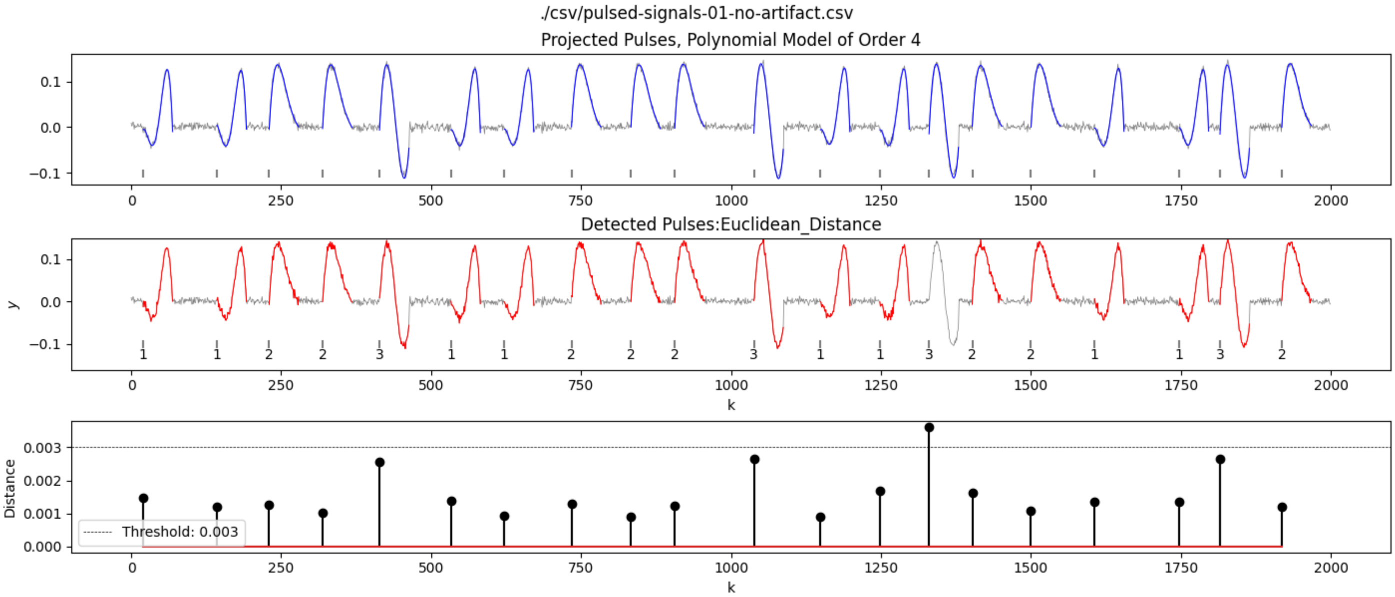
```python
"""
Distance Measures on Neural Spikes (Exercise Set 4)
===================================================
"""
import numpy as np
import matplotlib.pyplot as plt


# Select the CSV file to be load
FILENAME1 = "./csv/pulsed-signals-01-no-artifact.csv"
FILENAME2 = "./csv/pulsed-signals-02-motion-artifact.csv"
FILENAME3 = "./csv/pulsed-signals-03-motion-artifact-inversion.csv"
FILENAME4 = "./csv/pulsed-signals-04-baseline-drift.csv"
FILENAME5 = "./csv/pulsed-signals-05-strong-distortion.csv"

FILENAME=(FILENAME1,FILENAME2,FILENAME3,FILENAME4,FILENAME5)

PULSE_WIDTH = 50 # length of a single pulse, number of samples
POLYNOMIAL_ORDER = 4 # Polynomial order for shape modelling


# ========= Library Functions BEGIN =============================================
def load_CSV(filename):
    """
    Loads a single signal from the CSV file named filename.

    Expected CSV File Format:
    - Column 0: Sample Index k (int)
    - Column 1: Sample Value y_k (float)
    - Column 2: Spike Start (Boolean 0/1)
    - Column 3: Source neuron 1,2,3,... (int)

    Parameters
    ----------
    filename : string
               CSV filename e.g. "data.csv"

    Returns
    -------
    y  : numpy.ndarray of floats of shape (K,)
         Loaded sample data
    is_spike: numpy.ndarray of bool of shape (K,)
         indicates if a spike starts at this time index. "True" or "False".
    neuron_source: numpy.ndarray of float (K,)
         indicates neuron source 1,2,3,... for each time index (=Ground Truth). 0 for no
neuron.

