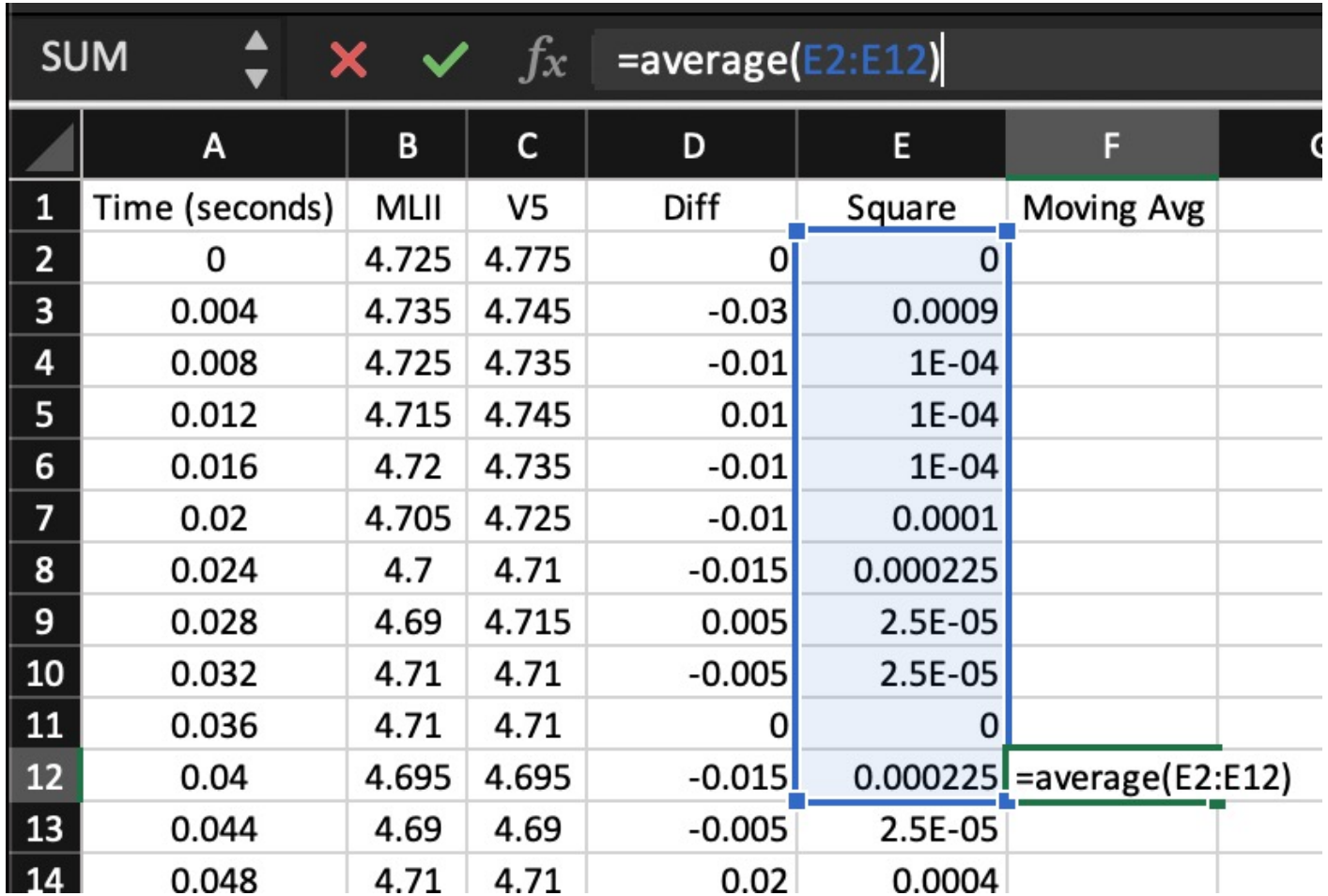
2. Type "=". Click the first entry in the Diff column. Type "*" and then re-select the entry again. Press enter

When Done, double click the bottom right of cell to copy down the entire column.



