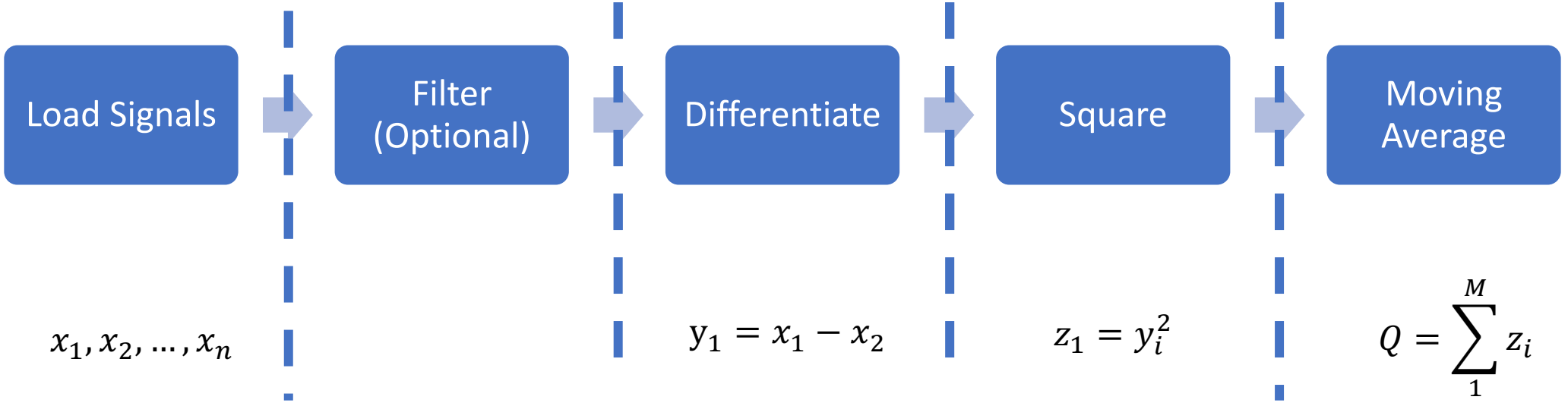
ENGR 298: Engineering Analysis and Decision Making – Signal Detection

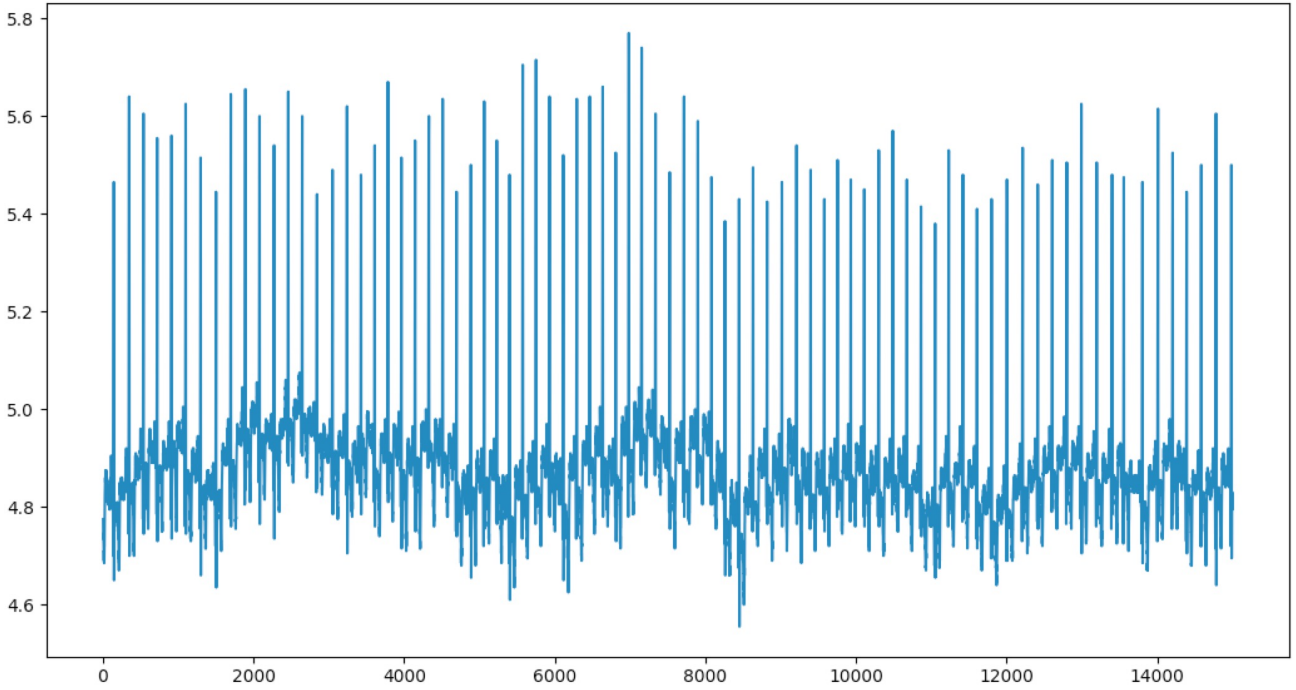
How to find the information once you've filtered it…

Dr. Jason Forsyth
Department of Engineering
James Madison University

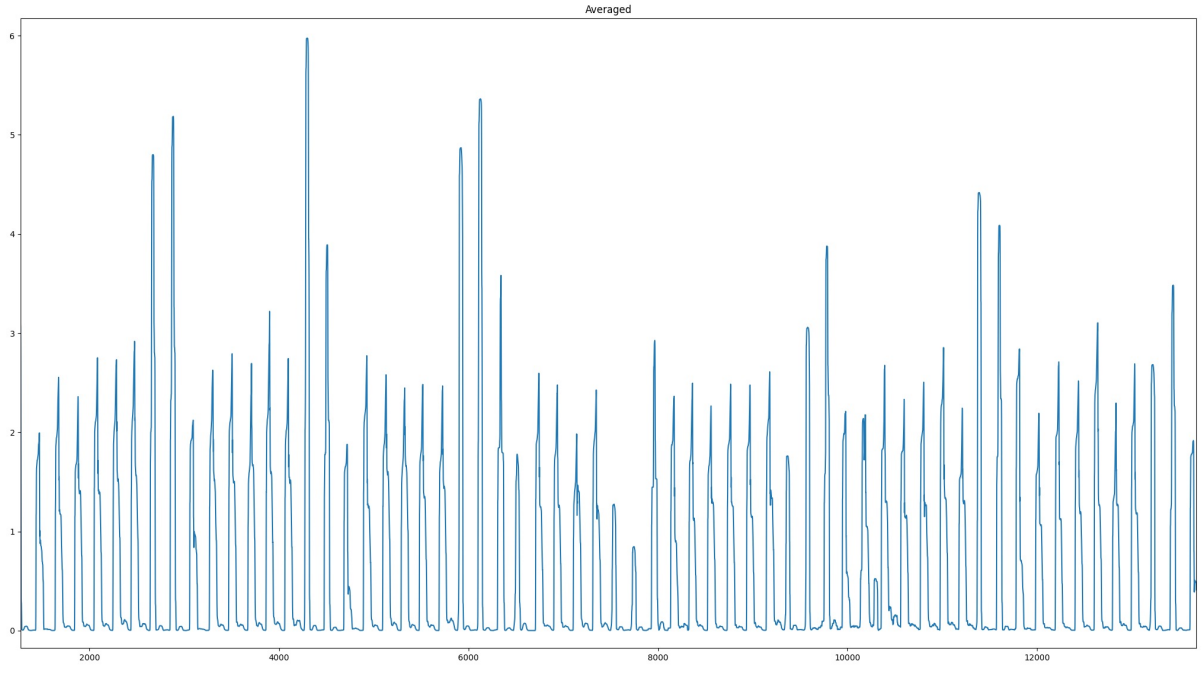
PT Algorithm Pipeline: How to Implement Each Phase



