Anand Sekar

Professor Shea-Brown

AMATH 342

2/21/17

AMATH 342 HW 3

1. FILTERING OF INPUTS

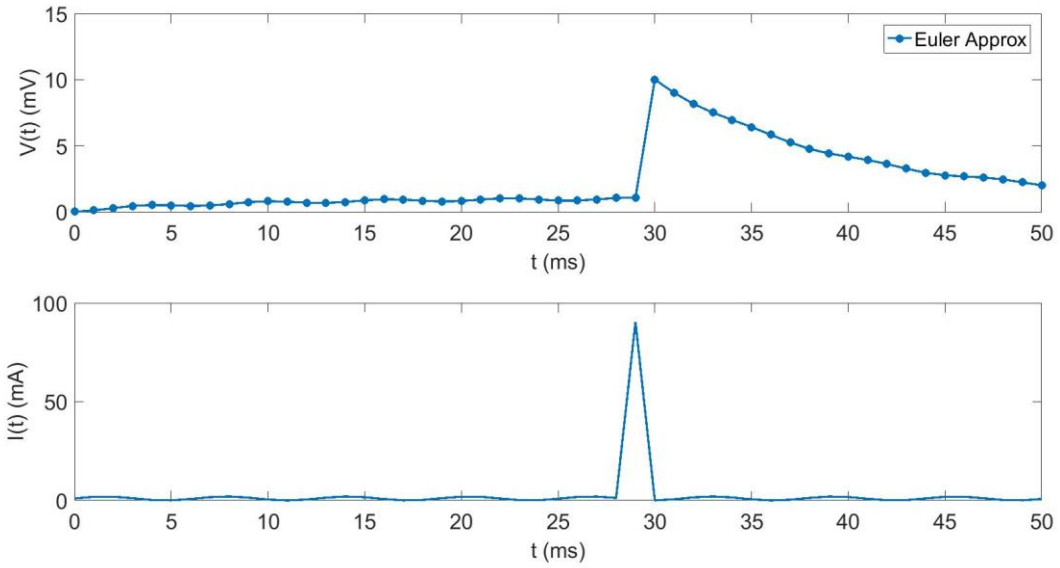
WHAT MATTERS IN DRIVING THE MEMBRANE RESPONSE?

In order to get $V(t_{now}) = 10mV$ at $t_{now} = 30ms$ with $RC = 10$, I tried out two different input currents. Here is a simple one where I simply set the current at t_{now} to 90.5 mA.

```
%euler method simulator
deltat=1; %timestep
Tmax=50;
tlist=linspace(0,Tmax,Tmax/deltat +1) ;
Vlist=zeros(1,length(tlist));

%initialize
V0=0;
Vlist(1)=V0;

%define input current
Iapplist=ones(1,length(tlist))+sin(tlist);
Iapplist(30) = 90.5;

%circuit parameters
R=1;
C=10;

for n=1:length(tlist)-1
    t=tlist(n);
    Vlist(n+1)=Vlist(n) + (-Vlist(n)/(R*C) + Iapplist(n)/C )*deltat;
end

set(0,'defaultaxesfontsize',20);
set(0,'defaulttextfontsize',20);

figure
set(gca,'FontSize',16)
subplot(211)
plot(tlist,Vlist,'.-','LineWidth',2,'MarkerSize',26); hold on
xlabel('t (ms)','Fontsize',20); ylabel('V(t) (mV)','Fontsize',20);
legend('Euler Approx')
```

```
subplot(212)
plot(tlist,Iapplist,'-','LineWidth',2); hold on
xlabel('t (ms)','Fontsize',20); ylabel('I(t) (mA)','Fontsize',20);
```

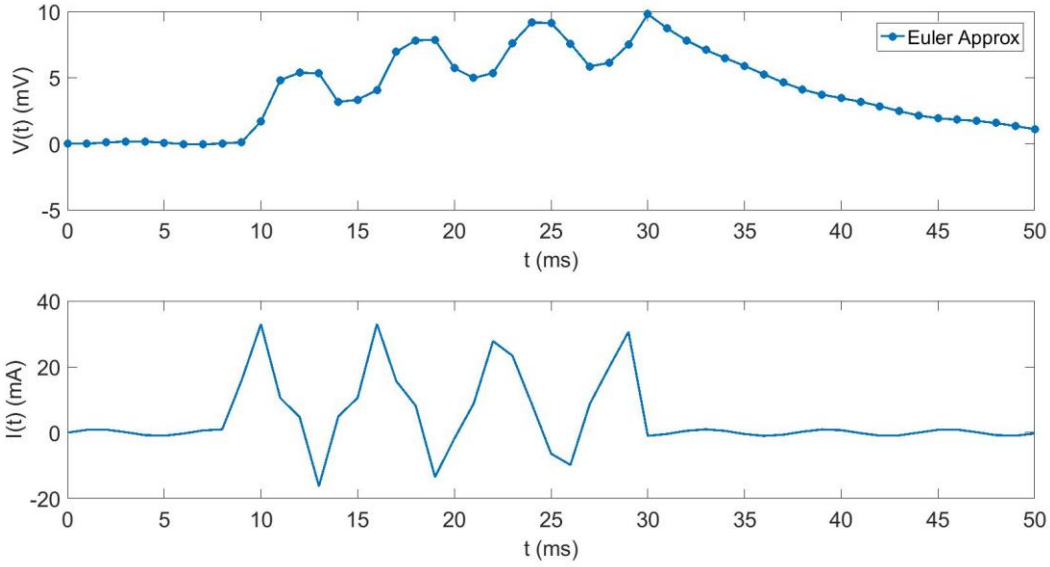
Result:



Then, I played around with a sin curve to get a very different-looking input current, but I was still able to get the conditions for voltage (10mv at 30ms). The code is same as the above except the part where I define the input current:

```
%define input currents
 Iapplist=sin(tlist);
 for t = 10:30
     Iapplist(t) = 25 * sin(tlist(t-2))^3+8.75;
 end;
```

Result:



ANALYSIS

The explicit solution for V(t) looks like this:

$$V = V_0 e^{-t/RC} + \int_0^t \frac{e^{-t/RC}}{C} \times I_A(t - t')\, dt$$

The current in this equation is within an integral. Simply focusing on that aspect, an integral is nothing but a sum of values. One can have different functions in which eventually sum up to a certain value at a certain point. In more concrete terms, there can be several input currents which can summate to producing the voltage $V(t_{now}) = 10mV$ at $t_{now} = 30ms$, as proved above with the graphs. With the first graph, the function simply produces a large instantaneous current (90.5 mA) at tnow. With the second graph, the function is somewhat sinusoidal in nature, producing currents with local maximums at around 30 mA; through integration, i.e. some form of summation, this can result in getting 10mv at 30ms.

2. SUMMATION OF SIMULTANEOUS IMPULSES

DO IMPULSES SUMMATE LINEARLY, SUBLINEARLY, OR SUPERLINEARLY?

