Student Name: Mohammad Mohammad Beigi
Student ID: 99102189
Subject: Neural Encoding

Advanced Topics in Neuroscience - Dr. Ali Ghazizadeh
Assignment 1

---

1 Integrate and Fire Neuron

Here we are going to simulate spike trains based on [Softky & Koch, 1993] and compare its statistics with real neural data. The simplest neuron model in this paper was a kind of integrate and fire neuron where it's inter-spike intervals (ISI) distribution explained by equation [9].
Temporarily, rather than direct implementation of equation [9] we are going to generate a spike train using renewal process. This procedure might help us understand why an Integrate and fire neuron, shows lower variability even with receiving irregular post synaptic inputs!

1. Generate a spike train using a Poisson process with r = 100 and $\Delta\tau$ = 1ms.

Answer:

We will represent spike trains as matrices. Each element of a matrix represents a time interval of 1ms. If there is a spike in this time interval, then we set the value of the element to 1, else we set it to 0.
Inputs of our function are:
1. firing rate fr 2. $\Delta\tau$ dt 3. duration of simulation tSim 4. number of trials nTrials
And outputs are:
1. spike matrix spikeMat 2. time vector tVec

The described functions is as below:

```
function [ spikeMat , tVec ] = poissonSpikeGen ( fr , dt, tSim , nTrials )
nBins = floor ( tSim / dt ) ;
spikeMat = rand ( nTrials , nBins ) < fr * dt ;
tVec = 0: dt : tSim - dt ;
end
```

Then we use the following command:

```
clc;
clear all;
close all;
[ spikeMat , tVec ] = poissonSpikeGen ( 100 , 1/1000, 1 , 100 );
```

2. Plot spike count probability histogram calculated from many Poisson spike trains, each of 1 sec duration with r = 100, superimposed with the theoretical (Poisson) spike count density.

Answer:

We will implement a function which gets a m by n matrix and returns a 1 by m matrix containing number of ones in each row(here each row is 1 trial) and plots histogram of the created matrix and a Poisson distribution with $\lambda = 100$.

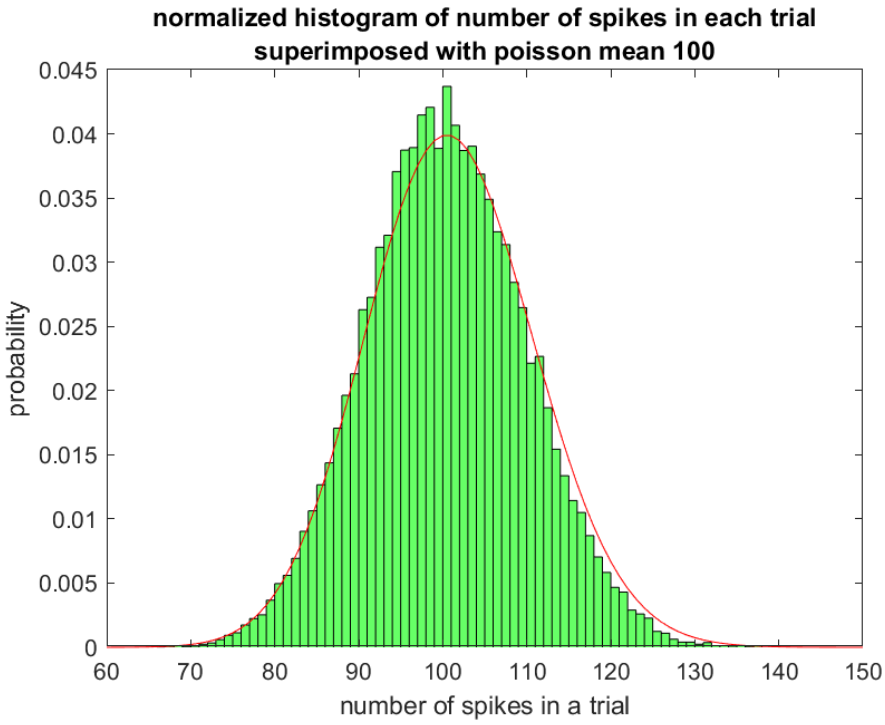
The described function is as below:

```
function numOfOnes = histogramPlot(spikeMat)
numOfOnes = sum(transpose(spikeMat(:, :)) == 1);
figure;
histogram(numOfOnes,'Normalization','probability', 'FaceColor','g');
hold on
x = 0:150;
y = poisspdf(x,100);
plot(60:150, y(60:150),"r")
xlabel("number of spikes in a trial")
ylabel("probability")
title(["normalized histogram of number of spikes in each trial",...
    "superimposed with poisson mean 100"])
end
```

By using the command

```
numOfOnes = histogramPlot(spikeMat);
```

We will get the following figure:



3. Plot Inter-spike interval (ISI) histogram calculated from the simulated Poisson spike trains, superimposed with the theoretical (exponential) inters-pike interval density.

Answer:

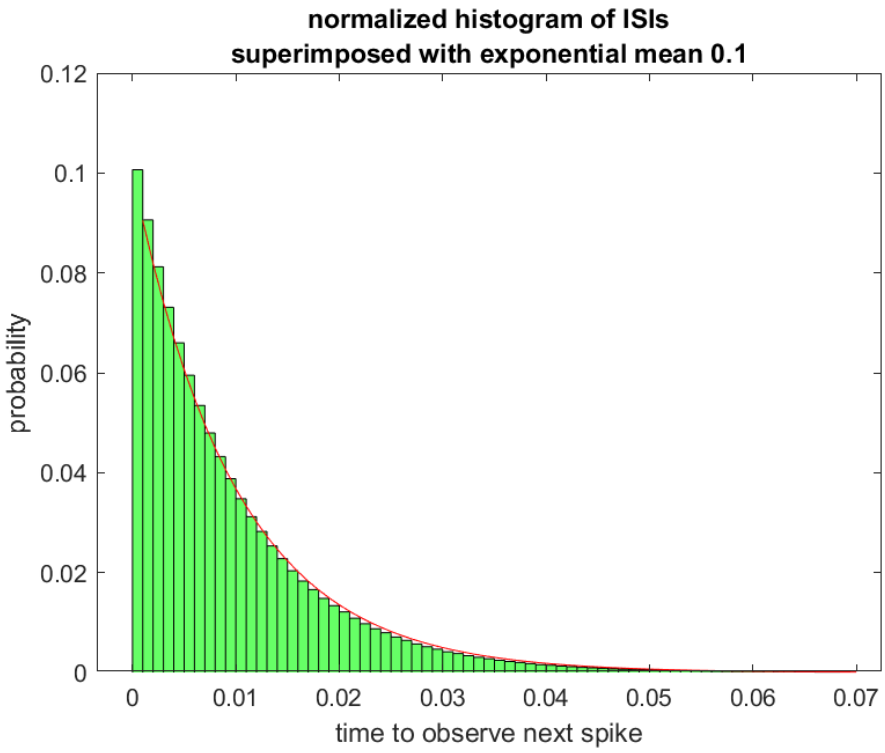
We will implement a function called ISIHistogramPlot to calculate differences of indexes of consecutive ones. To convert number of the generated matrix to time, we divide each element to 1000. Now we can plot histogram of the ISI. Also we plot an exponential PDF with $\lambda = 100 * \frac{1}{1000} = \frac{1}{10}$ and scale the indexes to reach to the exact corresponding time.

```
function spikeIntervals = ISIHistogramPlot(spikeMat)
[r, c] = size(spikeMat);
    idx = 1;
for i = 1 : r
    temp = 0;
    for j = 1 : c
        if(spikeMat(i, j) == 1 )
            spikeIntervals(idx) = temp + 1;
            idx = idx + 1;
            temp = 0;
        elseif(spikeMat(i, j) == 0 )
            temp = temp + 1;
        end
    end
end
figure;
histogram(spikeIntervals/1000, [0.000001:0.001:0.070],'Normalization',...
    'probability', 'FaceColor','g');
hold on
x = 0.001:0.001:0.07;
y = exppdf(x*1000,10);
plot(x, y,"r")
xlabel("time to observe next spike")
ylabel("probability")
title(["normalized histogram of ISIs",...
    "superimposed with exponential mean 0.1"])
end
```

By using the command

```
spikeIntervals = ISIHistogramPlot(spikeMat);
```

We will get the following figure:



4. A way to generate a renewal process spike train is to start with a Poisson spike train and delete all but every kth spike! This procedure is similar to integration over postsynaptic input with Poisson ISI distribution. Repeat steps a-c but using the above deleting spike procedure. Now look at the new plot, what distribution it is look like?

   We generate the spike train by the following function:

   (a) We will generate the spikes by means of the function below:

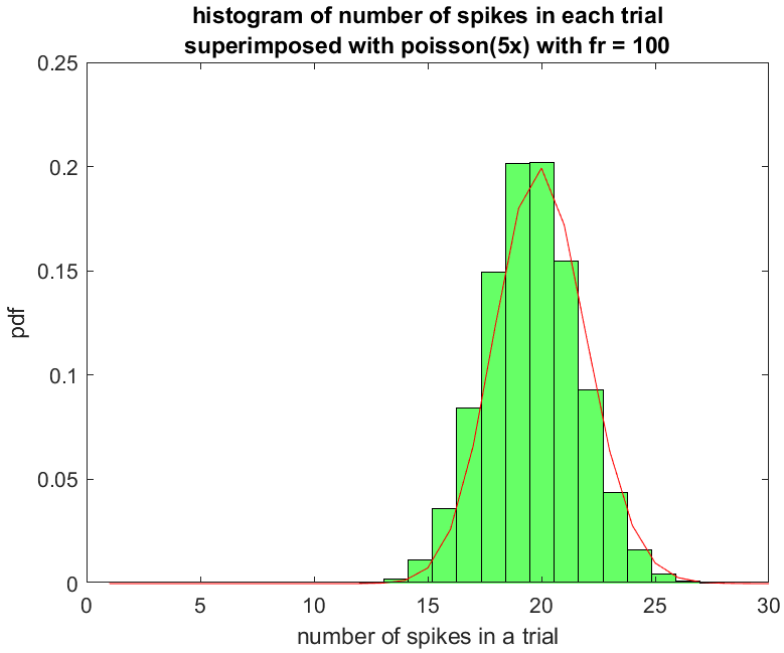
```
function [ spikeMat , tVec ] = erlangSpikeGen ( fr , dt, tSim , nTrials, k )
nBins = floor ( tSim / dt ) ;
spikeMat = rand ( nTrials , nBins ) < fr * dt ;
tVec = 0: dt : tSim - dt ;
[r, c] = size(spikeMat);
for i = 1 : r
    temp = 0;
    for j = 1 : c
        if(spikeMat(i, j) == 1)
            temp = temp + 1;
            if(temp ~= k)
                spikeMat(i, j) = 0;
            elseif(temp == k)
                temp = 0;
            end
        end
    end
end
end
```