Last week we examined how to clean up data

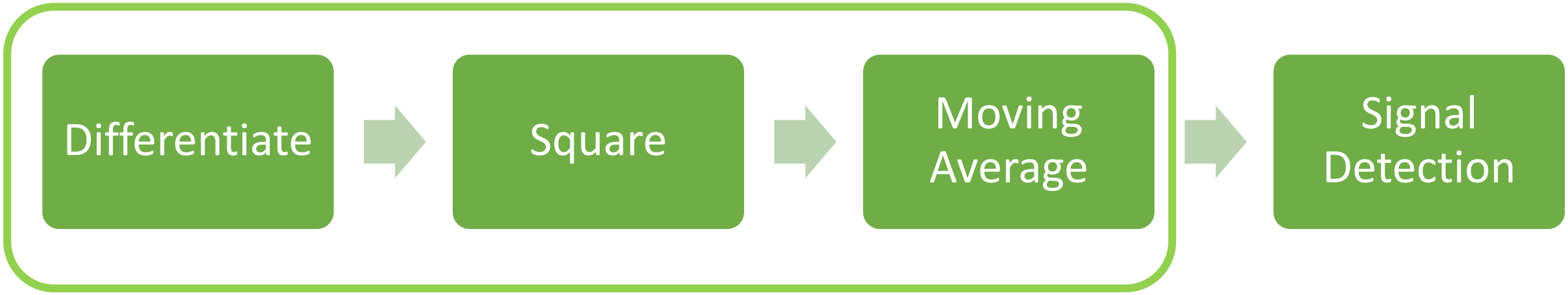


Raw ECG Data (before processing)



Post-processing and Filtered

Signal processing is only the first step in the pipeline



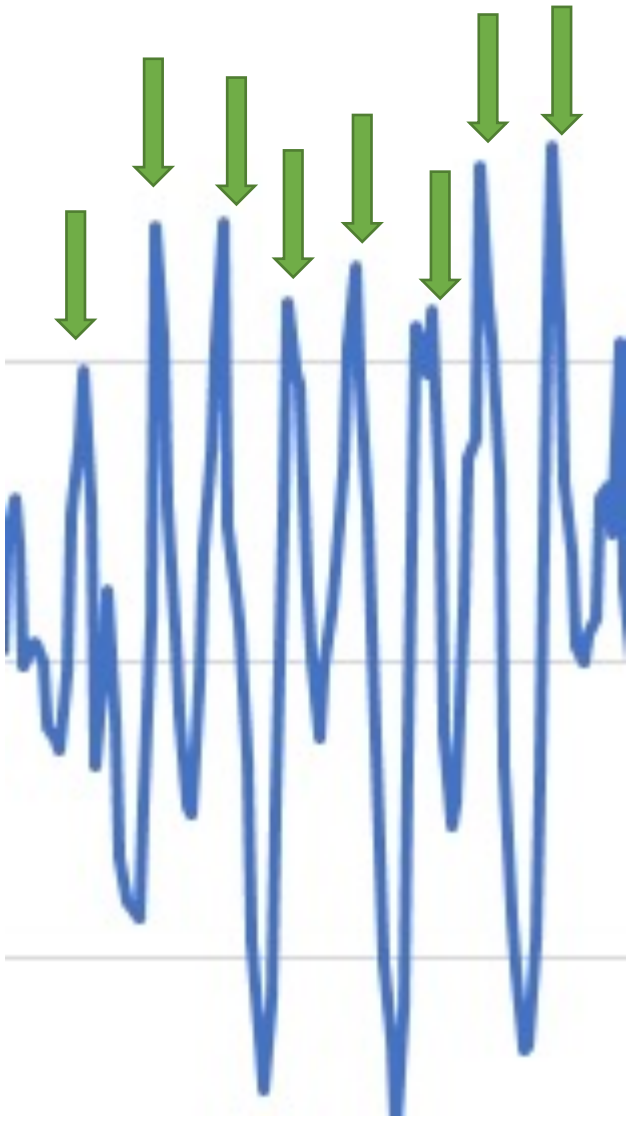
How detection works in practice.
Data can be noisy.

Looking for a heart 

- EKG data can be noisy due to motion or electrical interference
- Heart beats are very periodic but can vary based upon patient health, activity level…etc
- PT processing pipeline isolated heart beat “signal” from other noise producing a series of “peaks”. Each peak should be a beat.
- Must develop a method to count/enumerate these peaks. Will examine simple thresholding and advanced “peak finder” methods.

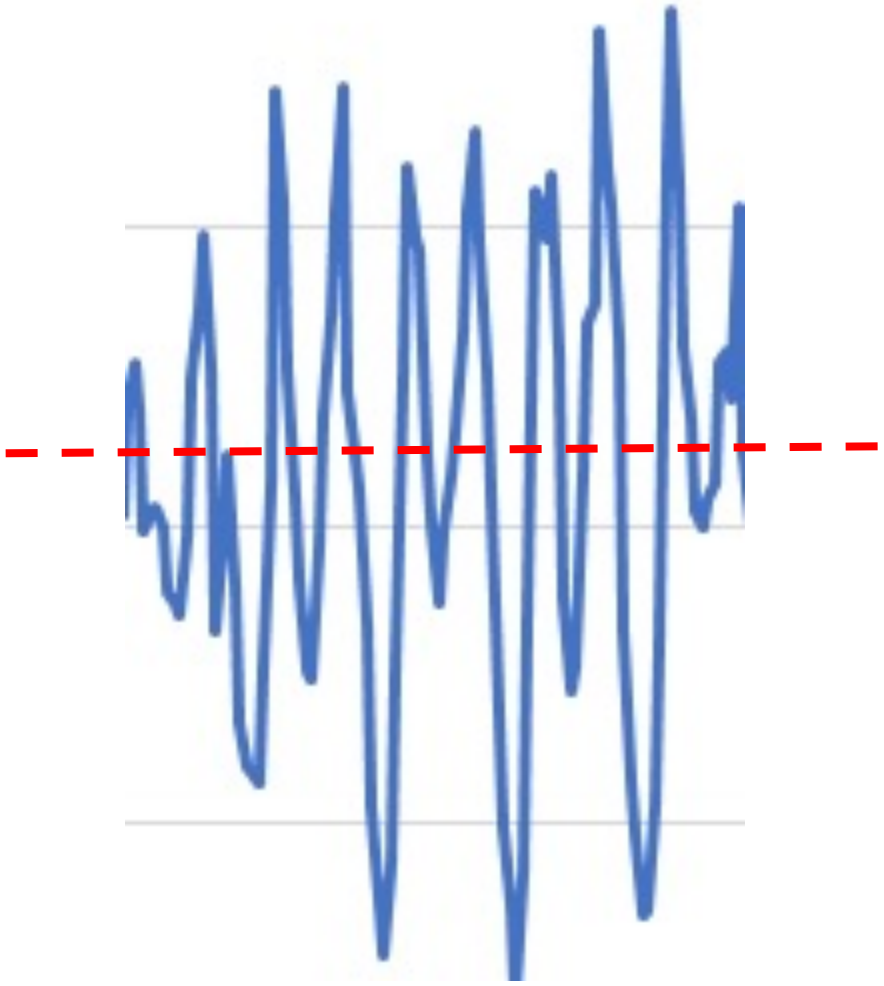
Counting Steps via Peaks

- How many maxima are detected?
- Would want to pair maxima with minima to ensure full stride
- Peaks can only occur so quickly
  - There's a maximum walking pace
  - Help reduce/remove noise