	A	B	C	D	E
1	Time (seconds)	MLII	V5	Diff	Square
2	0	4.725	4.775	0	0
3	0.004	4.735	4.745	-0.03	
4	0.008	4.725	4.735	-0.01	
5	0.012	4.715	4.745	0.01	

Make a Moving Average of 10pts. Can be adjusted.



	A	B	C	D	E	F
1	Time (seconds)	MLII	V5	Diff	Square	Moving Avg
2	0	4.725	4.775	0	0	
3	0.004	4.735	4.745	-0.03	0.0009	
4	0.008	4.725	4.735	-0.01	1E-04	
5	0.012	4.715	4.745	0.01	1E-04	
6	0.016	4.72	4.735	-0.01	1E-04	
7	0.02	4.705	4.725	-0.01	0.0001	
8	0.024	4.7	4.71	-0.015	0.000225	
9	0.028	4.69	4.715	0.005	2.5E-05	
10	0.032	4.71	4.71	-0.005	2.5E-05	
11	0.036	4.71	4.71	0	0	
12	0.04	4.695	4.695	-0.015	0.000225	=average(E2:E12)
13	0.044	4.69	4.69	-0.005	2.5E-05	
14	0.048	4.71	4.71	0.02	0.0004	

This time, when typing in the formula use the built-in average function.

Type “=average(“ then use the cursor to select a range of data (about 10 pts). Then type “)” to close the formula. Press enter.

Expand the formula down the column.

You should have something like this…

M24

	A	B	C	D	E	F
1	Time (seconds)	MLII	V5	Diff	Square	Moving Avg
2	0	4.725	4.775	0	0	
3	0.004	4.735	4.745	-0.03	0.0009	
4	0.008	4.725	4.735	-0.01	1E-04	
5	0.012	4.715	4.745	0.01	1E-04	
6	0.016	4.72	4.735	-0.01	1E-04	
7	0.02	4.705	4.725	-0.01	0.0001	
8	0.024	4.7	4.71	-0.015	0.000225	
9	0.028	4.69	4.715	0.005	2.5E-05	
10	0.032	4.71	4.71	-0.005	2.5E-05	
11	0.036	4.71	4.71	0	0	
12	0.04	4.695	4.695	-0.015	0.(25	0.00016364
13	0.044	4.69	4.69	-0.005	2.5E-05	0.00016591
14	0.048	4.71	4.71	0.02	0.0004	0.00012045
15	0.052	4.695	4.705	-0.005	2.5E-05	0.00011364
16	0.056	4.705	4.685	-0.02	0.0004	0.00014091
17	0.06	4.7	4.705	0.02	0.0004	0.00016818
18	0.064	4.71	4.715	0.01	1E-04	0.00016818
19	0.068	4.71	4.72	0.005	2.5E-05	0.00015
20	0.072	4.7	4.72	0	0	0.00014773
21	0.076	4.705	4.745	0.025	0.000625	0.00020227
22	0.08	4.72	4.785	0.04	0.0016	0.00034773
23	0.084	4.735	4.79	0.005	2.5E-05	0.00032955
24	0.088	4.74	4.79	0	0	0.00032727
25	0.092	4.73	4.805	0.015	0.000225	0.00031136

Some practical tips on detection (more on this next week)

- Identify peaks or zero crossings and then subsequent event soon after
- If corresponding event did not occur, then likely was a false alarm
- Don’t set static threshold; should float with your data

