In [12]:

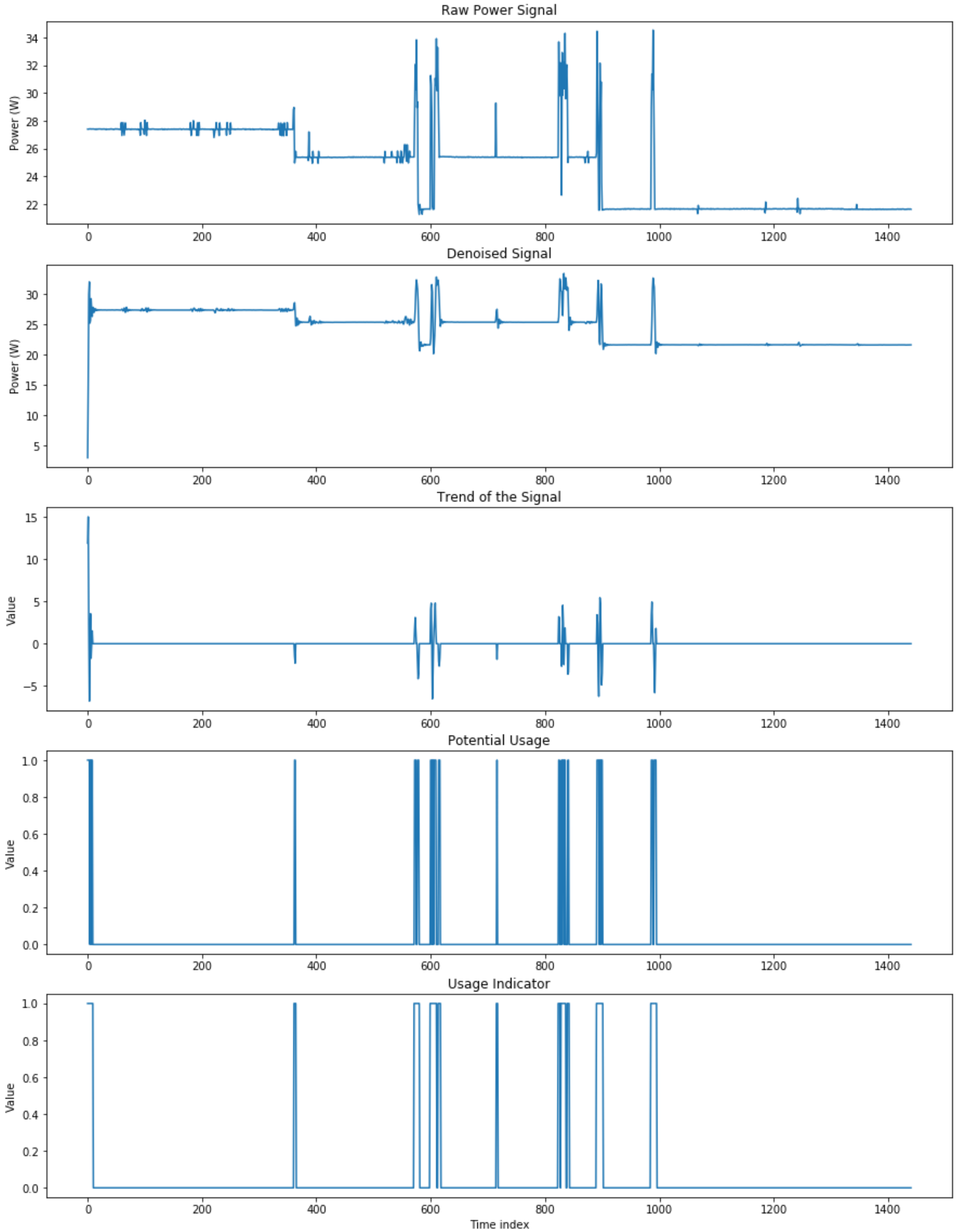
```
"""
This script analyzes a timestamped power consumption signal for an elect
rical device to detect active usage start/stop times.
The input timestamped power consumption signal is read from a csv file a
nd the output start/stop times are marked on a figure and printed out.
"""
import numpy as np
import pandas as pd
from dateutil.parser import parse
import matplotlib.pyplot as plt
from pandas.plotting import register_matplotlib_converters
from scipy.signal import butter, lfilter
register_matplotlib_converters()

def butter_lowpass(cutoff, fs, order=5):
    """
    Designing a Butterworth lowpass filter.

    @type   cutoff: number
    @param  cutoff: cutoff frequency, Hz
    @type   fs:     number
    @param  fs:     sampling frequency, Hz
    @type   order:  number
    @param  order:  filter order
    @rtype: a:      number
    @rparam a:      first coefficient of the filter
    @rtype: b:      number
    @rparam b:      second coefficinet of the filter
    """
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return b, a

def butter_lowpass_filter(signal, cutoff, fs, order=5):
    """
    Filter input signal with Butterworth lowpass filter.

    @type   signal: array
    @param  signal: Input signal
    @type   cutoff: number
    @param  cutoff: cutoff frequency, Hz
    @type   fs:     number
    @param  fs:     sampling frequency, Hz
    @type   order:  number
    @param  order:  filter order
    @rtype  y:      array
    @rparam y:      filtered signal
    """
    b, a = butter_lowpass(cutoff, fs, order=order)
    y = lfilter(b, a, signal)
    return y

def smoothness(deriv_signal, smooth_threshold = 1):
    """
    Smoothing the signal by setting to zero all the signal values around
```