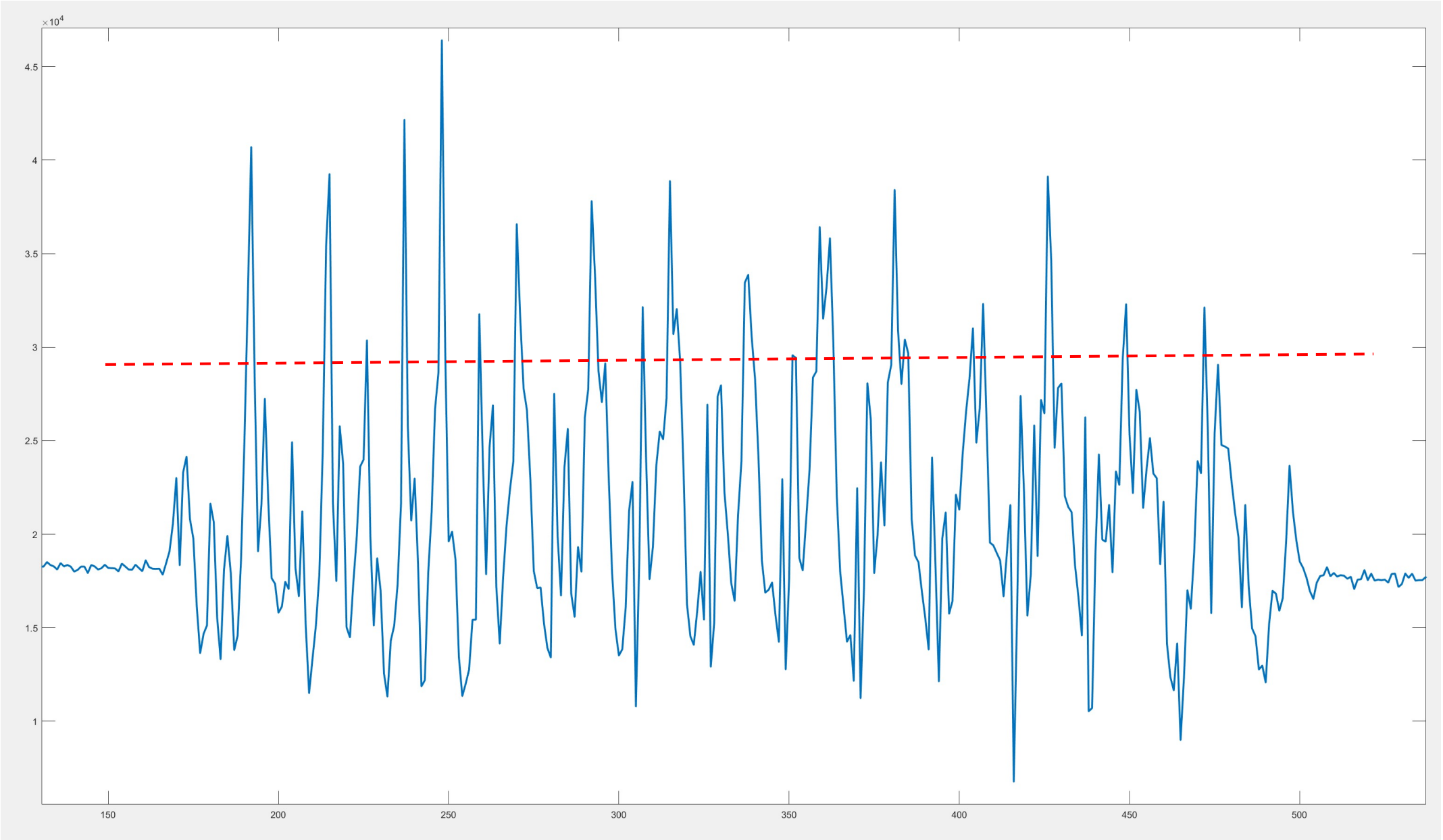


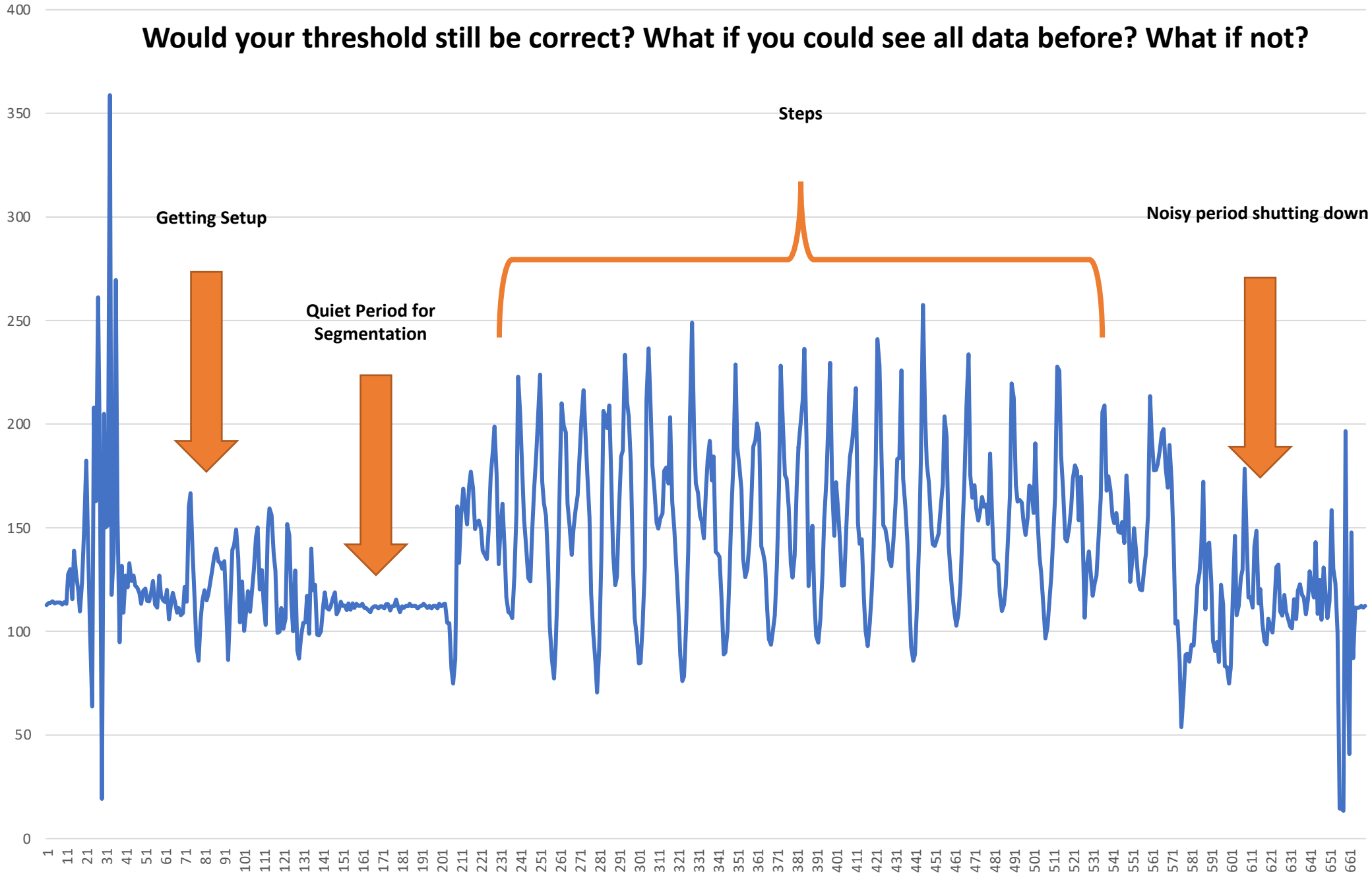
Counting Steps via Zero Crossing

- How many times does the signal cross a “zero” threshold?
- Accel will be at 1g when idle
- Need to detect rising and falling across threshold
  - Should occur relatively close in time

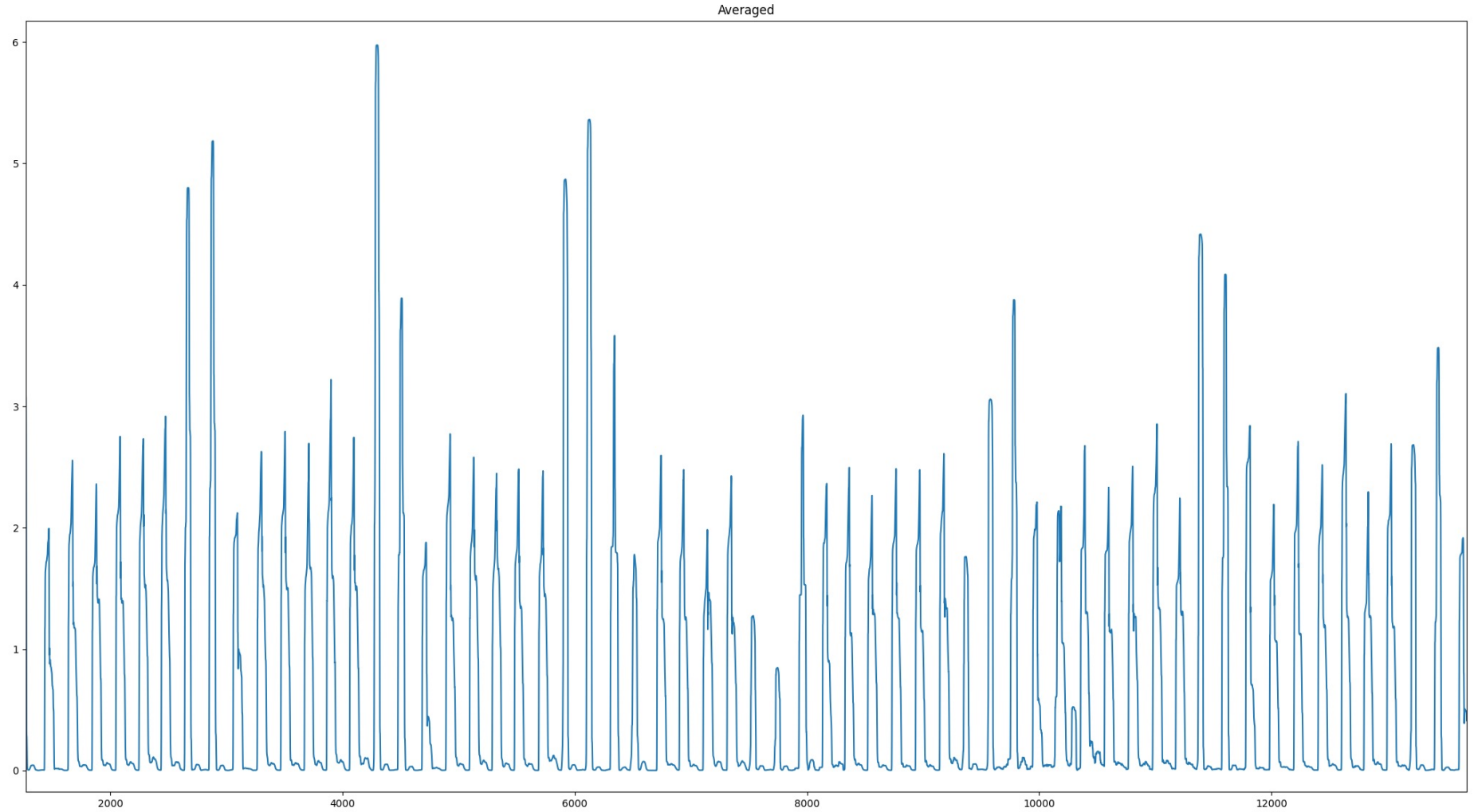


How many steps are in this accelerometer data? What threshold value would you pick to find them all?




Would your threshold still be correct? What if you could see all data before? What if not?
Steps
Getting Setup
Quiet Period for Segmentation
Noisy period shutting down
400
350
300
250
200
150
100
50
0
1
11
21
31
41
51
61
71
81
91
101
111
121
131
141
151
161
171
181
191
201
211
221
231
241
251
261
271
281
291
301
311
321
331
341
351
361
371
381
391
401
411
421
431
441
451
461
471
481
491
501
511
521
531
541
551
561
571
581
591
601
611
621
631
641
651
661

Will a single static threshold work?