CURRENT INPUT

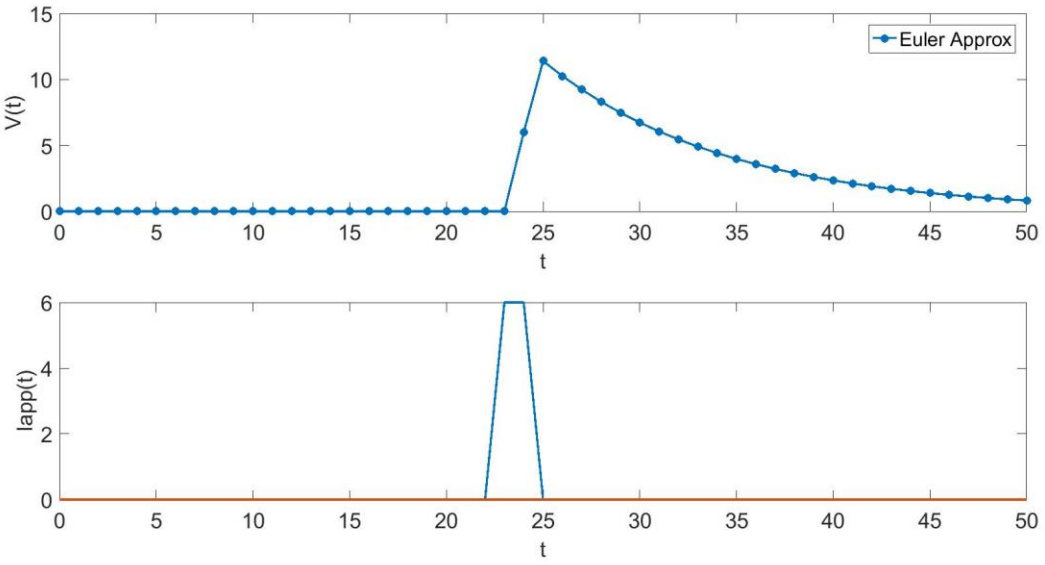
With RC = 10ms, I tested different currents (incrementing by 1) until I reached the spike generation threshold (10mv), with the initial voltage at zero. Having N simultaneous impulses is as simple as multiplying an input current by an integer value. I incremented this multiplier N until I reached the threshold value. Taking the fraction of the way to the threshold (dividing the

maximum voltage by ten) is analogous to percentage. I kept a list of this fraction, called f, and kept the list of corresponding N values. Here is the code:

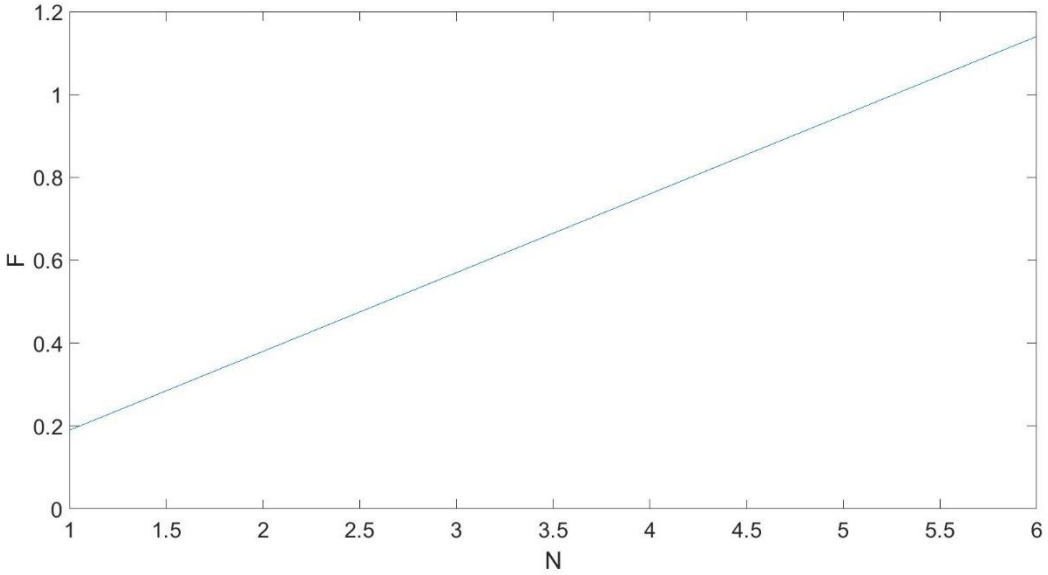
```
%euler method simulator
clear all;
deltat=1 ; %timestep
Tmax=50;
tlist=linspace(0,Tmax,Tmax/deltat +1) ;
maxVlist(1) = 1;
N = 1;
nlist = [];
flist = [];
while maxVlist(N) < 10

    Vlist=zeros(1,length(tlist));
    %initialize
    V0=0;
    Vlist(1)=V0;

    %define input currents
    %Iapplist=ones(1,length(tlist));
    Iapplist=zeros(length(tlist));
    Iapplist(24) = 1 * N;
    Iapplist(25) = 1 * N;
    %Iapplist=ones(1,length(tlist))+sin(tlist);

    %circuit parameters
    R=10;
    C=1;

    for n=1:length(tlist)-1
        t=tlist(n);
        Vlist(n+1)=Vlist(n) + (-Vlist(n)/(R*C) +
Iapplist(n)/C )*deltat;
    end

    maxVlist = [maxVlist max(Vlist)];
    fraction = max(Vlist)/ 10;
    flist = [flist fraction];
    nlist = [nlist N];
    N = N + 1;
end


set(0,'defaultaxesfontsize',20);
set(0,'defaulttextfontsize',20);

figure
set(gca,'FontSize',16)
subplot(211)
plot(tlist,Vlist,'.-','LineWidth',2,'MarkerSize',26); hold on
```

```
xlabel('t','Fontsize',20); ylabel('V(t)','Fontsize',20);
legend('Euler Approx')

subplot(212)
plot(tlist,Iapplist,'-','LineWidth',2); hold on
xlabel('t','Fontsize',20); ylabel('Iapp(t)','Fontsize',20);

% plotting f and n
figure
plot(nlist, flist);
xlabel('N');
ylabel('F');
```

When the code above is finished running, it gives me the graph of the final voltage and current which is just when the threshold is passed. This occurs at N = 7:



By plotting F against N, I'm able to see this relation:



By definition, F should equal 1 when the maximum voltage is the threshold. Here, it's obvious that F correlates linearly with N. This conclusion can be confirmed mathematically by manipulating the explicit integral solution for V(t):

Since we define $V_0 = 0$, the equation becomes simply:

$$V = \int_0^t \frac{e^{-t/RC}}{C} \times I_A(t - t')\, dt$$

We define f as:

$$f = \frac{\max(V(t))}{10} = \frac{\int_0^t \frac{e^{-t/RC}}{C} \times I_A(t - t')\, dt}{10}$$

N is simply a coefficient of the applied current, so we can put that into the equation, and for $f \geq 1$, we can define f_n as:

$$f_n = \frac{\max(V(t))}{10} = \frac{\int_0^t \frac{e^{-t/RC}}{C} \times N \times I_A(t - t')\, dt}{10}$$

Then, all we do is pull the N out of the integral:

$$f_n = \frac{\max(V(t))}{10} = N \times \frac{\int_0^t \frac{e^{-t/RC}}{C} \times I_A(t - t')\, dt}{10}$$