After Excel: Implementing the Algorithm in Python

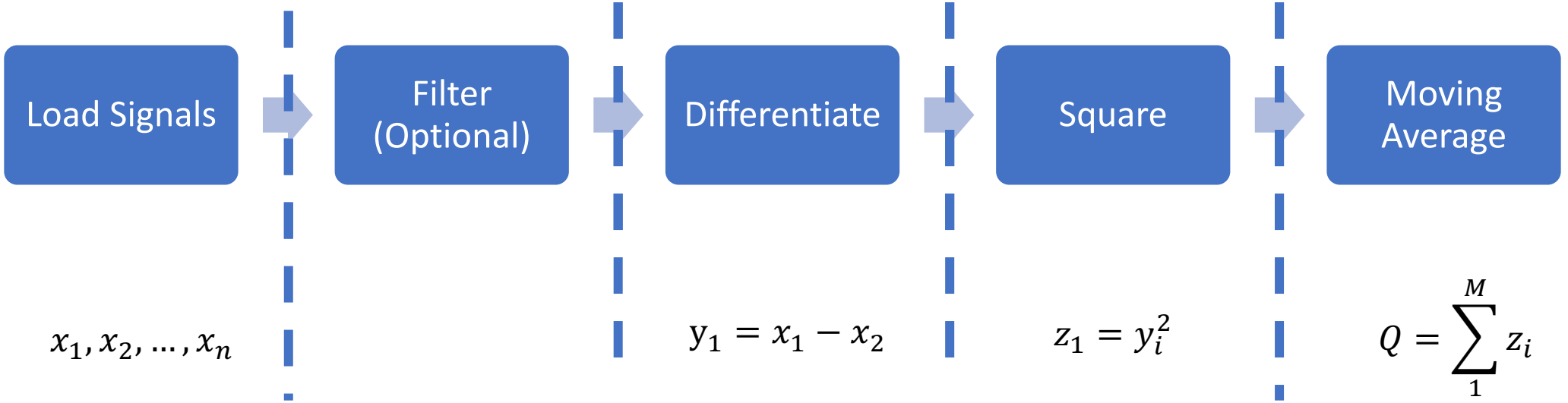
Pan-Tompkins Algorithm for QRS Detection

- To detect heart beats, we will follow a standard algorithm to isolate the “energy” from each heart beat. Will create a “pipeline” for processing the data.

- Each signal is an array/vector of sequential values from [1-n]: $x_1, x_2, \ldots, x_n$. Each operation will result in a new array/vector.

- Three/Four major operations to be performed:
  - Bandpass filter: remove noise outside of heartbeat bounds (optional)
  - Diff: take the difference between sequential points $y_1 = x_1 - x_2$
  - Square: take the result of Diff and perform ^2 operation $z_1 = y_1 * y_1$
  - Moving average: take the result from Diff and average some M values $Q = \sum_1^M z_i$

PT Algorithm Pipeline: How to Implement Each Phase



Go to Numpy documentation and look up operations

PT Algorithm Pipeline: How to Implement Each Phase

Operation:	$x_1, x_2, \ldots, x_n$	$y = x * w$	$y_1 = x_1 - x_2$	$z_1 = y_i^2$	$Q = \sum_1^M z_i$
Numpy:	np.loadtxt()	scipy.butter() np.convolve()	np.diff()	np.square()	np.convolve() with ones

Use Matplotlib to show resulting data

Matplotlib

Intro to pyplot

matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each `pyplot` function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

In **matplotlib.pyplot** various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the *axes* part of a figure and not the strict mathematical term for more than one axis).

Versatile plotting package that can have similar functionality to MATLAB. Also, can do much more ☺

