// size parameters of figures

var x\_base = 20;

var width\_graph = 780;

var height\_spike = 10;

var height\_graph = 60;

var height\_hist =54;

var max\_repeat = 500;

var max\_count = 10; // the number of initial binning positions taken for averaging

var res\_graph = 200;

var spike\_num;

var onset, offset;

var lv, np;

function LoadStart() {

return new Promise(function(Main){

document.getElementById("loading").style.visibility="visible";

Main();

});

}

function LoadEnd(){

document.getElementById("loading").style.visibility="hidden";

}

// data initialization

function ResetData() {

//document.data.spikes.value = "0.049 0.141 0.225 0.274 0.303 0.320 0.336 0.437 0.478 0.496 0.538 0.553 0.562 0.580 0.632 0.633 0.645 0.659 0.663 0.673 0.678 0.700 0.721 0.728 0.750 0.765 0.771 0.792 0.815 0.838 0.853 0.867 0.905 0.923 0.936 0.947 0.990 1.003 1.021 1.052 1.073 1.106 1.112 1.141 1.153 1.170 1.185 1.213 1.215 1.279 1.285 1.338 1.417 1.462 1.587 1.591 1.764 1.871 1.888 1.944 1.965 2.006 2.013 2.021 2.046 2.063 2.105 2.126 2.136 2.164 2.216 2.288 2.291 2.308 2.393 2.424 2.436 2.471 2.559 2.580 2.602 2.643 2.674 2.718 2.762 2.909 2.947 2.979 3.010 3.032 3.039 3.052 3.103 3.157 3.185 3.217 3.242 3.273 3.326 3.337 3.348 3.375 3.392 3.415 3.426 3.449 3.475 3.489 3.545 3.636 3.656 3.669 3.696 3.718 3.749 3.780 3.849 3.862 3.869 3.935 4.084 4.202 4.222 4.249 4.287 4.348 4.365 4.386 4.416 4.428 4.431 4.442 4.459 4.492 4.501 4.520 4.534 4.550 4.595 4.603 4.634 4.639 4.655 4.662 4.678 4.691 4.710 4.725 4.732 4.753 4.766 4.805 4.820 4.868 4.887 4.891 4.931 4.965 4.991 5.012 5.072 5.083 5.111 5.181 5.220 5.277 5.327 5.431 5.494 5.565 5.838 5.863 5.894 6.014 6.086 6.103 6.119 6.137 6.149 6.168 6.186 6.214 6.264 6.278 6.306 6.353 6.414 6.422 6.450 6.517 6.532 6.598 6.666 6.693 6.711 6.743 6.788 6.803 6.838 6.846 6.863 6.876 6.891 6.909 6.952 6.959 6.976 6.996 7.015 7.028 7.039 7.052 7.057 7.092 7.130 7.148 7.165 7.195 7.226 7.230 7.241 7.247 7.275 7.287 7.302 7.311 7.317 7.326 7.340 7.354 7.381 7.407 7.440 7.466 7.517 7.519 7.583 7.645 7.658 7.676 7.689 7.778 7.788 7.832 7.864 7.884 7.973 8.042 8.167 8.523 8.592 8.644 8.724 8.776 8.809 8.842 8.863 8.892 8.965 8.969 8.981 9.001 9.012 9.025 9.055 9.060 9.085 9.099 9.136 9.168 9.206 9.212 9.237 9.256 9.294 9.301 9.309 9.330 9.367 9.397 9.448 9.565 9.630 9.683 9.736 9.783 9.845 9.867 9.914 9.950 9.985";

document.data.spikes.value = "1.304 1.317 1.455 1.547 1.565 1.603 1.605 1.628 1.665 1.679 1.684 1.743 1.765 1.767 1.773 1.774 1.806 1.832 1.847 1.863 1.878 1.882 1.909 1.923 1.926 1.939 1.972 1.998 2.043 2.046 2.065 2.088 2.094 2.132 2.142 2.177 2.184 2.193 2.215 2.267 2.291 2.307 2.338 2.397 2.433 2.473 2.518 2.537 2.543 2.580 2.581 2.739 2.766 2.799 2.964 3.082 3.368 3.411 3.512 3.582 3.598 3.710 3.875 3.917 4.146 4.231 4.525 4.872 5.004 5.067 5.091 5.201 5.235 5.310 5.417 5.514 5.554 5.589 5.649 5.668 5.764 5.780 5.794 5.829 5.873 5.900 5.907 5.952 5.979 6.035 6.053 6.092 6.141 6.161 6.189 6.252 6.265 6.292 6.336 6.385 6.448 6.491 6.561 6.656 6.790 6.832 6.970 7.017 7.130 7.342 7.370 7.428 7.448 7.464 7.513 7.528 7.632 7.670 7.683 7.705 7.711 7.718 7.768 7.809 7.815 7.824 7.872 7.881 7.918 7.949 7.953 7.979 7.983 8.061 8.138 8.197 8.252 8.271 8.314 8.323 8.522 8.528 8.540 8.569 8.573 8.584 8.628 8.630 8.658 8.711 8.781 8.854 8.865 9.050 9.154 9.695 9.731 9.833 9.889 9.980";

return 0;

}

// generating data randomly

var MT = new MersenneTwister();

var Alpha = 2.0\*Math.PI\*MT.next();

var Beta = 2.0\*Math.PI\*MT.next();

var Theta = 2.0\*Math.PI\*MT.next();

var Amp = 0.3+1.2\*MT.next();

var SpikeData = new Array(3);

function RandomData() {

var t1,t2;

t1=Solve(0.0,Gamma(1.0));

document.data.spikes.value = Number(t1.toFixed(3));

var j=1;

Alpha = 2.0\*Math.PI\*MT.next();

Beta = 2.0\*Math.PI\*MT.next();

Theta = 2.0\*Math.PI\*MT.next();

Amp = 0.3+1.2\*MT.next();

var kappa = Math.random() \* 1.25 + 0.75;

while(1){

t2=t1+Solve(t1,Gamma(kappa));

if(t2>TIME) break;

document.data.spikes.value += " " + t2.toFixed(3);

t1=t2;

j++;

}

return 0;

}

var Base=30.0;

var Amplitude=10.0;

var TIME=10.0;

var Period=[2.0/Math.PI,1.41421356/Math.PI,0.8989898/Math.PI];

function Rate\_integral(prev\_time,new\_time){

return Base\*(new\_time-prev\_time) - Amplitude\*Period[0]\*Amp\*( Math.cos(Alpha+new\_time/Period[0]/Amp) - Math.cos(Alpha+prev\_time/Period[0]/Amp) ) - Amplitude\*Period[1]\*Amp\*( Math.cos(Beta+new\_time/Period[1]/Amp) - Math.cos(Beta+prev\_time/Period[1]/Amp) ) - Amplitude\*Period[2]\*Amp\*( Math.cos(Theta+new\_time/Period[2]/Amp) - Math.cos(Theta+prev\_time/Period[2]/Amp) );

}

function Solve(prev\_time,interval){

var boundary = new Array(2);

var new\_interval;

boundary[0]=0; boundary[1]=0.5/Base;

while( Rate\_integral(prev\_time,prev\_time+boundary[1]) < interval ){ boundary[1]+=0.5/Base; }

while( boundary[1]-boundary[0] > Math.pow(10.0,-6.0) ){

new\_interval=0.5\*(boundary[0]+boundary[1]);

if( Rate\_integral(prev\_time,prev\_time+new\_interval) > interval ) boundary[1]=new\_interval;

else boundary[0]=new\_interval;