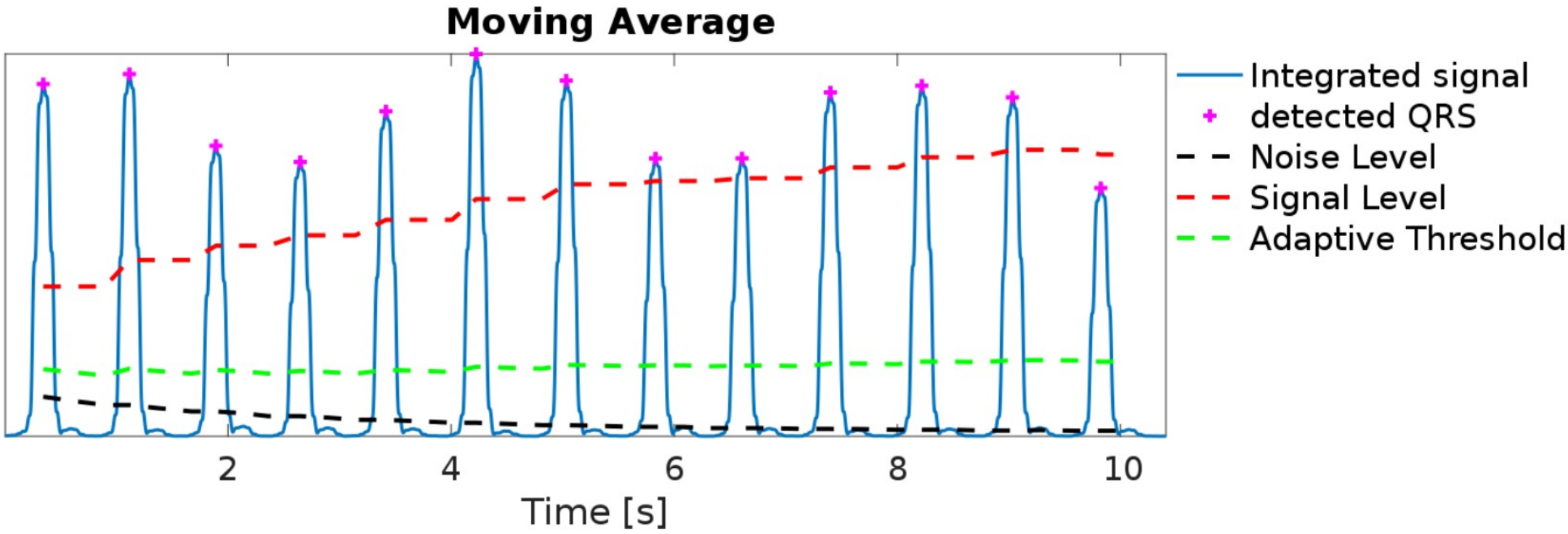
Signal Detection Approaches

- The methods presented here are two simple solutions. Many more approaches exist. Be inventive.

- Method 1: Simple Threshold Detection with Timeout
  - Insight: Once the signal passes a certain threshold it is likely a heartbeat; These events can only occur so often so we must “timeout” before accepting future values
  - Modification: Adapt thresholds based upon signal and noise floors

- Method 2: Peak Finding via Prominence and Width
  - Insight: Directly search signal for peaks of certain height and width. More computational “expensive” but appropriate for Python-like implementations.