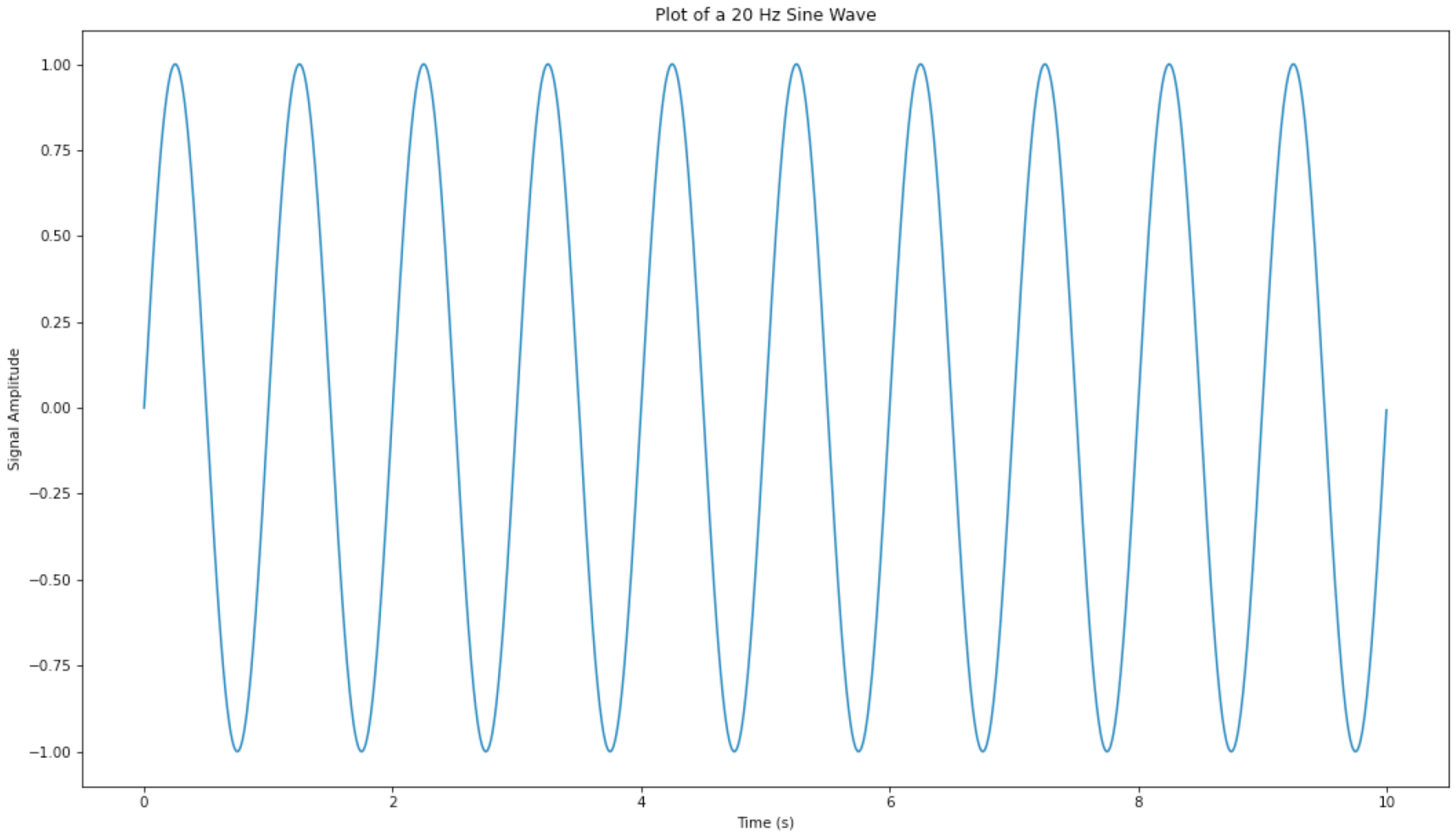
```
import numpy as np
import matplotlib.pyplot as plt

# sample a signal at a give rate (Hz)
sample_rate = 1000

# determine the sample period
period = 1/sample_rate

# over 10s, sample the signal
time = np.arange(0,10,period)

# generate a sine wave of frequency (Hz)
freq = 20
sine = np.sin(2*np.pi*time)
```

Use numpy to create a since wave based upon a time vector and frequency

```
# use matplotlib to plot sine wave
plt.plot(time,sine)

# label the x and y axis
plt.xlabel("Time (s)")
plt.ylabel("Signal Amplitude")

# provide a nice title
plt.title("Plot of a "+str(freq)+" Hz Sine Wave")

# now show the plot
plt.show()
```

Now plot the wave with matplotlib

Figure 1


Plot of a 20 Hz Sine Wave
Signal Amplitude
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
Time (s)

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

```
import numpy as np
import matplotlib.pyplot as plt
```