   (b) By using the function:

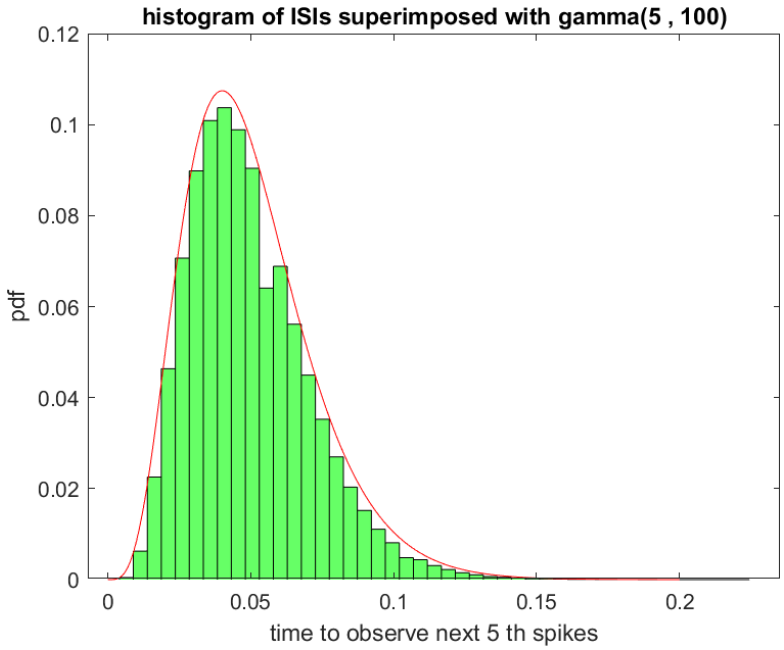
```
function numOfOnes = GammaHistogramPlot(spikeMat, k)
numOfOnes = sum(transpose(spikeMat(:, :)) == 1);
figure;
histogram(numOfOnes,15,'Normalization','probability', 'FaceColor','g');
```

```
hold on
x = 1:1:30;
for i = 1 : length(x)
   y(i) = k*exp(-100)*(100)^(k*x(i))/factorial(k*x(i));
end
plot(x, y,"r")
xlabel("number of spikes in a trial")
ylabel("pdf")
title(["histogram of number of spikes in each trial"...
    "superimposed with poisson(5x) with fr = 100"])
end
```

Histogram of Number of spikes in each trial generates a $P(N(t) = 5n) = \frac{e^{-\lambda t}(\lambda t)^{5n}}{5n!}$:



(c) And the following figure is the histogram of ISIs which can be superimposed with a Gamma distribution:



5. Now use the generated data and find out what is the C_v (Coefficient of Variation) for this spike trains? Compare it with Poisson process?

Answer:

By using the commands

```
[ spikeMat , tVec ] = erlangSpikeGen ( 100 , 1/1000, 1 , 100000, 5 );
spikeIntervals = ISIGammaHistogramPlot(spikeMat, 5);
gamma5 = sqrt(var(spikeIntervals))/mean(spikeIntervals)
[ spikeMat , tVec ] = erlangSpikeGen ( 100 , 1/1000, 1 , 100000, 9 );
spikeIntervals = ISIGammaHistogramPlot(spikeMat, 9);
gamma9 = sqrt(var(spikeIntervals))/mean(spikeIntervals)
[ spikeMat , tVec ] = erlangSpikeGen ( 100 , 1/1000, 1 , 100000, 20 );
spikeIntervals = ISIGammaHistogramPlot(spikeMat, 20);
gamma20 = sqrt(var(spikeIntervals))/mean(spikeIntervals)
```

We compute C_v for three different values of K(5, 9, 20):

```
gamma5 =

    0.4236


gamma9 =

    0.3159


gamma20 =

    0.2118
```

In Poisson distribution C_v equals $\frac{\sqrt{\lambda}}{\lambda} = \frac{1}{\sqrt{\lambda}}$ for all different values of λ. Here C_v equals $\frac{1}{\sqrt{100}} = 0.1$.
In exponential distribution C_v equals $\frac{\sqrt{\frac{1}{\lambda^2}}}{\frac{1}{\lambda}} = 1$.
Gamma distribution which has been created by calculating histogram values of ISI in second way of generating spikes, will have lower $C_v s$ while K increases.
We can also calculate $C_v s$ for second way of generating spikes:

```
secondSpikeGenerating5 =

    0.0977


secondSpikeGenerating9 =

    0.1024


secondSpikeGenerating20 =

    0.1249
```

6. Theoretically prove the C_v for the above distribution is equal to $\frac{1}{K^{0.5}}$

Answer:

Let X be a gamma random variable with parameters (r, λ). To find $E(X)$ and $\mathrm{Var}(X)$, note that for all $n \geq 0$,

$$E(X^n) = \int_0^\infty x^n \frac{\lambda e^{-\lambda x}(\lambda x)^{r-1}}{\Gamma(r)} dx = \frac{\lambda^r}{\Gamma(r)} \int_0^\infty x^{n+r-1} e^{-\lambda x}\, dx.$$

Let $t = \lambda x$; then $dt = \lambda\, dx$, so

$$E(X^n) = \frac{\lambda^r}{\Gamma(r)} \int_0^\infty \frac{t^{n+r-1}}{\lambda^{n+r-1}} e^{-t} \frac{1}{\lambda} dt = \frac{\lambda^r}{\Gamma(r)\lambda^{n+r}} \int_0^\infty t^{n+r-1} e^{-t}\, dt = \frac{\Gamma(n+r)}{\Gamma(r)\lambda^n}.$$

For $n = 1$, this gives

$$E(X) = \frac{\Gamma(r+1)}{\Gamma(r)\lambda} = \frac{r\Gamma(r)}{\lambda\Gamma(r)} = \frac{r}{\lambda}.$$

For $n = 2$,

$$E(X^2) = \frac{\Gamma(r+2)}{\Gamma(r)\lambda^2} = \frac{(r+1)\Gamma(r+1)}{\lambda^2\Gamma(r)} = \frac{(r+1)r\Gamma(r)}{\lambda^2\Gamma(r)} = \frac{r^2+r}{\lambda^2}.$$

Thus

$$\mathrm{Var}(X) = \frac{r^2+r}{\lambda^2} - \left(\frac{r}{\lambda}\right)^2 = \frac{r}{\lambda^2}.$$

We have shown that

For a gamma random variable with parameters r and λ,

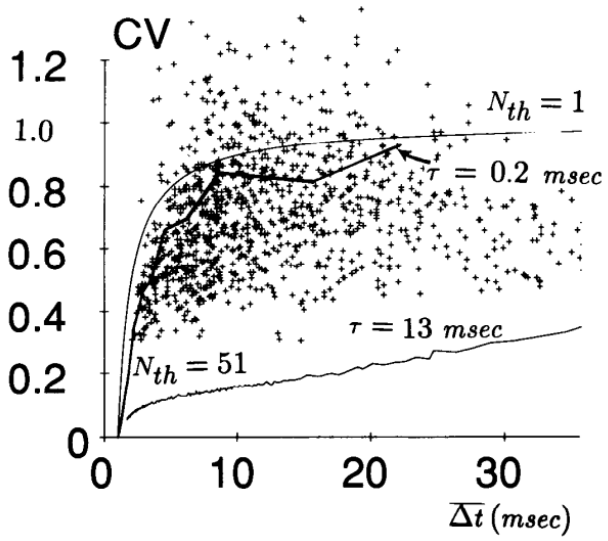
$$E(X) = \frac{r}{\lambda}, \qquad \mathrm{Var}(X) = \frac{r}{\lambda^2}, \qquad \sigma_X = \frac{\sqrt{r}}{\lambda}.$$

So C_v is equal to

$$\frac{\frac{\sqrt{K}}{\lambda}}{\frac{K}{\lambda}} = \frac{1}{\sqrt{K}}$$

7. Compare variability of simulated spike trains with real data-set in [Softky & Koch, 1993]

Answer:

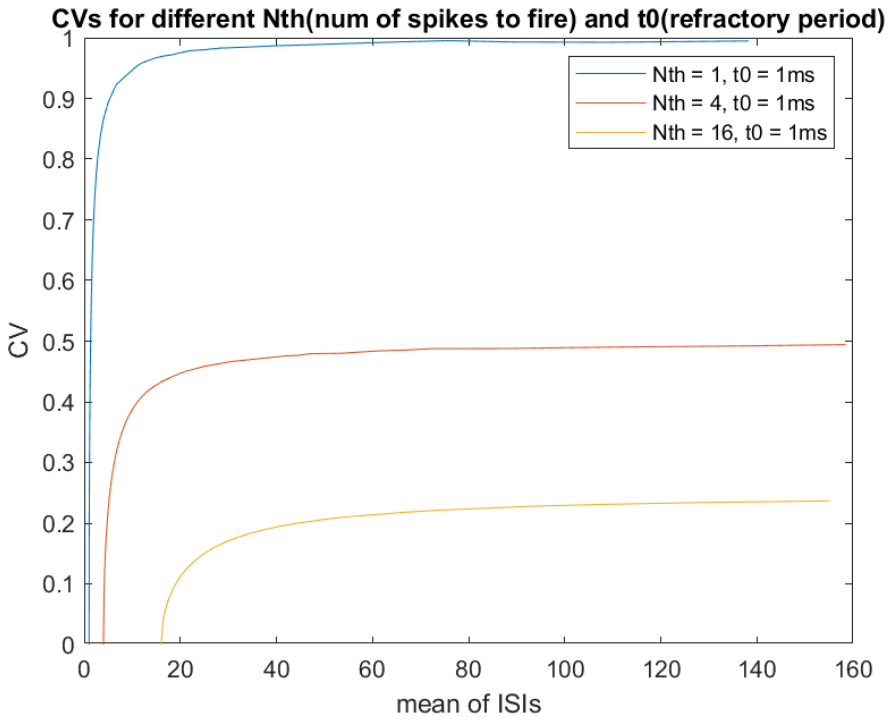


Comparison of macaque cortical neuron variability and leakyintegrator model: Scattered crosses show C_v, for macaque cortical neurons. The lower curve shows the simulated leaky-integrator model with parameter values in the accepted range ($N_{th} = 51$, T = 13 msec, and t0 = 1.0 msec). The middle curve shows the same simulation, still with to = 1.0 but 7 = 0.2 msec, a much shorter decay time than usually accepted for pyramical cells. The upper curve shows the theoretical upper bound on C, for a pure Poisson spike train with "dead-time" to = 1.0 msec. The observed C_v of macaque cortical cells lies much closer to the maximum possible than it does to the C_v predicted by a neuron model that performs significant temporal integration.
It is evident that in order to achieve high variability (i.e., $C_v > 0.7$) at high rates (which are comparable to those in our faster cells), τ has to be a fraction of a millisecond, or N_{th} must be only 1 or 2! In fact, the model best fitting the monkey data is that for a neuron that performs no temporal integration, having $N_{th} = 1$.

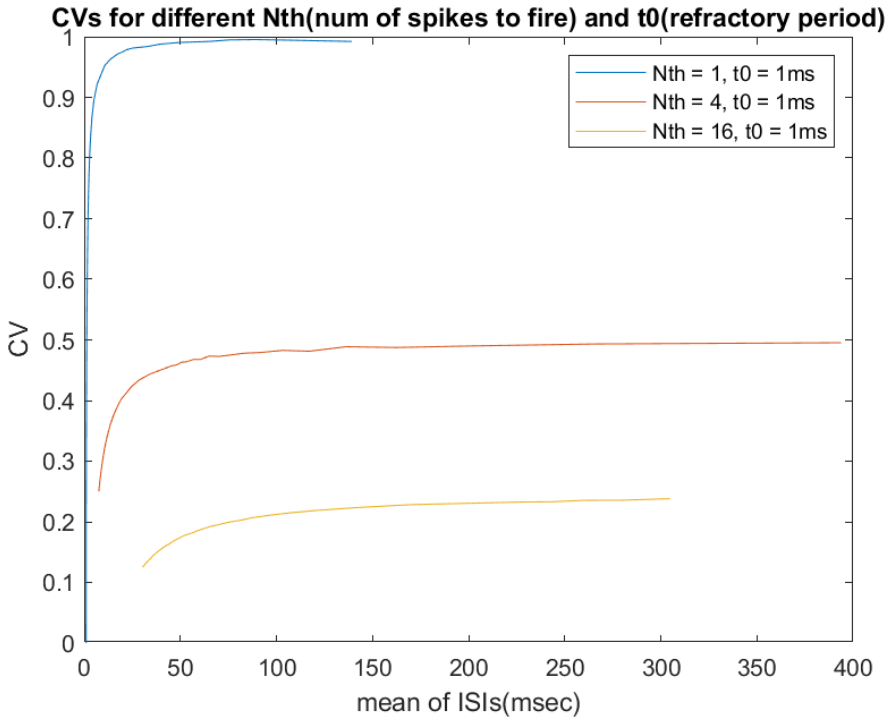
8. Contradicting with generated spike trains, real neural data contain less spikes within small intervals. The biological support for this claim is that each neuron has a refractory period which limit neuron to generate spikes within this period, so there is an upper limit for firing frequency in small intervals.Similar to equation [13] of paper, consider a refractory period for your spike trains and generate a plot similar to figure 6 of paper comparing C_v across different firing rates and different refractory periods.

Answer:

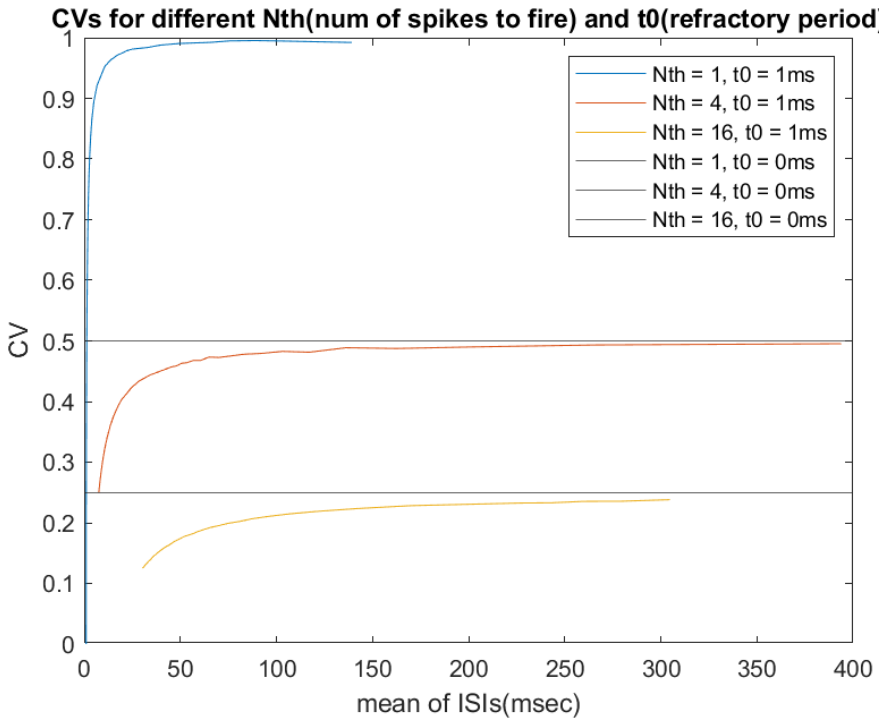
When resolution is equal to 1ms, the result is as below(means of ISIs are shown per millisecond):



By changing the resolution we obtain the figure:



To understand better:


CVs for different Nth(num of spikes to fire) and t0(refractory period)
Nth = 1, t0 = 1ms
Nth = 4, t0 = 1ms
Nth = 16, t0 = 1ms
Nth = 1, t0 = 0ms
Nth = 4, t0 = 0ms
Nth = 16, t0 = 0ms
CV
mean of ISIs(msec)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
50
100
150
200
250
300
350
400

2 Leaky Integrate and Fire Neuron

It is well known that depolarization do not persist forever, but that perturbations of membrane voltage tend to decay toward the resting potential. In this section we are going to implement a more realistic model of neuron, Leaky Integrate and fire (LIF) neuron, which consider the leakage of postsynaptic inputs. The leaky integrate-and-fire (LIF) neuron is probably one of the simplest spiking neuron models, but it is still very popular due to the ease with which it can be analyzed and simulated. In its simplest form, a neuron is modeled as a "leaky integrator" of its input I(t):

$$\tau_m \frac{dv}{dt} = -v(t) + RI(t)$$

where v(t) represents the membrane potential at time t, τ_m is the membrane time constant and R is the membrane resistance.

1. With stimulation by a constant input current of 20 mV, simulate the time-course of the membrane potential for 100ms. Take resting potential vr = 0 mV, and threshold voltage v_{th} = 15 mV.

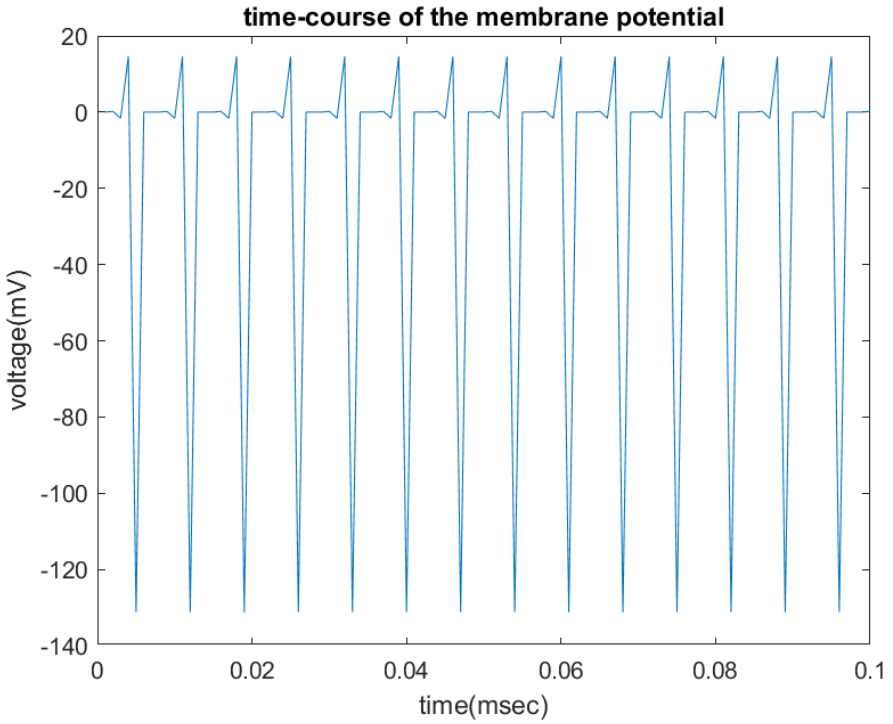
Answer:

By using the function below and setting parameters of LIF model:

```
function [rec_v, rec_sp] = run_LIF( Iinj, stop)
  % Simulate the LIF dynamics with external input current
  % Set parameters
  V_th = 15
  V_reset = 0
  tau_m = 10
  g_L = 1000
  V_init = 0
  E_L = 0
  dt = 0.1
  range_t = 100
  Lt = range_t
  tref = 0.1
  % Initialize voltage
  v = zeros(Lt, 1);
  v(1) = V_init
  % Set current time course
  Iinj = Iinj * ones(Lt, 1)
  % If current pulse, set beginning and end to 0
  if stop == 1
    Iinj(1:floor(length(Iinj) / 2) - 1000) = 0;
    Iinj(floor(length(Iinj) / 2) + 1000:end) = 0;
  end
  % Loop over time
  rec_spikes = []; % record spike times
  tr = 0; % the count for refractory duration
  for it = 1:Lt - 1
      it
    if tr > 0 % check if in refractory period
      v(it) = V_reset; % set voltage to reset
      tr = tr - 1; % reduce running counter of refractory period
    elseif v(it) >= V_th % if voltage over threshold
      rec_spikes(end+1) = it; % record spike event
      v(it) = V_reset; % reset voltage
      tr = tref / dt; % set refractory time
    end
```

```
    % Calculate the increment of the membrane potential
    dv = (-g_L * (v(it) - E_L) + Iinj(it)) / tau_m * dt;
    % Update the membrane potential
    v(it + 1) = v(it) + dv;
  end
  % Get spike times in ms
  rec_spikes = rec_spikes * dt;
  rec_v = v;
  rec_sp = rec_spikes;
end
```

The result will be as the following figure:



2. Write a mathematical equation explaining mean firing rate of neuron at a constant input current I, with considering a refractory period of Δtr .