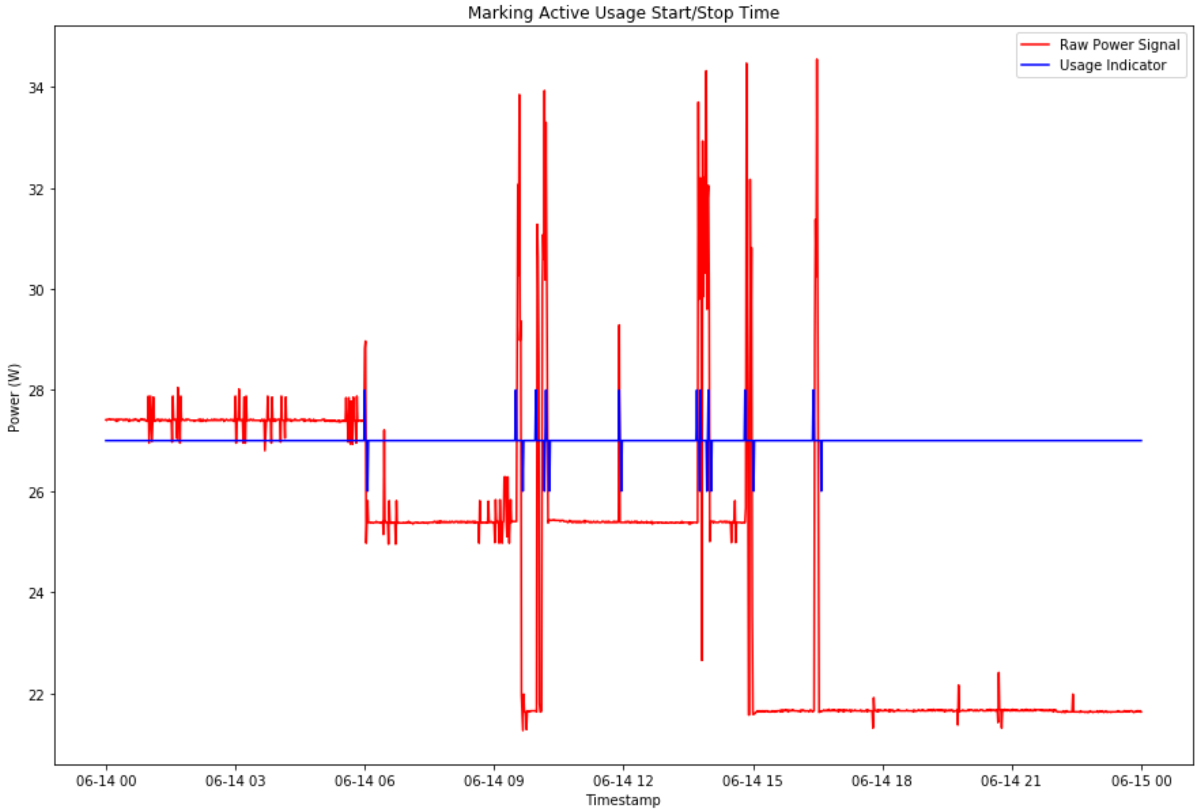
```
zero.
    This function can be applied to the first derivate of a signal to de
noise all constant levels.
    By smoothing a signal, there is no need to apply lowpass filter with
very small cutoff frequency
    which causes data loss.

    @type   deriv_signal:   1D array
    @param  deriv_signal:   first derivative of a signal
    @type   smooth_thre:    number
    @param  smooth_thr:     +- threshold value around zero for smoothnes
s
    @rtype  smooth_signal:  1D array
    @rparam smooth_signal:  smoothed signal
    """
    smooth_signal = deriv_signal
    for i in range(len(deriv_signal)):
        if -smooth_threshold < deriv_signal[i] < smooth_threshold:
            smooth_signal[i] = 0
    return(smooth_signal)

"""
Raw signal preparation.
"""
# parsing the csv file for reading timestamped power consumption signal
try:
    signal_df = pd.read_csv('data-sample.csv', parse_dates=['timestamp'
])
except:
    print("Either \"data-sample.csv\" is not available or in a wrong for
mat.")
else:
    print("timestamped power cosnumption signal is read from the csv fil
e.\n")

# decomposing the signal dataframe to timestamp and power
timestamp  = signal_df.values[:, 0]
power = signal_df.values[:, 1]
time_index = range(len(timestamp))

"""
Denoising the raw signal.
"""
# Butterworth low-pass filter parameters
fs = 10.0       # sampling frequency, Hz
cutoff = 3.0    # cutoff frequency of the filter, Hz

# Filter the signal with butterworth lowpass filter
power_denoised = butter_lowpass_filter(power, cutoff, fs)

"""
Specifying trend of the signal.
"""
# taking derivative of the signal with compensating for size of derivati
ve of the signal
power_trend = np.diff(np.append(power_denoised, power_denoised[len(power
_denoised)-1]))
```

```
# smoothing derivative of the signal
power_trend_smoothed = smoothness(power_trend, smooth_threshold = 1.5)

"""
Specifying the potential active usage periods
"""
# making an on/off potential usage period signal
potential_usage_period = np.zeros(len(power_trend_smoothed))
potential_usage_period [np.nonzero(power_trend_smoothed)] = 1

"""
Selection of true active usage periods using matched filter.
"""
# Defining matched filter (pattern) according to typical active usage pa
ttern
usage_pattern = np.array([0, 0, 0, 1, 1, 1, 0, 0, 0]) # pattern of typic
al usage

# Convolving the matched filter with the potential usage pweriod
usage_period_temp = np.convolve(usage_pattern, potential_usage_period,
'same')

# Converting to an on/off signal
usage_period = np.sign(usage_period_temp)

"""
Indicating start/stop time of active usage
"""
