**Qunatative Physiology 2 CLab 1: Cardiac Action Potentials**

Pranav Maddula

Washington University in St. Louis

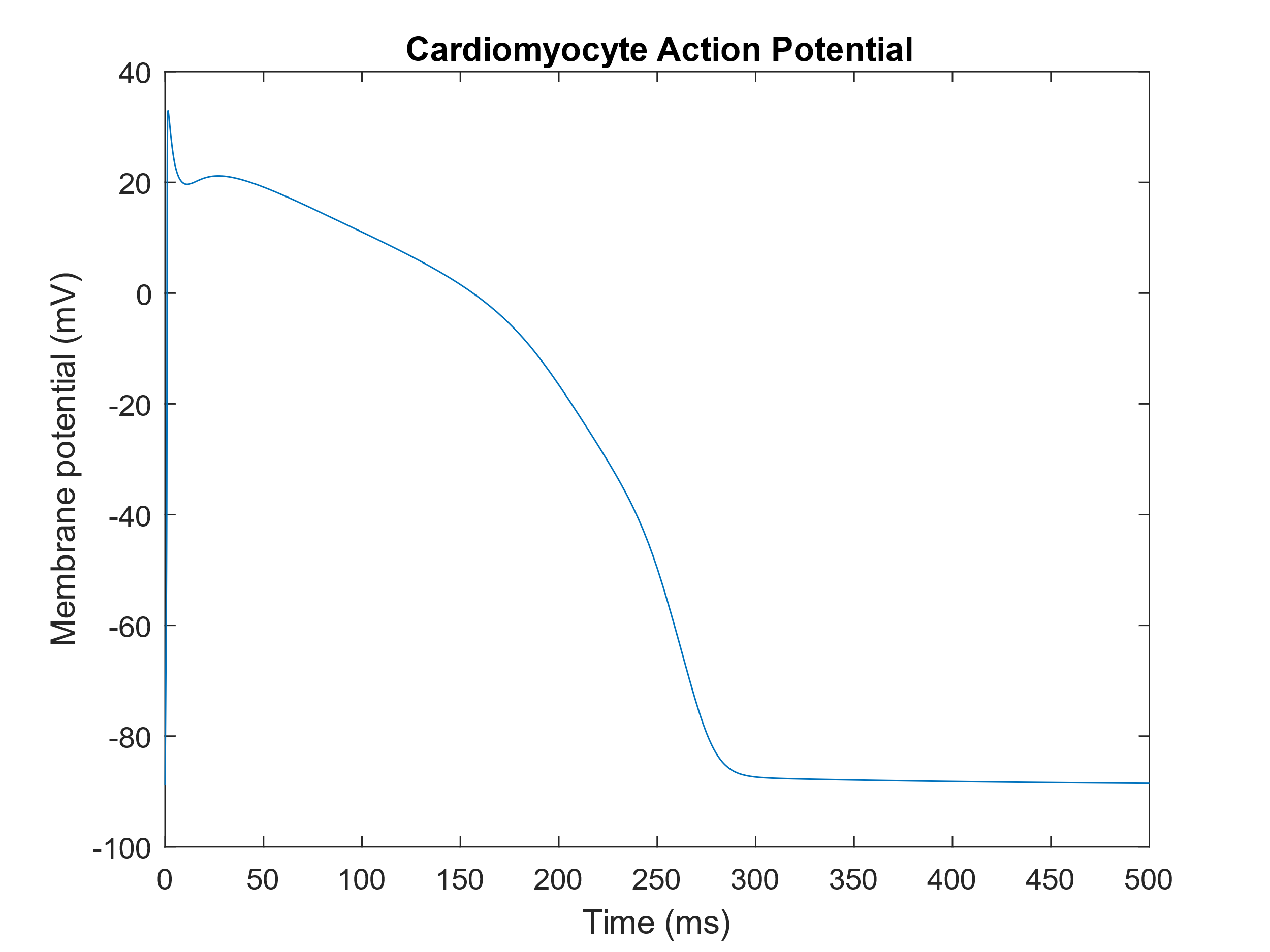
BME 301B

Lab Instructor: P. Widder

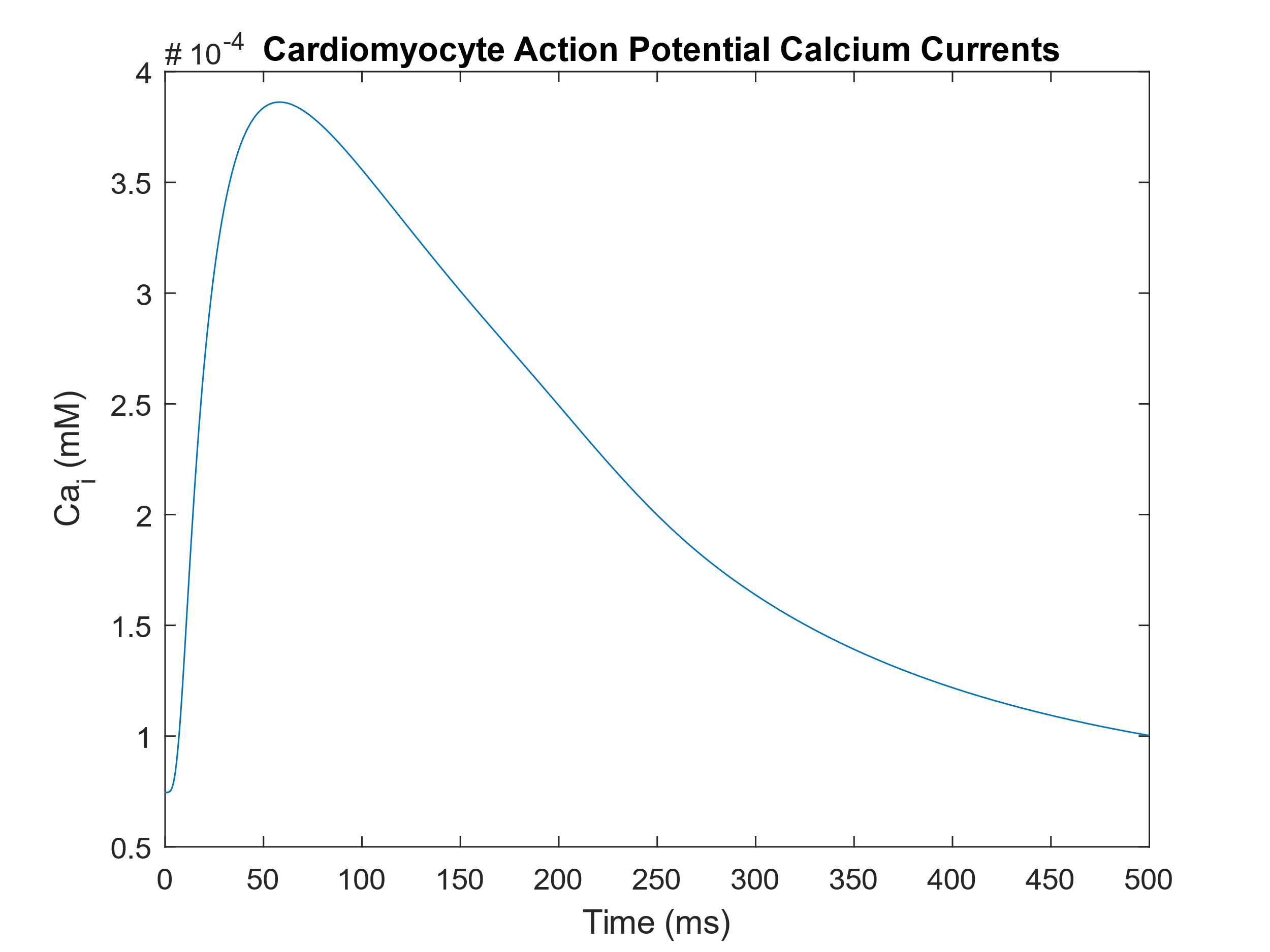
**Lab Date**: Friday, January 24, 2020

**Submission Date**: Tuesday, February 4, 2020

Question 1.1:



**Figure 1:** Plot of Cardiomyocyte Action Potential. 100 beats at a cycle length of 1 second were run, and the final beat was extracted and plotted.



**Figure 2:**  Plot of calcium currents during a Cardiomyocyte Action Potential. 100 beats at a cycle length of 1 second were run, and the final calcium conductance beat was extracted and plotted.

Question 1.1: Code

param.bcl = 1000; % basic cycle length in ms

param.model = @model\_Torord; % which model is to be used - right now, use model12. In general, any model with the same format of inputs/outputs as model12 may be simulated, which is useful when current formulations are changed within the model code, etc.

param.verbose = true; % printing numbers of beats simulated.

options = []; % parameters for ode15s - usually empty

beats = 100; % number of beats

ignoreFirst = beats - 1; % this many beats at the start of the simulations are ignored when extracting the structure of simulation outputs (i.e., beats - 1 keeps the last beat).