Answer:

Usually, we don't work with the spikes themselves, and the occurrence rate of spikes is more valuable for us. As a result, we should be able to Estimate the rate from a given spike sequence. The process of occurrence of spikes can be modeled as follows:

$$\rho(t) = \sum_{i=1}^{n} \delta(t - t_i)$$

One way to estimate the rate is to divide time into small intervals And in each interval, the number of spikes Divide by the length of that interval. In this way, we do not get a very continuous and smooth graph.
The second way is to use a moving window and estimate the rate like this:

$$r_{approx}(t) = \int_{-\infty}^{\infty} w(\tau)\rho(t-\tau)d\tau$$

We just need to take the window in such a way that the area under it is equal to 1.
Now a window looks like this:

$$w(t) = \begin{cases} \frac{1}{\Delta t} & \frac{-\Delta t}{2} \leq t \leq \frac{\Delta t}{2} \\ 0 & \text{otherwise} \end{cases}$$

If we choose a smoother window, for example Gaussian window, we can estimate a smoother rate.

$$w(t) = \frac{1}{\sqrt{2\pi}\sigma_w} e^{\frac{-t^2}{2\sigma_w^2}}$$

3. Repeat section a. with stimulating neuron by a time-varying input current I(t). For a general time-varying input current I(t), the solution of LIF equation, with the initial condition v(t0) = vr, is given by:

$$v(t) = v_r \exp(-\frac{t-t_0}{\tau_m}) + \frac{R}{\tau_m}\int_0^{t-t_0} \exp(-\frac{s}{\tau_m})I(t-s)ds$$

To generate a realistic I(t), first we define K spike train with Poisson distribution and then convolve them in an EPSC kernel, similar to what Softky and Koch did in their paper. As long as the membrane potential is well below the synapse's reversal potential, we can approximate the synaptic current by:

$$I_s(\mathrm{t}) \propto \mathrm{t}\, e^{(-t/t_{peak})}$$

$$I(t) = \sum_i \delta(t-t_i) . I_s(\mathrm{t})$$

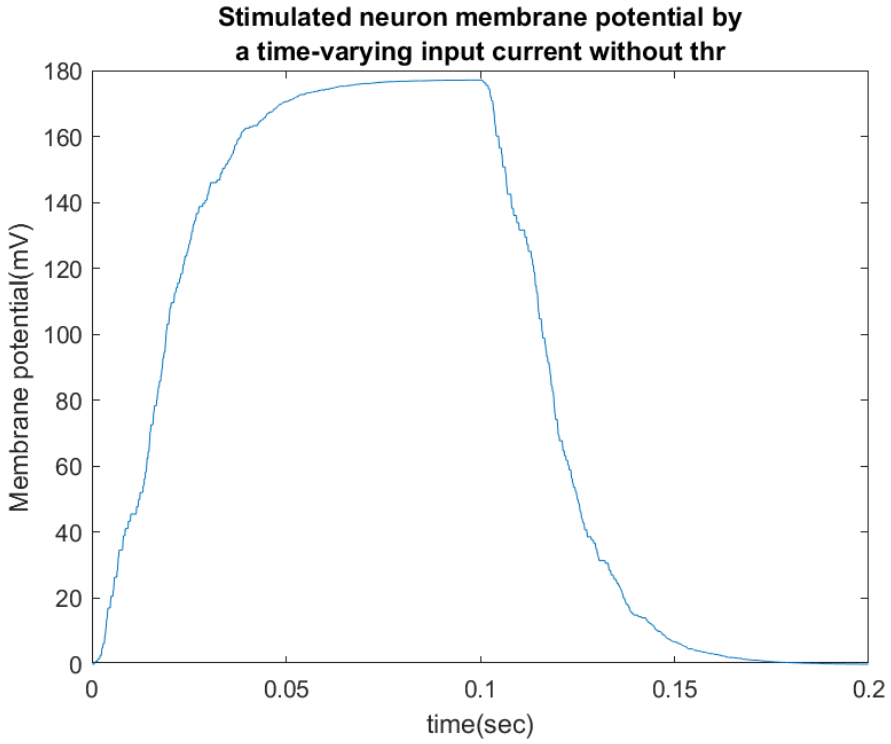
Answer:

By using the following MATLAB code(consider that here vr is equal to zero):

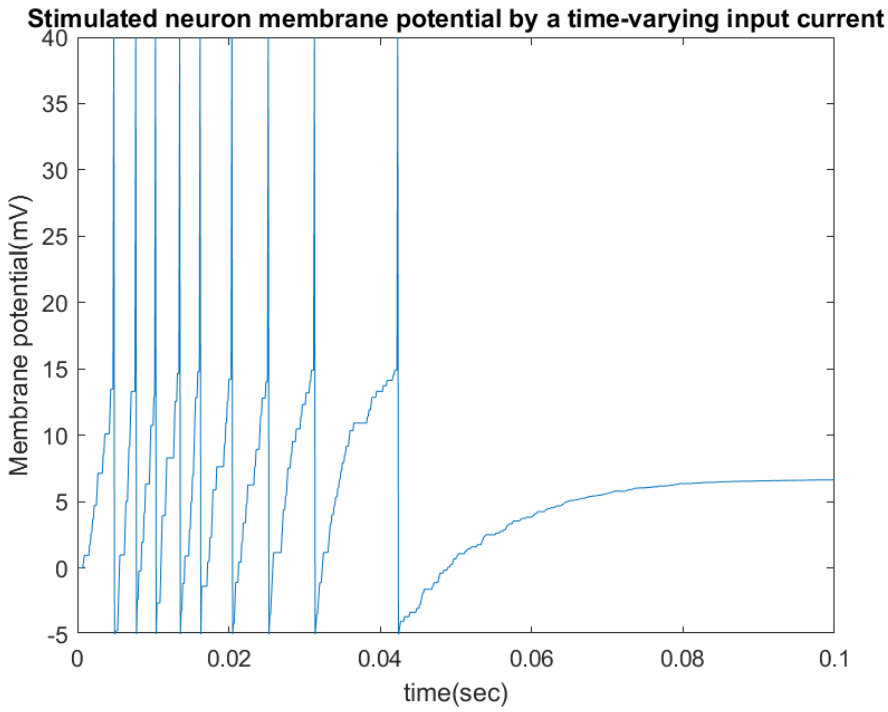
```
clc;clear all; close all;
tau_m = 30;
[ spikeMat , tVec ] = poissonSpikeGen ( 3000 , 1/10000, 0.1 , 1 );
for i = 1 : length(spikeMat)
    realisticCurrent(i) = 4.5*spikeMat(i) * i/10000 *exp(-i/100);
end
x = 1 : 1000;
y = exp(-x/10000/tau_m);
    voltages = 4000/tau_m*conv(realisticCurrent , y);
```

```
figure
plot(voltages)
title("volt")
thr = 15;
for i = 1 : length(voltages)
   if(voltages(i)>=thr)
       voltages(i) = 40;
       voltages(i + 1) = -5;
       voltages(i + 2 : length(voltages)) = voltages(i + 2 : length(voltages))...
           - 20*ones(1, length(i + 2 : length(voltages)));
      i = i + 1;
   end
end
figure
plot(0.0001 : 0.0001 : 0.1,voltages(1:1000))
title("Stimulated neuron membrane potential by a time-varying input current")
ylabel("Membrane potential(mV)")
xlabel("time(sec)")
```

We obtain the results: Without considering threshold:



With considering threshold:



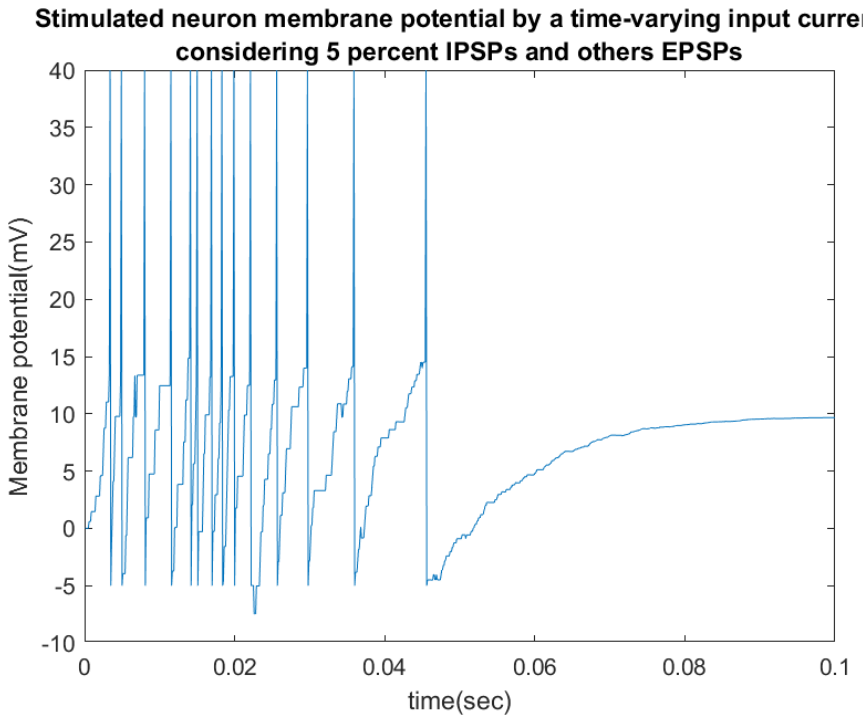
4. In this section we are going to include IPSPs effect in our simulations. To do so you just need to select a percentage of synaptic inputs to assign negative kernels. Repeat part c. for different percentages of inhibitory inputs and briefly explain (based on simulations!) how C_v depends on inhibition percentage of synaptic inputs and other simulation parameters.