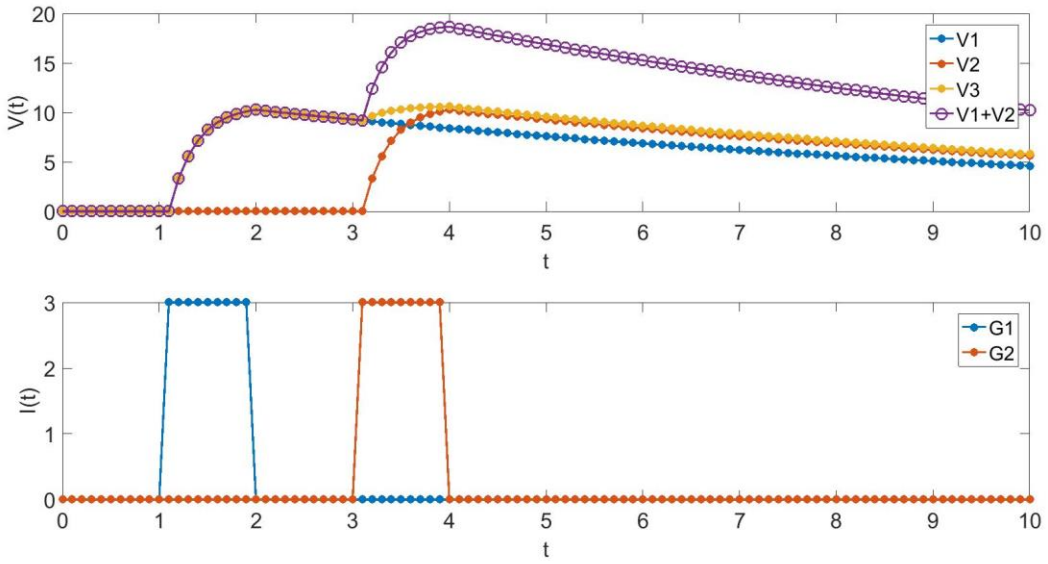
Finally we can substitute f back in on the right side to get

$$f_n = \frac{\max(V(t))}{10} = N \times f$$

This confirms the linear nature of summing current impulses.

CONDUCTANCE INPUT

To see how conductance input - with N again representing compounded inputs and F representing fraction to threshold – summates, I began by looking at the graph produced by the euler_illustrateRC_two_inputs_conductance.m code.



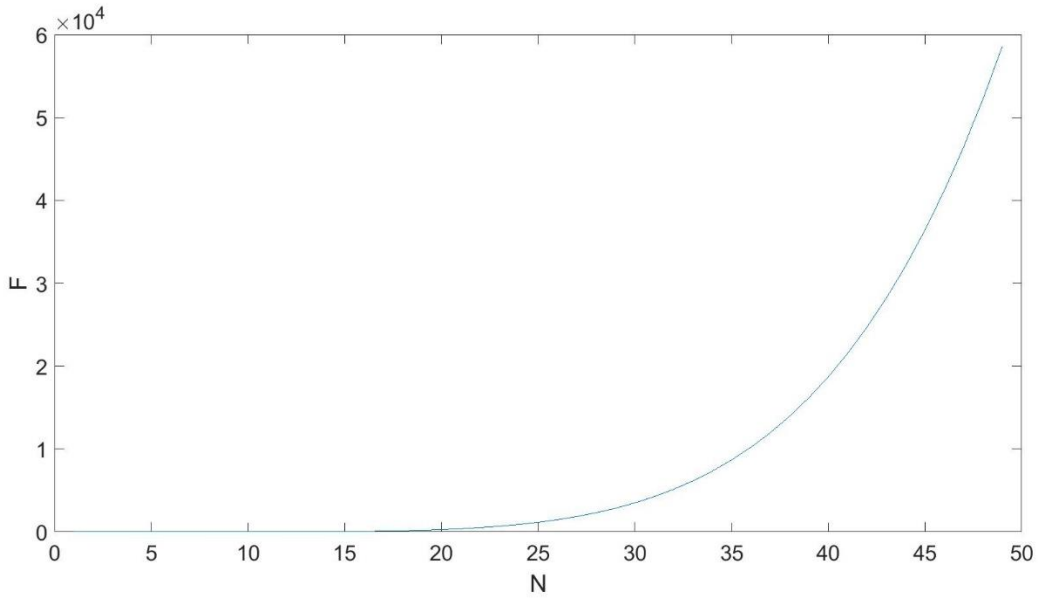
If the conductance were to summate linearly, then V3 – which is the voltage when the input is (G1 + G2), should equal V1 + V2. However, this is not the case. So to find out what the relationship between F and N is, I produced this code (which doesn't stop when the threshold is reached).

```
%euler method simulator

deltat=0.2 ; %timestep
Tmax=50;
tlist=linspace(0,Tmax,Tmax/deltat +1) ;
Vlist=zeros(1,length(tlist));

maxVlist(1) = 1;
N = 1;
nlist = [];
flist = [];

while N < 50
    %initialize
    V0=0;
    Vlist(1)=V0;

    %define input conductance
    gapplist=zeros(length(tlist));
```

```
    gapplist (25:30) = 1 * N;

    %circuit parameters
    R=10;
    C=1;
    E=11;

    for n=1:length(tlist)-1
        t=tlist(n);
        Vlist(n+1)=Vlist(n) + ( -Vlist(n)/(R*C) + gapplist(n)*(E-
Vlist(n)) )*deltat;
    end

    maxVlist = [maxVlist max(Vlist)];
    fraction = max(Vlist)/ 10;
    flist = [flist fraction];
    nlist = [nlist N];
    N = N + 1;

end

newmaxv = max(Vlist);
figure
set(gca,'FontSize',16)
subplot(211)
plot(tlist,Vlist,'.-','LineWidth',2,'MarkerSize',26); hold on
xlabel('V(t)','Fontsize',20); ylabel('V(t)','Fontsize',20);
%legend('Euler Approx')

subplot(212)
plot(tlist,gapplist,'-','LineWidth',2); hold on
xlabel('t','Fontsize',20); ylabel('gapp(t)','Fontsize',20);


%test linearity
Vlist1=zeros(1,length(tlist));
Vlist2=zeros(1,length(tlist));
Vlist=zeros(1,length(tlist));

%initialize
V0=0;
Vlist1(1)=V0;
Vlist2(1)=V0;
Vlist(1)=V0;

%define input conductance
gapplist1=ones(1,length(tlist));
gapplist2=1+sin(tlist);
gapplist=gapplist1+gapplist2;

for n=1:length(tlist)-1
```

```
    t=tlist(n);
    Vlist1(n+1)=Vlist1(n) + ( -Vlist1(n)/(R*C) + gapplist1(n)*(E-
Vlist1(n)) )*deltat;
    Vlist2(n+1)=Vlist2(n) + ( -Vlist2(n)/(R*C) + gapplist2(n)*(E-
Vlist2(n)) )*deltat;
    Vlist(n+1)=Vlist(n) + ( -Vlist(n)/(R*C) + gapplist(n)*(E-
Vlist(n)) )*deltat;
end
newmaxv = max(Vlist);

set(0,'defaultaxesfontsize',20);
set(0,'defaulttextfontsize',20);

figure
set(gca,'FontSize',16)
subplot(311)
plot(tlist,Vlist,'-','LineWidth',2,'MarkerSize',26); hold on
plot(tlist,Vlist1,'-','LineWidth',2,'MarkerSize',26); hold on
plot(tlist,Vlist2,'-','LineWidth',2,'MarkerSize',26); hold on
xlabel('V(t)','Fontsize',20); ylabel('V(t)','Fontsize',20);
%legend('Euler Approx')


set(gca,'FontSize',16)
subplot(312)
plot(tlist,Vlist,'.-','LineWidth',2,'MarkerSize',26); hold on
plot(tlist,Vlist1+Vlist2,'.-','LineWidth',2,'MarkerSize',26); hold on
xlabel('V(t)','Fontsize',20); ylabel('V(t)','Fontsize',20);
%legend('Euler Approx')


subplot(313)
plot(tlist,gapplist1,'-','LineWidth',2); hold on
plot(tlist,gapplist2,'-','LineWidth',2); hold on
plot(tlist,gapplist,'-','LineWidth',2); hold on
xlabel('t','Fontsize',20); ylabel('gapp(t)','Fontsize',20);

% plotting f and n
figure
plot(nlist, flist);
xlabel('N');
ylabel('F');
```