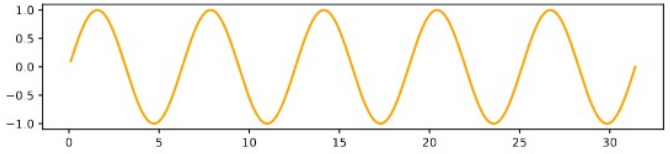
2 Prepare

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
fig.show()
```

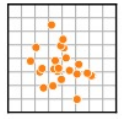
4 Observe



Choose

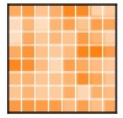
Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

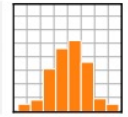
```
Z = np.random.uniform(0, 1, (8,8))

ax.imshow(Z)
```



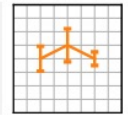
```
Z = np.random.uniform(0, 1, (8,8))

ax.contourf(Z)
```

```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```

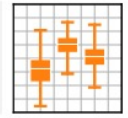
```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```



```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```



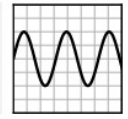
```
Z = np.random.normal(0, 1, (100,3))

ax.boxplot(Z)
```



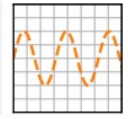
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

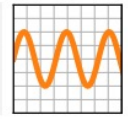
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



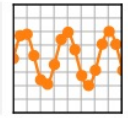
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



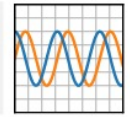
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



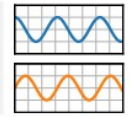
Organize

You can plot several data on the the same figure, but you can also split a figure in several subplots (named *Axes*):

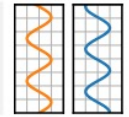
```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```



```
fig, (ax1, ax2) = plt.subplots((2,1))
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

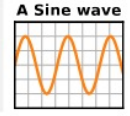


```
fig, (ax1, ax2) = plt.subplots((1,2))
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

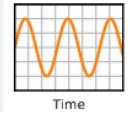


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```




```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

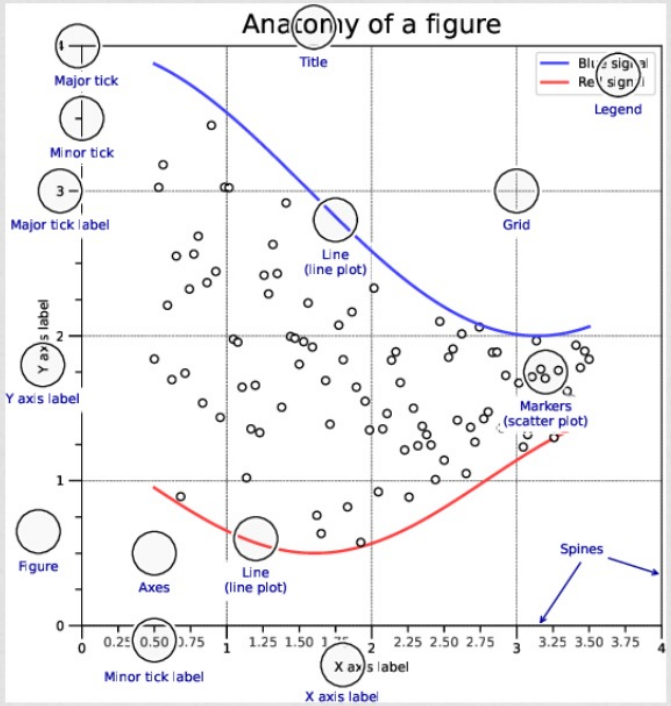
Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

Matplotlib for intermediate users

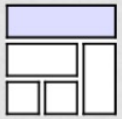
A matplotlib figure is composed of a hierarchy of elements that forms the actual figure. Each element can be modified.