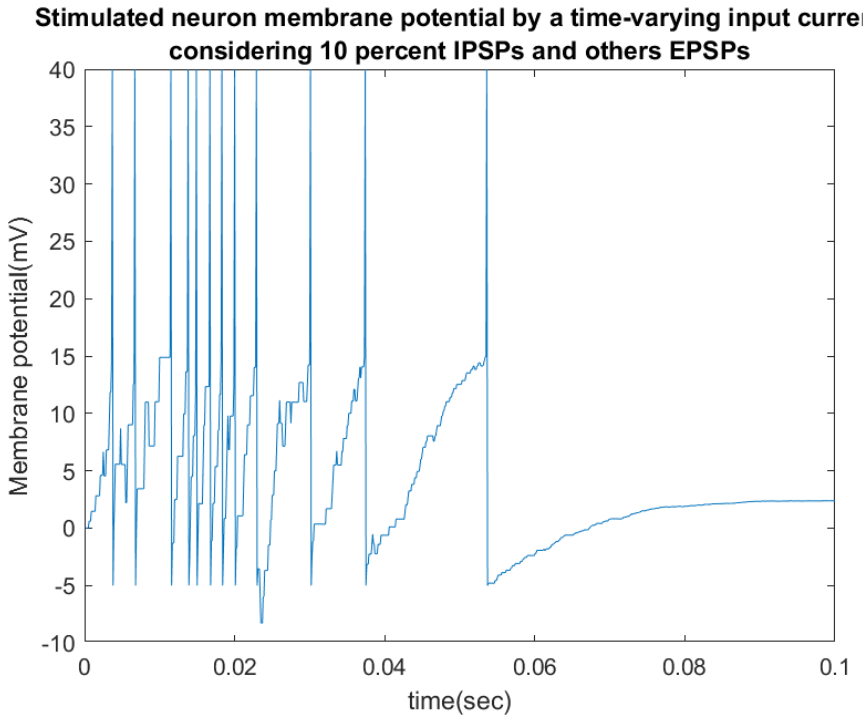
Answer:

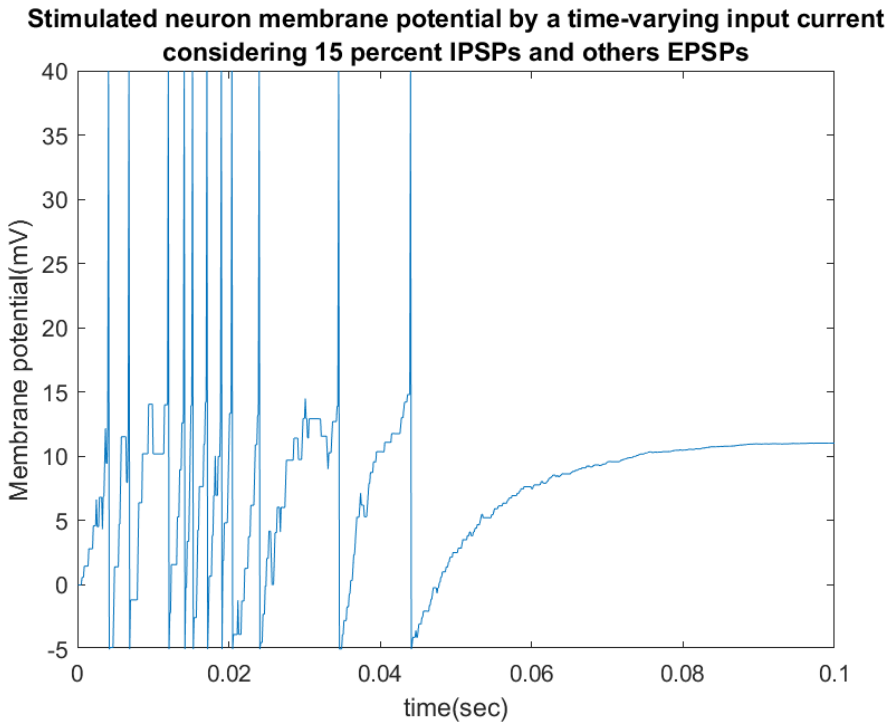
We add the following part to the code of previous part:

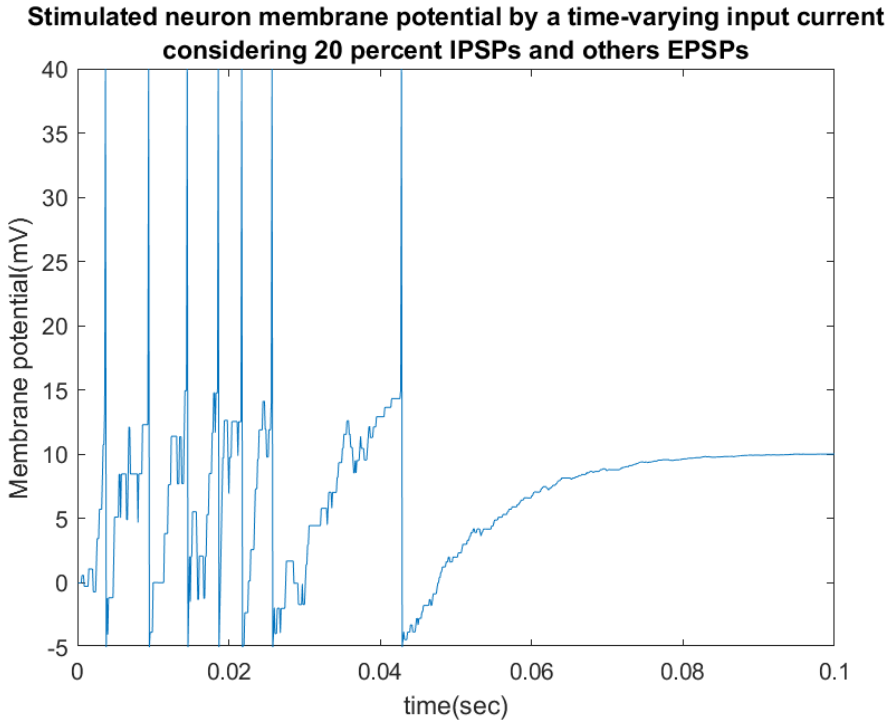
```
A = ones(1, 1000);
r = randperm(1000, 300);
A(r) = -1;
newRealisticCurrent = A.*realisticCurrent;
```

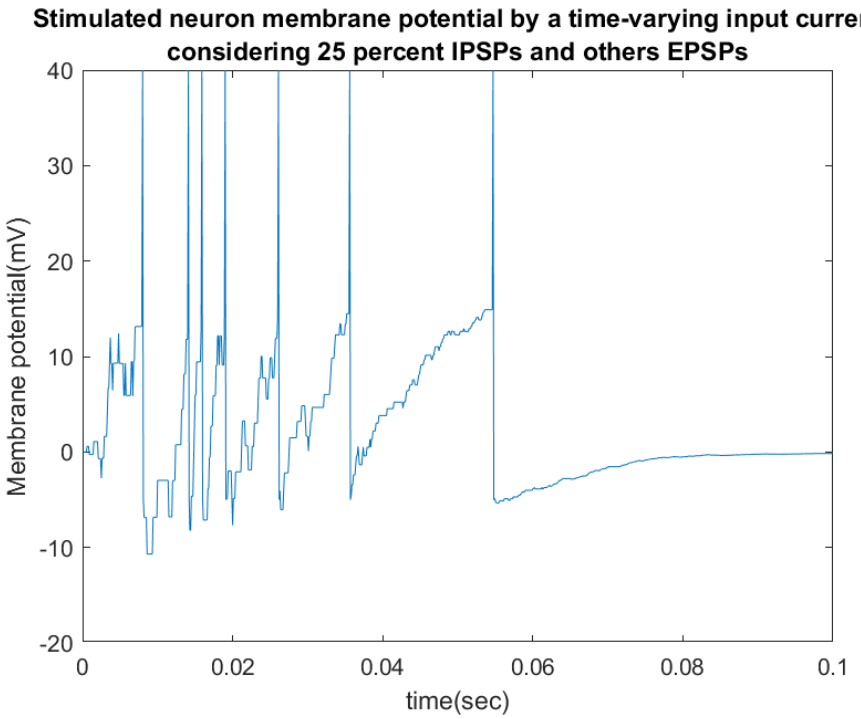
Results:

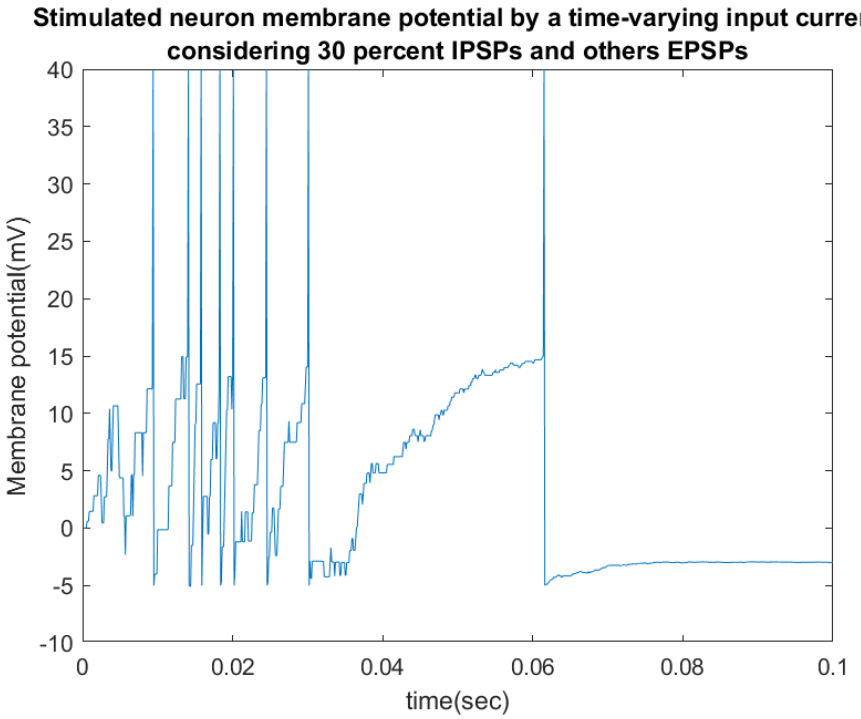

Stimulated neuron membrane potential by a time-varying input current
considering 5 percent IPSPs and others EPSPs
Membrane potential(mV)
time(sec)
40
35
30
25
20
15
10
5
0
-5
-10
0
0.02
0.04
0.06
0.08
0.1

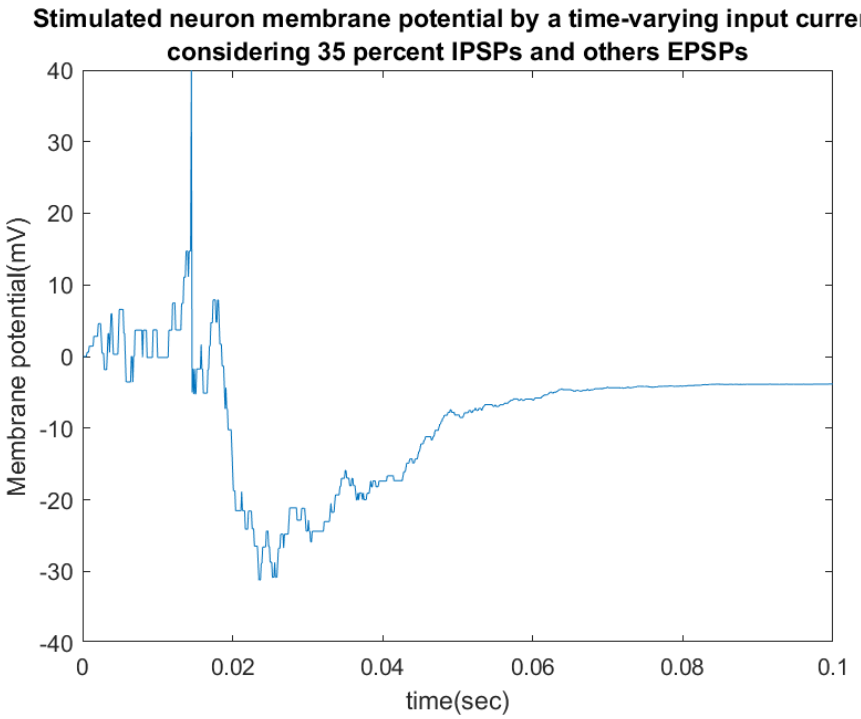

Stimulated neuron membrane potential by a time-varying input current
considering 10 percent IPSPs and others EPSPs
Membrane potential(mV)
time(sec)
40
35
30
25
20
15
10
5
0
-5
-10
0
0.02
0.04
0.06
0.08
0.1


Stimulated neuron membrane potential by a time-varying input current
considering 15 percent IPSPs and others EPSPs
Membrane potential(mV)
time(sec)
40
35
30
25
20
15
10
5
0
-5
0
0.02
0.04
0.06
0.08
0.1


Stimulated neuron membrane potential by a time-varying input current
considering 20 percent IPSPs and others EPSPs
Membrane potential(mV)
time(sec)
40
35
30
25
20
15
10
5
0
-5
0
0.02
0.04
0.06
0.08
0.1


Stimulated neuron membrane potential by a time-varying input current
considering 25 percent IPSPs and others EPSPs
Membrane potential(mV)
time(sec)
40
30
20
10
0
-10
-20
0
0.02
0.04
0.06
0.08
0.1


Stimulated neuron membrane potential by a time-varying input current
considering 30 percent IPSPs and others EPSPs
Membrane potential(mV)
time(sec)
40
35
30
25
20
15
10
5
0
-5
-10
0
0.02
0.04
0.06
0.08
0.1



Now we calculate ISIs for 1000 trials and then calculate CVs:

```
p =
    50

meanOfISIs =
   41.3658

CV =
    1.0672

p =
   100

meanOfISIs =
   45.3347

CV =
    1.0217

p =
   150

meanOfISIs =
   50.3131

CV =
    0.9748

p =
   200

meanOfISIs =
   56.6865

CV =
    0.9276

p =
   250

meanOfISIs =
   65.1286

CV =
    0.8909

p =
   300

meanOfISIs =
   76.4319

CV =
    0.8677
```

We can see that by increasing percentage of IPSPs, C_v decreases.
Now we decrease τ from 30ms to 15ms and observe that by decreasing this parameter, C_vs will increase:

```
p =
    50
meanOfISIs =
   24.7418
CV =
    1.2446

p =
   100
meanOfISIs =
   27.3312
CV =
    1.2176
```

```
p =
   150
meanOfISIs =
   30.5239
CV =
    1.1859

p =
   200
meanOfISIs =
   34.5796
CV =
    1.1508
```

```
p =
   250
meanOfISIs =
   39.9285
CV =
    1.1199

p =
   300
meanOfISIs =
   47.0831
CV =
    1.0998
```

Now we increase vth from 15mV to 30mV and observe that by increasing this parameter, C_vs will decrease:

```
p =
    50
meanOfISIs =
   46.7107
CV =
    0.9454

p =
   100
meanOfISIs =
   51.7478
CV =
    0.9130
```

```
p =
   150
meanOfISIs =
   57.9324
CV =
    0.8683

p =
   200
meanOfISIs =
   66.0836
CV =
    0.8332
```

```
p =
   250
meanOfISIs =
   77.0162
CV =
    0.8016

p =
   300
meanOfISIs =
   91.8910
CV =
    0.7770
```

5. Assume the neuron is doing coincidence detection of it excitatory inputs which have a Poisson distribution (assume neuron requires N out of M inputs to be active in a short D ms time window). What would be the C_v of the output neuron? How will it depend on the N/M and D? (you can use simulation to plot dependence, only consider excitatory inputs for this part).

Answer:

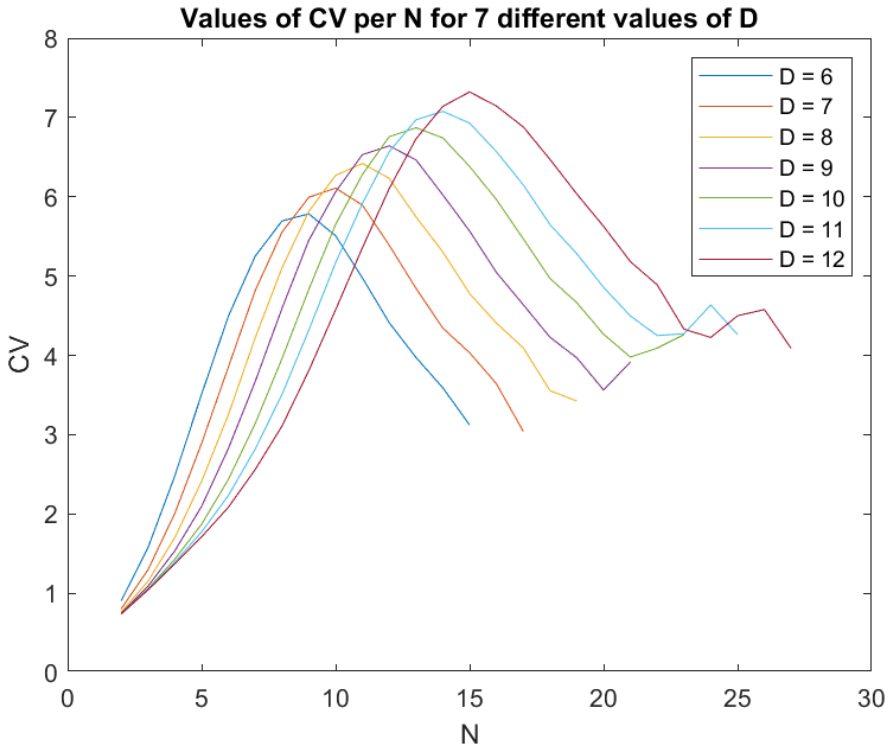
Function to detect coincidences:

```
function resultSpikes = coincidenceDetector(spikeMat, dt, D, N)
[r, c] = size(spikeMat);
resultSpikes = zeros(r, c);
D = D/1000/dt;
for i = 1 : r
    for j = 1 : c - D
        temp = 0;
        for k = 0 : D - 1
            if(spikeMat(i, j + k) == 1)
                temp = temp + 1;
            end
        end
        if(temp >= N)
            resultSpikes(i, j + D) = 1;
        end

    end
end
for i = 1 : r
    for j = 1 : D
        temp = 0;
        for k = 1 : j - 1
            if(spikeMat(i, k) == 1)
                temp = temp + 1;
            end
        end
        if(temp >= N)
            resultSpikes(i, j) = 1;
        end

    end
end
end
```

For this part we calculate CV for D = [6: 12]ms × N = [2: 8 : 29 - 2.1*abs(D - 13)]. So we calculate 56 number of CVs:

```
CV =

  Columns 1 through 17

    0.9040    1.5752    2.4795    3.5104    4.4958    5.2515    5.6954    5.7847    5.5074    4.9754    4.4116    3.9726    3.5907    3.1213         0         0         0
    0.8038    1.2958    2.0074    2.8912    3.8546    4.8181    5.5549    5.9933    6.1089    5.8940    5.3878    4.8449    4.3432    4.0326    3.6422    3.0418         0
    0.7617    1.1522    1.7023    2.4125    3.2537    4.2219    5.1044    5.8167    6.2692    6.4196    6.2319    5.7460    5.3037    4.7778    4.4073    4.0948    3.5545
    0.7471    1.0889    1.5349    2.0961    2.8270    3.6701    4.5942    5.4516    6.0634    6.5319    6.6424    6.4640    6.0236    5.5668    5.0460    4.6392    4.2280
    0.7406    1.0563    1.4305    1.8707    2.4360    3.1414    3.9597    4.8320    5.6506    6.2814    6.7580    6.8709    6.7419    6.3810    5.9650    5.4706    4.9697
    0.7385    1.0462    1.3913    1.7730    2.2335    2.8199    3.5071    4.3183    5.1684    5.9170    6.5675    6.9710    7.0773    6.9292    6.5690    6.1503    5.6484
    0.7375    1.0417    1.3702    1.7114    2.0849    2.5610    3.1086    3.8082    4.5754    5.3569    6.1036    6.7310    7.1406    7.3220    7.1442    6.8819    6.4710

  Columns 18 through 26

         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0
    3.4242         0         0         0         0         0         0         0         0
    3.9683    3.5623    3.9163         0         0         0         0         0         0
    4.6633    4.2658    3.9783    4.0916    4.2616         0         0         0         0
    5.2812    4.8569    4.4964    4.2496    4.2730    4.6359    4.2614         0         0
    6.0348    5.6262    5.1817    4.8889    4.3295    4.2259    4.4991    4.5778    4.0865
```

Totally we can see that for small N:
CV decreases while D increases.
For large N:
CV increases while D increases. For all values of D:
At first CV increases while N increases and then CV decreases.

