Big Challenge #A

Graphical User Digital Software

BME-513

ILLINOIS TECH

ILLINOIS INSTITUTE OF TECHNOLOGY

```
182   % eval(sprintf('%s',plotstrcell{plotflag}));
183
184   hold on;
185
186   if plotflag == 1
187       delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188       hold(ax, 'off');
189
190   end
```

Ignacio Hidalgo Power

Index:

Part 1 **3**
How is the code structured? 3
LLMs as Software Consultants. **8**
Understanding code through a simple prompt 8
Understanding code through a more complex prompt 9
Conclusion of simple prompt 10
Part 2 **11**
Adding the Plot to the GUI 11
FEBLAB A Task 2 11
Software Dev Considerations 13
Windows vs linux 13
Version Control 13

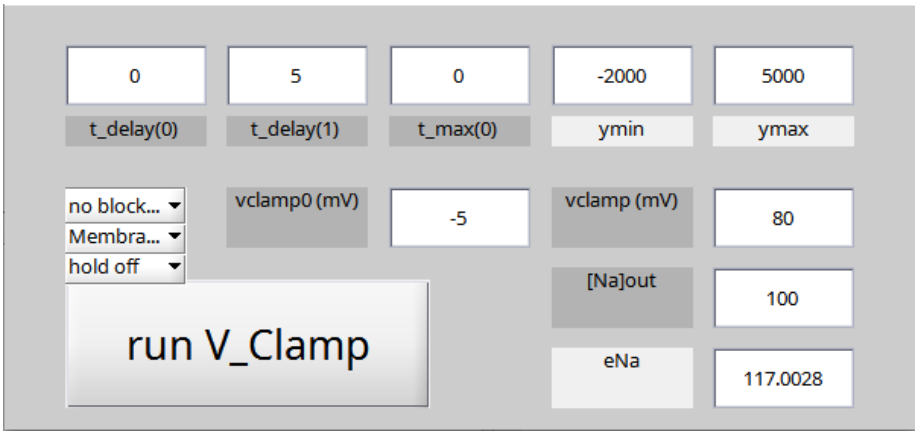
```
182   % eval(sprintf('%s',plotstrcell{plotflag}));
183
184   hold on;
185
186   if plotflag == 1
187       delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188       hold(ax, 'off');
189
190   end
```

Part 1

How is the code structured?

The code used in our H&H simulation is developed on two matlab files, vclamp.m and vcgui.m. The vcgui.m will be the Graphical User Interface (GUI) through which we change the variables for our experiment (Membrane Potential, Sodium Concentration outside the membrane, etc).

```
182   % eval(sprintf('%s',plotstrcell{plotflag}));
183
184   hold on;
185
186   if plotflag == 1
187       delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188       hold(ax, 'off');
189
190   end
```



Once we hit the “run V_Clamp” button in the GUI we will call the function declared on our vclamp.m file to run. The function will take the variables values that we gave in the vcgui, and use them in the implementation of the experiment. It is in vclamp where we have implemented the mathematical model that H&H made on their experiment.

In the vclamp code we first get the variables from the gui, using the methods findobj() and get().

```
19   EditHandle1=findobj(gcbf,'Tag','EditText1');
20   V_clamp1=str2num(get(EditHandle1,'String'));
21   EditHandle1a=findobj(gcbf,'Tag','EditText1a');
22   use_modified = get(EditHandle1a,'Visible');
23   EditHandle2=findobj(gcbf,'Tag','EditText2');
24   tdel1=str2num(get(EditHandle2,'String'));
25   EditHandle3=findobj(gcbf,'Tag','EditText3');
26   tend=str2num(get(EditHandle3,'String'));
```

From these variables we are getting information that is both important to the implementation of the model as for its visualization. After that we start the initialization of variables for the experiment that are constant, like the conductance for the different ions or the Nernst potential for potassium and sodium.

```
182  % eval(sprintf('%s',plotstrcell{plotflag}));
183
184  hold on;
185
186  if plotflag == 1
187      delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188      hold(ax, 'off');
189
190  end
```

```
73   DT = 0.05;      %time step
74   T_MAX = tend;   %Maximum Time
75   t = 0:DT:T_MAX;
76
77   V_REST = 0.0;   %Initial Membrane Voltage
78
79   G_NA = 120.0;   % mS/cm^2 % Sodium Conductance
80   G_K = 25.0;     % mS/cm^2 % Potassium Conductance
81   G_LEAK = 0.3;   % mS/cm^2 % Leakage Conductance will not be covered in 445/545.
82
83   E_K = -8.0;         % mV  % Nernst Potential Potassium
84   E_LEAK = 10.613;    % mV  % Leakage Voltage will not be covered in 445/545.
```