    with
```

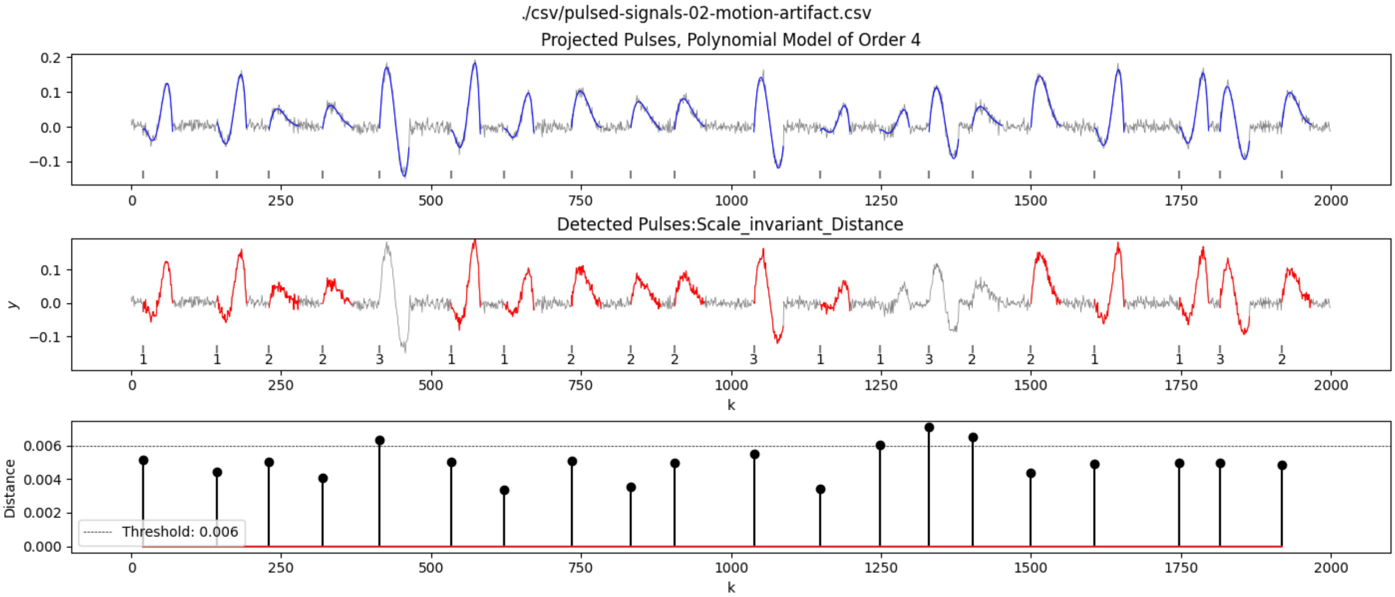
```python
    P: number of pulses
    K: number of samples in the observed signal y.
    """
    ys_csv = np.loadtxt(filename, delimiter=',') # load CSV
    index_k = ys_csv[:,0] # extract sample indices
    y = ys_csv[:,1] # extract signal y
    is_spike = (ys_csv[:,2]>0) # True for spike start at this index; False otherwise
    neuron_source = ys_csv[:,3].astype(int) # source neuron index 1, 2, 3, ...

    return (y, is_spike, neuron_source)

def project_poly(y, k0s, N, Q):
    """
    Projects multiple intervals of y into a space of polynomials of order Q.
    The polynomials are centered at index N//2.

    Parameters
    ----------
    y   : array_like of floats of shape=(K)
          observed signal of length K
    k0s : list of ints length P
          list with start indices of the spikes
    N : int
        length of a single pulse
    Q : int
        polynomial order

    Returns
    -------
    a_s  : numpy.ndarray of shape=shape of (P, N) of floats`
          coefficients of the polynomial models
          (each column contains the coefficients of the model of a single pulse)
    y_hat: numpy.ndarray of shape=shape of (N, P) of floats`
          projected pulse signals of length N
    S    : numpy.ndarray of shape=shape of (Q+1, Q+1) of floats`

    with
    P: number of pulses
    K: number of samples in the observed signal y.