X0 = getStartingState('Torord\_endo'); % starting state - can be also m12\_mid or m12\_epi for midmyocardial or epicardial cells respectively.

%% Simulation and extraction of outputs

% The structure param and other variables are passed to ORdRunner, which is

% an interface between user and the simulation code itself (which is in

% model12.m). The ORdRunner unpacks the structure of parameters given by

% the users, sets undefined parameters to default, and sends all that to

% @model12.

% time, X are cell arrays corresponding to stored beats (if 1 beat is

% simulated, this is 1-by-1 cell still), giving time vectors and state

% variable values at corresponding time points.

[time, X] = modelRunner(X0, options, param, beats, ignoreFirst);

% A structure of currents is computed from the state variables (see the

% function code for a list of properties extracted - also, hitting Tab

% following typing 'currents.' lists all the fields of the structure). Some

% state variables are also stored in a named way (time, V, Cai, Cass) so

% that the user can do most of necessary plotting simply via accessing the

% structure currents as shown below.

currents = getCurrentsStructure(time, X, param, 0);

%% Plotting membrane potential and calcium transient

figure(1);

plot(currents.time, currents.V);

xlabel('Time (ms)');

ylabel('Membrane potential (mV)');

xlim([0 500]);

title('Cardiomyocyte Action Potential')

figure(2);

plot(currents.time, currents.Cai);

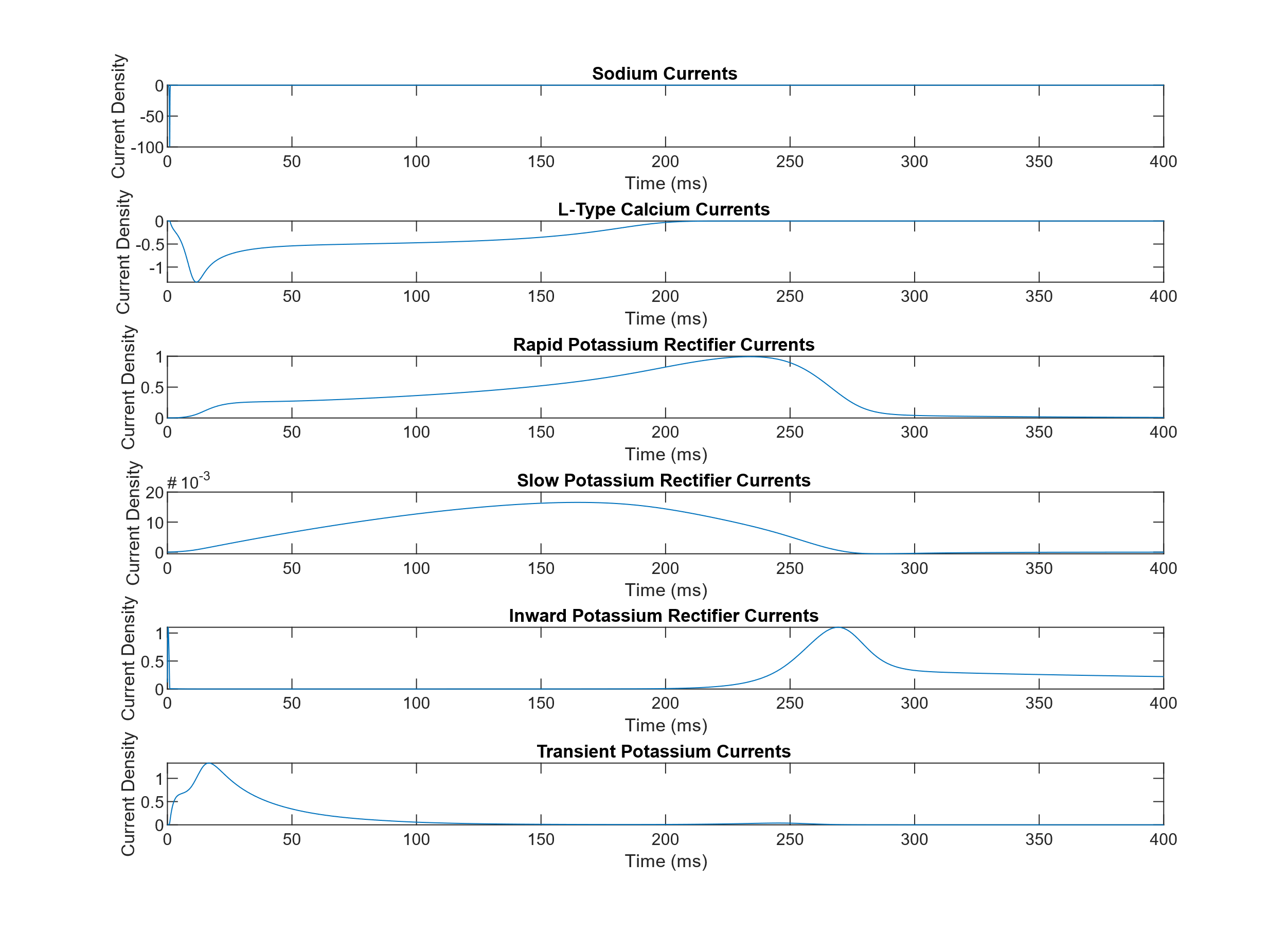
xlabel('Time (ms)');

ylabel('Ca\_i (mM)');

xlim([0 500]);

title('Cardiomyocyte Action Potential Calcium Currents')

Question 1.2:



**Figure 3**: This figure shows the individual ionic currents densities as a function of time for a standard Cardiomyocyte Action Potential. As in part 1, 100 beats at a cycle length of 1 second were run to establish initial conditions.

|  |  |
| --- | --- |
| Fast Sodium Current | Depolarizing outwards current during phase 0 of the AP |
| L-Type Calcium Current | Depolarizing inward calcium current during phase 1 of the AP |
| Rapid Delayed Rectifier Potassium Current | Repolarizing outwards potassium current during phase 3 |
| slow delayed rectifier potassium current | Repolarizing outwards potassium current during phase 2 |
| Inward rectifier potassium current | Repolarizing outwards potassium current during phase 3 |
| Transient outward potassium | Repolarizing outwards potassium current during phase 1 |

Table 1: This table holds answers to Question 1.2b

Question 1.2 Code:

param.bcl = 1000; % basic cycle length in ms

param.model = @model\_Torord; % which model is to be used - right now, use model12. In general, any model with the same format of inputs/outputs as model12 may be simulated, which is useful when current formulations are changed within the model code, etc.

param.verbose = true; % printing numbers of beats simulated.

options = []; % parameters for ode15s - usually empty

beats = 100; % number of beats

ignoreFirst = beats - 1; % this many beats at the start of the simulations are ignored when extracting the structure of simulation outputs (i.e., beats - 1 keeps the last beat).

X0 = getStartingState('Torord\_endo'); % starting state - can be also m12\_mid or m12\_epi for midmyocardial or epicardial cells respectively.

% time, X are cell arrays corresponding to stored beats (if 1 beat is

% simulated, this is 1-by-1 cell still), giving time vectors and state

% variable values at corresponding time points.

[time, X] = modelRunner(X0, options, param, beats, ignoreFirst);

% A structure of currents is computed from the state variables (see the

% function code for a list of properties extracted - also, hitting Tab

% following typing 'currents.' lists all the fields of the structure). Some

% state variables are also stored in a named way (time, V, Cai, Cass) so

% that the user can do most of necessary plotting simply via accessing the

% structure currents as shown below.

currents = getCurrentsStructure(time, X, param, 0);

%% Plotting

ina = currents.INa; % fast sodium current

ical = currents.ICaL\_i; % L-type calcium current

ikr = currents.IKr; % rapid delayed rectifier potassium current

iks = currents.IKs; % slow delayed rectifier potassium current

ik1 = currents.IK1; % inward rectifier potassium current

ito = currents.Ito; % transient outward potassium

time = currents.time

figure(3)

subplot(6,1,1)

plot(time,ina)

xlim([0 400]);

xlabel('Time (ms)');

ylabel('Current Density');

title('Sodium Currents')

subplot(6,1,2)

plot(time,ical)

xlim([0 400]);

xlabel('Time (ms)');

ylabel('Current Density');

title('L-Type Calcium Currents')

subplot(6,1,3)

plot(time,ikr)

xlim([0 400]);

xlabel('Time (ms)');

ylabel('Current Density');

title('Rapid Potassium Rectifier Currents')

subplot(6,1,4)

plot(time,iks)

xlim([0 400]);

xlabel('Time (ms)');

ylabel('Current Density');

title('Slow Potassium Rectifier Currents')

subplot(6,1,5)

plot(time,ik1)

xlim([0 400]);

xlabel('Time (ms)');

ylabel('Current Density');

title('Inward Potassium Rectifier Currents')

subplot(6,1,6)

plot(time,ito)