# indicating start/stop time
usage_time_indication = np.diff(np.append(usage_period, usage_period[len
(usage_period)-1]))

# dissmissing false detection at the begining of the signal
usage_time_indication[0:10] = 0

"""
Output results
"""
# Print out the results
start_stop_time_index = np.nonzero(usage_time_indication)
usage_ctr =1
print("Active usage events:\n")
for index in np.arange(0, len(start_stop_time_index[0]), 2):
    print("{:2d}- Start: ".format(usage_ctr) + np.str(timestamp[start_st
op_time_index[0][index]]) + ", " + "Stop: " + np.str(timestamp[start_sto
p_time_index[0][index+1]]) + ",\n")
    usage_ctr += 1

"""
Detection method step-by-step visualization.
"""
fig, visualization = plt.subplots(5, 1, figsize = (15, 20))

# raw signal
visualization[0].plot(time_index, power)
visualization[0].set_ylabel('Power (W)')
```

```
visualization[0].set_title('Raw Power Signal')

# denoised signal
visualization[1].plot(time_index, power_denoised)
visualization[1].set_ylabel('Power (W)')
visualization[1].set_title('Denoised Signal')

# signal trend
visualization[2].plot(time_index, power_trend_smoothed)
visualization[2].set_ylabel('Value')
visualization[2].set_title('Trend of the Signal')

# potential usage indicator
visualization[3].plot(time_index, potential_usage_period)
visualization[3].set_ylabel('Value')
visualization[3].set_title('Potential Usage')

# usage indicator
visualization[4].plot(time_index, usage_period)
visualization[4].set_xlabel('Time index')
visualization[4].set_ylabel('Value')
visualization[4].set_title('Usage Indicator')
plt.show(block = False)

"""
Visualizing the output results.
"""
# Plotting both raw power and usage indicator signals and marking start/
stop times
plt.figure(2, figsize = (15, 10))
plt.plot(timestamp, power, 'r', timestamp, np.add(usage_time_indication,
27), 'b')
plt.xlabel('Timestamp')
plt.ylabel('Power (W)')
plt.title('Marking Active Usage Start/Stop Time')
plt.legend(['Raw Power Signal', 'Usage Indicator'])
plt.show(block = False)
plt.show()
```

```
timestamped power cosnumption signal is read from the csv file.

Active usage events:

 1- Start: 2019-06-14 06:00:00, Stop: 2019-06-14 06:04:00,

 2- Start: 2019-06-14 09:30:00, Stop: 2019-06-14 09:40:00,

 3- Start: 2019-06-14 09:58:00, Stop: 2019-06-14 10:10:00,

 4- Start: 2019-06-14 10:12:00, Stop: 2019-06-14 10:17:00,

 5- Start: 2019-06-14 11:54:00, Stop: 2019-06-14 11:57:00,

 6- Start: 2019-06-14 13:42:00, Stop: 2019-06-14 13:46:00,

 7- Start: 2019-06-14 13:47:00, Stop: 2019-06-14 13:56:00,

 8- Start: 2019-06-14 13:58:00, Stop: 2019-06-14 14:02:00,

 9- Start: 2019-06-14 14:49:00, Stop: 2019-06-14 15:01:00,

10- Start: 2019-06-14 16:24:00, Stop: 2019-06-14 16:35:00,
```


