Program Manual for Marked-Point Process Goodness-of-Fit Toolbox

Based on "Assessing Goodness-of-Fit in Marked-Point Process Models of Neural Population Coding via Time and Rate Rescaling" paper

In this manual, we discuss different Matlab functions being written to implement transformations and techniques developed in the *"Assessing Goodness-of-Fit in Marked-Point Process Models of Neural Population Coding via Time and Rate Rescaling*" paper which are used to assess the goodness-of-fit of the model fitted to marked-point process data. In this manual, we start with an example problem that describes the processing pipeline of using the toolbox functions; we then provide a brief explanation of each function of this toolbox.

1- Example problem

In this example, we assume that events are 3-dimensional marks generated by a multi-variate Gaussian distribution. We run the two transformations proposed in the paper (IRCM, MDCI), which transform the events to uniformly distributed samples in a four-dimensional hypercube. The source code for this example can be found in example.m file. In the following subsections, we discuss how the data is being generated, and also how different transformations can be run using the toolbox.

1-1- Generate data

In the first step, we generate simulation data that contain spike times and marks. We generate simulated spiking data using a marked-point process intensity model consisting of a neuron encoding a position variable, x_t. x_t is modeled as an AR1 process,

$$x_t = 0.98x_{t-1} + \varepsilon_t \tag{1}$$

where, ε_t is a zero-mean white noise process with a standard deviation of 0.3. We generate the simulated spike data in discrete time using a step size of 1 millisecond - $dt = t_k - t_{k-1} = 1\ ms$ based on the following joint mark intensity function

$$\lambda(t, \vec{m}) = \lambda_x(x_t)\phi(\vec{m}; \vec{\mu}_m, \Sigma_m) \tag{2}$$

where, $\phi(\vec{m};\vec{\mu}_m,\Sigma_m)$ is the pdf of a multi-variate normal distribution with mean $\vec{\mu}_m = [8\ 16\ 28]^T$ and covariance $\Sigma_m = \begin{bmatrix} 0.3 & 0.001 & 0.002 \\ 0.001 & 0.04 & 0.003 \\ 0.002 & 0.003 & 0.4 \end{bmatrix}$. In Equation (2),

$$\lambda_x(x_t) = exp\left[\log(0.3) - \frac{(x_t-2)^2}{2\times\left(\sqrt{0.5}\right)^2}\right] \quad (3)$$

that represents the receptive field of a neuron. The receptive field of the cell is centered around x_k of 2 with a bandwidth of $sqrt(0.5)$ The maximum firing reate corresponds to 0.3 per each second. To generate data, we call the GenSpike function. To run this function, we first need to define the time interval and the ground intensity function which is defined by

$$\Lambda(t) = \int_M \lambda(t,\vec{m})d\vec{m} \quad (4)$$

The ground intensity is generated by the following function:

```
GndIntensity = CalGndIntensity(t2,M_l);    % compute ground intensity
```

where, $t2$ represents the time interval that is `t2=0:0.001:8` and M_l are mark intervals, and corresponding mark resolution. The output of this function is a vector with K rows that shows the ground intensity function - $\Lambda(t)$. To generate data, we run these lines of code:

```
%% choose the time and mark resolutions according to the Lambda function and
desired intervals
dt=0.001;
t2=0:dt:8;  % this is the simulation interval
M1_l = 5:0.06:11;
M2_l = 15:0.03:18;
M3_l = 24:0.1:34;
M_l = {M1_l M2_l M3_l};
%% compute ground intensity
GndIntensity = CalGndIntensity(t2,M_l);    % compute ground intensity
%% generate spikes for simulation, does not needed in real cases
Spikes = GenSpike(t2,GndIntensity);        % get the spike info from user,
this is provided by user, too
```

The GenSpike function is filled by the user and the output of it is a $n \times (d+1)$ matrix, where the number of rows shows the number of spikes (events) and the number of columns is equal to the dimension of the mark - d - plus one, where the first column keeps the time of spikes and other d columns keep the corresponding mark information.