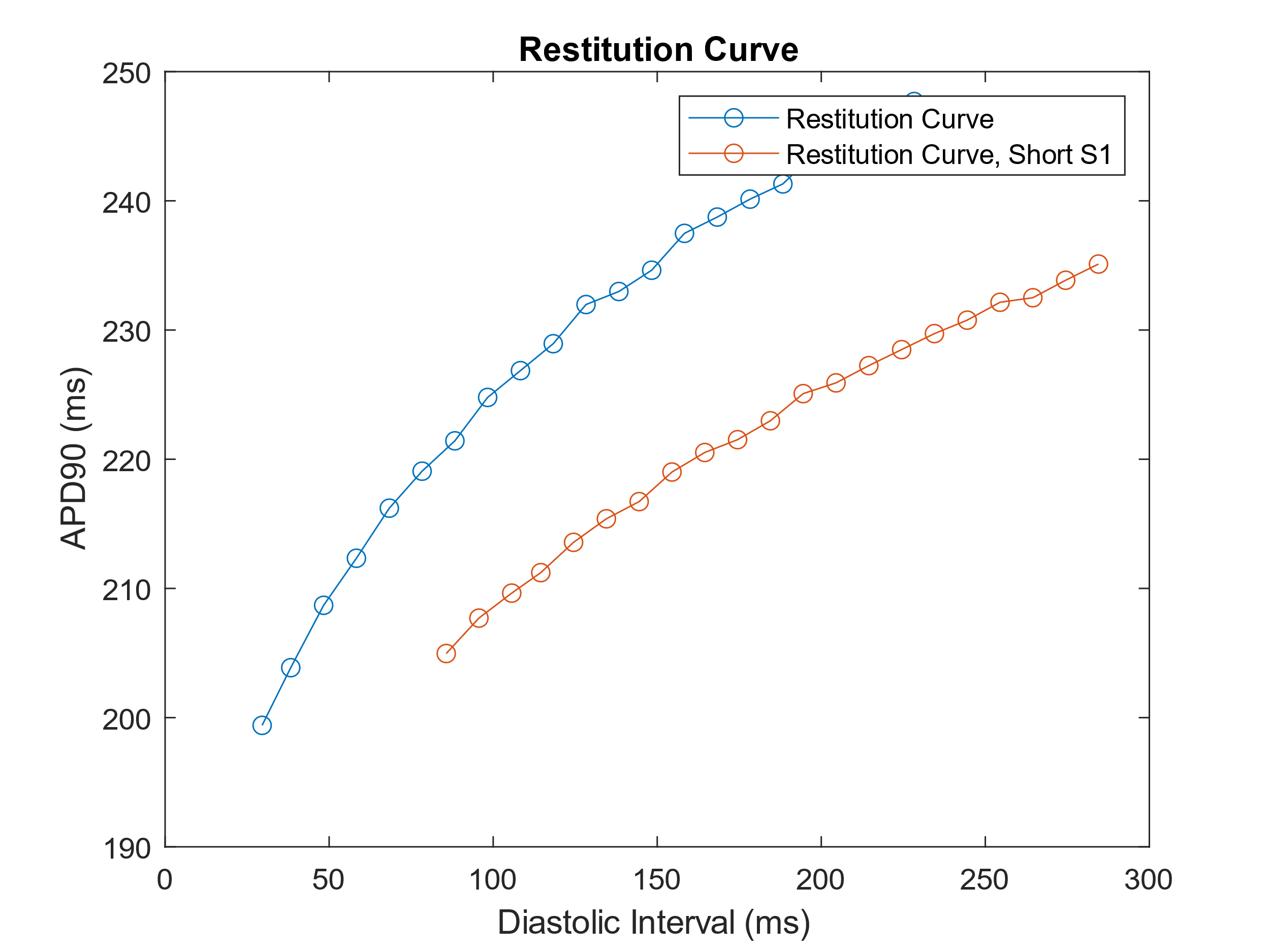
xlim([0 400]);

xlabel('Time (ms)');

ylabel('Current Density');

title('Transient Potassium Currents')

Question 2.1:



**Figure 4:** This figure shows the restitution curves generated by varying the cycle length of S1. For the initial curve, S1 had a cycle length of 1000ms, while for the second curve with a short S1, a cycle length of 350ms was used. The initial and final slopes of the long S2 are .45 and .26 (respectively). The initial and final slopes of the short S2 are .21 and .11 (respectively). The slopes between the two curves are different, and the slopes of the short S2 are lower, which is reflected as a flatter curve for short S2 in the figure.

Question 2.1 A,B,C,D Code:

%% Generate Restutition Curve

s2CL = [300:10:500]

for i = 1:length(s2CL)

[DInt(i), ADP90(i)] = Question\_2\_function(s2CL(i))

end

figure;

plot(DInt,ADP90,'-o')

xlabel('Diastolic Interval (ms)');

ylabel('APD90 (ms)');

title('Restitution Curve')

hold on

plot(DInt\_2,ADP90\_2,'-o')

legend('Restitution Curve','Restitution Curve, Short S1')

%

% %Changed 308 to .1 for tauh

function [DI,APD90\_2] = Question\_2\_function(s2CL)

clearvars -except s2CL DI DInt

X0 = getStartingState('Torord\_endo'); % starting state - can be also m12\_mid or m12\_epi for midmyocardial or epicardial cells respectively.

%%

param.bcl = 1000; % basic cycle length in ms

beats = 40; % number of beats

param.model = @model\_Torord; % which model is to be used - right now, use model12. In general, any model with the same format of inputs/outputs as model12 may be simulated, which is useful when current formulations are changed within the model code, etc.

param.verbose = true; % printing numbers of beats simulated.

options = []; % parameters for ode15s - usually empty

ignoreFirst = beats - 1

[time, X] = modelRunner(X0, options, param, beats, ignoreFirst);

lastX\_cell=X(end); lastX = cell2mat(lastX\_cell)

X02 = lastX(end,:)

%%

param2 = param

param2.bcl = s2CL; % s2 Cycle Length

beats2 = 1; % number of beats

param2.model = @model\_Torord; % which model is to be used - right now, use model12. In general, any model with the same format of inputs/outputs as model12 may be simulated, which is useful when current formulations are changed within the model code, etc.

param2.verbose = true; % printing numbers of beats simulated.

%ignoreFirst = beats - 1

ignoreFirst = 0

[time2, X2] = modelRunner(X02, options, param2, beats2, ignoreFirst);

lastX2\_cell=X2(end); lastX2 = cell2mat(lastX2\_cell)

X03 = lastX2(end,:)

currents2 = getCurrentsStructure(time2, X2, param2, 0);

%%

param3 = param

param3.bcl = 500; % basic cycle length in ms

beats3 = 1; % number of beats

param3.model = @model\_Torord; % which model is to be used - right now, use model12. In general, any model with the same format of inputs/outputs as model12 may be simulated, which is useful when current formulations are changed within the model code, etc.

param3.verbose = true; % printing numbers of beats simulated.

%ignoreFirst = beats - 1

ignoreFirst = 0

[time3, X3] = modelRunner(X03, options, param3, beats3, ignoreFirst);

currents3 = getCurrentsStructure(time3, X3, param3, 0);

%%

s1s2\_time = [currents2.time;currents3.time+currents2.time(end)];

s1s2\_Vm = [currents2.V;currents3.V];

figure; plot(s1s2\_time,s1s2\_Vm)

%% Calculations

%V Rest

V\_rest\_1 = mean(currents2.V(end-5:end))

%V Max

V\_max\_1 = max(currents2.V)

%dvdt Max

[dvdt\_max\_1,I\_1] = max(diff(currents2.V))

%V90

V90\_1 = V\_max\_1 - (0.9\*(V\_max\_1 - V\_rest\_1))

%tV90

offset = 30

[c\_1 index\_1] = min(abs(currents2.V(offset:end)-V90\_1))

index\_1 = index\_1+offset;