}

new\_interval=0.5\*(boundary[0]+boundary[1]);

if(new\_interval<Math.pow(10.0,-8.0)) new\_interval=Math.pow(10.0,-8.0);

return new\_interval;

}

function Gamma( kappa ){

var int\_kappa=Math.floor(kappa);

var frac\_kappa=kappa-Math.floor(kappa);

var x\_frac,x\_int;

/\*integer part\*/

x\_int=0;

for(var i=0;i<int\_kappa;i++){

x\_int+=-Math.log(MT.next());

}

/\*fractional part\*/

if( frac\_kappa < 0.01 ) x\_frac=0;

else{

var b=(Math.exp(1.0)+frac\_kappa)/Math.exp(1.0);

while(1){

var u=MT.next();

var p=b\*u;

var uu=MT.next();

if(p<=1.0){

x\_frac=Math.pow(p,1.0/frac\_kappa);

if(uu<=Math.exp(-x\_frac)) break;

}

if(p>1.0){

x\_frac=-Math.log((b-p)/frac\_kappa);

if(uu<=Math.pow(x\_frac,frac\_kappa-1.0)) break;

}

}

}

return (x\_int+x\_frac)/kappa;

}

var time\_old = new Array();

// produces a Data object, storing the local time .

var date\_obj = new Date();

// main function

function Main() {

var spike\_time = new Array();

PostData(spike\_time);

spike\_num = spike\_time.length;

// planning to implement a sort function

onset = spike\_time[0] - 0.001 \* (spike\_time[spike\_num - 1] - spike\_time[0]);

offset = spike\_time[spike\_num - 1] + 0.001 \* (spike\_time[spike\_num - 1] - spike\_time[0]);

// transform from absolute times to the passage of times

time\_old[0] = new Date().getTime();

SpikeRaster(spike\_time);

time\_old[1] = new Date().getTime();

DrawGraph\_SSOS(spike\_time); // old method new method

time\_old[3] = new Date().getTime();

DrawGraph\_Kernel12(spike\_time); // kernel smoother (with and without reflection boundary)

time\_old[5] = new Date().getTime();

//DrawGraph\_BayesNP(spike\_time); // Bayes method for non-poisson spike train

//time\_old[5] = new Date().getTime();

DrawGraph\_Bayes(spike\_time); // Bayes method

time\_old[6] = new Date().getTime();

DrawGraph\_HMM(spike\_time); // Hidden Markov Model

time\_old[7] = new Date().getTime();

document.getElementById("time").innerHTML = "<font size='2pt' face='Arial'>Spike Raster : " + (time\_old[1]-time\_old[0]) + " ms<br>(A) : " + (time\_old[2]-time\_old[1]) + " ms<br>(B)-(A) : " + (time\_old[3]-time\_old[2]) + " ms<br>(C) : " + (time\_old[4]-time\_old[3]) + " ms<br>(D)-(C) : " + (time\_old[5]-time\_old[4]) + " ms<br>(E) : " + (time\_old[6]-time\_old[5]) + " ms<br>(F) : " + (time\_old[7]-time\_old[6]) + " ms</font>";

//DrawGraph(spike\_time, SS(spike\_time), "SS"); // old mathod

//DrawGraph(spike\_time, OS(spike\_time), "OS"); // new method

//DrawGraph(spike\_time, Kernel(spike\_time), "Kernel"); // kernel smoother

//DrawGraph(spike\_time, Kernel(spike\_time), "Kernel2"); // kernel smoother with reflection boundary

//DrawGraph(spike\_time, 0, "HMM"); // Hidden Markov Model

}

// processing input data (sorting is not implemented here)

function PostData(spike\_time) {

var data\_text = document.data.spikes.value.replace(/\r?\n/g," ").replace(/^\s+|\s+$/g,"");

var data\_seq = data\_text.split(/[^0-9\.]+/);

//document.data.spikes.value = ""

for (var i = 0; i < data\_seq.length; i++) {

spike\_time[i] = Number(data\_seq[i]);

// document.data.spikes.value += Math.round(spike\_time[i]\*1000) + " ";

}

}

function SpikeRaster(spike\_time){

var names = ['SS','OS','Kernel','Kernel2','Bayes','HMM'];

names.forEach(function(name){

var wrap = d3.select('#raster\_' + name);

wrap.select("svg").remove();

var svg = wrap.append("svg").attr("width",800).attr("height",15);

var line = svg.append("line")

.attr("x1",x\_base)

.attr("y1",height\_spike)

.attr("x2",x\_base+width\_graph)

.attr("y2",height\_spike)

.attr("stroke","black")

.attr("stroke-width",1);

for (var i = 0; i < spike\_num; i++) {

x = (spike\_time[i] - onset) / (offset - onset);

var line = svg.append("line")

.attr("x1",x\_base+width\_graph \* x)

.attr("y1",0)

.attr("x2",x\_base+width\_graph \* x)

.attr("y2",height\_spike)

.attr("stroke","black")

.attr("stroke-width",1);

}

});

}

// Shimazaki-Shinomoto, Omi-Shinomoto

function SSOS(spike\_time) {

var binsize;

var count = new Array();

var cost\_SS, cost\_OS, cost\_SS\_min, cost\_OS\_min;

var w\_av, av, va;

var fano;

var opt\_binsize = new Array(); // [0]:SS, [1]:OS

lv = 0;

// compute the local variation Lv measuring the irregularity

for (var i = 0; i < spike\_num - 2; i++) {

var interval = new Array(2);

interval[0] = spike\_time[i + 1] - spike\_time[i];

interval[1] = spike\_time[i + 2] - spike\_time[i + 1];

if ((interval[0] + interval[1]) != 0) lv += 3 \* Math.pow(interval[0] - interval[1], 2.0) / Math.pow(interval[0] + interval[1], 2.0) / (spike\_num - 2);

else lv += 3.0 / (spike\_num - 2);

}

if (lv < 1) np = "regular";

else np = "bursty";

// vary the number of bins (max 500)

var TT = spike\_time.concat(spike\_time.map(function(element) {

return element + (offset - onset);

}));

for (var bin\_num = 1; bin\_num < max\_repeat; bin\_num++) {

binsize = (offset - onset) / bin\_num;

cost\_SS = 0;

cost\_OS = 0;

for (var cost\_count = 0; cost\_count < max\_count; cost\_count++) {

start = onset + cost\_count \* (binsize) / max\_count;

end = offset + cost\_count \* (binsize) / max\_count;

// initialization of the spike count

for (i = 0; i < bin\_num; i++) {

count[i] = 0;

}

//count the number of spikes

for (i = 0; TT[i] < end; i++) {

if (TT[i] >= start) {

count[Math.floor((TT[i] - start) / binsize)]++;

}

}

// computing the mean and variance of the numbers of spikes in a bin

av = 0;

va = 0;

w\_av = 0;

for (i = 0; i < bin\_num; i++) {

if (count[i] > 2) {

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

} else {

fano = 1.0;

}

w\_av += fano \* count[i] / bin\_num;

av += count[i] / bin\_num;

va += count[i] \* count[i] / bin\_num;

}

// computing the cost function

cost\_SS += (2.0 \* av - (va - av \* av)) / (binsize \* binsize);

cost\_OS += (2.0 \* w\_av - (va - av \* av)) / (binsize\*binsize);

}

cost\_SS /= max\_count

cost\_OS /= max\_count

// updates if the cost is smaller

if (cost\_SS < cost\_SS\_min || bin\_num == 1) {

cost\_SS\_min = cost\_SS;

opt\_binsize[0] = binsize;