    """

    # --- Generating Base Matrix of Subspace of Polynomials ---
    M = Q+1 # number of base vectors needed
    S = np.zeros((N, M)) # basis matrix of subspace
    for i in range(M): # generating base vectors  x^i
        S[:,i] = np.power(np.arange(N)-N//2, i)

    # Note: The polynomial is centered at base vector index N//2.
```

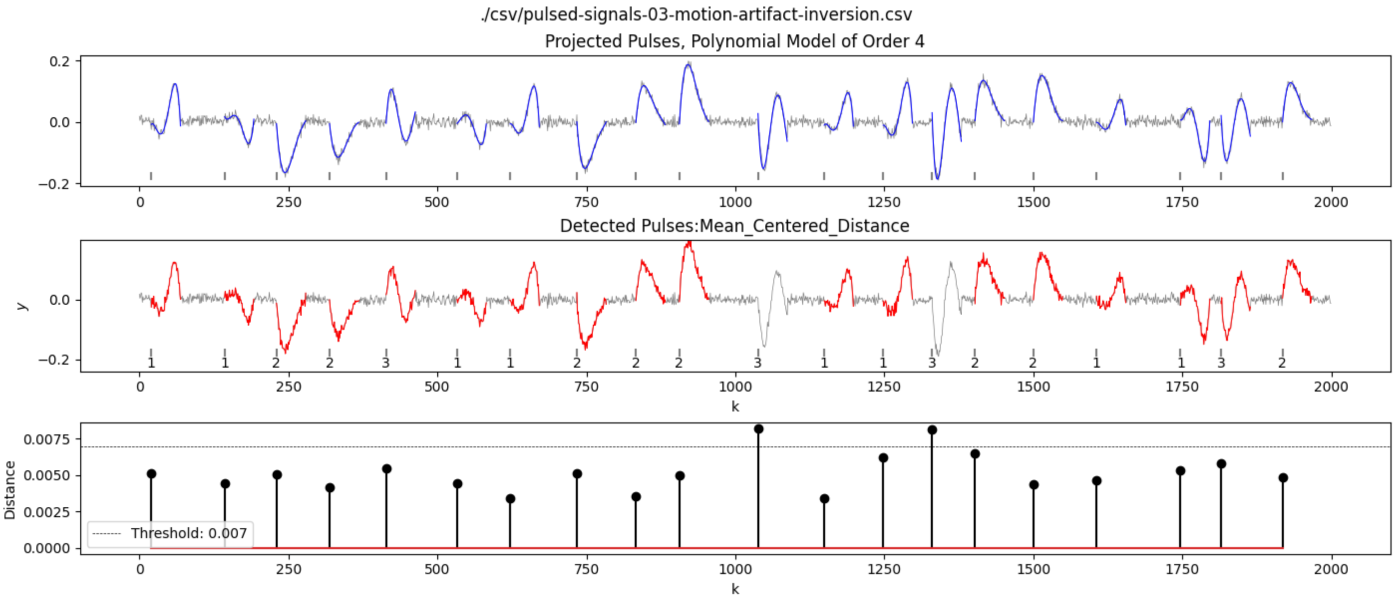
```
    P = len(k0s)
    a_hat_s = np.zeros( (P, M) ) # Memory for coefficients of polynomial approximations
(=model parameters)
    y_intervals_hat = np.zeros( (N, P) ) # memory for projected pulses
    for p in range(P):
       # --- Projection of Signal to Subspace ---
       y_interval = y[indices_spikes[p]:indices_spikes[p]+N ]
       a_hat_s[p,:] = np.linalg.inv(S.transpose()@S)@S.transpose()@y_interval
       y_intervals_hat[:,p] = S@a_hat_s[p,:].T

    return (a_hat_s, y_intervals_hat, S)



# ========= MAIN Programm START =============================================
N=PULSE_WIDTH
# Loading CSV File
for t in range(5):
    filename=FILENAME[t]
    ys_csv=np.loadtxt(filename,delimiter=',')

    (y, is_pulse, neuron_sources) = load_CSV(filename)
    indices_spikes = np.where(is_pulse!=False)[0]  # finde indices of all spike beginning
    (a_hat_s, y_intervals_hat, S) = project_poly(y, indices_spikes, N=PULSE_WIDTH,
Q=POLYNOMIAL_ORDER) # project each pulse to the subspace of

    NOF_PULSES = y_intervals_hat.shape[1] # total number of pulses in the csv file ===p
    K = y.shape[0] # number of samples
    M = a_hat_s.shape[1] # subspace dimension

    # ------ Dummy Code - To be Completed ------
    if t==0: # for clean spike train recording with only very little artifacts
        METHOD_STR = "Euclidean_Distance" # Displayed on the plot
        DISTANCE_THRESHOLD = 0.003 # max. distance for spike detection (to be tuned
manually)
        distances = np.arange(NOF_PULSES) # Distance measure per pulse ** DUMMY CODE ***
        D=np.zeros(NOF_PULSES) # bp.zeros(2)=[0.0]默认浮点数, np.zeros(3, dtype=int)
