"""

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

    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(mean\_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))

        # ------ 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()