}

if (cost\_OS < cost\_OS\_min || bin\_num == 1) {

cost\_OS\_min = cost\_OS;

opt\_binsize[1] = binsize;

}

}

return opt\_binsize;

}

// kernel smoother

function Kernel(spike\_time){

var width = new Array(50);

var cost = new Array(width.length);

var cost\_min,min\_index;

for (var i=0; i<width.length; i++) {

width[i] = (offset - onset) / (i+1);

cost[i] = KernelCost(spike\_time, width[i]);

if(cost[i]<cost\_min || i==0){

cost\_min = cost[i];

min\_index = i;

}

}

return width[min\_index];

}

// cost function of the kernel smoother

function KernelCost(spike\_time, width) {

var A = 0;

for (var i=0; i<spike\_time.length; i++) {

for (var j=i+1; j<spike\_time.length; j++) {

var x = spike\_time[i]-spike\_time[j];

if (x < 5\*width) {

A = A + 2\*Math.exp(-x\*x/4/width/width) - 4\*Math.sqrt(2)\*Math.exp(-x\*x/2/width/width);

}

}

}

return (spike\_time.length/width + A/width) / 2 / Math.sqrt(Math.PI);

}

function Bayes(spike\_time){

var n=0; // 10^n < x < 10^(n+1)

if(offset-onset>1){

while((spike\_time[spike\_time.length-1]-spike\_time[0])>Math.pow(10,n+1)){

n += 1;

}

}else{

while((spike\_time[spike\_time.length-1]-spike\_time[0])<Math.pow(10,n)){

n -= 1;

}

}

}

function DrawGraph\_SSOS(spike\_time){

//SS

var wrap = d3.select('#graph\_SS');

wrap.select("svg").remove(); // initialization

var svg = wrap.append("svg").attr("width",x\_base+width\_graph).attr("height",height\_graph);

var opt = new Array();

opt = SSOS(spike\_time);

var opt\_rate\_SS = new Array();

var rate\_max = EstimateRate(spike\_time, opt[0], opt\_rate\_SS);

var x,y,xx,yy;

for (var i = 0; i < opt\_rate\_SS.length; i++) {

x = i \* opt[0] / (offset - onset);

y = opt\_rate\_SS[i] / rate\_max;

xx = x\_base + width\_graph \* x;

yy = height\_hist \* y;

if (onset + (i + 1) \* opt[0] < offset){

svg.append("rect").attr("x", xx).attr("y", height\_graph-yy).attr("width", width\_graph \* opt[0] / (offset - onset)).attr("height", yy).attr("fill","#87CEFA").attr("stroke","#67AEDA");

}else{

svg.append("rect").attr("x", xx).attr("y", height\_graph-height\_hist \* y).attr("width", width\_graph - width\_graph \* x).attr("height", height\_hist \* y).attr("fill","#87CEFA").attr("stroke","#67AEDA");

}

}

svg.append("rect").attr("x", x\_base).attr("y", 0).attr("width", width\_graph).attr("height", height\_graph).attr("stroke","black").attr("stroke-width",1).attr("fill","none");

document.getElementById("optimal\_SS").innerHTML = "Optimal bin size = <font color=\"red\">" + opt[0].toFixed(2) + "</font>";

time\_old[2] = new Date().getTime();

//OS

wrap = d3.select('#graph\_OS');

wrap.select("svg").remove(); // initialization

svg = wrap.append("svg").attr("width",x\_base+width\_graph).attr("height",height\_graph);

var opt\_rate\_OS = new Array();

rate\_max = EstimateRate(spike\_time, opt[1], opt\_rate\_OS);

for (var i = 0; i < opt\_rate\_OS.length; i++) {

x = i \* opt[1] / (offset - onset);

y = opt\_rate\_OS[i] / rate\_max;

xx = x\_base + width\_graph \* x;

yy = height\_hist \* y;

if (onset + (i + 1) \* opt[1] < offset){

svg.append("rect").attr("x", xx).attr("y", height\_graph-yy).attr("width", width\_graph \* opt[1] / (offset - onset)).attr("height", yy).attr("fill","#7FFFD4").attr("stroke","#5FDFB4");

}else{

svg.append("rect").attr("x", xx).attr("y", height\_graph-height\_hist \* y).attr("width", width\_graph - width\_graph \* x).attr("height", height\_hist \* y).attr("fill","#7FFFD4").attr("stroke","#5FDFB4");

}

}

svg.append("rect").attr("x", x\_base).attr("y", 0).attr("width", width\_graph).attr("height", height\_graph).attr("stroke","black").attr("stroke-width",1).attr("fill","none");

document.getElementById("optimal\_OS").innerHTML = "Optimal bin size = <font color=\"red\">" + opt[1].toFixed(2) + "</font>&nbsp;&nbsp;&nbsp;&nbsp;Irregularity is estimated as Lv = <font color=\"red\">" + lv.toFixed(2) + "</font>";

}

function DrawGraph\_Kernel12(spike\_time){

// Kernel(C)

var wrap = d3.select('#graph\_Kernel');

wrap.select("svg").remove(); // initialization

var svg = wrap.append("svg").attr("width",x\_base+width\_graph).attr("height",height\_graph);

//var opt = Kernel(spike\_time);

var opty1 = new Array();

var opty2 = new Array();

//var maxy = kern12(spike\_time, opt, opty1, opty2);

var res = kernel\_rate(spike\_time,opty1,opty2);

var maxy = res[0];

var opt = res[1];

var xy1 = new Array();

for (var i = 0;i<opty1.length;i++) {

xy1[i] = [x\_base + Math.round(i\*width\_graph/(opty1.length-1)), height\_graph - Math.round(height\_graph\*opty1[i]/(1.2\*maxy))];

}

xy1.unshift([x\_base, height\_graph]);

xy1.push([x\_base+width\_graph, height\_graph]);

var line = d3.svg.line()

.x(function(d) {return d[0];})

.y(function(d) {return d[1];});

svg.append("path").attr("d", line(xy1) ).attr("fill","#F0E68C").attr("stroke","#D0C66C");

svg.append("rect").attr("x", x\_base).attr("y", 0).attr("width", width\_graph).attr("height", height\_graph).attr("stroke","black").attr("stroke-width",1).attr("fill","none");

document.getElementById("optimal\_Kernel").innerHTML = "Optimal bandwidth = <font color=\"red\">" + opt.toFixed(2) + "</font>";

time\_old[4] = new Date().getTime();

// Kernel2(D)

var wrap = d3.select('#graph\_Kernel2');

wrap.select("svg").remove(); // initialization

var svg = wrap.append("svg").attr("width",x\_base+width\_graph).attr("height",height\_graph);

var xy2 = new Array();

for (var i = 0;i<opty2.length;i++) {

xy2[i] = [x\_base + Math.round(i\*width\_graph/(opty2.length-1)), height\_graph - Math.round(height\_graph\*opty2[i]/(1.2\*maxy))];

}

xy2.unshift([x\_base, height\_graph]);

xy2.push([x\_base+width\_graph, height\_graph]);

var line = d3.svg.line()

.x(function(d) {return d[0];})

.y(function(d) {return d[1];});

svg.append("path").attr("d", line(xy2) ).attr("fill","#FFDEAD").attr("stroke","#DFBE8D");

svg.append("rect").attr("x", x\_base).attr("y", 0).attr("width", width\_graph).attr("height", height\_graph).attr("stroke","black").attr("stroke-width",1).attr("fill","none");

document.getElementById("optimal\_Kernel2").innerHTML = "Optimal bandwidth = <font color=\"red\">" + opt.toFixed(2) + "</font>";