%APD90

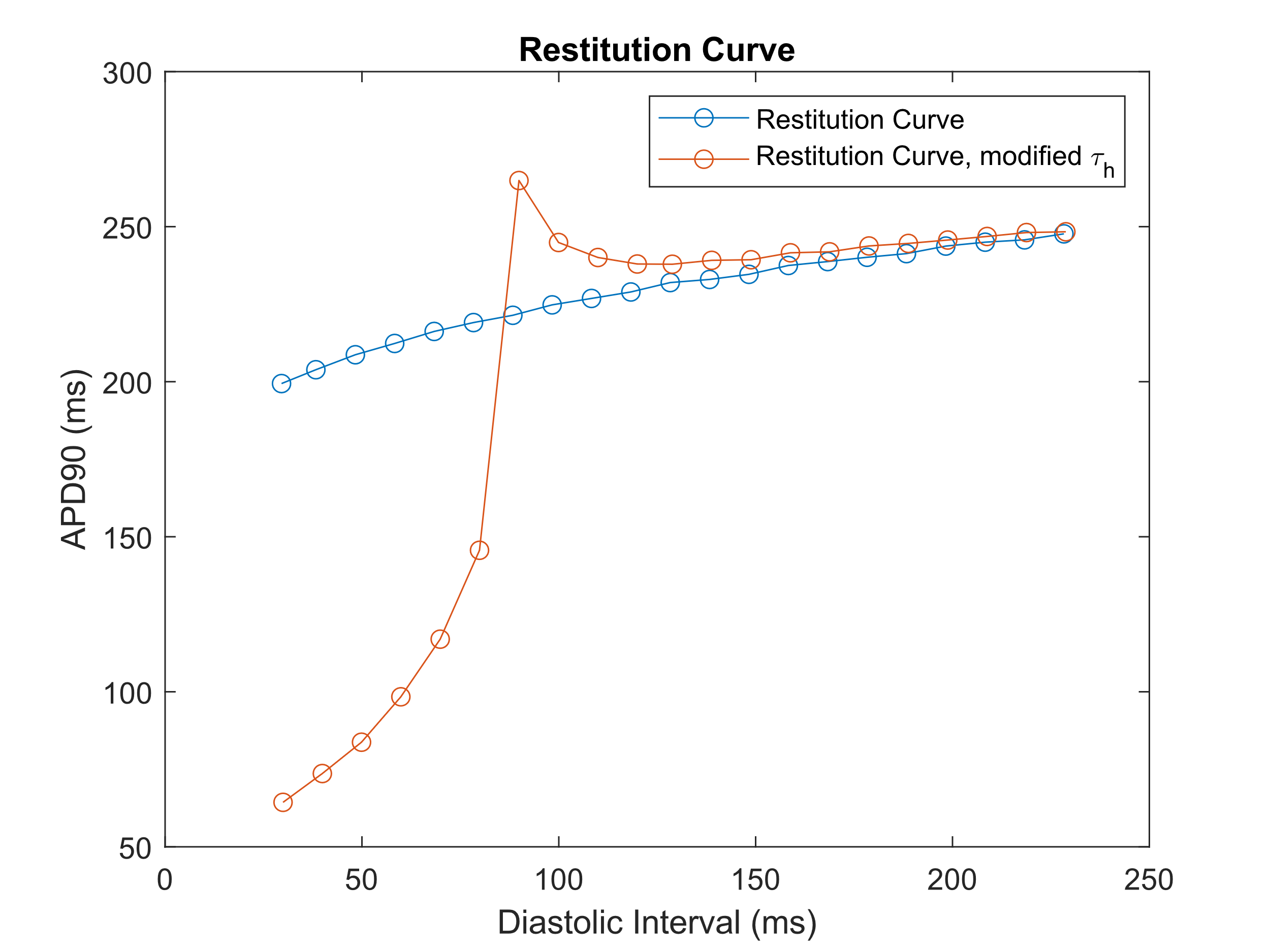
APD90\_1 = currents2.time(index\_1) - currents2.time(I\_1)

T2 = s1s2\_time(I\_2 + length(currents2.time))

T1 = s1s2\_time(index\_1)

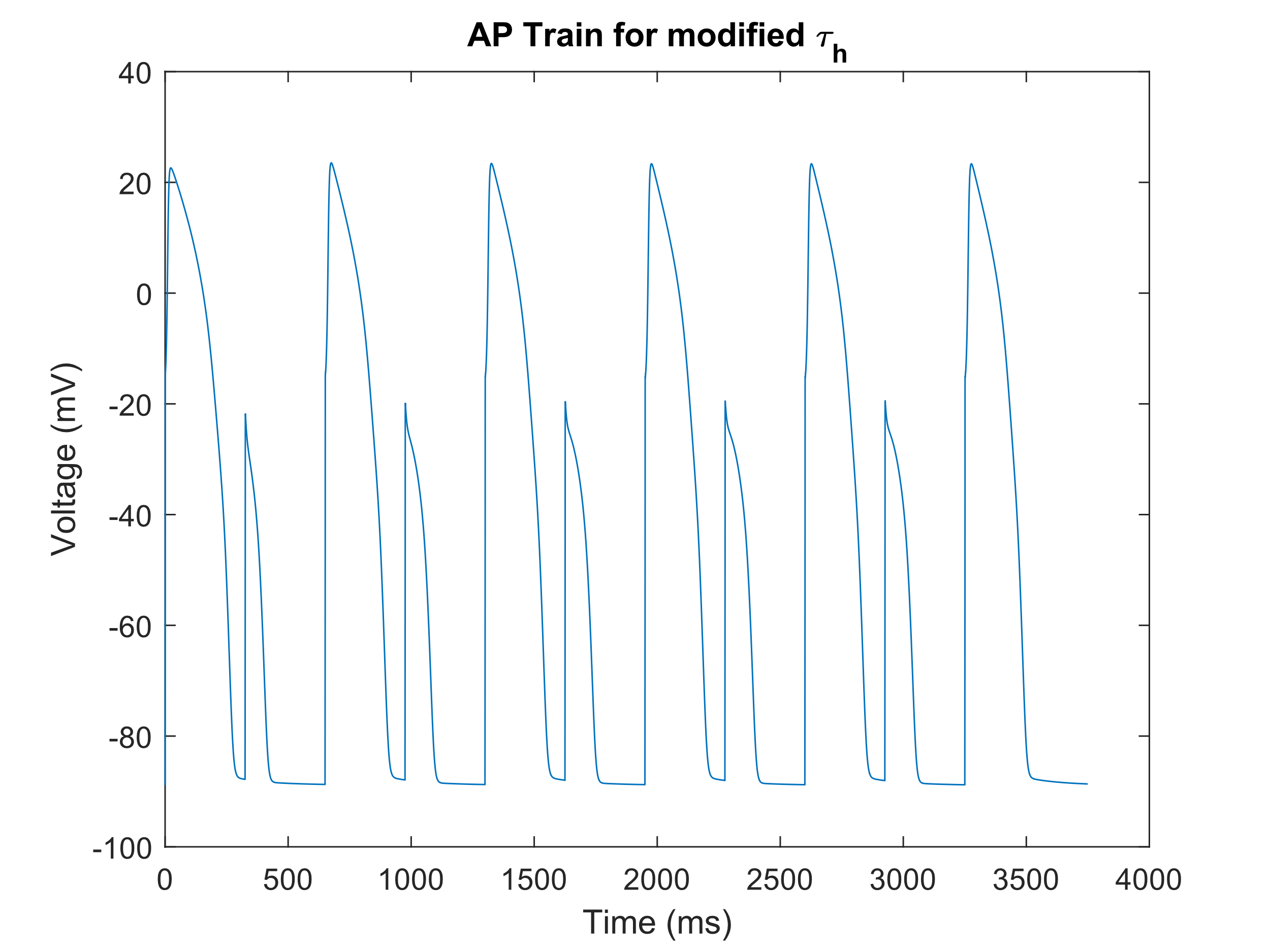
DI = T2-T1

end



**Figure 5:** This figure shows that it is, in fact, possible to achieve an initial slope greater than 1 for the restitution curve. This is done by modifying the τh parameter, which is the inactivation time constant for the sodium gate. The figure shows the initial restitution curve with an S1 cycle length of 1000ms in blue, and a curve generated using the modified τh parameter with the same S1 cycle length. The initial and final slopes of the modified restitution curve are 1.13 and .09 (respectively).

The results in **Fig. 5** were generated by reducing the value of the τh parameter by a factor 10. This results in a the sodium channel being inhibited more quickly, which may lead to some sort of periodic instability in the model as every other AP will be inhibited more quickly, limiting the magnitude of the AP. This hypothesis is supported by the results from the AP train, in Fig. 6.

**Figure 6:** This figure shows the AP train that is a result of the modification of the τh parameter. Periodic instability is seen as hypothesized as every other beat is overly inhibited due to the increased sodium recovery times.

Question 2.1 E Code:

%% Generate Restutition Curve

s2CL = [300:10:500]

for i = 1:length(s2CL)

[DInt(i), ADP90(i)] = Question\_2\_function(s2CL(i))

end

figure;

plot(DInt,ADP90,'-o')

xlabel('Diastolic Interval (ms)');

ylabel('APD90 (ms)');

title('Restitution Curve')

hold on

plot(DInt\_2,ADP90\_2,'-o')

legend('Restitution Curve','Restitution Curve, Short S1')

% %Changed 308 to .1 for tauh

function [DI,APD90\_2] = Question\_2\_function(s2CL)

clearvars -except s2CL DI DInt

X0 = getStartingState('Torord\_endo'); % starting state - can be also m12\_mid or m12\_epi for midmyocardial or epicardial cells respectively.