This produces the following figure:



Clearly, F and N correlate exponentially (rather than linearly with compounded simultaneous current input). This can be proven with the explicit solution of V(t): (insert starred equation and take out g, zero out stuff, and see how G is exponentiated).

$$V(t) = V_0 exp \int_0^t \frac{1}{\tau} + g(s)ds + E \int_0^t dt' g(t') \, exp \int_{t'}^t \left(\frac{1}{\tau} + g(s)\right) ds$$

Since V(0) = 0, this becomes

$$V(t) = E \int_0^t dt' g(t') \, exp \int_{t'}^t \left(\frac{1}{\tau} + g(s)\right) ds$$

Then, similar to how we defined N and f in the previous, current-based input, we can modify the equation as such:

$$f = \frac{V(t)}{10} = \frac{E \int_0^t dt' g(t') \, exp \int_{t'}^t \left(\frac{1}{\tau} + \times N \times g(s)\right) ds}{10}$$

Since N is a constant, it can be pulled out of the integral:

$$f = \frac{V(t)}{10} = \frac{E \int_0^t dt' g(t') \, exp \, N \int_{t'}^t \left(\frac{1}{\tau N} + g(s)\right) ds}{10}$$

The term exp is used to represent $e^{something}$, and now that N is outside the integral, f correlates with something along the lines of e^N. This shows a rather exponential, than linear, relationship between f and N with conductance-based inputs. This makes sense, if one were to simplify the relationship between current, conductance, and resulting voltage with Ohm's law. Voltage can correlate more directly with current. If this were integrated, it would make sense that once more

currents are added, it would summate linearly. Voltage also correlates directly with resistance – but resistance is the inverse of conductance. Summing values which are inversed already hints at a non-linear behavior. Additionally, when something of the nature $\frac{1}{x}$ is integrated, it can result in something of the nature $\ln(x)$, which can result in something of the nature e^x. The last few sentences are not meant to be a legitimate explanation of the phenomena, but rather serve as a way to bridge intuitive understanding.

3. HH MODEL

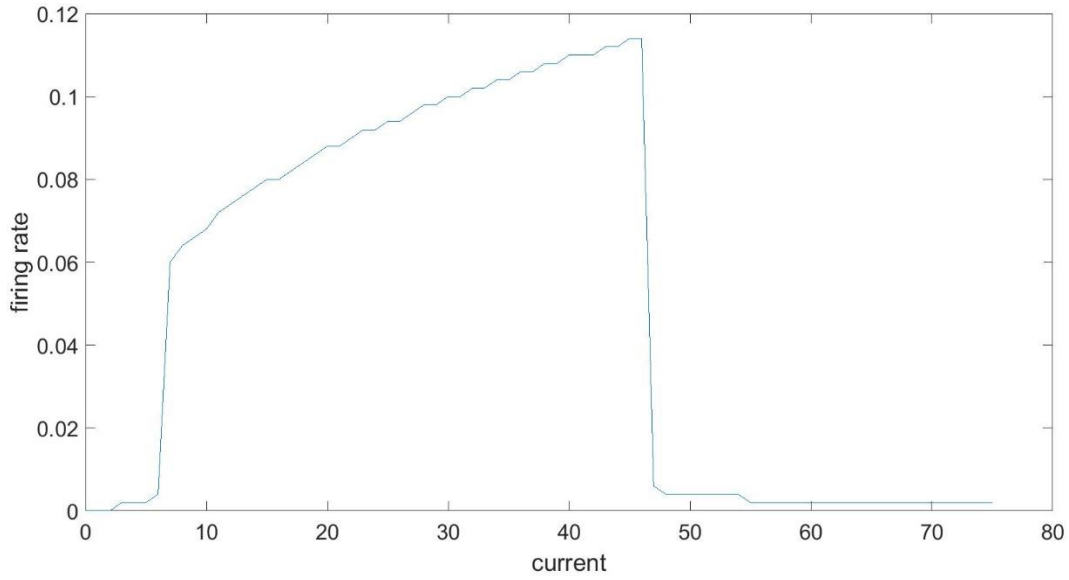
TUNING CURVE

Code:

```
% Hodgkin/Huxley Equations (V_HH = -V-65), with current definition
% of membrane potential (V=Vin-Vout)
clear all;
vna=50;  %set the constants
vk=-77;
vl=-54.4;
gna=120;
gk=36;
gl=.3;
c=1;
I=6.35;

v_init=-65;  %the initial conditions
m_init=.052;
h_init=.596;
n_init=.317;

npoints=50000;  %number of timesteps to integrate
dt=0.01;        %timestep

m=zeros(npoints,1); %initialize everything to zero
n=zeros(npoints,1);
h=zeros(npoints,1);
v=zeros(npoints,1);
time=zeros(npoints,1);

m(1)=m_init; %set the initial conditions to be the first entry in the
vectors
n(1)=n_init;
h(1)=h_init;
v(1)=v_init;
time(1)=0.0;

currentList = [];
peaksList = [];
```

```
for current = 0:75
    currentList = [currentList current];
    I = current;
    numpeak = 0;
    tic
    for step=1:npoints-1,
        v(step+1)=v(step)+((I - gna*h(step)*(v(step)-
vna)*m(step)^3 ...
                  -gk*(v(step)-vk)*n(step)^4-gl*(v(step)-vl))/c)*dt;
        m(step+1)=m(step)+ (alpha_m(v(step))*(1-m(step))-
beta_m(v(step))*m(step))*dt;
        h(step+1)=h(step)+ (alpha_h(v(step))*(1-h(step))-
beta_h(v(step))*h(step))*dt;
        n(step+1)=n(step)+ (alpha_n(v(step))*(1-n(step))-
beta_n(v(step))*n(step))*dt;
        time(step+1)=time(step)+dt;

        %condition for spike detection:  decreasing now, increased
        %before, over threshold
        thresh = 10;
        if ((step>1) & (v(step+1)<v(step)) & (v(step)>v(step-1)) &
v(step)>thresh)
          numpeak = numpeak + 1;
        end
    end
    toc
    peaksList = [peaksList numpeak];
end

set(0,'defaultaxesfontsize',20);
set(0,'defaulttextfontsize',20);

figure
rateList = peaksList / 500;
plot(currentList, rateList);
xlabel('current');
ylabel('firing rate');
```