}

function DrawGraph\_HMM(spike\_time){

var wrap = d3.select('#graph\_HMM');

wrap.select("svg").remove(); // initialization

var svg = wrap.append("svg").attr("width",x\_base+width\_graph).attr("height",height\_graph);

var x,y,maxy;

var opty;

var opt = ((offset-onset)/(spike\_time.length-1)) \* 5; // step width = ISI \* 5

opty = get\_hmm\_ratefunc(spike\_time, opt);

for(var i=0; i<opty.length; i++){

if(i==0 || maxy<opty[i][1]) maxy=opty[i][1];

}

var x,y,xx,yy;

for (var i = 0; i < opty.length; i++) {

var x\_pos=x\_base+i\*width\_graph/opty.length;

var height=height\_hist\*opty[i][1]/maxy;

if (onset + i \* opt < offset){

svg.append("rect").attr("x", x\_pos).attr("y", height\_graph-height).attr("width", width\_graph/opty.length+1).attr("height", height).attr("fill","#DA75F3");

}else{

x\_pos=offset;

svg.append("rect").attr("x", x\_pos).attr("y", height\_graph-height).attr("width", width\_graph/opty.length+1).attr("height", height).attr("fill","#DA75F3");

}

}

svg.append("rect").attr("x", x\_base).attr("y", 0).attr("width", width\_graph).attr("height", height\_graph).attr("stroke","black").attr("stroke-width",1).attr("fill","none");

}

/\*

DrawGraph\_Bayes

estimates the firing rate with Kalman filtering and draw the rate.

arguments:

spike\_time: spike train

output

draw the estimated firing rates

internal parameters

wrap, svg, maxy, xy: parameters for drawing a figure

kalman\_data: firing rate estimated by Kalman filtering\*/

function DrawGraph\_Bayes(spike\_time){

var wrap = d3.select('#graph\_Bayes');

wrap.select("svg").remove(); // initialization

var svg = wrap.append("svg").attr("width",x\_base+width\_graph).attr("height",height\_graph);

var maxy;

var xy = new Array();

var kalman\_data = SecondStage(spike\_time);

// ThirdStage(spike\_time,beta);

for(var i=0; i<kalman\_data[0].length; i++){

if(i==0 || maxy<kalman\_data[0][i]) maxy=kalman\_data[0][i];

}

for (var i = 0;i<spike\_time.length-1;i++) {

xy[i] = [x\_base + width\_graph\*(spike\_time[i]/2+spike\_time[i+1]/2-spike\_time[0])/(spike\_time[spike\_time.length-1]-spike\_time[0]), height\_graph - height\_graph\*kalman\_data[0][i]/(1.2\*maxy)];

}

xy.unshift([x\_base, height\_graph - height\_graph\*kalman\_data[0][0]/(1.2\*maxy)]);

xy.unshift([x\_base, height\_graph]);

xy.push([x\_base+width\_graph, height\_graph - height\_graph\*kalman\_data[0][spike\_time.length-2]/(1.2\*maxy)]);

xy.push([x\_base+width\_graph, height\_graph]);

var line = d3.svg.line()

.x(function(d) {return d[0];})

.y(function(d) {return d[1];});

svg.append("path").attr("d", line(xy) ).attr("fill","#FFC0CB").attr("stroke","#DFA0AB");

svg.append("rect").attr("x", x\_base).attr("y", 0).attr("width", width\_graph).attr("height", height\_graph).attr("stroke","black").attr("stroke-width",1).attr("fill","none");

}

/\* dft method \*/

function org\_dft(x){

var n = x.length;

var y = new Array();// y[0] = y\_re[]; y[1] = y\_im[];

y[0] = new Array();

y[1] = new Array();

for(var i=0;i<n;i++){

y[0][i] = 0;

y[1][i] = 0;

for(var j=0;j<n;j++){

y[0][i] += x[j]\*Math.cos(2\*Math.PI/n\*i\*j);

y[1][i] += x[j]\*(-Math.sin(2\*Math.PI/n\*i\*j));

}

}

return y;

}

function org\_idft(y){

var n = y[0].length;

// input : y[0] = y\_re[]; y[1] = y\_im[];

var w\_re = Math.cos(-2\*Math.PI/n);

var w\_im = -Math.sin(-2\*Math.PI/n);

var x = new Array();

for(var i=0;i<n;i++){

x[i] = 0;

for(var j=0;j<n;j++){

x[i]+= (y[0][j]\*Math.cos(2\*Math.PI/n\*i\*j) - y[1][j]\*Math.sin(2\*Math.PI/n\*i\*j))/n;

// calculate real part only

}

}

return x;

}

function org\_fft(n,re,im){

var theta = 2\*Math.PI/n;

var firstr = new Array();

var firsti = new Array();

var secondr = new Array();

var secondi = new Array();

for(var i=0;i<n/2;i++){

firstr[i] = re[2\*i];

firsti[i] = im[2\*i];

}

for(var i=0;i<n/2;i++){

secondr[i] = re[2\*i+1];

secondi[i] = im[2\*i+1];

}

if(n/2>1){

org\_fft(n/2,firstr,firsti);

org\_fft(n/2,secondr,secondi);

}

for(var i=0;i<n/2;i++){

var wr = Math.cos(theta \* i);

var wi = Math.sin(theta \* i);

re[i] = firstr[i] + wr\*secondr[i] - wi\*secondi[i];

im[i] = firsti[i] + wr\*secondi[i] + wi\*secondr[i];

wr = Math.cos(theta \* (i+n/2));

wi = Math.sin(theta \* (i+n/2));

re[i+n/2] = firstr[i] + wr\*secondr[i] - wi\*secondi[i];

im[i+n/2] = firsti[i] + wr\*secondi[i] + wi\*secondr[i];

}

}

function org\_ifft(n,re,im){

for(var i=0;i<n;i++){

im[i] = -im[i];

}

org\_fft(n,re,im);

for(var i=0;i<n;i++){

re[i] = re[i]/n

im[i] = -im[i]/n;

}

}

/\*

Function kernel\_rate returns optimized kernel densities estimate using a Gauss kernel function.

Input arguments

spike\_time: sample data list.

Output arguments

y1: Estimated density using a Gauss kernel function.

y2: Estimated density using a Gauss kernel function with reflection boundary.

maxy: Maximum value of y2.

optw: Optimal kernel bandwidth.

Optimization principle:

The optimal bandwidth is obtained as a minimizer of the fromula,

sum\_{i, j} \int k(x - x\_i) k(x - x\_j) dx - 2 sum\_{i~=j} k(x\_i - x\_j),

where k(x) is the kernel function, according to

Hideaki Shimazaki and Shigeru Shinomoto

Kernel Bandwidth Optimization in Spike Rate Estimation

Journal of Computational Neuroscience 2010

http://dx.doi.org/10.1007/s10827-009-0180-4

The above optimization is based on a principle of minimizing

expected L2 loss function between the kernel estimate and an

unknown underlying density function. An assumption is merely

that samples are drawn from the density independently each other.