        # print(distances)
        # print(y[indices_spikes[distances[0]]:indices_spikes[distances[0]]+N ])
        for d in distances: # d==p
            # print(d)
            D[d]=np.sum((y[indices_spikes[d]:(indices_spikes[d]+N)]-
y_intervals_hat[:,d])**2)
        indices_spikes_detected = indices_spikes[D<DISTANCE_THRESHOLD]
        print("The detected pulses using " + METHOD_STR + "of file " + str(t+1)+":"
+str(indices_spikes_detected))

    elif t==1: # for spike train with variable spike amplitudes due to altering distances
between sensor and neurons
```

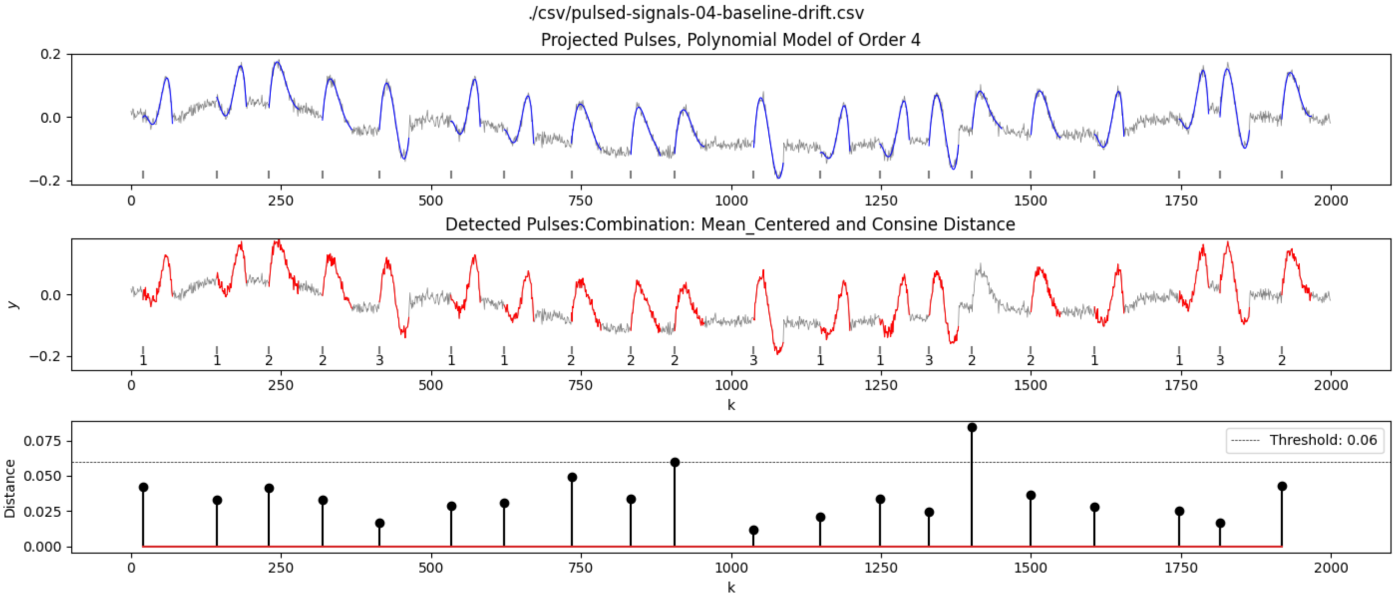
```
        METHOD_STR = "Scale_invariant_Distance" # Displayed on the plot
        DISTANCE_THRESHOLD = 0.006 # max. distance for spike detection (to be tuned
manually)
        distances = np.arange(NOF_PULSES)
        D=np.zeros(NOF_PULSES)
        λ_hat=np.zeros(NOF_PULSES)
        for d in distances: # d==p
            λ_hat[d]=np.dot(y[indices_spikes[d]:indices_spikes[d]+N ],y_intervals_hat[:,d])
/np.square(np.linalg.norm(y_intervals_hat[:,d]))
            D[d]=np.square(np.linalg.norm(y[indices_spikes[d]:indices_spikes[d]+N]))-
λ_hat[d]*np.dot(y[indices_spikes[d]:indices_spikes[d]+N],y_intervals_hat[:,d])
        indices_spikes_detected = indices_spikes[D<DISTANCE_THRESHOLD]
        print("The detected pulses using " + METHOD_STR + "of file " + str(t+1)+":"
+str(indices_spikes_detected))

    elif t==2:
        METHOD_STR = "Mean_Centered_Distance" # Displayed on the plot
        DISTANCE_THRESHOLD = 0.007 # max. distance for spike detection (to be tuned
manually)
        distances = np.arange(NOF_PULSES)
        D=np.zeros(NOF_PULSES)
        for d in distances:
            y_mean=np.mean(y[indices_spikes[d]:indices_spikes[d]+N ])
            y_mean_hat=np.mean(y_intervals_hat[:,d])
            mean_centered_y=y[indices_spikes[d]:indices_spikes[d]+N ]-y_mean
            mean_centered_y_intervals_hat=y_intervals_hat[:,d]-y_mean_hat
            D[d]=np.sum(np.square(mean_centered_y-mean_centered_y_intervals_hat))