Figure (1) shows the ground intensity along with spike times and marks on 3-dimensions space.

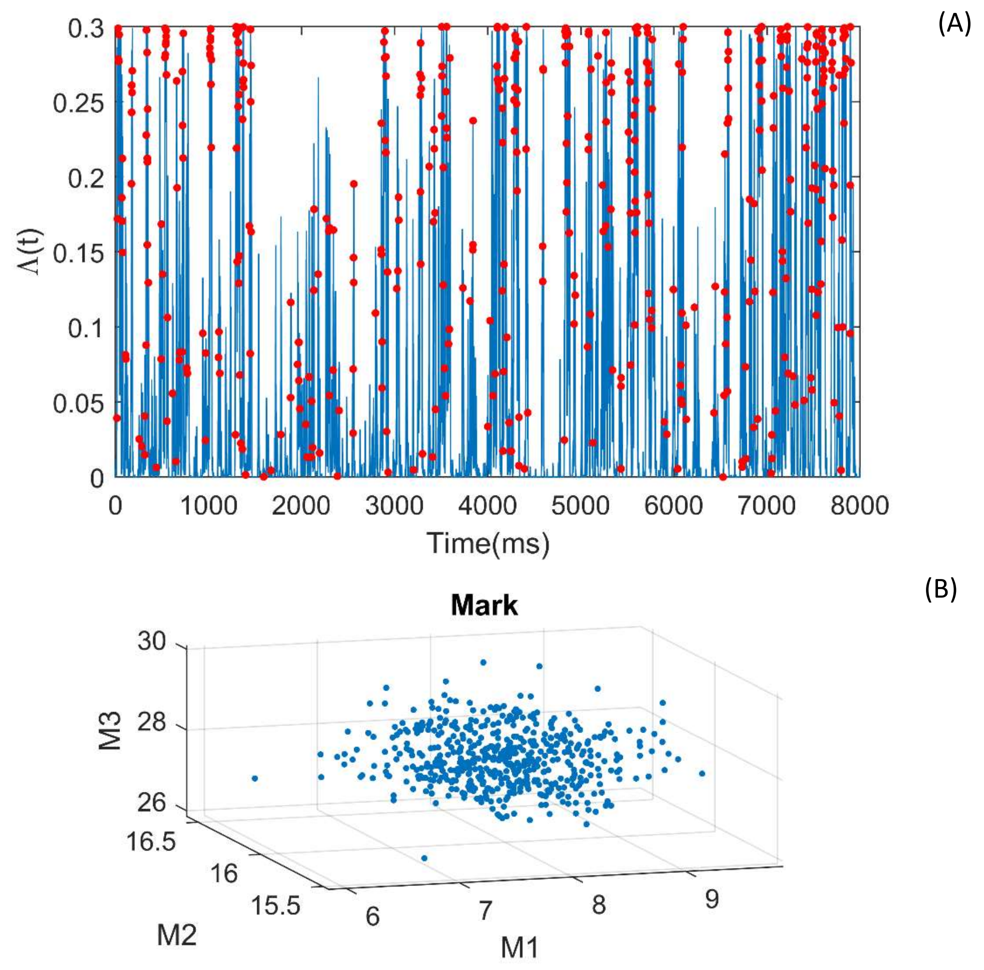


Figure (1): Simulation data including the spike times and marks along with the ground intensity function. (A) The ground intensity along with the spike times, (B) Mark data points

1-2- Marked-Point Process to uniform transformation

In this section, we run two transformations (IRCM and MDCI) proposed in the paper that use a marked point process model to take a dataset of spike times and waveforms to a set of uniformly identically distributed samples under true model on the hypercube $[0\ \ 1]^{d+1}$, where d is the dimension of the mark used to describe the spike waveform features in the model.