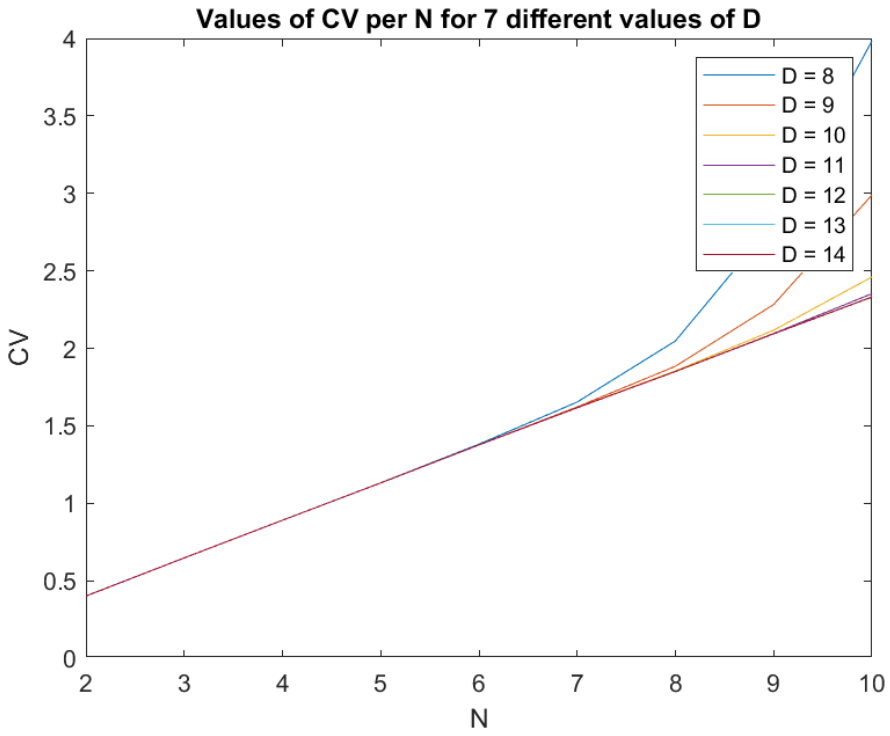


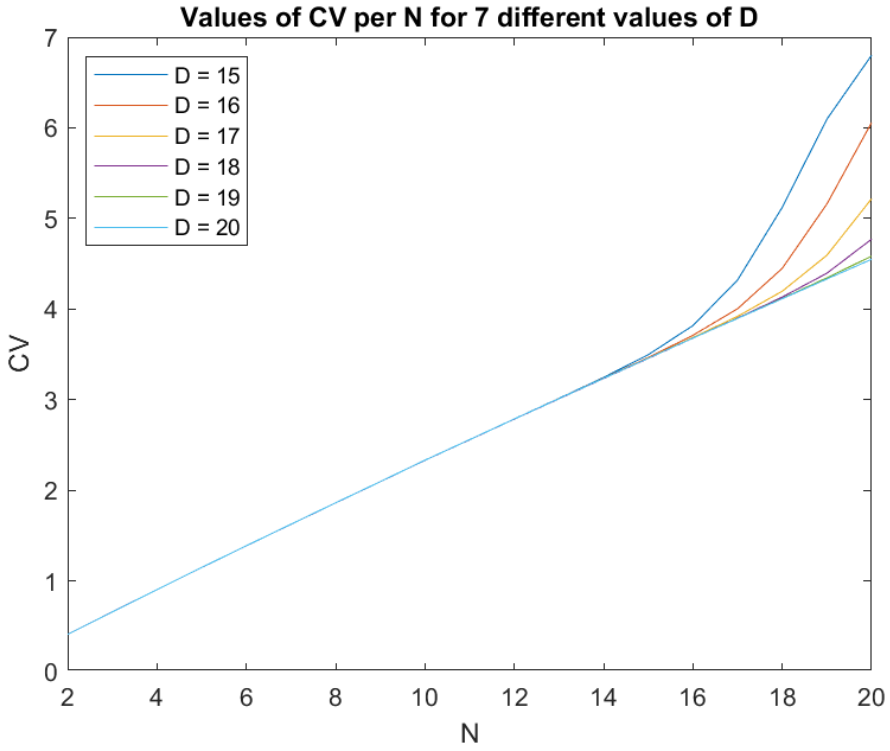
Note that M is constant. So N/M is proportional to N. So we plotted the lines per N(not per N/M).

Now by considering a small refractory period we may observe that CV doesn't change while D is increasing:

```
CV =

    0.4008    0.6470    0.8902    1.1317    1.3812    1.6534    2.0474    2.8108    3.9837
    0.4018    0.6474    0.8903    1.1311    1.3770    1.6217    1.8843    2.2829    2.9892
    0.4008    0.6470    0.8902    1.1313    1.3769    1.6173    1.8543    2.1171    2.4605
    0.4018    0.6474    0.8903    1.1311    1.3764    1.6164    1.8503    2.0960    2.3527
    0.4008    0.6470    0.8902    1.1313    1.3768    1.6170    1.8505    2.0939    2.3328
    0.4008    0.6470    0.8902    1.1313    1.3768    1.6170    1.8505    2.0936    2.3309
    0.4008    0.6470    0.8902    1.1313    1.3768    1.6170    1.8505    2.0936    2.3308
    0.4008    0.6470    0.8902    1.1313    1.3768    1.6170    1.8505    2.0936    2.3308
    0.4008    0.6470    0.8902    1.1313    1.3768    1.6170    1.8505    2.0936    2.3308
    0.4008    0.6470    0.8902    1.1313    1.3768    1.6170    1.8505    2.0936    2.3308
    0.4018    0.6474    0.8903    1.1311    1.3764    1.6164    1.8497    2.0926    2.3297
    0.4008    0.6470    0.8902    1.1313    1.3768    1.6170    1.8505    2.0936    2.3308
    0.4008    0.6470    0.8902    1.1313    1.3768    1.6170    1.8505    2.0936    2.3308
```

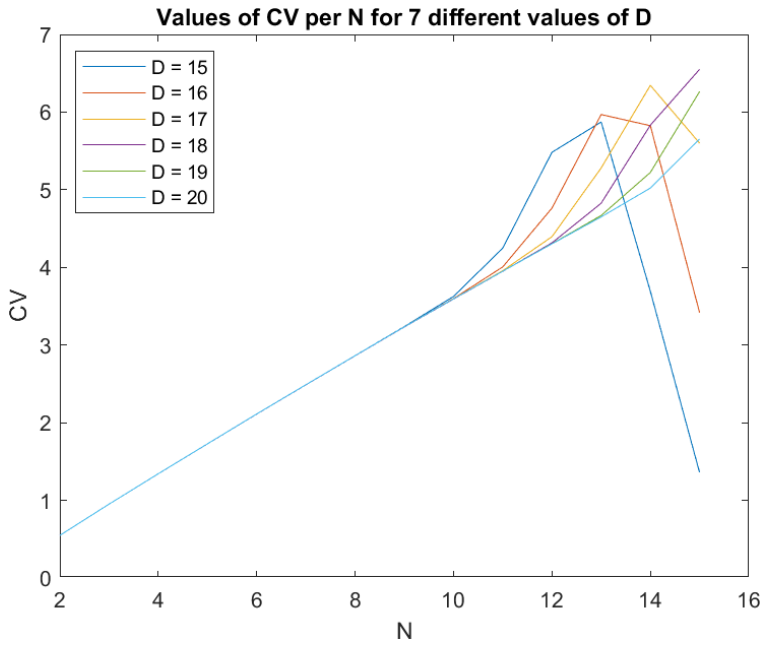
But for a special D, CV increases while N increases(for small N):





We can observe that for some small values of N, CVs for different D values are equal to each other. Now we consider a bigger refractory period:

```
CV =

    0.5485    0.9516    1.3428    1.7283    2.1124    2.4891    2.8636    3.2377    3.6261    4.2481    5.4820    5.8730    3.6918    1.3599
    0.5485    0.9516    1.3428    1.7283    2.1124    2.4891    2.8636    3.2350    3.5989    4.0059    4.7639    5.9686    5.8250    3.4190
    0.5485    0.9516    1.3428    1.7283    2.1124    2.4891    2.8636    3.2345    3.5955    3.9576    4.3954    5.2804    6.3443    5.5995
    0.5491    0.9516    1.3424    1.7276    2.1114    2.4879    2.8621    3.2328    3.5926    3.9497    4.3155    4.8260    5.8320    6.5503
    0.5485    0.9516    1.3428    1.7283    2.1124    2.4891    2.8636    3.2345    3.5945    3.9508    4.3047    4.6682    5.2229    6.2634
    0.5485    0.9516    1.3428    1.7283    2.1124    2.4891    2.8636    3.2345    3.5946    3.9507    4.3029    4.6501    5.0218    5.6547
```



6. What will happen if the neuron is coincidence detector but we also add inhibitory neurons. Here the neuron will fire if number of excitatory pulses received minus number of inhibitory pulses (Nnet=Nx-Ni) is bigger than a threshold in D ms time window. What would be the Cv of the output neuron? How will it depend on the Nnet and D? (you can use simulations to plot dependence).