%%

param.bcl = 1000; % basic cycle length in ms

beats = 40; % number of beats

param.model = @model\_Torord; % which model is to be used - right now, use model12. In general, any model with the same format of inputs/outputs as model12 may be simulated, which is useful when current formulations are changed within the model code, etc.

param.verbose = true; % printing numbers of beats simulated.

options = []; % parameters for ode15s - usually empty

ignoreFirst = beats - 1

[time, X] = modelRunner(X0, options, param, beats, ignoreFirst);

lastX\_cell=X(end); lastX = cell2mat(lastX\_cell)

X02 = lastX(end,:)

%%

param2 = param

param2.bcl = s2CL; % s2 Cycle Length

beats2 = 1; % number of beats

param2.model = @model\_Torord; % which model is to be used - right now, use model12. In general, any model with the same format of inputs/outputs as model12 may be simulated, which is useful when current formulations are changed within the model code, etc.

param2.verbose = true; % printing numbers of beats simulated.

%ignoreFirst = beats - 1

ignoreFirst = 0

[time2, X2] = modelRunner(X02, options, param2, beats2, ignoreFirst);

lastX2\_cell=X2(end); lastX2 = cell2mat(lastX2\_cell)

X03 = lastX2(end,:)

currents2 = getCurrentsStructure(time2, X2, param2, 0);

%%

param3 = param

param3.bcl = 500; % basic cycle length in ms

beats3 = 1; % number of beats

param3.model = @model\_Torord; % which model is to be used - right now, use model12. In general, any model with the same format of inputs/outputs as model12 may be simulated, which is useful when current formulations are changed within the model code, etc.

param3.verbose = true; % printing numbers of beats simulated.

%ignoreFirst = beats - 1

ignoreFirst = 0

[time3, X3] = modelRunner(X03, options, param3, beats3, ignoreFirst);

currents3 = getCurrentsStructure(time3, X3, param3, 0);

%%

s1s2\_time = [currents2.time;currents3.time+currents2.time(end)];

s1s2\_Vm = [currents2.V;currents3.V];

figure; plot(s1s2\_time,s1s2\_Vm)

%% Calculations

%V Rest

V\_rest\_1 = mean(currents2.V(end-5:end))

%V Max

V\_max\_1 = max(currents2.V)

%dvdt Max

[dvdt\_max\_1,I\_1] = max(diff(currents2.V))

%V90

V90\_1 = V\_max\_1 - (0.9\*(V\_max\_1 - V\_rest\_1))

%tV90

offset = 30

[c\_1 index\_1] = min(abs(currents2.V(offset:end)-V90\_1))

index\_1 = index\_1+offset;

%APD90

APD90\_1 = currents2.time(index\_1) - currents2.time(I\_1)

T2 = s1s2\_time(I\_2 + length(currents2.time))

T1 = s1s2\_time(index\_1)

DI = T2-T1

end

train\_time = s1s2\_time;

train\_Vm = s1s2\_Vm;

T1 = train\_time;

T2 = T1;

V1 = train\_Vm;

V2 = V1;

for i = 1:1:8

T2 = [T2;T1+T2(end)];

V2 = [V2;V1];

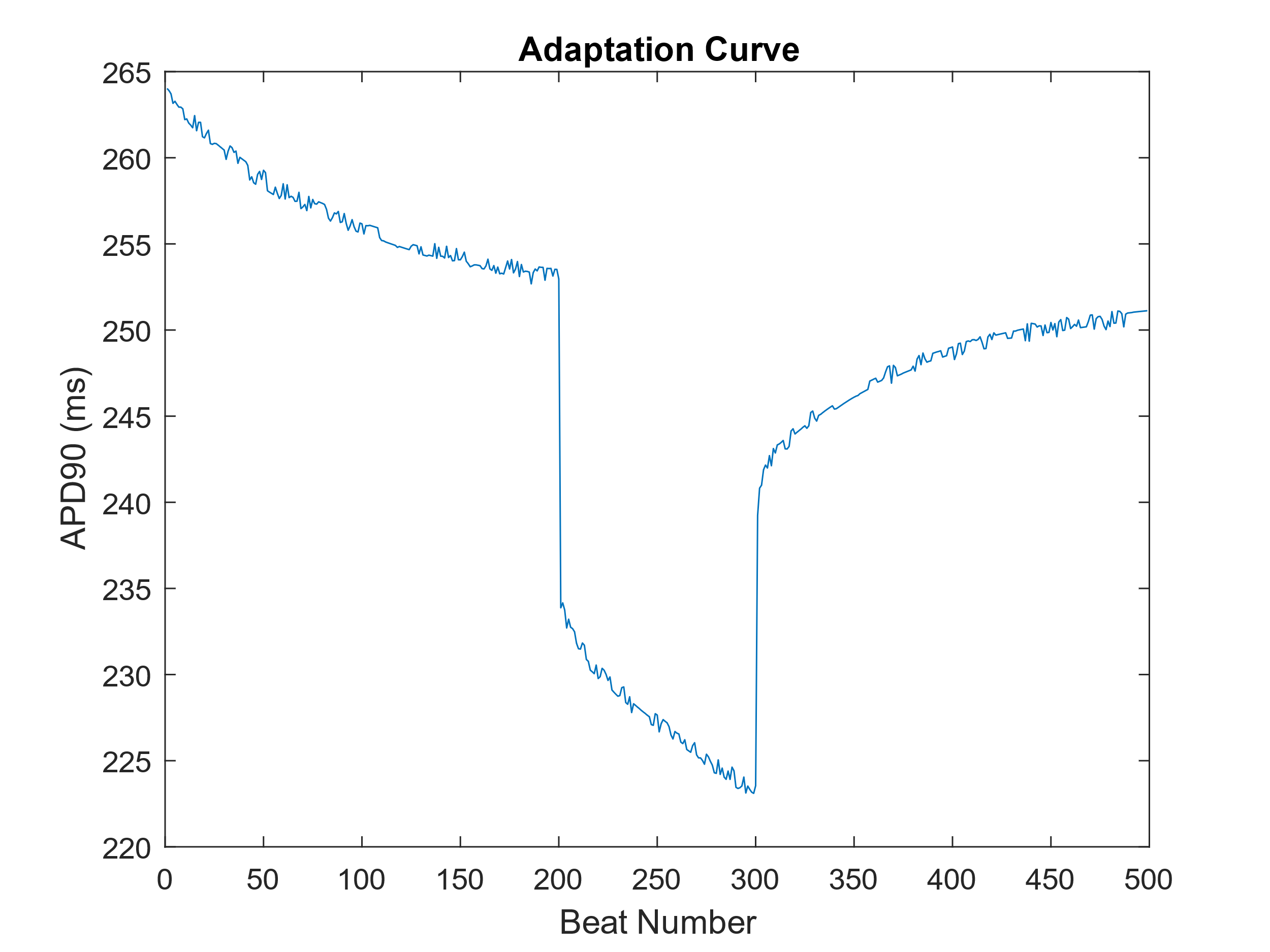
end

train\_time = T2;

train\_Vm = V2;

figure; plot(train\_time,train\_Vm)

Question 2.2:



**Figure 7:** This figure shows the APD adaptation dynamics for varied cycle length. For this adaptation curve, 200 beats of S1 are run at a normal cycle length of 750ms, followed by 100 beats of S2 at a shortened cycle length of 480ms, finished off by 200 beats of S3 at a regular cycle length of 750ms. It can be seen that a shortened cycle length leads to a shortened APD and that the shortening of the APD happens almost instantly, while the recovery of APD as cycle length returns to normal is much slower.

Question 2.2 Code:

X0 = getStartingState('Torord\_endo')

param.bcl = 750; % basic cycle length in ms

beats = 200; % number of beats

param.model = @model\_Torord

param.verbose = true; % printing numbers of beats simulated.

options = []; % parameters for ode15s - usually empty

ignoreFirst = 0

[time, X] = modelRunner(X0, options, param, beats, ignoreFirst);

lastX\_cell=X(end); lastX = cell2mat(lastX\_cell)

X02 = lastX(end,:)

currents = getCurrentsStructure(time, X, param, 0);

%%

param2 = param

param2.bcl = 480; % basic cycle length in ms

beats2 = 100; % number of beats

param2.model = @model\_Torord;

param2.verbose = true; % printing numbers of beats simulated.

%ignoreFirst = beats - 1

ignoreFirst = 0

[time2, X2] = modelRunner(X02, options, param2, beats2, ignoreFirst);

lastX2\_cell=X2(end); lastX2 = cell2mat(lastX2\_cell)

X03 = lastX2(end,:)

currents2 = getCurrentsStructure(time2, X2, param2, 0);

%%

param3 = param

param3.bcl = 750; % basic cycle length in ms

beats3 = 200; % number of beats

param3.model = @model\_Torord;

param3.verbose = true; % printing numbers of beats simulated.