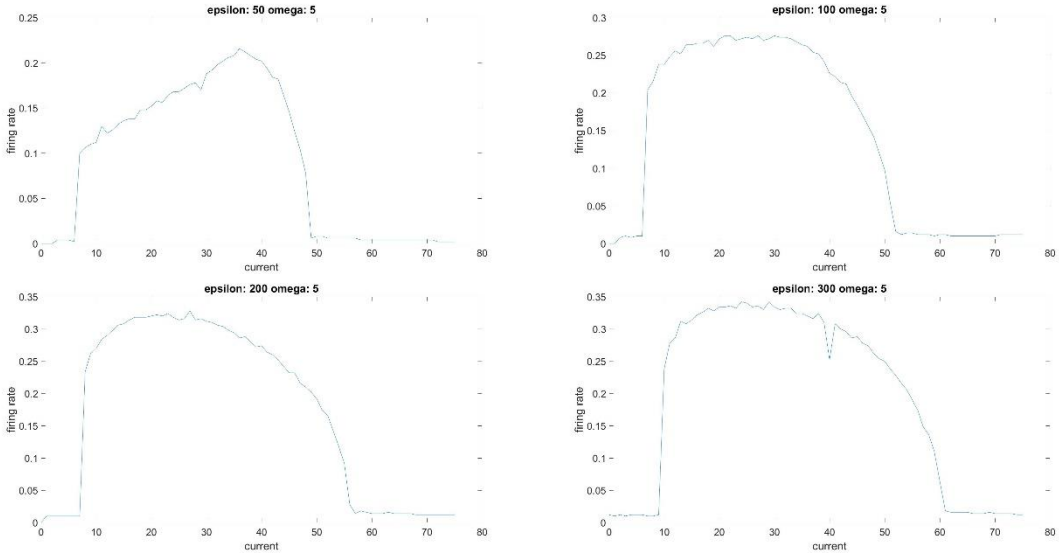
Result:

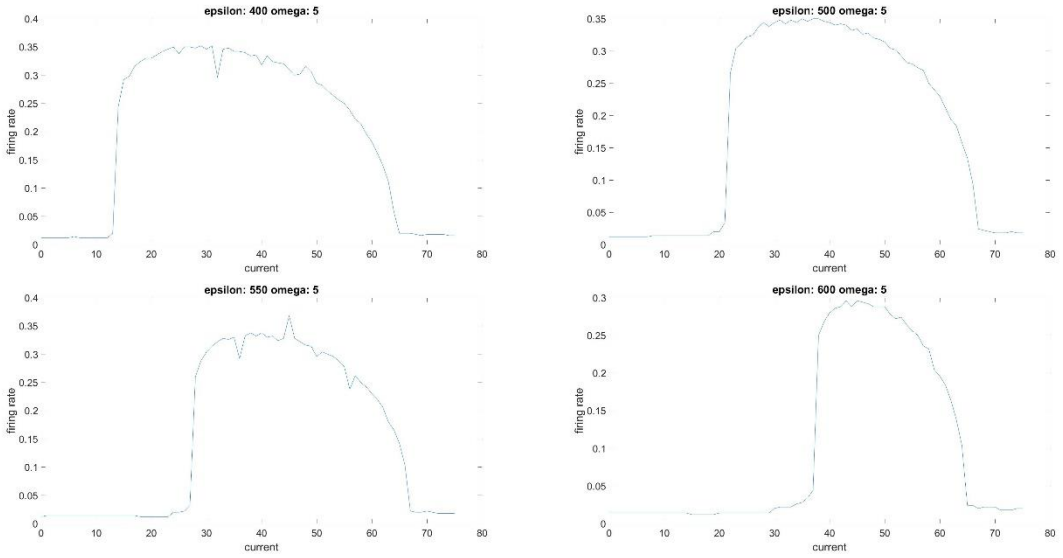


This is what the tuning curve looks like when simply incrementing the continuously-applied currents. Now, we add a sinusoidal background current. The code is very similar to the tuning curve above, except I define an epsilon and omega in the beginning, and add this line immediately after the for loop which calculates voltage using steps:

```
    for step=1:npoints-1,
        I = current + (epsilon * sin(2 * pi * time(step) * omega));
```

Here are the results for various manipulations of epsilon:



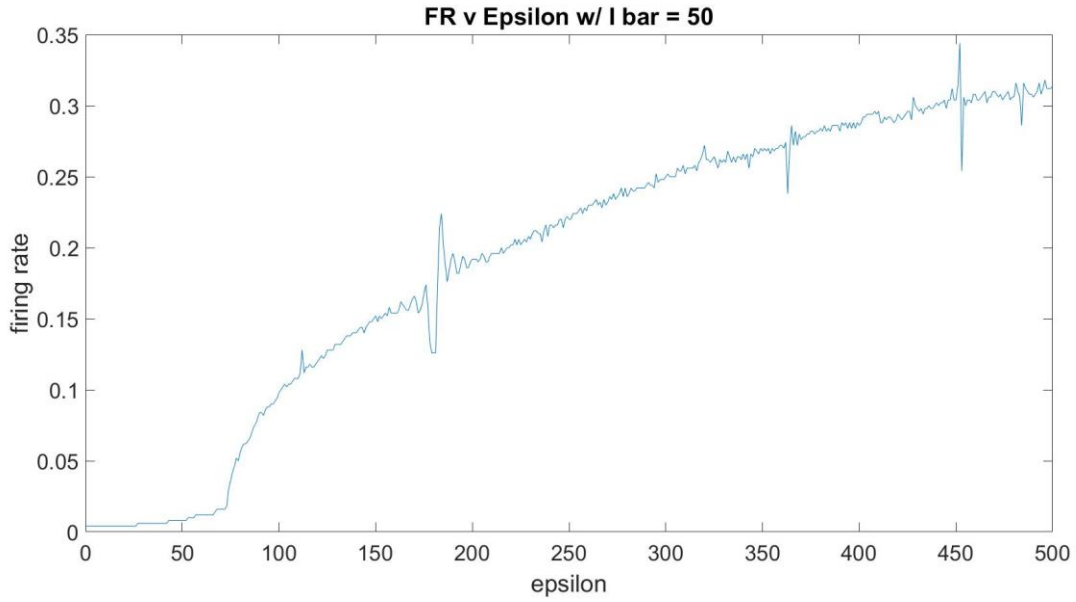


I'm not sure how to analyze these different graphs, so I plotted a tuning curve. Here is the code and resulting graph:

```
% constants above this code
epsilonList =[];
peaksList = [];
omega = 5;
current = 50;
for epsilon = 0:500
    epsilonList = [epsilonList epsilon];
    numpeak = 0;
    tic
    for step=1:npoints-1,
        I = current + (epsilon * sin(2 * pi * time(step) * omega));

        v(step+1)=v(step)+((I - gna*h(step)*(v(step)-
vna)*m(step)^3 ...
                  -gk*(v(step)-vk)*n(step)^4-gl*(v(step)-vl))/c)*dt;
        m(step+1)=m(step)+ (alpha_m(v(step))*(1-m(step))-
beta_m(v(step))*m(step))*dt;
        h(step+1)=h(step)+ (alpha_h(v(step))*(1-h(step))-
beta_h(v(step))*h(step))*dt;
        n(step+1)=n(step)+ (alpha_n(v(step))*(1-n(step))-
beta_n(v(step))*n(step))*dt;
        time(step+1)=time(step)+dt;



        %condition for spike detection:  decreasing now, increased
              %before, over threshold
              thresh = 10;
```