Answer:

We just add two parts to our code:

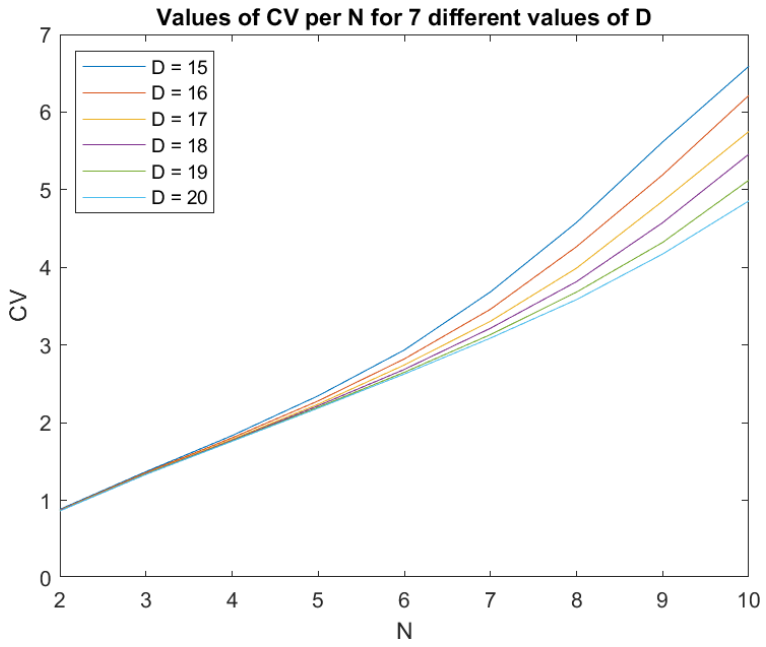
```
4 -    p = 100;
5 -    A = ones(1, 1000);
6 -    r = randperm(1000, p);
7 -    A(r) = -1;
8 -    spikeMat = A.*spikeMat;
```

```
elseif(spikeMat(i, j + k) == -1)
    temp = temp - 1;
```

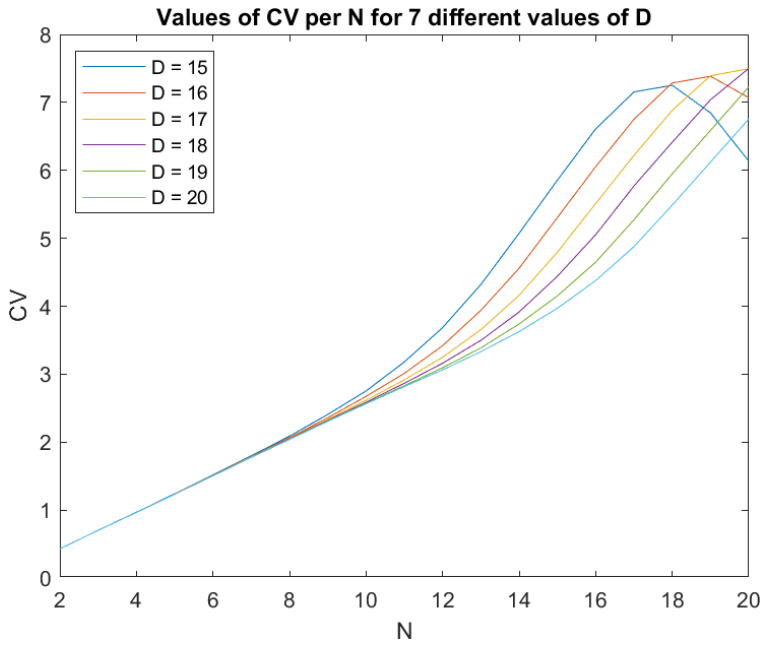
Result:

For 10% IPSP:

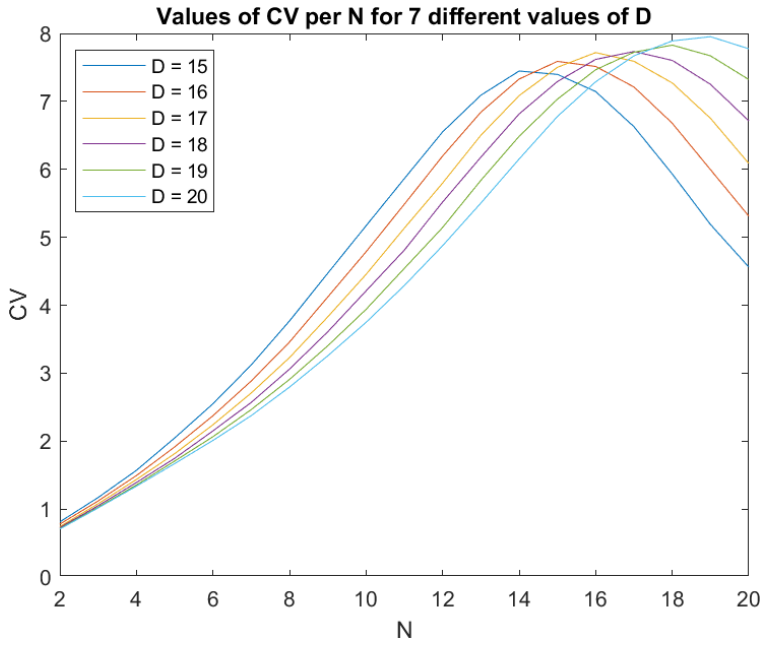
```
CV =

    0.8844    1.3700    1.8341    2.3465    2.9364    3.6845    4.5785    5.6155    6.5898
    0.8756    1.3585    1.8057    2.2805    2.8209    3.4611    4.2674    5.1922    6.2148
    0.8708    1.3476    1.7860    2.2388    2.7432    3.3067    3.9904    4.8529    5.7502
    0.8670    1.3416    1.7730    2.2187    2.6866    3.2162    3.8168    4.5764    5.4574
    0.8634    1.3380    1.7652    2.1995    2.6496    3.1340    3.6821    4.3222    5.1232
    0.8631    1.3341    1.7604    2.1877    2.6275    3.0890    3.5854    4.1724    4.8561
```



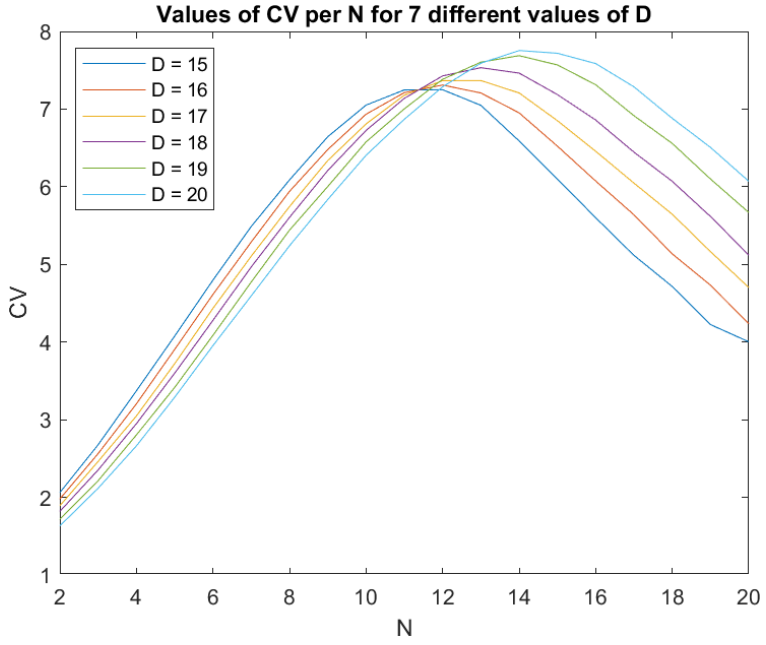
For smaller refractory period:



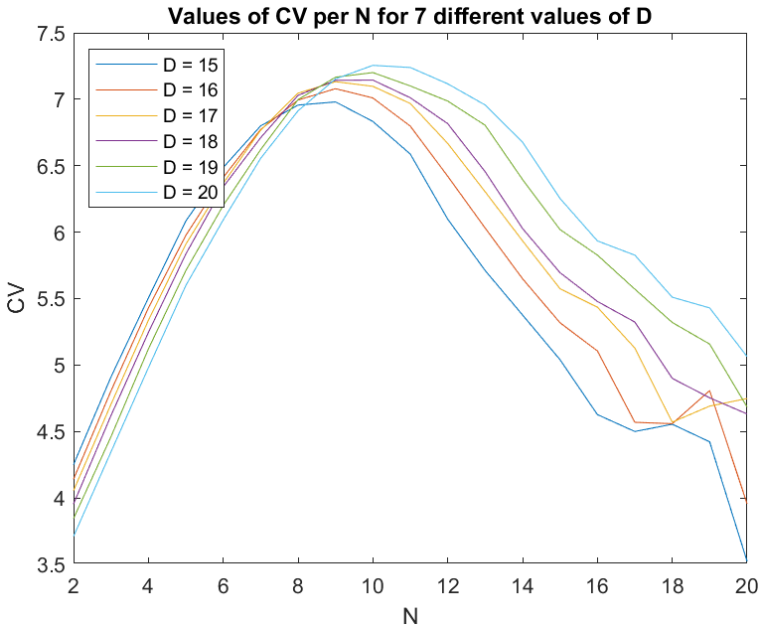
For 20% IPSP:



For 30% IPSP:



For 40% IPSP:


Values of CV per N for 7 different values of D
D = 15
D = 16
D = 17
D = 18
D = 19
D = 20
CV
N
7.5
7
6.5
6
5.5
5
4.5
4
3.5
2
4
6
8
10
12
14
16
18
20