%ignoreFirst = beats - 1

ignoreFirst = 0

[time3, X3] = modelRunner(X03, options, param3, beats3, ignoreFirst);

currents3 = getCurrentsStructure(time3, X3, param3, 0);

%%

T1 = currents.time;

T2 = currents2.time;

T3 = currents3.time;

T12 = [T1;(T2+T1(end))];

T123 = [T1;(T2+T1(end));(T3+T1(end)+T2(end))];

s1s2\_time = T123

s1s2\_Vm = [currents.V;currents2.V;currents3.V];

figure; plot(s1s2\_time,s1s2\_Vm)

%%

TF = islocalmin(s1s2\_Vm);

s1s2\_mod = s1s2\_Vm;

A = 1

for i=1:length(TF)

if(TF(i)==1)

if (A==1)

TF(i)=0;

A=0;

else

A = 1

end

end

end

figure; plot(s1s2\_time,s1s2\_Vm,s1s2\_time(TF),s1s2\_Vm(TF),'r\*')

vec = s1s2\_mod;

cut = TF;

cutsum = cumsum(cut);

cutsum(cut == 1) = NaN; %Don't include the cut indices themselves

sumvals = unique(cutsum); % Find the values to use in indexing vec for the output

sumvals(isnan(sumvals)) = []; %Remove NaN values from sumvals

output = {};

for i=1:numel(sumvals)

Beats\_split{i} = vec(cutsum == sumvals(i)); %#ok<SAGROW>

end

s1s2\_time\_mod = s1s2\_time;

vec = s1s2\_time\_mod;

cut = TF;

cutsum = cumsum(cut);

cutsum(cut == 1) = NaN; %Don't include the cut indices themselves

sumvals = unique(cutsum); % Find the values to use in indexing vec for the output

sumvals(isnan(sumvals)) = []; %Remove NaN values from sumvals

output = {};

for i=1:numel(sumvals)

Time\_split{i} = vec(cutsum == sumvals(i)); %#ok<SAGROW>

end

A = size(Time\_split)

for i = 1:A(2)

x = Beats\_split{i}

y = Time\_split{i}

APD(i) = APD90(x,y)

end

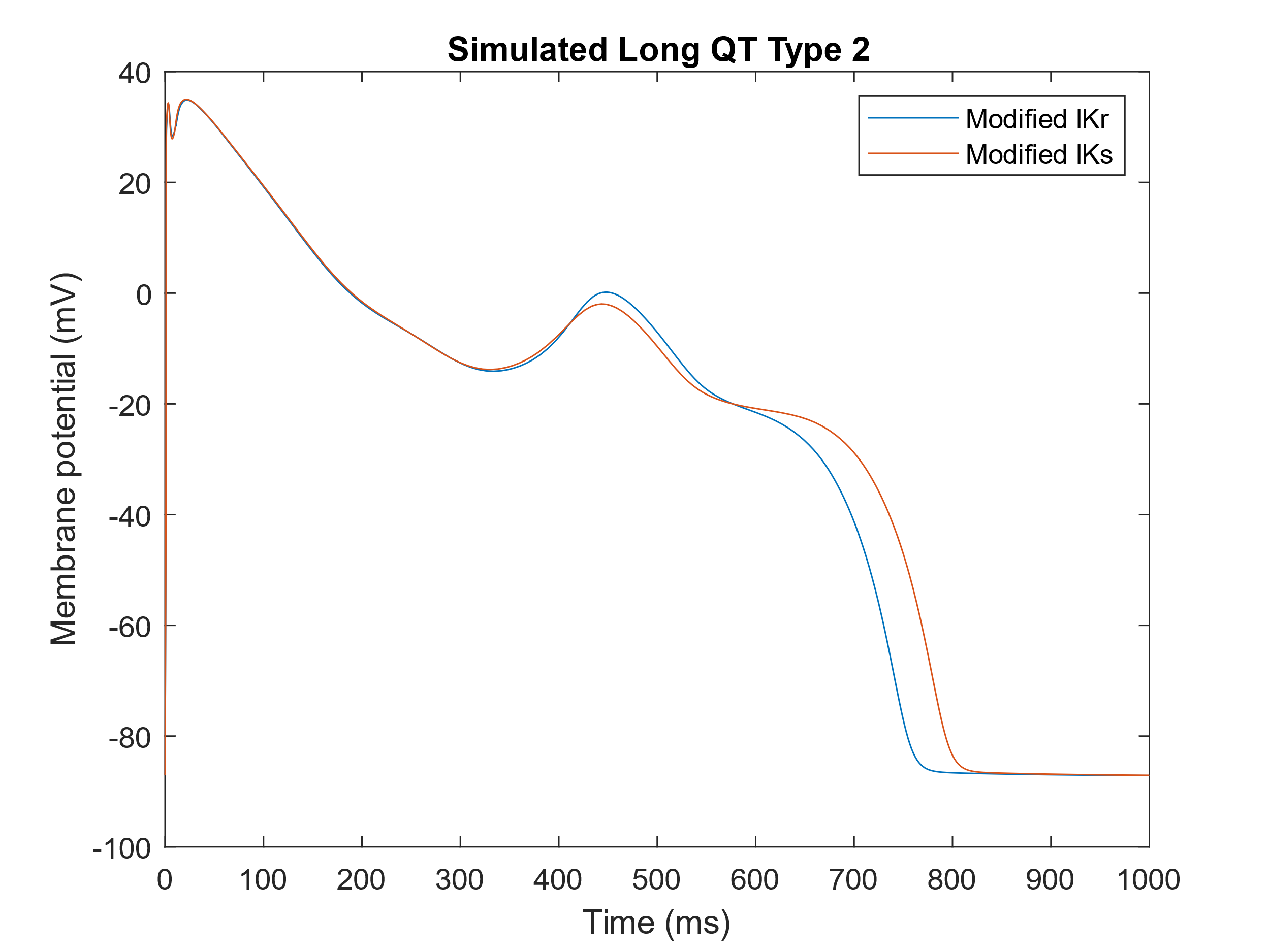
figure; plot(APD(2:end))

xlabel('Beat Number');

ylabel('APD90 (ms)');

title('Adaptation Curve')

Question 2.3:



**Figure 8**: This figure shows what an Early After Depolarization (EAD) for a Cardiomyocyte Action Potential looks like. Two EADs are shown, the first caused by a change in IKr conductance and extracellular calcium (blue) and the second caused by a change in IKr conductance, extracellular calcium, and IKs conductance (orange). For both of the curves, IKr conductance was set to .5, and extracellular calcium concentration is set to 6mM. For the second curve (orange), IKs was also modified to be .5.

At longer cycle lengths, and larger extracellular calcium concentrations, EADs are more probable. Likewise, reducing IKr also increases the probability of EAD formation. This is then coupled with the effects of the reduction of IKs to produce larger and more pronounced EADs, as can be seen in the orange curve of Fig. 1. This effect of IKs on the probability of producing EADs is important as many drugs and pharmaceutical compounds affect the IKs channel, and thus the IKs conductance. This is important as these drugs can cause arrhythmia as an unintended side-effect, which is important to note for groups that are at risk of arrhythmias and fibrillation.

Question 2.3 Code:

% Cardiac model ToR-ORd

% Copyright (C) 2019 Jakub Tomek. Contact: jakub.tomek.mff@gmail.com

%

% This program is free software: you can redistribute it and/or modify

% it under the terms of the GNU General Public License as published by

% the Free Software Foundation, either version 3 of the License, or

% (at your option) any later version.

%

% This program is distributed in the hope that it will be useful,

% but WITHOUT ANY WARRANTY; without even the implied warranty of

% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

% GNU General Public License for more details.

%

% You should have received a copy of the GNU General Public License

% along with this program. If not, see <https://www.gnu.org/licenses/>.

%% This is a simple script which runs the control endocardial model for 100

% beats and plots membrane potential and calcium transient.

% Setting parameters

%clear

% Param is the structure of model parameters that the user may wish to

% change compared to default simulation. The full list is given in the

% function ORdRunner, and it mainly includes cell type, current

% multipliers, extracellular ionic concentrations, or fraction of NCX and ICaL

% localisation in junctional subspace.

param.bcl = 1000; % basic cycle length in ms

param.model = @model\_Torord; % which model is to be used - right now, use model12. In general, any model with the same format of inputs/outputs as model12 may be simulated, which is useful when current formulations are changed within the model code, etc.

param.verbose = true; % printing numbers of beats simulated.

options = []; % parameters for ode15s - usually empty

beats = 5; % number of beats

ignoreFirst = beats - 1; % this many beats at the start of the simulations are ignored when extracting the structure of simulation outputs (i.e., beats - 1 keeps the last beat).

X0 = getStartingState('Torord\_endo'); % starting state - can be also m12\_mid or m12\_epi for midmyocardial or epicardial cells respectively.

%% Simulation and extraction of outputs

% The structure param and other variables are passed to ORdRunner, which is

% an interface between user and the simulation code itself (which is in

% model12.m). The ORdRunner unpacks the structure of parameters given by

% the users, sets undefined parameters to default, and sends all that to

% @model12.

% time, X are cell arrays corresponding to stored beats (if 1 beat is

% simulated, this is 1-by-1 cell still), giving time vectors and state

% variable values at corresponding time points.