PT Algorithm Utilizes Adaptive Thresholds

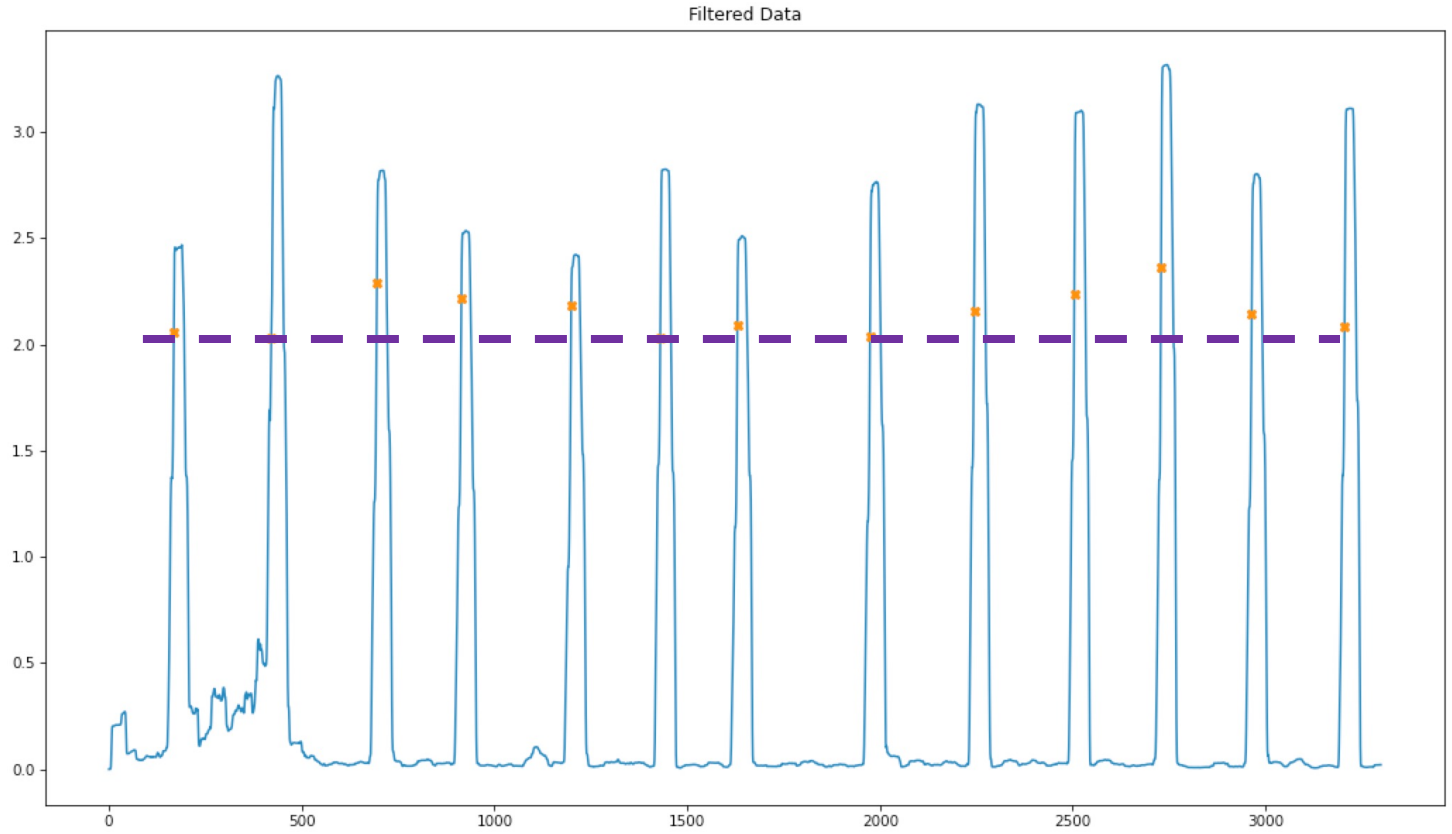


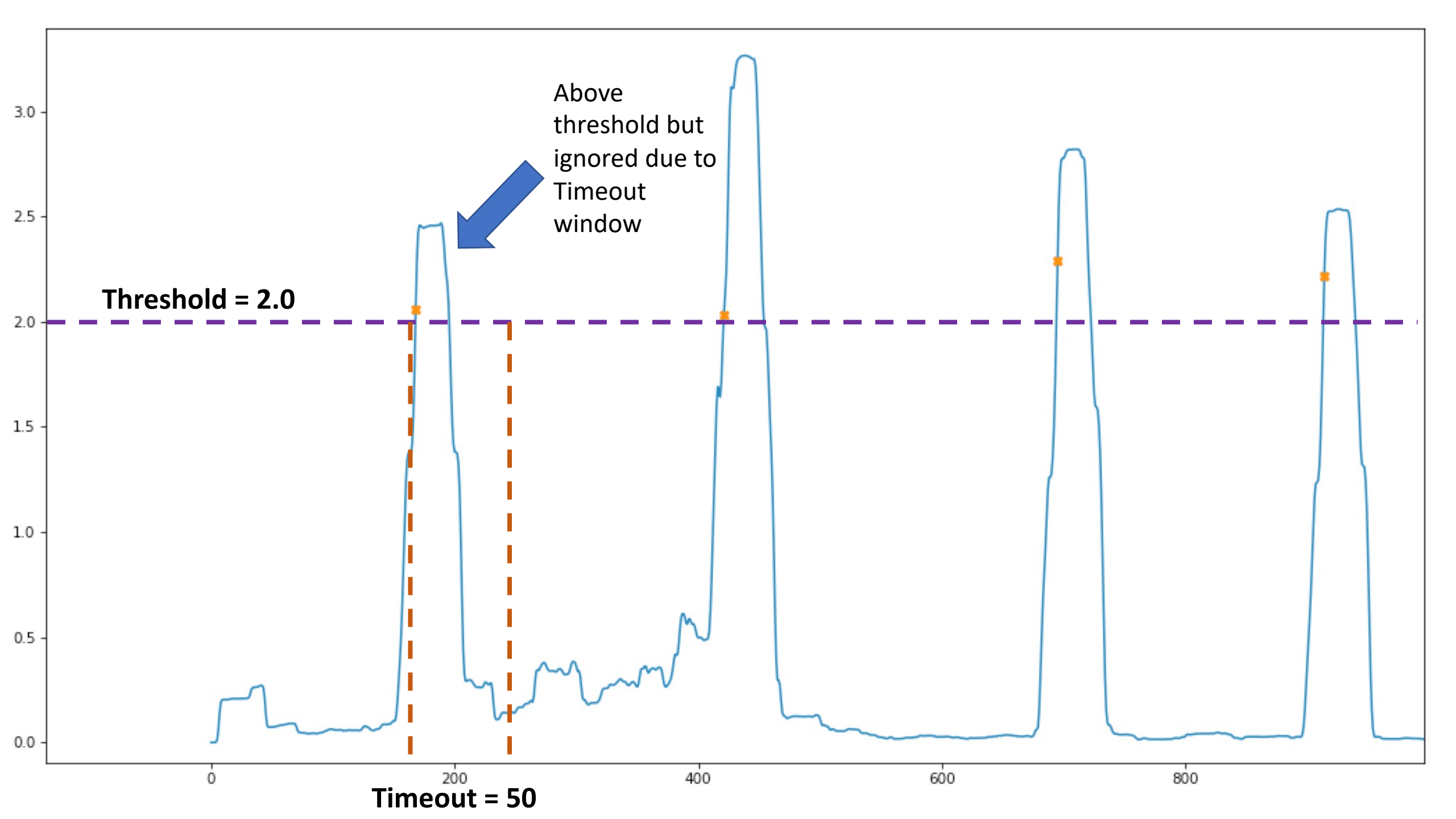
https://commons.wikimedia.org/wiki/File:ECGrita.svg#/media/File:ECGrita.svg

Simple Threshold Detection with Time Out

- Select a **threshold** that when the signal crosses (positive) it is considered a heart beat

- Ignore all other points until a **timeout** has based so that additional values in the peak are not accidently counted




Above threshold but ignored due to Timeout window
Threshold = 2.0
Timeout = 50
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800

A simple Python implementation for thresholding and timeout – Part 1

```
# set a detection threshold
detection_threshold = 2.0

# set a heart beat time out
detection_time_out = 50

# track the last time we found a beat
last_detected_index = -1

# keep not of where we are in the data
current_index = 0

# store indices of all found beats
time_detected = list()
```

A simple Python implementation for thresholding and timeout – Part 2