For more information, please visit

http://2000.jukuin.keio.ac.jp/Shimazaki

\*/

function kernel\_rate(spike\_time, y1, y2){

var T = spike\_time[spike\_time.length-1] - spike\_time[0];

var dt\_samp = spike\_time[1]-spike\_time[0];

for (var i=0;i<spike\_time.length-1;i++){

if(dt\_samp>spike\_time[i+1]-spike\_time[i]) dt\_samp = spike\_time[i+1]-spike\_time[i];

}

var t\_num=1000;

if(Math.ceil(T/dt\_samp)<t\_num){

t\_num = Math.ceil(T/dt\_samp);

}

var t = new Array();

t[0] = spike\_time[0]

for(var i=0; i<t\_num-1;i++){

t[i+1]=t[i]+T/(t\_num);

}

var dt = t[1]-t[0];

for(var i=0;i<t.length-1;i++){

if(dt>t[i+1]-t[i]){

dt=t[i+1]-t[i];

}

}

var y\_hist = new Array();

for(var i=0;i<t.length;i++){

y\_hist[i]=0;

}

for(var i=0;i<spike\_time.length;i++){

for(var j=0;j<t.length-1;j++){

if(spike\_time[i]>=t[j]-dt/2 && spike\_time[i]<t[j+1]-dt/2) y\_hist[j]++;

}

if(spike\_time[i]>=t[t.length-1]-dt/2) y\_hist[t.length-1]++;

}

var L = y\_hist.length;

var N = 0;

for(var i=0;i<L;i++){

N+=y\_hist[i];

}

for (var i=0;i<t.length;i++){

y\_hist[i] = y\_hist[i]/N/dt; // density

}

var Wmin = 2\*dt;

var Wmax = 1\*(spike\_time[spike\_time.length-1] - spike\_time[0]);

var tol = Math.pow(10,-5);

var phi = (Math.sqrt(5) + 1)/2; //golden ratio

// a = Wmin; b = Wmax;

var a=ilogexp(Wmin);

var b=ilogexp(Wmax);

var c1 = (phi-1)\*a + (2-phi)\*b;

var c2 = (2-phi)\*a + (phi-1)\*b;

var f1 = kernel\_cost\_function(y\_hist,N,logexp(c1),dt);

var f2 = kernel\_cost\_function(y\_hist,N,logexp(c2),dt);

var k = 1;

var W = new Array();

var C = new Array();

var optw;

while (Math.abs(b-a) > tol\*(Math.abs(c1)+Math.abs(c2)) && k <= 20){

if (f1 < f2) {

b = c2;

c2 = c1;

c1 = (phi - 1)\*a + (2 - phi)\*b;

f2 = f1;

f1 = kernel\_cost\_function(y\_hist,N,logexp(c1),dt);

// 170926 fix

// W[k] = Math.log(1+Math.exp(c1));

W[k] = logexp(c1)

C[k] = f1;

//var optw = Math.log(1+Math.exp(c1));

optw = logexp(c1);

//y = yh1./sum(yh1.\*dt); //make the final output a density

}else{

a = c1;

c1 = c2;

c2 = (2 - phi)\*a + (phi - 1)\*b;

f1 = f2;

f2 = kernel\_cost\_function(y\_hist,N,logexp(c2),dt);

//W[k] = Math.log(1+Math.exp(c2));

W[k] = logexp(c2);

C[k] = f2;

//var optw = Math.log(1+Math.exp(c2));

optw = logexp(c2);

//y = yh2./sum(yh2.\*dt);

}

k = k + 1;

}

var yh = new Array();

yh = fftkernel(y\_hist,optw/dt);

var sum\_yh = 0;

for(var i=0;i<yh.length;i++){

sum\_yh += yh[i];

}

for(var i=0;i<yh.length;i++){

y1[i] = yh[i]/sum\_yh/dt;

}

/\* reflection part \*/

var yh2 = new Array();

yh2 = fftkernel\_ref(y\_hist,optw/dt);

sum\_yh = 0;

for(var i=0;i<yh2.length;i++){

sum\_yh += yh2[i];

}

var maxy = yh2[0]/sum\_yh/dt;

for(var i=0;i<yh2.length;i++){

y2[i] = yh2[i]/sum\_yh/dt;

if(maxy<y2[i]) maxy = y2[i];

}

var res = new Array(2);

res[0] = maxy;

res[1] = optw;

return res;

}

function kernel\_cost\_function(y\_hist, N, w, dt){

var yh = fftkernel(y\_hist,w/dt); // density

var sumyh = 0;

for(var i=0;i<yh.length;i++){

sumyh += Math.pow(yh[i],2);

}

var sumyh\_hist = 0;

for(var i=0;i<yh.length;i++){

sumyh\_hist += yh[i]\*y\_hist[i];

}

// formula for density

var C = sumyh\*dt - 2\* sumyh\_hist\*dt + 2\*1/Math.sqrt(2\*Math.PI)/w/N;

C = C \* N\* N;

return C;

}

/\*

fftkernel(x, width)

Function fftkernel applies the Gauss kernel smoother to an input signal using FFT algorithm.

Input argument

x: Sample signal vector

width: Kernel bandwidth (the standard deviation) in unit of the sampling resplution of x.

Output argument

re.slice(0, L): Smoothed signal.

MAY 5 / 23, 2012 Author Hideaki Shimazaki

RIKEN Brain Science Institute

http://2000.jukuin.keio.ac.jp/shimazaki

\*/

function fftkernel(x, width){

var L=x.length;

var Lmax = Math.floor(L+3\*width);

var n=1;

while(n<Lmax){

n=n\*2;

}

/\*

var x\_buf=new Float64Array(n);

for (var k=0;k<n;k++){

x\_buf[k]=0;

}

for (var k=0;k<x.length;k++){

x\_buf[k]=x[k];

}

var y\_new = new Array();

y\_new = org\_dft(x\_buf);

\*/

var re=new Float64Array(n);

var im=new Float64Array(n);

for (var k=0;k<n;k++){

re[k]=0;

im[k]=0;

}

for (var k=0;k<x.length;k++){

re[k]=x[k];

}

org\_fft(n,re,im);

var f\_old = new Array();

for (var k=0;k<n;k++){

f\_old[k]=k/n;

}

var f = new Array();

var k=0;

for (;k<Math.ceil(n/2)+1;k++){

f[n-1-k]=f\_old[k+1];

f[k]=-f\_old[k];

}

var K = new Array();

for(var j=0;j<n;j++){

K[j]=Math.exp(-0.5\*Math.pow(width\*2\*Math.PI\*f[j],2));

}

/\*

for(var j=0;j<n;j++){

y\_new[0][j] = y\_new[0][j]\*K[j];

y\_new[1][j] = y\_new[1][j]\*K[j];

}

var x\_new = new Array();

x\_new = org\_idft(y\_new);

return x\_new.slice(0,L);

\*/

for(var j=0;j<n;j++){

re[j] = re[j]\*K[j];

im[j] = im[j]\*K[j];

}

org\_ifft(n,re,im);

return re.slice(0,L);

}

/\*

fftkernel\_ref(x, width)

Function fftkernel applies the Gauss kernel smoother to an input signal using FFT algorithm with reflection boundary.

Input argument

x: Sample signal vector

width: Kernel bandwidth (the standard deviation) in unit of the sampling resplution of x.

Output argument

y: Smoothed signal.