Raw Power Signal
Power (W)
Denoised Signal
Power (W)
Trend of the Signal
Value
Potential Usage
Value
Usage Indicator
Value
Time index


Marking Active Usage Start/Stop Time
Raw Power Signal
Usage Indicator
Power (W)
34
32
30
28
26
24
22
06-14 00
06-14 03
06-14 06
06-14 09
06-14 12
06-14 15
06-14 18
06-14 21
06-15 00
Timestamp

Here are the steps taken to detect the start/stop time of active usages:

1- Denoising the raw signal: A butterworth lowpass filter is designed and the raw signal is filtered out to eliminate the high-frequency noises.

```
Challenges: cutoff frequency has to be selected carefully in order to keep i
mportant information of the signal specially signal edges which also consist
s of high-frequency components.
```

2- Finding trend of the signal: This step is done for detecting the edges.

```
Challenges: 1- Size of first derivative of the signal is decreased by 1 that
must be correctly compensated for finding the corresponding timestamp.
2- To completely denoise the signal remained from the first step due to limi
tation on cutoff frequency, signal smoothness must be done by specifying a p
roper threshold value.
```

3- To indicate the potential usage periods, non-zero values of the first derivative of the signal get 1 and zero values get 0 as an on/off signal.

```
   Challenges: There are many false detections that must be still eliminate
d.
```

4- Matched filter method is used in detecting the true usuage periods and removing false detections. A typical usage pattern is defind and convolved with the potential usage period signal to detect the true usage periods. Sign function is applied to show the result as an on/off signal.

```
   Challenges: 1- Defining the kernel for matched filter, i.e., typical usag
e pattern, is always tricky and has to be done carefully by studying the typ
ical duration of true active usage periods.
```

5- Finally, derivative of the usage on/off signal is taken to indicate start/stop time as a separate signal.

```
Challenges: 1- Size of first derivative of the signal is decreased by 1 that
must be correctly compensated for finding the corresponding timestamp.
2-Please note that always the first couple of samples due to filtering and t
aking derivatives are not reliable that must be eliminated from the final re
sults.
```

```
In [ ]:
```