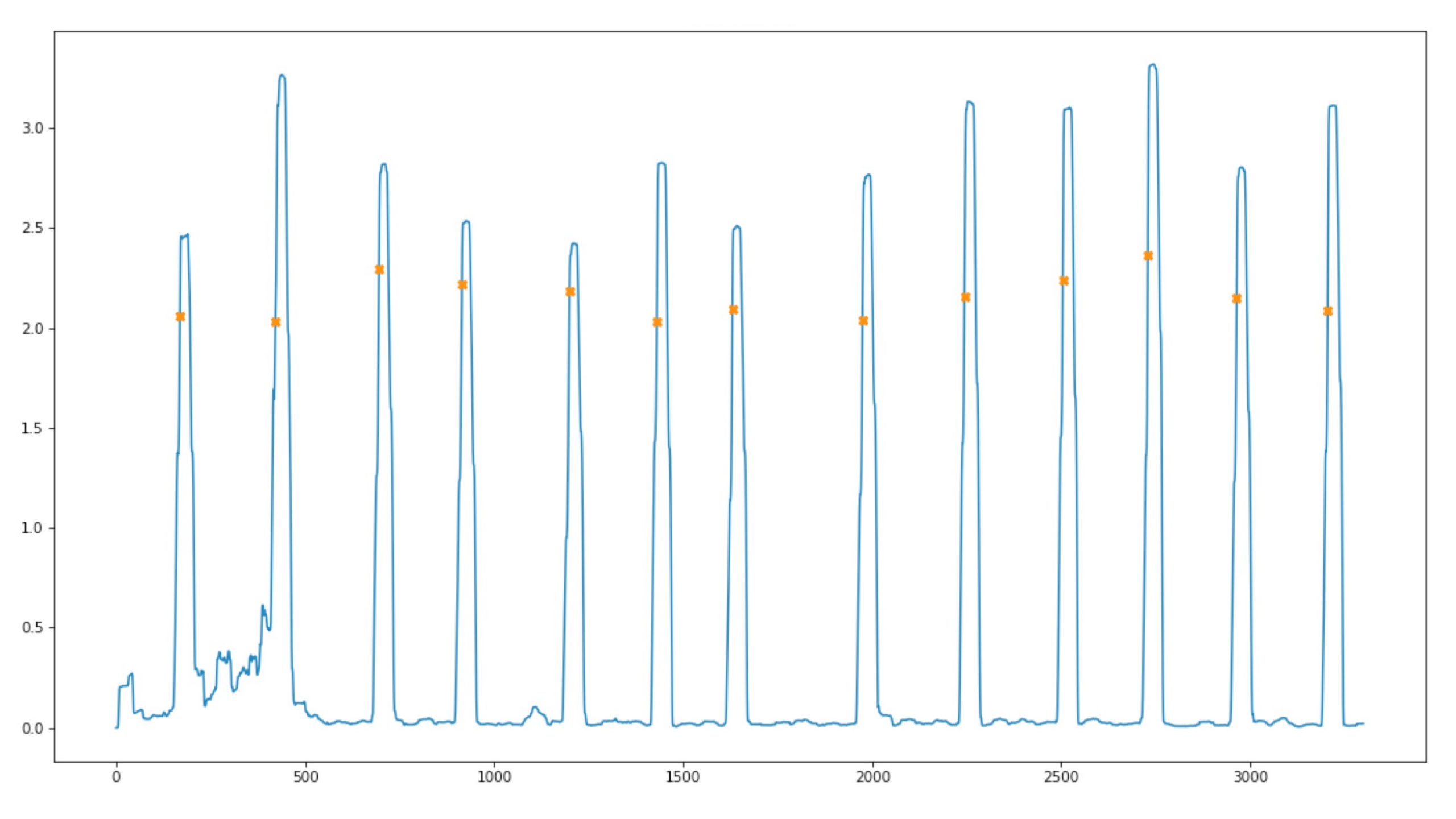
```
# loop through signal finding beats
for value in signal:

    # if current value is greater than threshold AND it has been
    # long enough since our previous heart beat detection
    if value > detection_threshold \
            and abs(current_index - last_detected_index) > detection_time_out:

        # update the last time we found a beat
        last_detected_index = current_index

        # add this index to the list of our beats
        beats_detected.append(current_index)

    current_index += 1
```


3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000

Some notes on simple threshold

- Very easy to implement (just a loop and if statement)
- Most detections occur “early” from the peak but this may be OK since the processing pipeline introduces delays (from moving average and filters)
- Work well for periodic and stable signal but not resistant to signal drift or rise in noise (without adaptive approach).
- Timeout only works when periodicity is stable; may cause mis-detect if heart rate changes.