[time, X] = modelRunner(X0, options, param, beats, ignoreFirst);

% A structure of currents is computed from the state variables (see the

% function code for a list of properties extracted - also, hitting Tab

% following typing 'currents.' lists all the fields of the structure). Some

% state variables are also stored in a named way (time, V, Cai, Cass) so

% that the user can do most of necessary plotting simply via accessing the

% structure currents as shown below.

currents = getCurrentsStructure(time, X, param, 0);

%% Plotting membrane potential and calcium transient

figure(1);

plot(currents.time, currents.V);

xlabel('Time (ms)');

ylabel('Membrane potential (mV)');

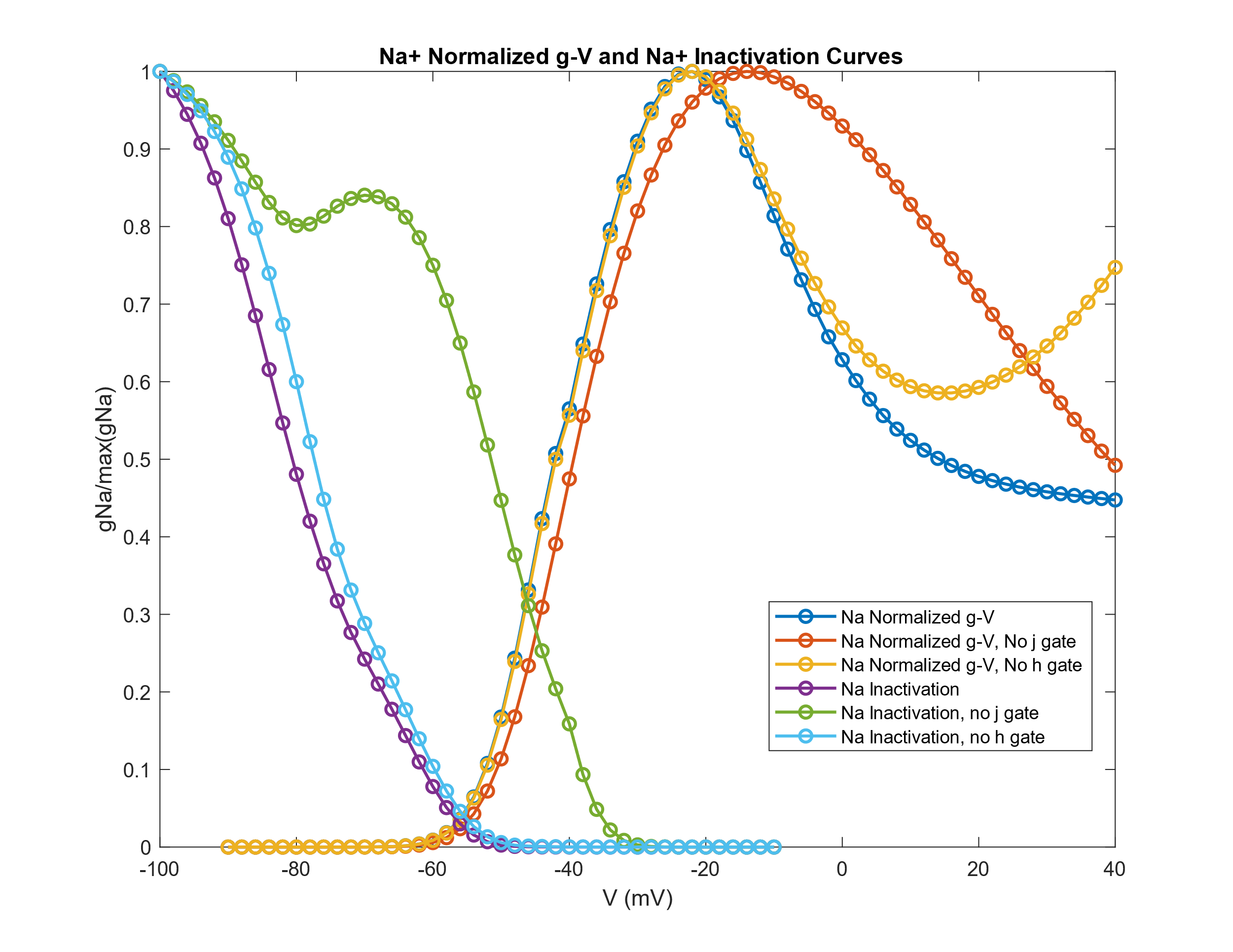
title('Simulated Long QT Type 2')

hold on

plot(currents2.time, currents2.V);

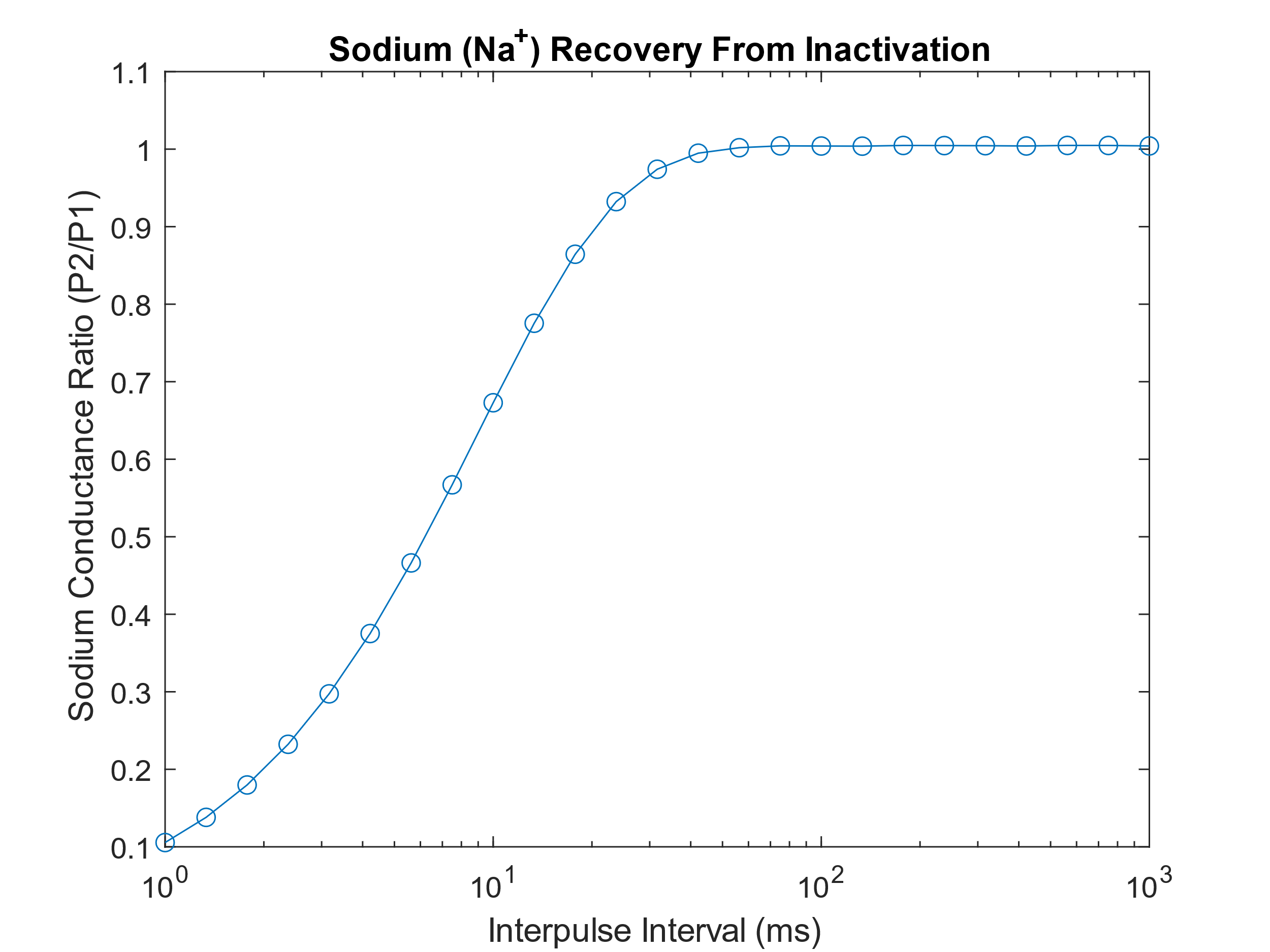
legend('Modified IKr','Modified IKs')

Question 3:



**Figure 9:** This figure shows the Na channel activation and inactivation levels across various voltage levels, with activation represented via the normalized g-V curve. Plots for the sodium activation and inactivation are also shown for various configurations with deactivated gates.

From this figure, it can be seen that the sodium inactivation curve without the h gate is quite similar to the inactivation curve with both the j and h gates. However, once the j gate is removed, the shape of the inactivation curve changes significantly. This is the case as the inactivation is elongated, and there is a second local peak during the inactivation that is not found in the unmodified inactivation curve. Thus, the j gate appears to be most important in deciding the inactivation curve. However, neither seems to affect the activation curve significantly, but the h gate appears to have the greatest impact on activation tendencies at higher voltages.