MAY 5 / 23, 2012 Author Hideaki Shimazaki

RIKEN Brain Science Institute

http://2000.jukuin.keio.ac.jp/shimazaki

\*/

function fftkernel\_ref(x,width){

var yh = fftkernel(x,width);

var halflen = Math.ceil(x.length/2);

var remlen = x.length - halflen;

var x\_revleft = new Array();

for(var i=0;i<remlen;i++){

x\_revleft[i] = 0;

}

for(var i=0;i<halflen;i++){

x\_revleft[remlen+i] = x[i];

}

var addleft = fftkernel(x\_revleft,width);

var x\_revright = new Array();

for(var i=0;i<halflen;i++){

x\_revright[i] = x[halflen+i];

}

for(var i=0;i<remlen;i++){

x\_revright[halflen+i] = 0;

}

var addright = fftkernel(x\_revright,width);

var y = new Array();

for(var i=0;i<Math.ceil(yh.length/2);i++){

y[i] = yh[i] + addleft[halflen-1-i];

y[x.length-i-1] = yh[x.length-i-1] + addright[halflen+i];

}

return y;

}

function logexp(x){

var y = new Array();

if(x<100){

y = Math.log(1+Math.exp(x));

}else{

y = x;

}

return y;

}

function ilogexp(x){

// ilogexp = @(x) log(exp(x)-1);

var y = new Array();

if(x<100){

y = Math.log(Math.exp(x)-1);

}else{

y = x;

}

return y;

}

function kern12(spike\_time, width, y1, y2) {

/\*

var x = new Array(res\_graph)

x[0] = onset;

var maxy=0;

var gauss;

var addNumber = 0;

for (var i=0; i<res\_graph; i++) {

x[i+1] = x[i] + (offset-onset)/(res\_graph-1);

}

var maxy=0;

var gauss;

for (var i=0; i<res\_graph; i++) {

addNumber = 0;

y1[i] = 0;

for (var j in spike\_time) {

if((x[i]-5\*width <= spike\_time[j]) && (spike\_time[j] <= x[i]+5\*width)){

gauss = 1/Math.sqrt(2\*Math.PI)/width\*Math.exp(-(x[i]-spike\_time[j])\*(x[i]-spike\_time[j])/2/width/width);

y1[i] = y1[i] + gauss / spike\_time.length;

}

if (x[i]-5\*width < onset) {

if (-(x[i]-5\*width)+2\*onset > spike\_time[j]){

gauss = 1/Math.sqrt(2\*Math.PI)/width\*Math.exp(-(x[i]-(onset-(spike\_time[j]-onset)))\*(x[i]-(onset-(spike\_time[j]-onset)))/2/width/width);

addNumber = addNumber + gauss / spike\_time.length;

}

}else if(x[i]+5\*width > offset){

if(-(x[i]+5\*width)+2\*offset > spike\_time[i]){

gauss = 1/Math.sqrt(2\*Math.PI)/width\*Math.exp(-(x[i]-(offset+(offset-spike\_time[j])))\*(x[i]-(offset+(offset-spike\_time[j])))/2/width/width);

addNumber = addNumber + gauss / spike\_time.length;

}

}

}

y2[i] = y1[i] + addNumber;

if(maxy<y2[i]) maxy=y2[i];

}

\*/

//////////////////

var L=spike\_time.length;

var Lmax = L+3\*width;

var n=1;

while(n<Lmax){

n=n\*2;

}

var imag=new Float64Array(n);

var real=new Float64Array(n);

for (var k=0;k<n;k++){

imag[k]=0;

real[k]=0;

}

for (var k=0;k<spike\_time.length;k++){

real[k]=spike\_time[k];

}

fftnoasm=new FftModule(n,false);

fftnoasm.fft(real,imag,0);

var f\_old = new Array();

for (var k=0;k<real.length;k++){

f\_old[k]=k/real.length;

}

var f = new Array();

var k=0;

for (;k<Math.ceil(real.length/2)+1;k++){

f[real.length-1-k]=f\_old[k+1];

f[k]=-f\_old[k];

}

var K = new Array();

for(var j=0;j<real.length;j++){

K[j]=Math.exp(-0.5\*Math.pow(width\*2\*Math.PI\*f[j],2));

}

for(var j=0;j<real.length;j++){

y1[j] = real[j]\*K[j];

imag[j] = imag[j]\*K[j];

}

fftnoasm.fft(y1,imag,1);

document.data.spikes.value = y1;

var maxy = 0;

for(var i = 0;i<y1.length;i++){

if(maxy<y1[i]) maxy=y1[i];

}

//////////////////

return maxy;

}

// estimate the firing rate, given the parameters of binsize and binrate.

function EstimateRate(spike\_time, opt\_binsize, opt\_rate) {

var opt\_binnum = Math.ceil((spike\_time[spike\_num - 1] - onset) / opt\_binsize);

var rate\_max;

for (var i = 0; i < opt\_binnum; i++) {

opt\_rate[i] = 0;

}

for (i = 0; i < spike\_num; i++) {

opt\_rate[Math.floor((spike\_time[i] - onset) / opt\_binsize)] += 1.0 / opt\_binsize;

}

for (i = 0; i < opt\_binnum; i++) {

if (i == 0 || opt\_rate[i] > rate\_max) rate\_max = opt\_rate[i];

}

return rate\_max;

}

// output

function GenerateOutputFileMessage(message) {

return "<div id='Output'></div> <script type='text/javascript'>var myBlob = new Blob([\"" + message + "\"], {type: 'text/html'}); var url = URL.createObjectURL(myBlob); document.getElementById('Output').innerHTML = '<a href=' + url + ' download=datasheet.csv>download as csv</a>';</script>";

}

function OutputResults\_SS() {

var result;

var spike\_time = new Array();

PostData(spike\_time);

var opt\_binsize = new Array();

var opt\_rate = new Array();

opt\_binsize = SSOS(spike\_time);

EstimateRate(spike\_time, opt\_binsize[0], opt\_rate);

//save as csv

var filemessage = "X-AXIS,Y-AXIS\\n";

filemessage += onset.toFixed(2) + ",0\\n";

for (var i = 0; i < opt\_rate.length; i++) {

filemessage += (onset + i \* opt\_binsize[0]).toFixed(2) + "," + opt\_rate[i].toFixed(2) + "\\n";

filemessage += (onset + (i + 1) \* opt\_binsize[0]).toFixed(2) + "," + opt\_rate[i].toFixed(2) + "\\n";

}

filemessage += (onset + opt\_rate.length \* opt\_binsize[0]).toFixed(2) + ",0\\n";

WIN\_RESULTS = window.open();

WIN\_RESULTS.document.open();

WIN\_RESULTS.document.writeln("<title>Data Sheet of the Optimized Histogram</title>");

WIN\_RESULTS.document.writeln("<h2>Histgram: Poissonian optimization</h2>");

WIN\_RESULTS.document.writeln("Optimal binsize: <b>"+opt\_binsize[0].toFixed(2)+"</b><br><br>");

WIN\_RESULTS.document.writeln(GenerateOutputFileMessage(filemessage));

WIN\_RESULTS.document.writeln("<table border=1><tr align=center><td width=150> X-AXIS (time) </td><td width=150> Y-AXIS (density)</td>");

WIN\_RESULTS.document.writeln("<tr align=right><td>" + onset.toFixed(2) + "</td><td>0.00</td></tr>");

for (var i=0;i<opt\_rate.length;i++) {

WIN\_RESULTS.document.writeln("<tr align=right><td>" + (onset + i \* opt\_binsize[0]).toFixed(2) + "</td><td>" + opt\_rate[i].toFixed(2) + "</td></tr>");

WIN\_RESULTS.document.writeln("<tr align=right><td>" + (onset + (i + 1) \* opt\_binsize[0]).toFixed(2) + "</td><td>" + opt\_rate[i].toFixed(2) + "</td></tr>");