A. IRCM

IRCM algorithm requires the ground intensity ($\Lambda(t) = \int_M \lambda(t, \vec{m}) d\vec{m}$) which is created by CalGndIntensity.m function that is already called in the previous section for the data generation process.

After calling CalGndIntensty, we can run the following lines which runs IRCM:

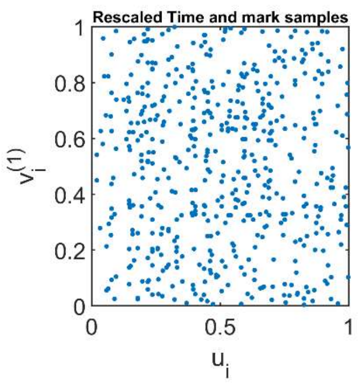
```
%% IRCM
M1_h = 5:0.006:11;  % about 250 points seems good
M2_h = 15:0.003:18;  % more than 500 points ok
M3_h = 24:0.01:34;  % more than 500 points ok
M_h = {M1_h M2_h M3_h};
[u_1,v_1] = IRCM(GndIntensity,Spikes,M_h,M_l,dt);
```

where, $\{M\_l, M\_h\}$ are low and high-resolution mark intervals. To modify this function, since this function is memory consuming, we considered two groups of mark intervals containing high and low-resolution intervals functions are accepting two different resolutions for the mark space. The algorithm check the mark space defined by the finer resolution first; if the necessary memory space is not available on the PC, the algorithm switches to a rough resolution letting the algorithm to be run in the machine. Note that, each dimension might have different resolution and thus, we accept two set of mark spaces instead of giving a particular resolution

The output of this function includes a vector with n rows and a $n \times d$ matrix that contains the rescaled time - u_i – samples, and $v_i^{(l)}$ samples, where $l = 1, \ldots, d$ indicates the l^{th} dimension of the mark, respectively.

Figure (2.A) shows an example projection of u_i and $v_i^{(1)}$ samples and figure (2.B) shows the rescaled mark - $v_i^{(l)}$ – samples in a 3-d hypercube. Given we know the true model that this data is being generated from, we expect transformed data points will be uniformly distributed in a 4-dimensional hypercube.

(A)



(B)

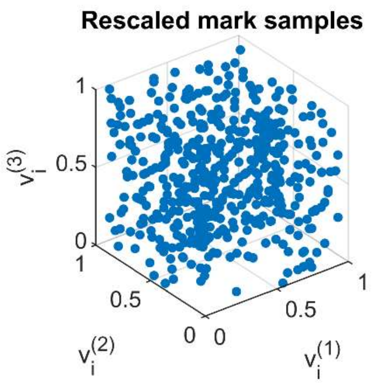


Figure (2): The rescaled data under the IRCM algorithm: (A). The rescaled time samples (u_i) via one dimension of mark -$v_i^{(1)}$ (B).The $v_i^{(l)}$ samples uniformly distributed in the 3-d hypercube

B. MDCI

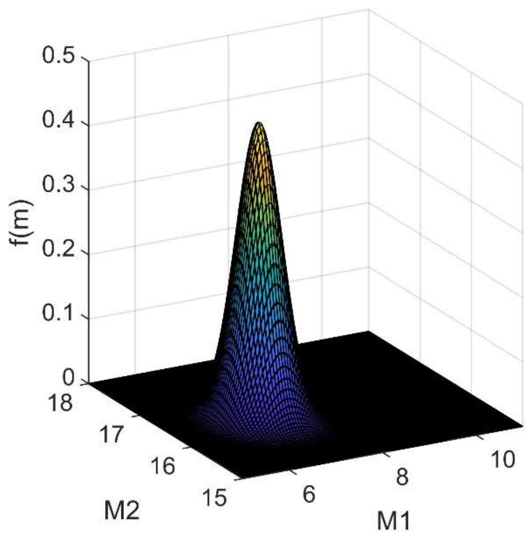
As a part of the MDCI algorithm, we need to calculate the Mark density defined by

