##########

# SS\_v2.py computes the optimal number of bins of time-histogram based on the optimization method proposed by Shimazaki and Shinomoto 2007.

# needs libraries: (matplotlib, numpy).

# Instruction

# put SS\_v2.py in a folder.

# import SS\_v2

# then you may obtain SS\_v2.().

# you need only SS function.

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

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

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

# references:

# H. Shimazaki and S. Shinomoto, A method for selecting the bin size of a time histogram. Neural Computation (2007) 19:1503-1700.

# Shigeru Shinomoto (2010) Estimating the firing rate. in "Analysis of Parallel Spike Train Data" (eds. S. Gruen and S. Rotter) (Springer, New York).

# Contact:

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

##########

##########

import matplotlib.pyplot as plt

import numpy as np

def SS(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)

# computes the cost function by changing the number of bins

# adopts the number of bins that minimizes the cost function

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

cost = cost\_av(spike\_times, onset, offset, bin\_num, 10)

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

cost\_min = cost

optimal\_bin\_num = bin\_num

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

return optimal\_bin\_num

##########

# cost\_f

# computes the cost function defined by Shimazaki and Shinomoto

# arguments:

# spike\_times: spike train

# start: time of the initial spike

# end: time of the final spike

# bin\_num: number of bins

# returns the cost function

##########

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

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

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

av = np.mean(hist)

va = np.mean(hist \* hist)

return ((2.0 \* av - (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

# 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, bin\_num, times) :

temp = 0.0

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

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

# averages the cost with respect to the starting positions.

# times: number of starting positions.

for i in range(0, times) :

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

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

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

return temp / times

##########

# drawSS

# draws a histogram

# arguments:

# spike\_times: a spike train

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

##########

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

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

plt.yticks([])

plt.show()