}

WIN\_RESULTS.document.writeln("<tr align=right><td>" + (onset + opt\_rate.length \* opt\_binsize[0]).toFixed(2) + "</td><td>0.00</td></tr>");

WIN\_RESULTS.document.writeln("</table><br>");

WIN\_RESULTS.document.close();

}

function OutputResults\_OS() {

var result;

var spike\_time = new Array();

PostData(spike\_time);

var opt\_binsize = new Array();

var opt\_rate = new Array();

opt\_binsize = SSOS(spike\_time);

EstimateRate(spike\_time, opt\_binsize[1], opt\_rate);

//save as csv

var filemessage = "X-AXIS,Y-AXIS\\n";

filemessage += onset.toFixed(2) + ",0\\n";

for (var i = 0; i < opt\_rate.length; i++) {

filemessage += (onset + i \* opt\_binsize[1]).toFixed(2) + "," + opt\_rate[i].toFixed(2) + "\\n";

filemessage += (onset + (i + 1) \* opt\_binsize[1]).toFixed(2) + "," + opt\_rate[i].toFixed(2) + "\\n";

}

filemessage += (onset + opt\_rate.length \* opt\_binsize[1]).toFixed(2) + ",0\\n";

WIN\_RESULTS = window.open();

WIN\_RESULTS.document.open();

WIN\_RESULTS.document.writeln("<title>Data Sheet of the Optimized Histogram</title>");

WIN\_RESULTS.document.writeln("<h2>Histgram: Non-Poissonian optimization</h2>");

WIN\_RESULTS.document.writeln("Optimal binsize: <b>"+opt\_binsize[1].toFixed(2)+"</b><br>Lv: <b>" + lv.toFixed(2) + "</b> (" + np + " firing)<br><br>");

WIN\_RESULTS.document.writeln(GenerateOutputFileMessage(filemessage));

WIN\_RESULTS.document.writeln("<table border=1><tr align=center><td width=150> X-AXIS (time) </td><td width=150> Y-AXIS (density) </td></tr>");

WIN\_RESULTS.document.writeln("<tr align=right><td>" + onset.toFixed(2) + "</td><td>0.00</td></tr>");

for (var i=0;i<opt\_rate.length;i++) {

WIN\_RESULTS.document.writeln("<tr align=right><td>" + (onset + i \* opt\_binsize[1]).toFixed(2) + "</td><td>" + opt\_rate[i].toFixed(2) + "</td></tr>");

WIN\_RESULTS.document.writeln("<tr align=right><td>" + (onset + (i + 1) \* opt\_binsize[1]).toFixed(2) + "</td><td>" + opt\_rate[i].toFixed(2) + "</td></tr>");

}

WIN\_RESULTS.document.writeln("<tr align=right><td>" + (onset + opt\_rate.length \* opt\_binsize[1]).toFixed(2) + "</td><td>0.00</td></tr>");

WIN\_RESULTS.document.writeln("</table><br>");

WIN\_RESULTS.document.close();

}

function xaxisForKernel(spike\_time) {

var x = new Array(res\_graph);

var data\_max = spike\_time[spike\_time.length - 1];

var data\_min = spike\_time[0];

x[0] = data\_min;

for (var i = 0; i < res\_graph - 1; i++) {

x[i + 1] = x[i] + (data\_max - data\_min) / (res\_graph - 1);

}

return x;

}

function kern(spike\_time, width, y) {

var x = new Array(res\_graph)

x[0] = onset;

for (var i=0; i<res\_graph; i++) {

x[i+1] = x[i] + (offset-onset)/(res\_graph-1);

}

var maxy=0;

var gauss;

for (var i=0; i<res\_graph; i++) {

y[i] = 0;

for (var j in spike\_time) {

if((x[i]-5\*width <= spike\_time[j]) && (spike\_time[j] <= x[i]+5\*width)){

gauss = 1/Math.sqrt(2\*Math.PI)/width\*Math.exp(-(x[i]-spike\_time[j])\*(x[i]-spike\_time[j])/2/width/width);

y[i] = y[i] + gauss;

}

}

if(maxy<y[i]) maxy=y[i];

}

return maxy;

}

function kern2(spike\_time, width, y) {

var x = new Array(res\_graph)

x[0] = onset;

for (var i=0; i<res\_graph; i++) {

x[i+1] = x[i] + (offset-onset)/(res\_graph-1);

}

var maxy=0;

var gauss;

var addNumber = 0;

for (var i=0; i<res\_graph; i++) {

addNumber = 0;

y[i] = 0;

for (var j in spike\_time) {

if((x[i]-5\*width <= spike\_time[j]) && (spike\_time[j] <= x[i]+5\*width)){

gauss = 1/Math.sqrt(2\*Math.PI)/width\*Math.exp(-(x[i]-spike\_time[j])\*(x[i]-spike\_time[j])/2/width/width);

y[i] = y[i] + gauss;

}

if (x[i] - 5\*width<onset) {

if (-(x[i]-5\*width)+2\*onset > spike\_time[j]){

gauss = 1/Math.sqrt(2\*Math.PI)/width\*Math.exp(-(x[i]-(onset-(spike\_time[j]-onset)))\*(x[i]-(onset-(spike\_time[j]-onset)))/2/width/width);

addNumber = addNumber + gauss;

}

}else if(x[i]+5\*width>offset){

if(-(x[i]+5\*width)+2\*offset > spike\_time[i]){

gauss = 1/Math.sqrt(2\*Math.PI)/width\*Math.exp(-(x[i]-(offset+(offset-spike\_time[j])))\*(x[i]-(offset+(offset-spike\_time[j])))/2/width/width);

addNumber = addNumber + gauss;

}

}

}

y[i] += addNumber;

if(maxy<y[i]) maxy=y[i];

}

return maxy;

}

function OutputResults\_Kernel() {

var spike\_time = new Array();

PostData(spike\_time);

var opt = Kernel(spike\_time);

var opty = new Array();

kern(spike\_time, opt, opty);

var xaxis = xaxisForKernel(spike\_time);

//save as csv

var filemessage = "X-AXIS,Y-AXIS\\n";

filemessage += xaxis[0].toFixed(3) + ",0\\n";

for (var i = 0; i < xaxis.length; i++) {

filemessage += xaxis[i].toFixed(3) + "," + opty[i].toFixed(3) + "\\n";

}

filemessage += xaxis[xaxis.length - 1].toFixed(3) + ",0\\n";

WIN\_RESULTS = window.open();

WIN\_RESULTS.document.open();

WIN\_RESULTS.document.writeln("<title>Data Sheet of the Optimized Histogram</title>");

WIN\_RESULTS.document.writeln("<h2>Histgram: Kernel Density Estimation</h2>");

WIN\_RESULTS.document.writeln("Optimal Bandwidth: <b>"+opt.toFixed(3)+"</b><br><br>");

WIN\_RESULTS.document.writeln(GenerateOutputFileMessage(filemessage));

WIN\_RESULTS.document.writeln("<table border=1><tr align=center><td width=150> X-AXIS (time) </td><td width=150> Y-AXIS (density) </td></tr>");

WIN\_RESULTS.document.writeln("<tr align=right><td>"+xaxis[0].toFixed(3)+"</td><td>0.00</td></tr>");

for (var i=0;i<xaxis.length;i++) {

WIN\_RESULTS.document.writeln("<tr align=right><td>"+xaxis[i].toFixed(3)+"</td><td>" + opty[i].toFixed(3) + "</td></tr>");