        indices_spikes_detected = indices_spikes[D<DISTANCE_THRESHOLD]
        print("The detected pulses using " + METHOD_STR + "of file " + str(t+1)+":"
+str(indices_spikes_detected))

    else:
        METHOD_STR='Combination: Mean_Centered and Consine Distance'
        DISTANCE_THRESHOLD = 0.06 # max. distance for spike detection (to be tuned
manually)
        distances = np.arange(NOF_PULSES)
        D=np.zeros(NOF_PULSES)
        for d in distances:
            y_mean=np.mean(y[indices_spikes[d]:indices_spikes[d]+N ])
            y_mean_hat=np.mean(y_intervals_hat[:,d])
            mean_centered_y=y[indices_spikes[d]:indices_spikes[d]+N ]-y_mean
            mean_centered_y_intervals_hat=y_intervals_hat[:,d]-y_mean_hat
            sc_cos=np.dot(mean_centered_y,mean_centered_y_intervals_hat)/((np.linalg.norm(m
ean_centered_y))*(np.linalg.norm(mean_centered_y_intervals_hat)))
            D[d]=2*(1-sc_cos)
        # Select pulses of low distance
        indices_spikes_detected = indices_spikes[D<DISTANCE_THRESHOLD]
        print("The detected pulses using " + METHOD_STR + "of file " + str(t+1)+":"
+str(indices_spikes_detected))
```

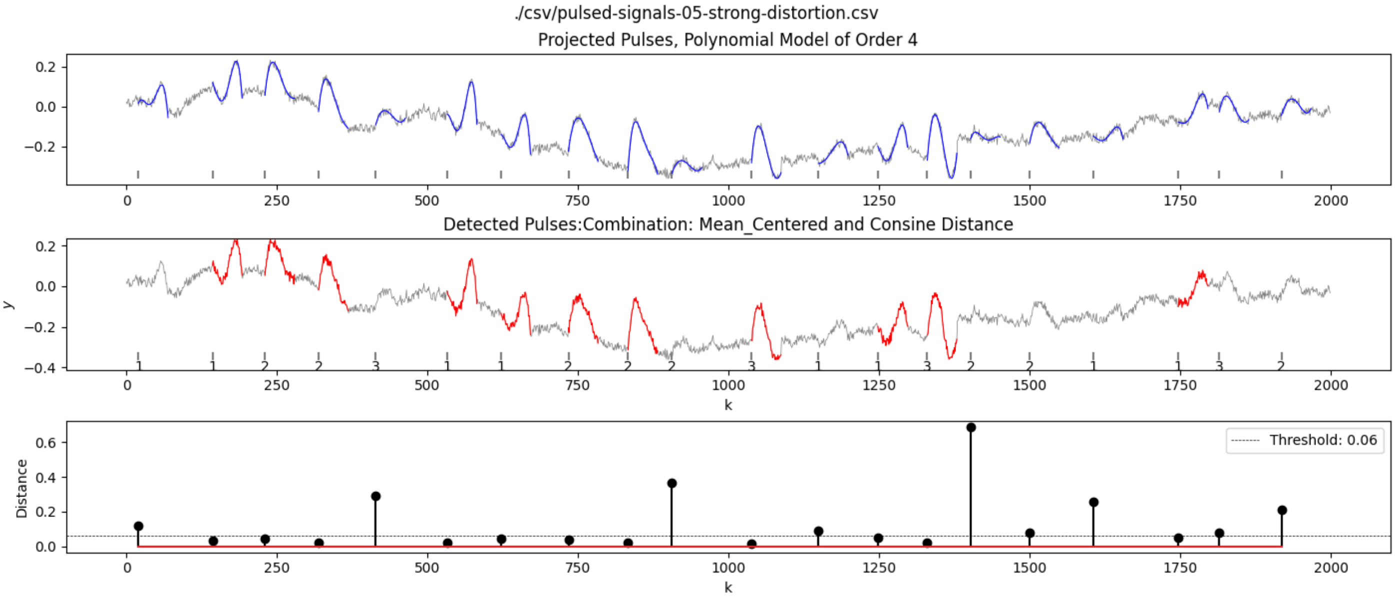
```python
        # ------ Dummy Code - END -----------------





    # ========= PLOTTING of RESULTS =============================================
    fig = plt.figure(figsize=(14,6), constrained_layout=True)
    spec = fig.add_gridspec(3, 1)
    ax0 = fig.add_subplot(spec[0, :])
    ax1 = fig.add_subplot(spec[1, :], sharex=ax0)
    ax2 = fig.add_subplot(spec[2, :], sharex=ax0)

    plt.suptitle(filename)

    # Subplot 1 : plot signal with projections
    ax0.set_title("Projected Pulses, Polynomial Model of Order "+str(POLYNOMIAL_ORDER))
    ax0.plot(range(K), y, lw=0.5, c='tab:gray', label='observation')
    ax0.scatter(indices_spikes, np.ones_like(indices_spikes)*y.min(), marker=2,
c='tab:gray') # markers & reference marker
    for y_interval, k0, is_true_pulse in zip(y_intervals_hat.transpose(), indices_spikes,
