##########

# OS\_v1.py computes the optimal number of bins of time-histogram based on the optimization method proposed by Omi and Shinomoto, which may be applicable to non-Poisson spike trains.

# needs libraries: (matplotlib, numpy, pandas).

# Instruction

# put OS\_v1.py in a folder.

# import OS\_v1

# then you may obtain OS\_v1.().

# you need only OS function.

# the function OS takes a spike train as an argument.

# spike train could be given by a list or numpy.array.

# the program selects the optimal bin size for a given spike train and draws the histogram.

# references:

# Takahiro Omi & Shigeru Shinomoto, "Optimizing time histograms for non-Poissonian spike trains", Neural Computation 23, 3125 (2011).

# Contact:

# Shigeru Shinomoto: shinomoto@scphys.kyoto-u.ac.jp

##########

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import math

def OS(spike\_times) :

spike\_times = np.array(spike\_times)

max\_value = max(spike\_times)

min\_value = min(spike\_times)

onset = min\_value - 0.001 \* (max\_value - min\_value)

offset = max\_value + 0.001 \* (max\_value - min\_value)

lv = 0

ISI = np.diff(spike\_times)

for i in range(0, len(spike\_times) - 2) :

interval1 = ISI[i]

interval2 = ISI[i + 1]

if(interval1 + interval2 != 0) :

lv += 3 \* pow(interval1 - interval2, 2) / (pow(interval1 + interval2, 2) \* (len(spike\_times) - 2))

else :

lv += 3 / (len(spike\_times) - 2)

for bin\_num in range(1, 500) :

times = 10

cost = cost\_av(spike\_times, onset, offset, lv, bin\_num, times)

if (bin\_num == 1 or cost < cost\_min) :

cost\_min = cost

optimal\_bin\_num = bin\_num

drawOS(spike\_times, optimal\_bin\_num)

##########

# cost\_f

# computes the cost function defined by Omi and Shinomoto

# arguments:

# spike\_times: spike train

# start: time of the initial spike

# end: time of the final spike

# lv: the value of local variation Lv, which measures the spiking irregularity

# bin\_num: number of bins

# returns the cost function

##########

def cost\_f(spike\_times, start, end, lv, bin\_num) :

bin\_width = (end - start) / bin\_num

hist = np.histogram(spike\_times, np.linspace(start, end, bin\_num + 1))[0]

fano = 2.0 \* lv / (3.0 - lv)

av = np.mean(hist)

va = np.mean(hist \* hist)

w\_av = np.mean(hist \* fano)

fano\_bin = np.where(hist > 2, fano, 1.0)

return ((2.0 \* np.mean(hist \* fano\_bin) - (va - av \* av)) / (bin\_width \* bin\_width))

##########

# cost\_av

# computes an average cost function with respect to initial binning positions.

# arguments:

# spike\_times: spike train

# onset: time of an initial spike

# offset: time of a final spike

# lv: the value of local variation Lv, which measures the spiking irregularity

# bin\_num: the number of bins

# times: the number of initial binning positions

# returns the averaged cost function

##########

def cost\_av(spike\_times, onset, offset, lv, bin\_num, times) :

temp = 0.0

bin\_width = (offset - onset) / bin\_num

TT = np.hstack([spike\_times, spike\_times + (offset - onset)])

for i in range(0, times) :

start = onset + i \* bin\_width / times

end = offset + i \* bin\_width / times

temp += cost\_f(TT, start, end, lv, bin\_num)

return temp / times

##########

# drawOS

# draws a histogram

# arguments:

# spike\_times: a spike train

# optimal\_bin\_num: an optimal number of bins

##########

def drawOS(spike\_times, optimal\_bin\_num):

plt.hist(spike\_times, optimal\_bin\_num)

plt.yticks([])

plt.show()