##########

# BRE\_v2.py returns Bayesian estimation of the rate of event occurrence.

# needs libraries: (matplotlib, numpy).

# Instruction

# put BRE\_v2.py on a folder in a path.

# import BRE

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

# you need only BRE function

# the function BRE(Bayesian Rate Estimate) takes a spike train as an argument.

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

# a parameter beta is determined by EM algorithm, and a figure is drawn from the rate estimated with the Kalman filter.

# references:

# S. Koyama and S. Shinomoto, Empirical Bayes interpretations of random point events. J. Phys. A (2005) 38:L531-L537.

# T. Shimokawa and S. Shinomoto, Estimating instantaneous irregularity of neuronal firing. Neural Computation (2009) 21:1931-1951.

# 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

##########

# BRE\_v2.py

# Junpei Naito 2017/9/19

##########

import matplotlib.pyplot as plt

import numpy as np

import math

def BRE(spike\_times) :

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

max\_value = max(spike\_times)

min\_value = min(spike\_times)

ISI = np.diff(spike\_times)

mu = len(spike\_times) / (max\_value - min\_value)

beta0 = pow(mu, -3)

beta = EMmethod(ISI, beta0)

kalman = KalmanFilter(ISI, beta)

drawBRE(spike\_times, kalman)

####

# EMmethod

# estimates a parameter beta with the EM algorithm.

# arguments:

# ISI: inter spike interval

# beta0: initial value of the parameter beta

# returns the estimated beta.

####

def EMmethod(ISI, beta0) :

N = len(ISI)

beta = 0

beta\_new = beta0

for j in range(0, 100) :

beta = beta\_new

kalman = KalmanFilter(ISI, beta)

beta\_new = 0

t0 = 0

#------------- revision in version 2 (2017/11/24) begin

indexes = np.where(ISI[:-1] > 0)[0]

beta\_new = sum((kalman[1][indexes + 1] + kalman[1][indexes] - 2 \* kalman[2][indexes]

+ (kalman[0][indexes + 1] - kalman[0][indexes])

\* (kalman[0][indexes + 1] - kalman[0][indexes])) / ISI[indexes])

t0 = N - 1 - len(indexes)

# revised by Naito 2017/11/24

#------------- revision in version 2 (2017/11/24) end

beta\_new = (N - t0 - 1) / (2 \* beta\_new)

return beta\_new

####

# KalmanFilter

# estimates the rate of event occurrence with the Kalman filtering.

# arguments:

# ISI: inter spike interval

# beta: the parameter

# returns the rate of event occurrence.

####

def KalmanFilter(ISI, beta) :

N = len(ISI)

IEL = N / sum(ISI)

IVL = pow(IEL / 3, 2)

A = IEL - ISI[0] \* IVL

EL = np.empty([2, N])

VL = np.empty([2, N])

EL\_N = np.empty(N)

VL\_N = np.empty(N)

COVL\_N = np.empty(N)

EL[0][0] = (A + math.sqrt(A \* A + 4 \* IVL)) / 2

VL[0][0] = 1 / (1 / IVL + 1 / pow(EL[0][0], 2))

# prediction and filtering

for i in range(0, N - 1) :

EL[1][i] = EL[0][i]

VL[1][i] = VL[0][i] + ISI[i] / (2 \* beta)

A = EL[1][i] - ISI[i + 1] \* VL[1][i]

EL[0][i + 1] = (A + math.sqrt(A \* A + 4 \* VL[1][i])) / 2

VL[0][i + 1] = 1 / (1 / VL[1][i] + 1 / pow(EL[0][i + 1], 2))

# smoothing

EL\_N[N - 1] = EL[0][N - 1]

VL\_N[N - 1] = VL[0][N - 1]

for i in range(0, N - 1) :

i = N - 2 - i

H = VL[0][i] / VL[1][i]

EL\_N[i] = EL[0][i] + H \* (EL\_N[i + 1] - EL[1][i])

VL\_N[i] = VL[0][i] + H \* H \* (VL\_N[i + 1] - VL[1][i])

# COVL\_N[i] = H \* VL\_N[i + 1]

COVL\_N = (VL[0][:-1] / VL[1][:-1]) \* VL\_N[1:]

return [EL\_N, VL\_N, COVL\_N]

####

# drawBRE

# draws the rate of event occurrence.

# arguments:

# spike\_times: a spike train

# kalman: estimated rate

####

def drawBRE(spike\_times, kalman) :

xaxis = []

yaxis = kalman[0][:]

for i in range(0, len(spike\_times) - 1) :

xaxis.append((spike\_times[i] + spike\_times[i + 1]) / 2)

plt.stackplot(xaxis, yaxis)

plt.show()