Figure, axes & spines

```
fig, axs = plt.subplots(3,3)
axs[0,0].set_facecolor("#ddddff")
axs[2,2].set_facecolor("#ffffdd")
```

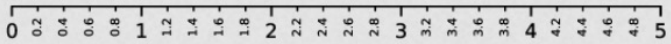
```
gs = fig.add_gridspec(3, 3)
ax = fig.add_subplot(gs[0, :])
ax.set_facecolor("#ddddff")
```



```
fig, ax = plt.subplots()
ax.spines["top"].set_color("None")
ax.spines["right"].set_color("None")
```

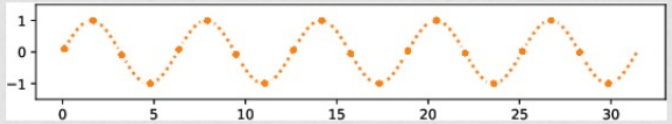
Ticks & labels

```
from mpl.ticker import MultipleLocator as ML
from mpl.ticker import ScalarFormatter as SF
ax.xaxis.set_minor_locator(ML(0.2))
ax.xaxis.set_minor_formatter(SF())
ax.tick_params(axis='x',which='minor',rotation=90)
```



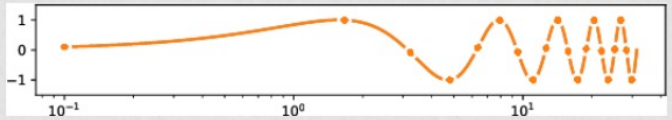
Lines & markers

```
X = np.linspace(0.1, 10*np.pi, 1000)
Y = np.sin(X)
ax.plot(X, Y, "C1o:", markevery=25, mec="1.0")
```



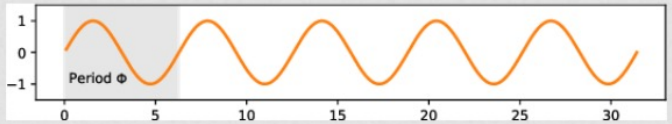
Scales & projections

```
fig, ax = plt.subplots()
ax.set_xscale("log")
ax.plot(X, Y, "C1o-", markevery=25, mec="1.0")
```



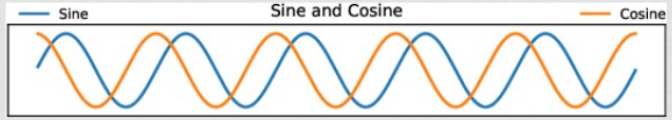
Text & ornaments

```
ax.fill_betweenx([-1,1],[0],[2*np.pi])
ax.text(0, -1, r" Period $\Phi$")
```



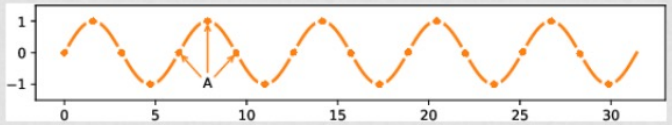
Legend

```
ax.plot(X, np.sin(X), "C0", label="Sine")
ax.plot(X, np.cos(X), "C1", label="Cosine")
ax.legend(bbox_to_anchor=(0,1,1,.1),ncol=2,
          mode="expand",  loc="lower left")
```



Annotation

```
ax.annotate("A", (X[250],Y[250]),(X[250],-1),
  ha="center", va="center",arrowprops =
  {"arrowstyle" : "->", "color": "C1"})
```



Colors

Any color can be used, but Matplotlib offers sets of colors:

Size & DPI

Consider a square figure to be included in a two-columns A4 paper with 2cm margins on each side and a column separation of 1cm. The width of a figure is (21 - 2*2 - 1)/2 = 8cm. One inch being 2.54cm, figure size should be 3.15×3.15 in.

```
fig = plt.figure(figsize=(3.15,3.15), dpi=50)
plt.savefig("figure.pdf", dpi=600)
```

Weekly Assignments

- **Be EXCELlent**: Use Excel to import a CSV file from the ekg_data set. Perform the Pan Tompkins processing steps to transform the signal into a final moving average plot. Upload an image of the final plot for the first 10 seconds.

- **PythonBeats**. Using *numpy* and *matplotlib*, load a dataset from ekg_data. Plot the first 10 seconds of a signal. Apply appropriate labels and title to figure. Upload code and figure.

- **PanTompkins Algorithm**. Implement the entire Pan Tompkins algorithm in Python. Generate a plot for each stage of the pipeline showing only the first 10 seconds of data. Pipeline should process all data but plots only for limited set.