$$f(\vec{m}) = \Gamma\,(\vec{m}) / \int_{\mathcal{M}} \Gamma\,(\vec{m}) d\vec{m} \tag{5}$$

where, $\Gamma\,(m) = \int_0^T \lambda\,(t, m) dt$ is mark intensity function. We can calculate $f(\vec{m})$ by calling CalMarkIntensity.m function:

```
%% compute Mark intensity
MarkIntensity = CalMarkIntensity(t2,M_l,dt);
```

where, argument $t2$ represents the time samples, dt is time resolution, and M_l is the low-resolution mark intervals. The output argument is a d-dimension array with $m_i\,, i = 1, .., d$ components per each dimension which shows the mark density function. At this example, this array has 3 dimensions where each dimension is corresponding to each dimension of the mark. Figure (3) shows the mark intensity function for the first and the second dimension of the mark.



Figure(3): Mark intensity function for two dimensions mark

To run this algorithm, we need to run following function:

```
%% MDCI
[u_2,v_2] = MDCI(t2,MarkIntensity,Spikes,M_h,M_l,dt);
```

The rescaled data derived from the MDCI algorithm, under the true model, are uniformly distributed on the 4 dimensions hypercube $[0 \;\; 1]^4$. Similar to the IRCM algorithm, we have two mark intervals with low and high-resolution. The output of this function contains two output arguments, the first one is a vector with n rows that shows the number of spikes and contains the rescaled time - u_i – samples, and a $n \times d$ matrix that the number of rows shows the number of spikes (events) and the number of columns is equal to the dimension of the mark - d. The columns of this matrix contain $v_i^{(l)}$ samples, where $l = 1, \dots, d$ indicates the l^{th} dimension of the mark.

Figure (4.A) shows an example projection of u_i and $v_i^{(1)}$ samples and figure (4.B) shows the rescaled mark - $v_i^{(l)}$ – samples in a 3 hypercube. Given, we know the true model generates this data, we expect to see data points that are uniformly distributed.

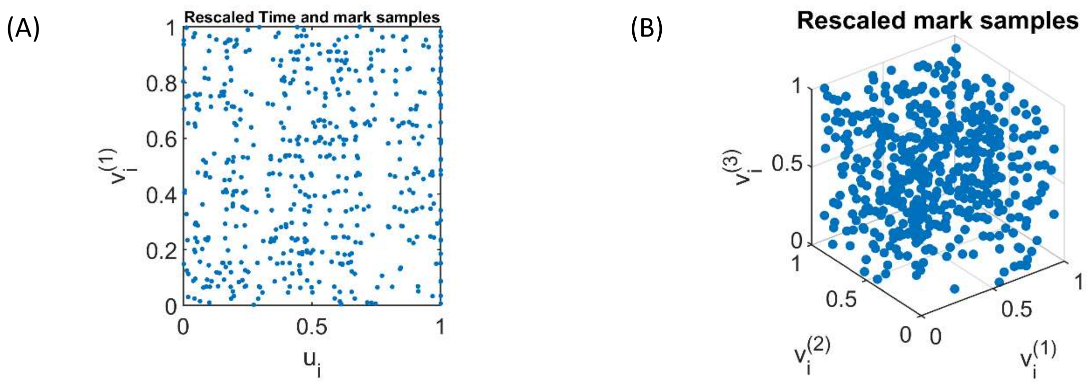


Figure (4): The rescaled data under the MDCI algorithm: (A). The rescaled time samples (u_i) via one dimension of mark -$v_i^{(1)}$ (B).The $v_i^{(l)}$ samples uniformly distributed in the 3-d hypercube

1-3- Uniformity test

In this section, we applied two different uniformity tests, Multivariate KS test, and Pearson χ^2 test.