}

WIN\_RESULTS.document.writeln("<tr align=right><td>"+xaxis[xaxis.length-1].toFixed(3)+"</td><td>0.00</td></tr>");

WIN\_RESULTS.document.writeln("</table><br>");

WIN\_RESULTS.document.close();

}

function OutputResults\_Kernel2() {

var spike\_time = new Array();

PostData(spike\_time);

var opt = Kernel(spike\_time);

var opty = new Array();

kern2(spike\_time, opt, opty);

var xaxis = xaxisForKernel(spike\_time);

//save as csv

var filemessage = "X-AXIS,Y-AXIS\\n";

filemessage += xaxis[0].toFixed(3) + ",0\\n";

for (var i = 0; i < xaxis.length; i++) {

filemessage += xaxis[i].toFixed(3) + "," + opty[i].toFixed(3) + "\\n";

}

filemessage += spike\_time[spike\_time.length - 1] + ",0\\n";

WIN\_RESULTS = window.open();

WIN\_RESULTS.document.open();

WIN\_RESULTS.document.writeln("<title>Data Sheet of the Optimized Histogram</title>");

WIN\_RESULTS.document.writeln("<h2>Histgram: Kernel Density Estimation</h2>");

WIN\_RESULTS.document.writeln("Optimal Bandwidth: <b>"+opt.toFixed(3)+"</b><br><br>");

WIN\_RESULTS.document.writeln(GenerateOutputFileMessage(filemessage));

WIN\_RESULTS.document.writeln("<table border=1><tr align=center><td width=150> X-AXIS (time) </td><td> Y-AXIS (density) </td></tr>");

WIN\_RESULTS.document.writeln("<tr align=right><td>"+xaxis[0].toFixed(3)+"</td><td>0.00</td></tr>");

for (var i=0;i<xaxis.length;i++) {

WIN\_RESULTS.document.writeln("<tr align=right><td>"+xaxis[i].toFixed(3)+"</td><td>" + opty[i].toFixed(3) + "</td></tr>");

}

WIN\_RESULTS.document.writeln("<tr align=right><td>"+xaxis[xaxis.length -1].toFixed(3)+"</td><td>0.00</td></tr>");

WIN\_RESULTS.document.writeln("</table><br>");

WIN\_RESULTS.document.close();

}

function OutputResults\_HMM() {

var spike\_time = new Array();

PostData(spike\_time);

var opty;

var opt = (offset-onset)/(spike\_time.length-1);

var time = onset;

opty = get\_hmm\_ratefunc(spike\_time, opt); // step size = 5\* (mean inter-spike interval)

//save as csv

var filemessage = "X-AXIS,Y-AXIS\\n";

filemessage += time.toFixed(3) + ",0\\n";

filemessage += time.toFixed(3) + "," + opty[0][1].toFixed(3) + "\\n";

time += opt;

for (var i = 1; i < opty.length; i++) {

if (opty[i][1] != opty[i - 1][1]) {

filemessage += time.toFixed(3) + "," + opty[i - 1][1].toFixed(3) + "\\n";

filemessage += time.toFixed(3) + "," + opty[i][1].toFixed(3) + "\\n";

}

time += opt;

}

filemessage += time.toFixed(3) + "," + opty[opty.length - 1][1].toFixed(3) + "\\n";

filemessage += time.toFixed(3) + ",0\\n";

time = onset;

WIN\_RESULTS = window.open();

WIN\_RESULTS.document.open();

WIN\_RESULTS.document.writeln("<title>Data Sheet of the Optimized Histogram</title>");

WIN\_RESULTS.document.writeln("<h2>Histgram: Two state hidden Markov model</h2>");

WIN\_RESULTS.document.writeln(GenerateOutputFileMessage(filemessage));

WIN\_RESULTS.document.writeln("<table border=1><tr align=center><td width=150> X-AXIS (time) </td><td width=150> Y-AXIS (density) </td></tr>");

//WIN\_RESULTS.document.writeln("<tr align=right><td>0.000</td><td>0.000</td></tr>");

WIN\_RESULTS.document.writeln("<tr align=right><td>" +time.toFixed(2)+"</td><td>0.000</td></tr>");

WIN\_RESULTS.document.writeln("<tr align=right><td>" + time.toFixed(2) + "</td><td>" + opty[0][1].toFixed(3) + "</td></tr>");

time+=opt;

for (var i=1;i<opty.length;i++) {

if(opty[i][1]!=opty[i-1][1]){

WIN\_RESULTS.document.writeln("<tr align=right><td>"+time.toFixed(2)+"</td><td>" + opty[i-1][1].toFixed(3)+"</td></tr>");

WIN\_RESULTS.document.writeln("<tr align=right><td>"+time.toFixed(2)+"</td><td>" + opty[i][1].toFixed(3) + "</td></tr>");

}

time+=opt;

}

WIN\_RESULTS.document.writeln("<tr align=right><td>"+ time.toFixed(2) +"</td><td>" + opty[opty.length-1][1].toFixed(3) + "</td></tr>");

WIN\_RESULTS.document.writeln("<tr align=right><td>"+ time.toFixed(2) +"</td><td>0.000</td></tr>");

WIN\_RESULTS.document.writeln("</table><br>");

WIN\_RESULTS.document.writeln("</blockquote>");

WIN\_RESULTS.document.close();

}

function OutputResults\_Bayes(){

var spike\_time = new Array();

PostData(spike\_time);

var opty;

var kalman\_data = SecondStage(spike\_time);

// ThirdStage(spike\_time,beta);

//save as csv

var filemessage = "X-AXIS,Y-AXIS\\n";

filemessage += ((spike\_time[0] + spike\_time[1]) / 2).toFixed(3) + ",0\\n";

for (var i = 0; i < spike\_time.length - 1; i++) {

filemessage += ((spike\_time[i] + spike\_time[i + 1]) / 2).toFixed(3) + "," + kalman\_data[0][i].toFixed(3) + "\\n";

}

filemessage += ((spike\_time[spike\_time.length - 2] + spike\_time[spike\_time.length - 1]) / 2).toFixed(3) + ",0\\n";

WIN\_RESULTS = window.open();

WIN\_RESULTS.document.open();

WIN\_RESULTS.document.writeln("<title>Data Sheet of the Optimized Histogram</title>");

WIN\_RESULTS.document.writeln("<h2>Histgram: Bayesian model Estimation</h2>");

WIN\_RESULTS.document.writeln("<br><br>");

WIN\_RESULTS.document.writeln(GenerateOutputFileMessage(filemessage));

WIN\_RESULTS.document.writeln("<table border=1><tr align=center><td width=150> X-AXIS (time) </td><td width=150> Y-AXIS (density) </td></tr>");

WIN\_RESULTS.document.writeln("<tr align=right><td>"+((spike\_time[0] + spike\_time[1]) / 2).toFixed(3)+"</td><td>0.00</td></tr>");

for (var i=0;i<spike\_time.length - 1;i++) {

WIN\_RESULTS.document.writeln("<tr align=right><td>"+((spike\_time[i] + spike\_time[i + 1]) / 2).toFixed(3)+"</td><td>" + kalman\_data[0][i].toFixed(3) + "</td></tr>");

}

WIN\_RESULTS.document.writeln("<tr align=right><td>"+((spike\_time[spike\_time.length - 2] + spike\_time[spike\_time.length - 1]) / 2).toFixed(3) + "</td><td>0.00</td></tr>");

WIN\_RESULTS.document.writeln("</table><br>");

WIN\_RESULTS.document.close();

}