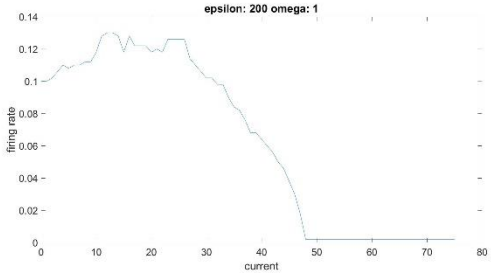
```
                if ((step>1) & (v(step+1)<v(step)) & (v(step)>v(step-
1)) & v(step)>thresh)
                  numpeak = numpeak + 1;
                end
    end
    toc

    peaksList = [peaksList numpeak];
end

set(0,'defaultaxesfontsize',20);
set(0,'defaulttextfontsize',20);

figure
plot(time,v);
xlabel('t');
ylabel('V');

figure
rateList = peaksList / 500;
plot(epsilonList, rateList);
title(['FR v Epsilon w/ I bar = ' num2str(current)]);
xlabel('epsilon');
ylabel('firing rate');
```

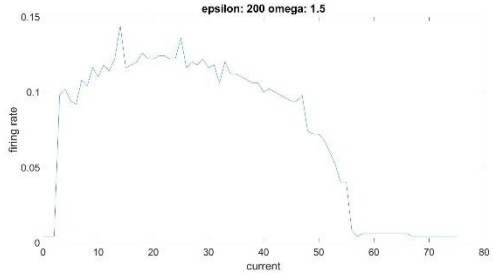
Result:

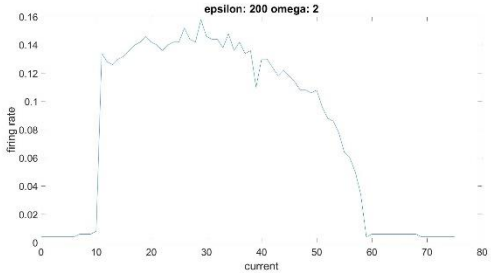


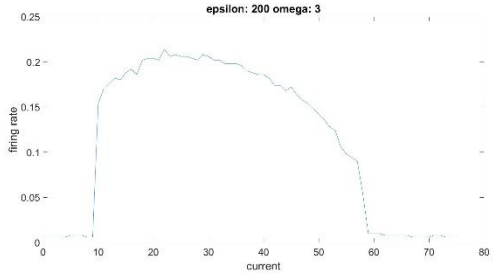
It's clear from this graph that with larger values of epsilon, the firing rate increases, and the rate of increasing decreases with larger values of epsilon.

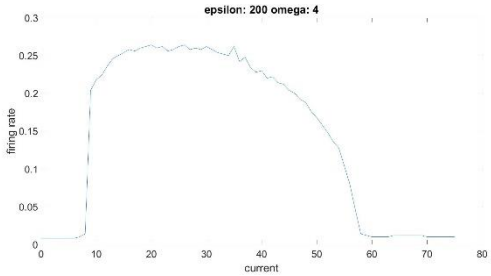
Here are the results for various manipulations of omega:

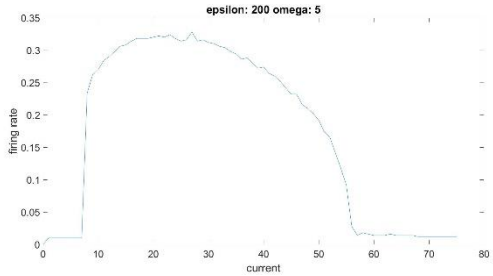

epsilon: 200 omega: 1
firing rate
current

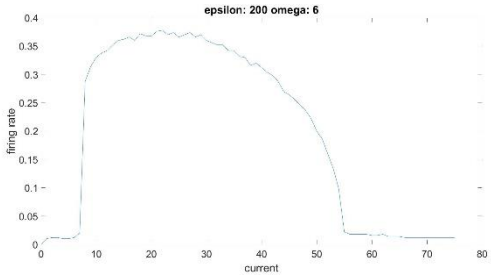

epsilon: 200 omega: 1.5
firing rate
current

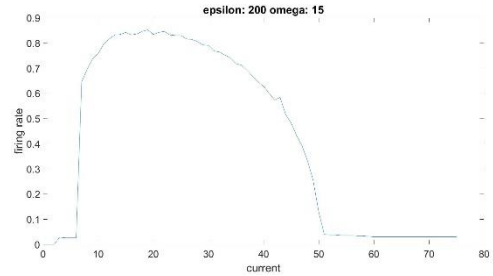

epsilon: 200 omega: 2
firing rate
current

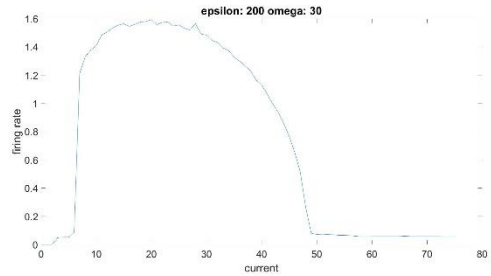

epsilon: 200 omega: 3
firing rate
current

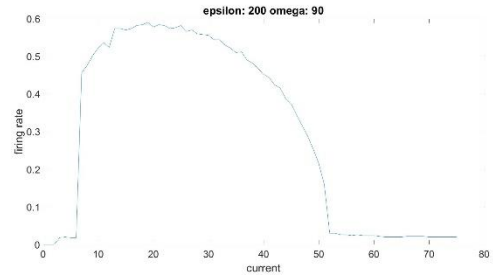

epsilon: 200 omega: 4
firing rate
current

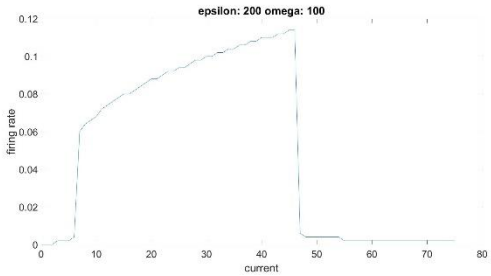

epsilon: 200 omega: 5
firing rate
current


epsilon: 200 omega: 6
firing rate
current


epsilon: 200 omega: 15
firing rate
current


epsilon: 200 omega: 30
firing rate
current

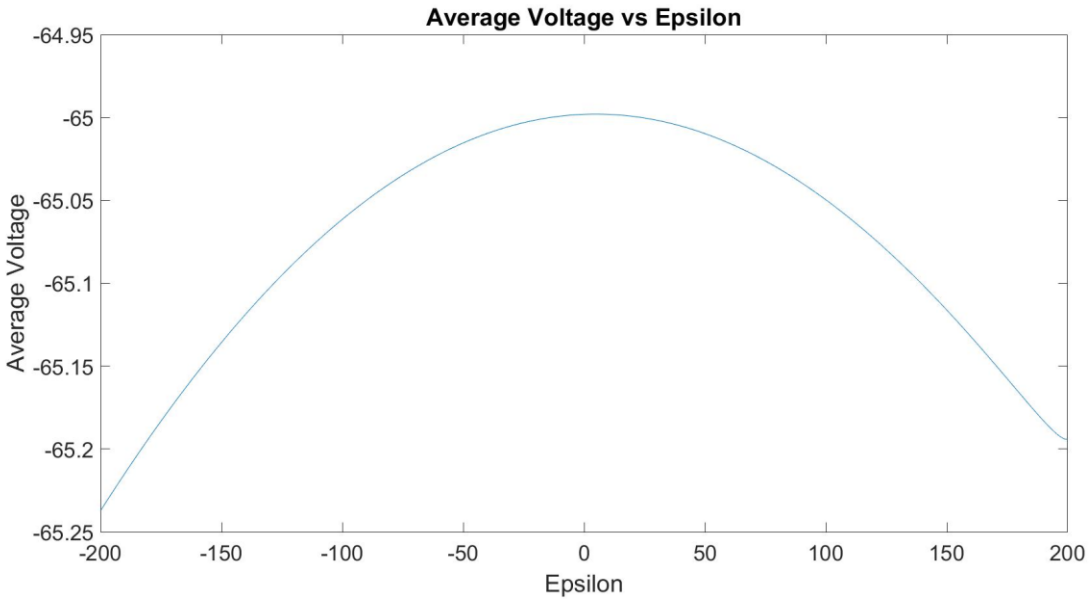

epsilon: 200 omega: 90
firing rate
current


epsilon: 200 omega: 100
firing rate
current

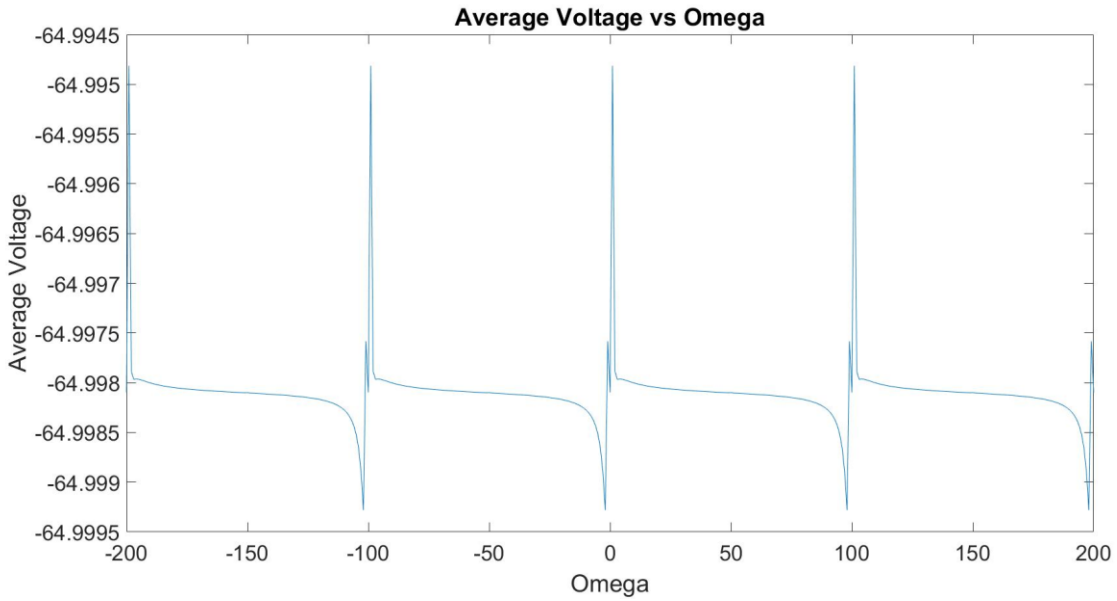
From 1 to 99, omega keeps increasing the firing rate (as observed by the changing scale of the y-axis). Most importantly, the curve gets smoother and smoother – approaching the shape of the regular tuning curve. This makes sense because the frequency, as it gets higher and to the value of 100, with a relatively small epsilon – you pretty much get the original tuning curve.

OTHER ASPECTS OF THE NEURAL RESPONSE

There are a few aspects of the neural response beyond the firing rate that come to mind. The resulting action potentials from the sinusoidal current input have their own aspects such as amplitude or width/ duration. Average voltage – as part of a tuning curve over varying epsilon and omega – is also an aspect that can be analyzed. I would expect the average voltage to increase as epsilon increases and omega to have a periodic effect, but these are simply guesses. This is what the tuning curve for average looks like for varying epsilon:

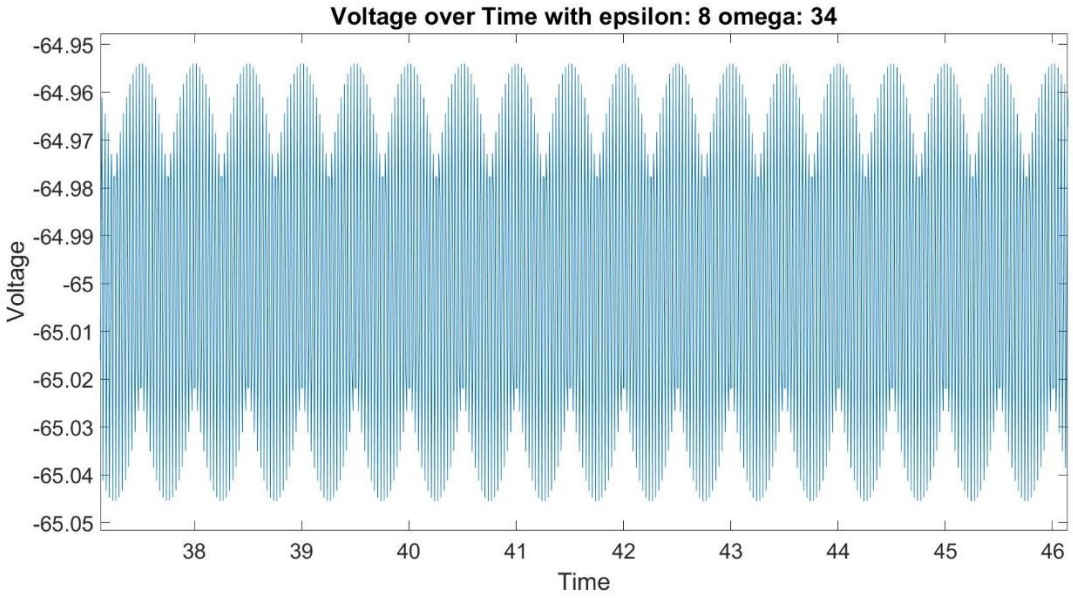


And this is what it looks like while varying omega:



It makes sense how average voltage would vary with omega – given that omega affects the frequency of the sinusoidal function (it would be repetitive). It also makes sense that the average voltage vs. epsilon graph is symmetric because the amplitude increases either way – and the larger fluctuations cause and overall lower average voltage (because the graph seems somewhat shifted downwards).