A. Multivariate KS test

To run this algorithm, we need to run the following lines:

```
%% Uniformity test
% Multivariate KS test:
X = [u_1',v_1];
C_a = 1.12; % it is calculated by Monte Carlo method (Table 2 in paper)
MKS_1 = Gof_MKS(X,C_a);
if (MKS_1 == 1)
    disp('MKS:The null hypothesis is passed under MDCI')
else
    disp('MKS:The null hypothesis is rejected under MDCI')
end
```

The input arguments of Gof_MKS function are the rescaled samples matrix with n rows - the number of spikes - and $d + 1$ columns (the first column is corresponding to the u_i samples and the other columns are related to $v_i^{(l)}$ samples given by IRCM or MDCI algorithms), and the critical value C_a calculated by the Monte Carlo simulation – see the table 2 of paper.

This function calculates the D^{KS} value defined in the Table 2 in the paper. If $D^{KS} > C_\alpha$, the null hypothesis is being rejected (this implicates the uniformity of samples). The output argument – MKS – is 1 when the uniformity test is failed to reject the null hypothesis and otherwise 0.

At this example MKS was 1 given X generated by the IRCM and MDCI algorithms.

B. Pearson χ^2 test

To run this algorithm, we need to run the following function:

```
DF = 26; % Degree of freedom
P = 0.05; % p-value
chi_squ_1 = Gof_Pearson(X,DF,P);
if (chi_squ_1 == 1)
    disp('ChiS:The null hypothesis is passed under IRCM')
else
    disp('ChiS:The null hypothesis is rejected under IRCM')
end
```

The input arguments of Gof_Pearson function are the rescaled samples matrix with n rows - the number of spikes - and $d+1$ columns. The first column corresponds to the u_i samples and the other columns are $v_i^{(l)}$ samples generated by IRCM or MDCI algorithms. DF is the degree of freedom of the Chi-square test for the corresponding probability - P. The output argument is a value - chi_squ – of zero or one; where the value of 1 indicates that the uniformity test is failed to reject the null hypothesis and 0 shows the uniformity test rejects the null hypothesis (that is the uniformity of samples).
At this example $chi\_squ = 1$ under both algorithms, the IRCM and MDCI algorithms.

2- Functions

In this section, we explain the different functions of this toolbox.

CalGndIntensity

It creates the ground intensity defined in the eq.(4). The Ground intensity is used in generating data s, IRCM, and MDCI algorithms.

Syntax

[GndIntensity] = CalGndIntensity(Time,M)

Description

The function returns the ground intensity. The output of this function is a vector with T rows where T represents the number of sample times.

Input Arguments

Time	A vector representing the real-time samples
M	A $d \times 1$ cell representing the mark intervals

Output Arguments

GndIntensity	A vector representing ground intensity at time points defined in Time input argument

CalMarkIntensity

It creates the mark intensity defined by eq.(6) that used in the MDCI algorithm.

Syntax

[MarkIntensity] = CalMarkIntensity(Time,M,dt)

Description

The function returns the mark intensity. The mark intensity is a $m_1 \times m_2 \times \ldots \times m_d$ array, where m_i is related to the i^{th} dimension of the mark.

Input Arguments

Time	A vector representing the real-time samples
M	A $d \times 1$ cell representing the mark intervals
dt	Time resolution

Output Arguments

MarkIntensity	An array representing mark intensity in the given Time samples

GenSpike

It generates spike times and marks given the real-time samples and ground intensity defined in eq(4). This function is modified by the user based on model data.

Syntax

[Spikes] = GenSpike(Time,GroundIntensity)

Description

The function returns the spike times and marks given ground intensity based on model data.

Input Arguments

Time	A vector representing the real-time samples
GroundIntensity	A vector representing the ground intensity

Output Arguments

Spikes	A $n \times (d+1)$ matrix; the number of rows shows the number of spikes and the number of columns is equal to the dimension of the mark - d - plus one

Lambda

It generates the joint mark intensity function - $\lambda(t,\vec{m})$. This function is modified by the user based on model data. Because this function is a memory-consuming function if you define a vast range for time and mark intervals, this massage appears:

```
6.593926e+05 megabytes needed, like to proceed?
y: to continue
any other key: terminate m file
```

If you want to continue you might select 'y' and otherwise, you need to change the time interval or mark intervals.