Peak Finding in SciPy

- Peaks are local maximums in the signal.
- Will use find_peaks in SciPy package to locate in signal.

Let $A = a_0, a_1, \ldots, a_{n-1}$ be an array of integers of length n

0	1	2	3	4	5	6	7	8	9
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9

Definition: (Peak)
Integer a_i is a *peak* if adjacent integers are not larger than a_i

Example:

4	3	9	10	14	8	7	2	2	2

RELEASE 1.9.0.DEV0+1590.9E8DF2E

SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. With SciPy, an interactive Python session becomes a data-processing and system-prototyping environment rivaling systems, such as MATLAB, IDL, Octave, R-Lab, and SciLab.

The additional benefit of basing SciPy on Python is that this also makes a powerful programming language available for use in developing sophisticated programs and specialized applications. Scientific applications using SciPy benefit from the development of additional modules in numerous niches of the software landscape by developers across the world. Everything from parallel programming to web and data-base subroutines and classes have been made available to the Python programmer. All of this power is available in addition to the mathematical libraries in SciPy.

SciPy Organization

SciPy is organized into subpackages covering different scientific computing domains. These are summarized in the following table:

Subpackage	Description
`cluster`	Clustering algorithms
`constants`	Physical and mathematical constants
`fftpack`	Fast Fourier Transform routines
`integrate`	Integration and ordinary differential equation solvers
`interpolate`	Interpolation and smoothing splines
`io`	Input and Output
`linalg`	Linear algebra
`ndimage`	N-dimensional image processing

scipy.signal.find_peaks

Many parameters

```
scipy.signal.find_peaks(x, height=None, threshold=None, distance=None,
prominence=None, width=None, wlen=None, rel_height=0.5,
plateau_size=None)
```

[source]

Find peaks inside a signal based on peak properties.

This function takes a 1-D array and finds all local maxima by simple comparison of neighboring values. Optionally, a subset of these peaks can be selected by specifying conditions for a peak's properties.

Returns:

peaks : *ndarray*

Indices of peaks in *x* that satisfy all given conditions.

Returns two objects as tuple

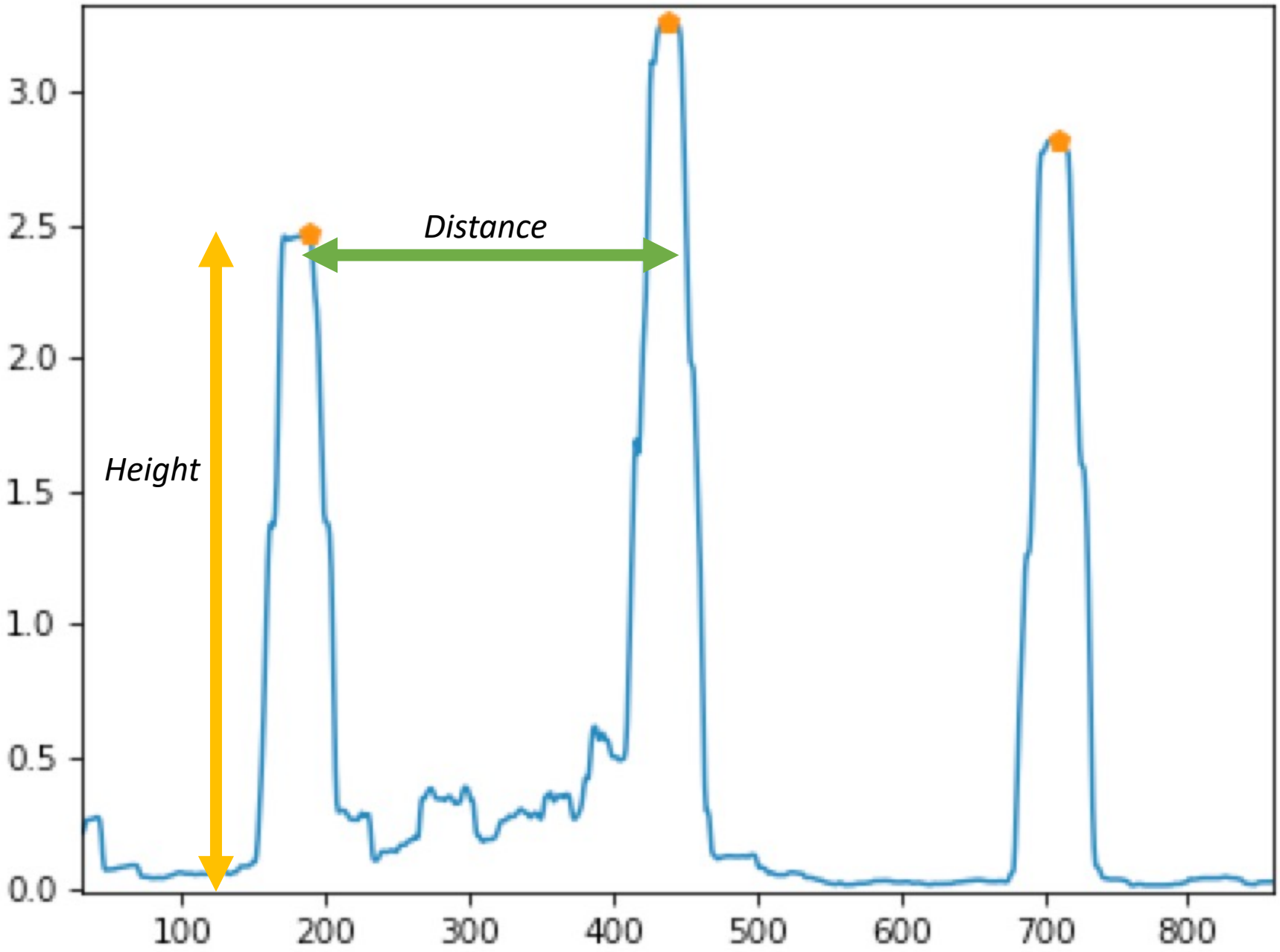
properties : *dict*

A dictionary containing properties of the returned peaks which were calculated as intermediate results during evaluation of the specified conditions:

Height = minimum height of peak; measured in value (y-axis)

Distance = minimum distance between peaks; measured in samples

What are some good values for Height and Distance?