Another aspect can be analyzing the "beats" created by the pattern of spikes. For example, below is a graph that shows a zoomed-in graph of voltage over current with a set omega and epsilon:



There is an enveloping sinusoidal function occurring – and the properties of this function can be more quantifiable aspects. I wrote code to detect the frequency of this sinusoidal function, and

was going to run it to get a graph of average frequency over different values of epsilon or omega. Here is the code:

```
% Hodgkin/Huxley Equations (V_HH = -V-65), with current definition
% of membrane potential (V=Vin-Vout)
clear all;
vna=50;  %set the constants
vk=-77;
vl=-54.4;
gna=120;
gk=36;
gl=.3;
c=1;
I=0;

v_init=-65;  %the initial conditions
m_init=.052;
h_init=.596;
n_init=.317;

npoints=50000;  %number of timesteps to integrate
dt=0.01;        %timestep

m=zeros(npoints,1); %initialize everything to zero
n=zeros(npoints,1);
h=zeros(npoints,1);
v=zeros(npoints,1);
time=zeros(npoints,1);

m(1)=m_init; %set the initial conditions to be the first entry in the
vectors
n(1)=n_init;
h(1)=h_init;
v(1)=v_init;
time(1)=0.0;

freqList = [];
omegaList = [];
epsilon = 8;
for omega = 31:35,
    maxList = [];
    for step=1:npoints-1,
        I = 0 + (epsilon * sin(2 * pi * time(step) * omega));

        v(step+1)=v(step)+((I - gna*h(step)*(v(step)-
vna)*m(step)^3 ...
                  -gk*(v(step)-vk)*n(step)^4-gl*(v(step)-vl))/c)*dt;
        m(step+1)=m(step)+ (alpha_m(v(step))*(1-m(step))-
beta_m(v(step))*m(step))*dt;
        h(step+1)=h(step)+ (alpha_h(v(step))*(1-h(step))-
beta_h(v(step))*h(step))*dt;
```

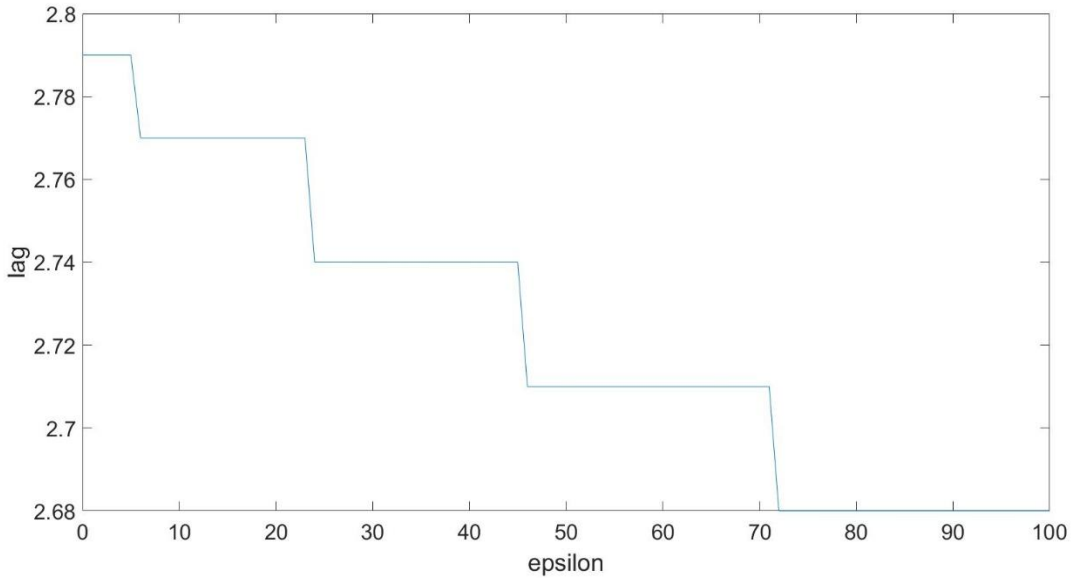
```
        n(step+1)=n(step)+ (alpha_n(v(step))*(1-n(step))-
beta_n(v(step))*n(step))*dt;
        time(step+1)=time(step)+dt;

        if ((step>1) & (v(step+1)<v(step)) & (v(step)>v(step-1)))
            %local max detected
            maxList = [maxList [v(step);step]];
        end
    end

    %get index of maximums
    indexList = [];
    for n = 1:length(maxList)-1
        if ((maxList(n+1)<maxList(n)) & (maxList(n)>maxList(n-1)))
            indexList = [indexList n];
        end
    end

    timeDiffList = [];
    for n = 1:length(indexList)-1
        t1 = maxList(2, indexList(n));
        t2 = maxList(2, indexList(n+1));
        tdiff = t2 - t1;
        timeDiffList = [timeDiffList tdiff];
    end

    meanPeriod = mean(timeDiffList);
    meanFrequency = 1/meanPeriod;

    omegaList = [omegaList omega];
    freqList = [freqList meanFrequency];

    disp(meanPeriod);
end
set(0,'defaultaxesfontsize',20);
set(0,'defaulttextfontsize',20);

figure
plot(omegaList, freqList);
title('Beat Frequency over Omega');
ylabel('Beat Frequency');
xlabel('Omega');
```

I was able to get average frequency for single omega values, but unfortunately Matlab glitches out – with its debugger – when put in this omega for loop. Professor Shea-Brown also agreed that there was no logical explanation (as in the debugger was reporting the wrong error), so there must be some computing error occurring. I could manually plot the points, but unfortunately I don't have that time, so I have decided on analyzing yet another aspect of the neural response. However, I'm sure that if this code runs, it would yield some significant results.

Spike lag – the time to the first spike – could be another aspect of the neural response. This is the graph (along with code) of the time to the first spike over values of epsilon.

```
% all the repetitive HH constants would be above this
omega = 34;
current = 6.35;
epsilonList = [];
lagList = [];
for epsilon=0:100
    spikeTimes = [];
    for step=1:npoints-1,
        I = current + (epsilon * sin(2 * pi * time(step) * omega));

        v(step+1)=v(step)+((I - gna*h(step)*(v(step)-
vna)*m(step)^3 ...
                  -gk*(v(step)-vk)*n(step)^4-gl*(v(step)-vl))/c)*dt;
        m(step+1)=m(step)+ (alpha_m(v(step))*(1-m(step))-
beta_m(v(step))*m(step))*dt;
        h(step+1)=h(step)+ (alpha_h(v(step))*(1-h(step))-
beta_h(v(step))*h(step))*dt;
        n(step+1)=n(step)+ (alpha_n(v(step))*(1-n(step))-
beta_n(v(step))*n(step))*dt;
        time(step+1)=time(step)+dt;

         %condition for spike detection:  decreasing now, increased
         %before, over threshold
         thresh = 10;
         if ((step>1) & (v(step+1)<v(step)) & (v(step)>v(step-1)) &
v(step)>thresh)
             spikeTimes = [spikeTimes (step*dt)];
         end
    end
    lagList = [lagList spikeTimes(1)];
    epsilonList = [epsilonList epsilon];
end

disp(spikeTimes(1));

set(0,'defaultaxesfontsize',20);
set(0,'defaulttextfontsize',20);

figure
plot(epsilonList,lagList);
xlabel('epsilon');
ylabel('lag');
```

Result:



As the graph clearly shows, as values of epsilon increases, the time lag to the first spike decreases (i.e. getting to the first spike is faster), and eventually it doesn't spike. This makes sense due to how epsilon affects the amplitude of the applied current – getting a quicker voltage spike.