**Figure 10:** This figure shows the Na channel recovery from inactivation. It can be seen that it takes around 100ms to fully recover the channel function and reset from inactivation.

From Figs. 9 & 10, and the other results discussed above, it is not reasonable to assume that the activation is at the peak at steady state. This is the case as the normalized g-V curves representing activation are at their maximum at around -20mv, and drop off as voltage continues to increase past this point. Thus, it can be concluded that activation is at steady state at the peak is not a reasonable assumption.

Question 3 A,C Code :

function odetest4

V=-90:2:40;

m0=findsteadystatem(-120);

j0=findsteadystatej(-120);

h0=findsteadystateh(-120);

for i=1:length(V)

[t,y]=ode45(@totderiv,[0 10],[m0 j0 h0],[],V(i));

m=y(:,1);

j=y(:,2);

h=y(:,3);

gNatot(i)=max((m.^3).\*j.\*h);

gNanoj(i)=max((m.^3).\*h);

gNanoh(i)=max((m.^3).\*j);

end

figure

plot(V,gNatot/max(gNatot),'-o','LineWidth',1.5)

hold on

plot(V,gNanoj/max(gNanoj),'-o','LineWidth',1.5)

hold on

plot(V,gNanoh/max(gNanoh),'-o','LineWidth',1.5)

clear gNatot

clear gNanoj

clear gNanoh;

V=-100:2:-10;

for i=1:length(V)

[t,y]=ode45(@totderiv,[0 100],[m0 j0 h0],[],V(i))

[t,y]=ode45(@totderiv,[0 10],y(length(y),:),[],-10);

m=y(:,1);

j=y(:,2);

h=y(:,3);

gNatot(i)=max((m.^3).\*j.\*h);

gNanoj(i)=max((m.^3).\*h);

gNanoh(i)=max((m.^3).\*j);

end

plot(V,gNatot/max(gNatot),'-o','LineWidth',1.5)

hold on

plot(V,gNanoj/max(gNanoj),'-o','LineWidth',1.5)

hold on

plot(V,gNanoh/max(gNanoh),'-o','LineWidth',1.5)

xlabel('V (mV)')

ylabel('gNa/max(gNa)')

legend ('Na Normalized g-V','Na Normalized g-V, No j gate','Na Normalized g-V, No h gate','Na Inactivation','Na Inactivation, no j gate','Na Inactivation, no h gate')

title('Na+ Normalized g-V and Na+ Inactivation Curves')

Question 3 B Code:

function odetest\_2

%dm, dh, dj

V = -100:2:40;

%we hold voltage at -120 to find m

m0 = findsteadystatem(-120);

h0 = findsteadystateh(-120);

j0 = findsteadystatej(-120);

%% part b #3

% x axis is interpulse interval on log scale

% y axis is p2/p1 (pulse 2/pulse 1)

timeinterval = logspace(0,3,25);

for i = 1:length(timeinterval);

% setup

[t,ym] = ode45(@derivm,[0 100], [m0], [], -120);

[t, yh] = ode45(@derivh, [0 100], [h0],[],-120);

[t,yj] = ode45(@derivj, [0 100], [j0] , [], -120);

%pulse 1

[t,ym] = ode45(@derivm,[0 500], [ym(length(ym),:)], [], -20);

[t, yh] = ode45(@derivh, [0 500], [yh(length(yh),:)],[],-20);

[t,yj] = ode45(@derivj, [0 500], [yj(length(yj),:)] , [], -20);

m = ym(:,1);

h = yh(:,1);

j = yj(:,1);

maxm(i) = max(m.^3);

maxh(i) = max(h);

maxj(i) = max(j);

%interval

[t,ym] = ode45(@derivm,[0 timeinterval(i)], [ym(length(ym),:)], [], -120);

[t, yh] = ode45(@derivh, [0 timeinterval(i)], [yh(length(yh),:)],[],-120);

[t,yj] = ode45(@derivj, [0 timeinterval(i)], [yj(length(yj),:)] , [], -120);

%pulse 2

[t,ym] = ode45(@derivm,[0 30], [ym(length(ym),:)], [], -20);

[t, yh] = ode45(@derivh, [0 30], [yh(length(yh),:)],[],-20);

[t,yj] = ode45(@derivj, [0 30], [yj(length(yj),:)] , [], -20);

m = ym(:,1);

h = yh(:,1);

j = yj(:,1);

maxm2(i) = max(m.^3);

maxh2(i) = max(h);

maxj2(i) = max(j);

end

P1 = ((maxm/(max(maxm)).^3).\*(maxh/max(maxh)).\*(maxj/max(maxj)));

P2 = ((maxm2/max(maxm2)).^3).\*(maxh2/max(maxh2)).\*(maxj2/max(maxj2));

figure(4)

semilogx(timeinterval, P2./P1, '-o')

title('Sodium (Na^+) Recovery From Inactivation')

ylabel('Sodium Conductance Ratio (P2/P1)')

xlabel('Interpulse Interval (ms)')

x=3

Question 3 Derivative Functions:

function yptot=totderiv(t,y,V)

m=y(1);

minf = 1 / ((1 + exp( -(56.86 + V) / 9.03 ))^2);

taum = 0.1292 \* exp(-((V+45.79)/15.54)^2) + 0.06487 \* exp(-((V-4.823)/51.12)^2);

dmdt=(minf - m) / taum;

yptot(1)=dmdt;

h=y(2);

[ah,bh]=ratesh(V);

tauh = 1 / (ah + bh);

hinf = 1 / ((1 + exp( (V + 71.55)/7.43 ))^2);

dhdt= (hinf - h) / tauh;

yptot(2)=dhdt;

j=y(3);

[aj,bj]=ratesj(V);

tauj = 1 / (aj + bj);

jinf = 1 / ((1 + exp( (V + 71.55)/7.43 ))^2);

djdt=(jinf - j) / tauj;

yptot(3)=djdt;

yptot=yptot';

function ypm=derivm(t,y,V)

m=y(1);

minf = 1 / ((1 + exp( -(56.86 + V) / 9.03 ))^2);

taum = 0.1292 \* exp(-((V+45.79)/15.54)^2) + 0.06487 \* exp(-((V-4.823)/51.12)^2);

dmdt=(minf - m) / taum;

ypm(1)=dmdt;

ypm=ypm';

function yph=derivh(t,y,V)

h=y(1);

[ah,bh]=ratesh(V);

tauh = 1 / (ah + bh);

hinf = 1 / ((1 + exp( (V + 71.55)/7.43 ))^2);

dhdt= (hinf - h) / tauh;

yph(1)=dhdt;

yph=yph';

function ypj=derivj(t,y,V)

j=y(1);

[aj,bj]=ratesj(V);

tauj = 1 / (aj + bj);

jinf = 1 / ((1 + exp( (V + 71.55)/7.43 ))^2);

djdt=(jinf - j) / tauj;

ypj(1)=djdt;

ypj=ypj';

function [ah,bh]=ratesh(V)

ah =(V >= -40) \* (0)...

+ (V < -40) \* (0.057 \* exp( -(V + 80) / 6.8 ));

bh= (V >= -40) \* (0.77 / (0.13\*(1 + exp( -(V + 10.66) / 11.1 )))) ...

+ (V < -40) \* ((2.7 \* exp( 0.079 \* V) + 3.1\*10^5 \* exp(0.3485 \* V)));

function [hinf]=findsteadystateh(V)

[ah, bh]=ratesh(V); % Compute rates at V

hinf = 1 / ((1 + exp( (V + 71.55 + 6)/7.43 ))^2); % Find j-infinity

function [aj,bj]=ratesj(V)

aj= (V >= -40) \* (0) ...

+(V < -40) \* (((-2.5428 \* 10^4\*exp(0.2444\*V) - 6.948\*10^-6 \* exp(-0.04391\*V)) \* (V + 37.78)) / ...

(1 + exp( 0.311 \* (V + 79.23) )));

bj = (V >= -40) \* ((0.6 \* exp( 0.057 \* V)) / (1 + exp( -0.1 \* (V + 32) ))) ...

+ (V < -40) \* ((0.02424 \* exp( -0.01052 \* V )) / (1 + exp( -0.1378 \* (V + 40.14) )));

function [jinf]=findsteadystatej(V)

[aj, bj]=ratesj(V); % Compute rates at V

jinf = 1 / ((1 + exp( (V + 71.55)/7.43 ))^2); % Find j-infinity

% function [am,bm]=ratesm(V)

%

% am=0.32\*(V+47.13)/(1-exp(-0.1\*(V+47.13)));

% bm=0.08\*exp(-V/11);

function [minf]=findsteadystatem(V)

minf=1 / ((1 + exp( -(56.86 + V) / 9.03 ))^2); % Find m-infinity