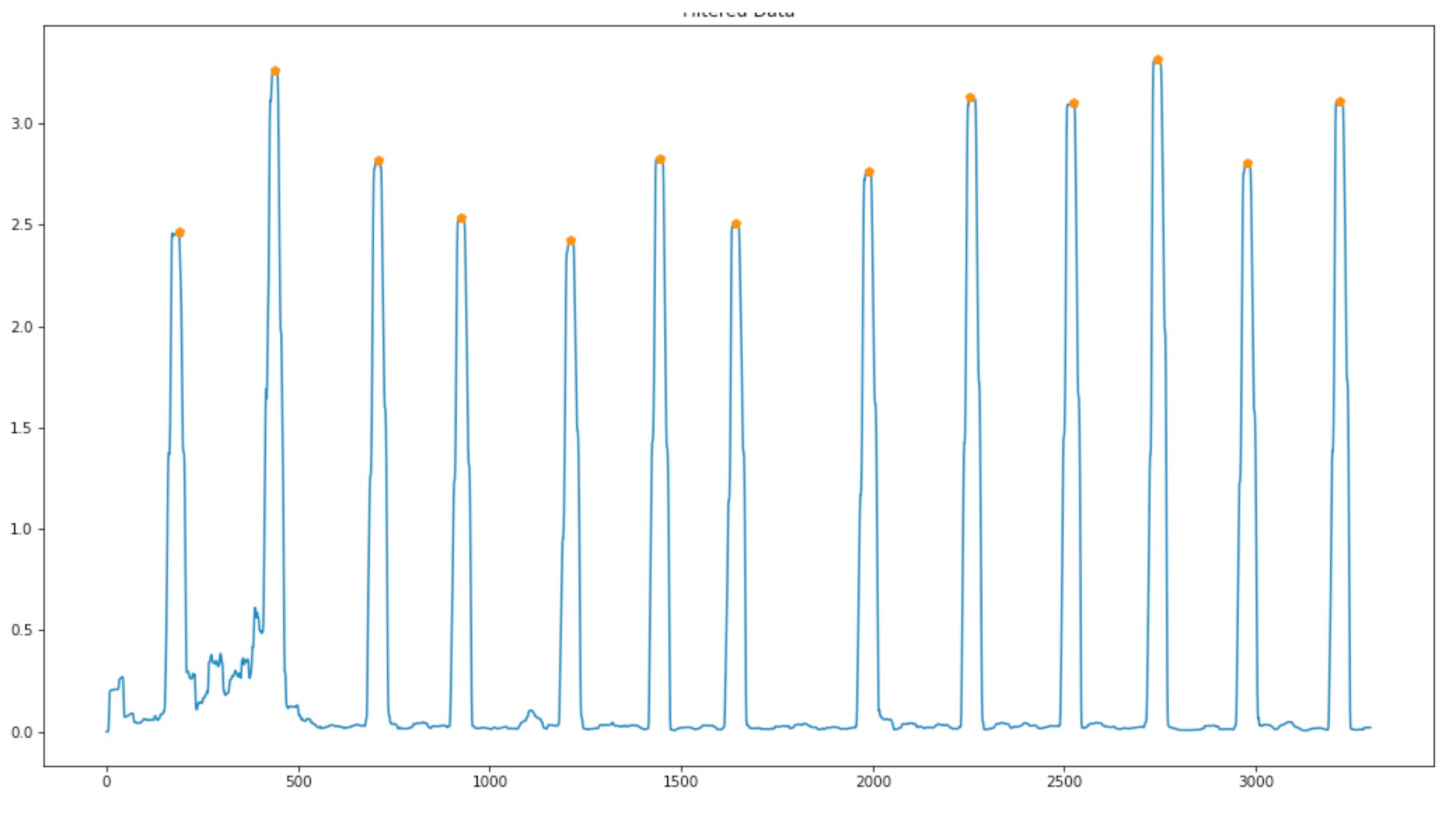


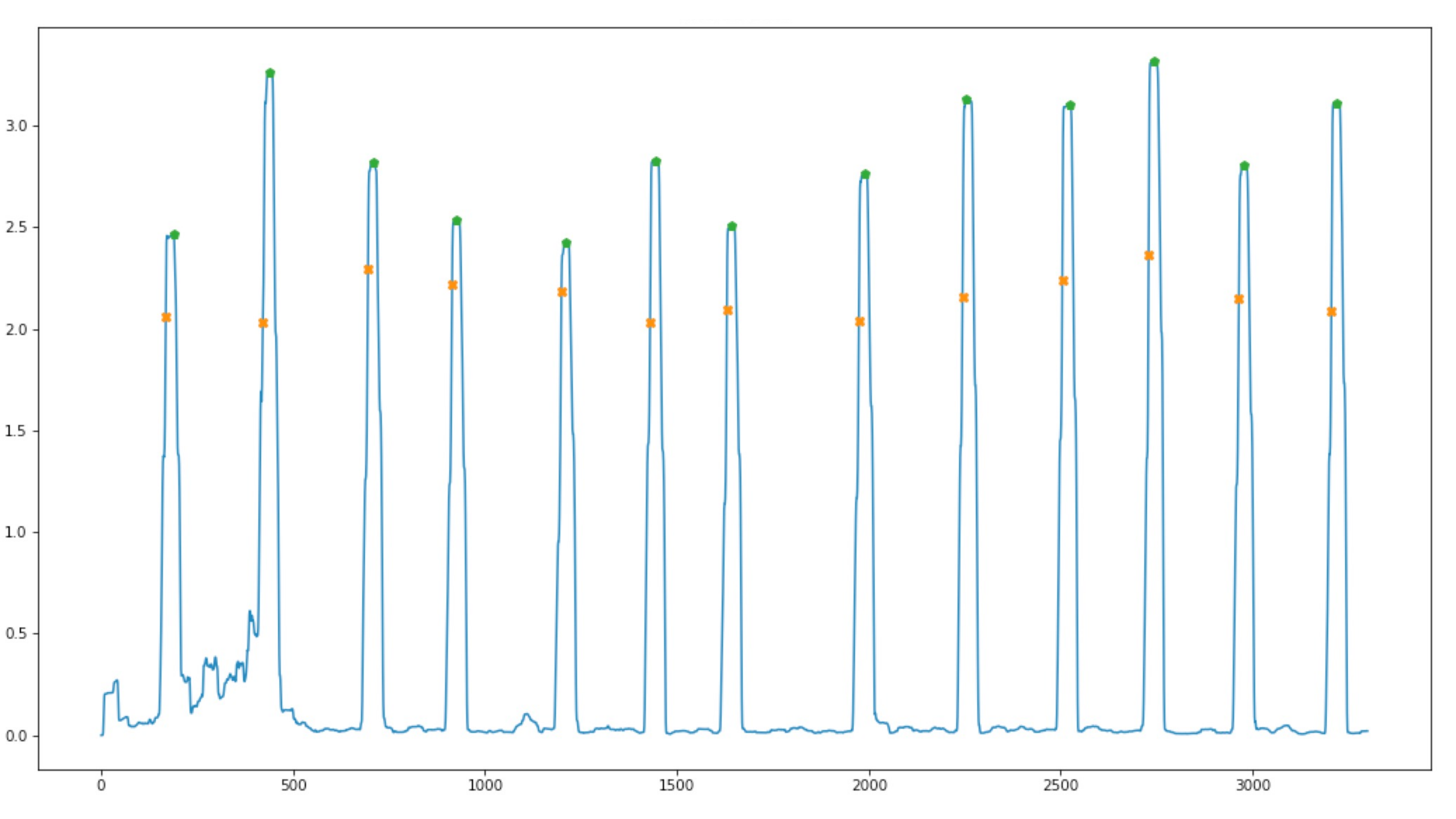
Much simpler than thresholds…

Intentionally name the parameters to be passed. Parameter order does not matter when done this way.

```
# Method 2: Use Find Peaks
peaks, _ = find_peaks(signal, distance=100, height=1)
```

Since function returns two objects, store array as 'peaks' and use _ to drop/ignore the dictionary return


Filtered Data
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000


3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000

Connecting back to evaluation metrics…

- Your program will take a data file and pass back the indices of the potential heart beats.

- My program will compare those samples/indices in the annotations file.

- Automatically calculate accuracy, precision, F1…etc. Highest score to the best performance across all datasets (some of which you will not have seen ☺)