Syntax

[Lambda] = Lambda(Time,M1,M2,M3)

Description

The function returns $\lambda(t,\vec{m})$ given time spike and mark.

Input Arguments

Time	A vector representing the real-time samples
M1	A vector representing the first dimension of mark interval
M2	A vector representing the second dimension of mark interval
M3	A vector representing the third dimension of mark interval

Output Arguments

Lambda	A $(d+1)$ −array, the first dimension represents the time samples and other d dimensions show the mark information per time sample

IRCM

It converts a dataset of spike times and marks to a set of uniformly identically distributed samples on the hypercube $[0\ \ 1]^{d+1}$.

Syntax

[u,v]=IRCM(GndIntensity,Spikes,M_h,M_l,dt)

Description

The function returns rescaled samples given ground intensity, mark intervals, and time resolution.

Input Arguments

GndIntensity	A vector representing the Ground intensity derived by CalGndIntensity function
Spikes	A $n \times (d+1)$ matrix derived by GenSpike function
M_h	A cell representing the high-resolution mark interval
M_l	A cell representing the low-resolution mark interval
dt	The time resolution

Output Arguments

u	A vector with n columns that shows the rescaled time samples under the IRCM algorithm
v	A $n \times d$ matrix, the number of rows shows the number of spikes and each column is related to each dimension of rescaled mark data $v_i^{(l)}$ under the IRCM algorithm

MDCI

It converts a dataset of spike times and marks to a set of uniformly identically distributed samples on the hypercube $[0\ \ 1]^{d+1}$. Despite of the IRCM algorithm, at this algorithm, we do not need to use the high-resolution mark intervals.

Syntax

[u,v]=MDCI(Time,MarkIntensity,Spikes,M_l,dt)

Description

The function returns rescaled samples given real-time samples, mark intensity, mark intervals, and time resolution.

Input Arguments

Time	A vector representing the real-time samples
MarkIntensity	A matrix representing the mark intensity derived by CalMarkIntensity function
Spikes	A $n \times (d+1)$ matrix derived by GenSpike function
M_l	A cell representing the low-resolution mark interval
dt	The time resolution

Output Arguments

u	A vector with n columns that shows the rescaled time samples under the MDCI algorithm

v	A $n \times d$ matrix, the number of rows shows the number of spikes and each column is related to each dimension of rescaled mark data $v_i^{(l)}$ under the MDCI algorithm

Gof_MKS

It returns the result of the uniformity test under the multivariate KS test.

Syntax

MKS = Gof_MKS(X,C_a)

Description

It returns the result of the uniformity test under the multivariate KS test. If $MKS = 0$ the null hypothesis is rejected (that is the uniformity of samples) otherwise, the null hypothesis is passed.

Input Arguments

C_a	A number that shows the critical value of multivariate KS test calculated by the Monte Carlo estimation – Table 2 of paper
X	A $n \times (d + 1)$ matrix representing the rescaled samples

Output Arguments

MKS	A logical value representing the result of MKS test

Gof_Pearson

It returns the result of the uniformity test under the Pearson χ^2test.

Syntax

chi_squ = Gof_Pearson(X,DF,P)

Description

It returns the result of the uniformity test under the Pearson χ^2test. If $chi\_squ = 0$ the null hypothesis is rejected (that is the uniformity of samples) otherwise, the null hypothesis is passed.

Input Arguments

X	A $n \times (d + 1)$ matrix representing the rescaled samples
DF	The degree of freedom for the chi-square distribution
P	The corresponding probability of chi-square distribution

Output Arguments

Chi_squ	A logical value representing the result of the Pearson χ^2 test