neuron_sources):
        ax0.plot(range(PULSE_WIDTH)+k0, y_interval, lw=.75, c='b', label='Pulse') #
projected polynomials

    # Subplot 2 : plot signal with detected pulses
    ax1.set(title="Detected Pulses:"+METHOD_STR)
    ax1.plot(range(K), y, lw=0.5, c='gray', label='observation')
    ax1.scatter(indices_spikes, np.ones_like(indices_spikes)*y.min(), marker=2,
c='tab:gray') # markers & reference marker
    for k0 in indices_spikes_detected: # draw projected polynomials
        ax1.plot(range(PULSE_WIDTH)+k0, y[range(PULSE_WIDTH)+k0], lw=.75, c='r',
label='Pulse')
    for index_spike in indices_spikes:  # add spike source labels
        ax1.annotate(str(neuron_sources[index_spike]), (index_spike, y.min()), ha='center',
va='top')
    ax1.set(xlabel='k', ylabel=r'$y$')
    ax1.set_ylim([y.min()-0.05, y.max()])

    # Subplot 3 : plot distance measure with threshold (dashed line)
    ax2.stem(indices_spikes, D, 'k', markerfmt='ko', label='')
    ax2.axhline(y = DISTANCE_THRESHOLD, ls='--', lw=0.5, c='k', label="Threshold:
"+str(DISTANCE_THRESHOLD))
    ax2.set(xlabel='k', ylabel=r'Distance')
    ax2.legend()

    plt.show()
```


./csv/pulsed-signals-01-no-artifact.csv
Projected Pulses, Polynomial Model of Order 4
Detected Pulses:Euclidean_Distance
y
k
Distance
Threshold: 0.003
k


./csv/pulsed-signals-02-motion-artifact.csv
Projected Pulses, Polynomial Model of Order 4
Detected Pulses:Scale_invariant_Distance
y
k
Distance
Threshold: 0.006
k


./csv/pulsed-signals-03-motion-artifact-inversion.csv
Projected Pulses, Polynomial Model of Order 4
Detected Pulses:Mean_Centered_Distance
y
k
Distance
Threshold: 0.007
k


./csv/pulsed-signals-04-baseline-drift.csv
Projected Pulses, Polynomial Model of Order 4
Detected Pulses:Combination: Mean_Centered and Consine Distance
y
k
Distance
Threshold: 0.06


./csv/pulsed-signals-05-strong-distortion.csv
Projected Pulses, Polynomial Model of Order 4
Detected Pulses:Combination: Mean_Centered and Consine Distance
y
k
Distance
Threshold: 0.06