After that we declare the activation and inactivation variables ‘m’ ‘h’ and ‘n’ using support functions that are provided at the bottom of the file.

```
113   clear v  m  h  n            % clear old varibles
114   v = V_REST;                 % initial membrane voltage
115   m = alpha_m(v)/(alpha_m(v)+beta_m(v));  % initial (steady-state) m
116   h = alpha_h(v)/(alpha_h(v)+beta_h(v));  % initial (steady-state) h
117   n = alpha_n(v)/(alpha_n(v)+beta_n(v));  % initial (steady-state) n
118   idel=floor(tdel1/.05);
119   idel2=floor(tdel2/.05);
```

```
217   % Support functions are provided below:
218   %*********************
219   function rate = alpha_h(v)
220       rate = zeros(size(v));
221       rate = 0.07*exp(-v/20.0);
222
223   %************************
224   function rate = beta_h(v)
225       rate = zeros(size(v));
226       rate =  1.0 ./ (exp((-v+30.0)/10.0) + 1.0);
227
228   %************************
229   function rate = alpha_m(v)
230       rate = zeros(size(v));      % DEFAULT RATE TO ZERO
231       rate = 0.1.*(25.-v)./(exp((25.-v)./10)-1);
232
233   %**********************
234   function rate =  beta_m(v)
235       rate = zeros(size(v));
236       rate = 4.0*exp(-v/18.0);
237
238   %************************
239   function rate = alpha_n(v)
240       rate = zeros(size(v));
241       rate = 0.01.*(10.-v)./(exp((10.-v)/10)-1);
242
243   %**********************
244   function rate = beta_n(v)
245       rate = zeros(size(v));
246       rate = 0.125*exp(-v/80.0);
```

```
182   % eval(sprintf('%s',plotstrcell{plotflag}));
183
184   hold on;
185
186   if plotflag == 1
187       delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188       hold(ax, 'off');
189
190   end
```

Once all that is ready we create two arrays to store the data of the changes of conductance in sodium and potassium and begin the for loop to see the change in the membrane current through out the specified time

```
121   g_K = [];
122   g_Na = [];
123
124   for i=1:length(t)
```

The next steps are recalculating the steady state values, constants, gating variables and conductance values depending on the iteration of the loop. These change throughout iterations thanks to the if else statements which change the membrane voltage depending on the time step.

```
124     if i<=idel
125         V=V_REST;
126     elseif i >= idel2
127         V=V_clamp1;
128     else
129         V=V_clamp0;
130     end
131
132     m_inf = alpha_m(V)/(alpha_m(V)+beta_m(V));
133     h_inf = alpha_h(V)/(alpha_h(V)+beta_h(V));
134     n_inf = alpha_n(V)/(alpha_n(V)+beta_n(V));
135     tau_m = 1/(alpha_m(V)+beta_m(V));
136     tau_h = 1/(alpha_h(V)+beta_h(V));
137     tau_n = 1/(alpha_n(V)+beta_n(V));
138     M     = m_inf-(m_inf-m)*exp(-(i-(idel+1))*DT/tau_m);
139     H     = h_inf-(h_inf-h)*exp(-(i-(idel+1))*DT/tau_h);
140     N     = n_inf-(n_inf-n)*exp(-(i-(idel+1))*DT/tau_n);
141     gNa   = G_NA * M^3 * H;
142     gK    = G_K  * N^4;
143
144     if i >= idel2
145         V=V_clamp1;
146     else
147         V=V_clamp0;
148     end
```

When the new steady state values, constants and membrane voltage have been calculated, that's when we can use them to get the membrane current, implementing the mathematical model taught on lecture notes 3. Once the membrane current has been calculated we store the sodium and potassium current for that step in time and the conductance of those ions in the previously declared arrays.

```
182   % eval(sprintf('%s',plotstrcell{plotflag}));
183
184   hold on;
185
186   if plotflag == 1
187       delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188       hold(ax, 'off');
189
190   end
```

```
150     Im(i) = (gNa*(V-E_NA) + gK*(V-E_K));        %Membrane Current
151     %++BME445/545 *** USEFUL CODE COMMENTED OUT (Used in LAB_B)
152     %Im(i) = (gNa*(V-ENA) + gK*(V-EK) + GLEAK*(V-ELEAK));
153     %Im(i) = gNa*(V-ENA) + gK*(V-EK)+GLEAK*(V-ELEAK)+(v(i)-v(i-1))/DT;
154     %Im(i) = gNa*(V-ENA) + gK*(V-EK)+(v(i)-v(i-1))/DT;
155     %--2022.01.24
156     INa(i)= gNa*(V-E_NA);
157     IK(i) = gK*(V-E_K);
158     g_Na(i)= gNa;
159     g_K(i)= gK;
160
161 end
```

The next part of the code is the display of the experiment using a plot. The display will only be shown depending on what value the variable ‘displayflag’ got from the GUI.

```
171 % Actual plotting.
172 if displayflag == 1
173
174     plot(t,Im,'b-');
175     axis([0 T_MAX ymin ymax]);
176     title('Measured Membrane Current vs Time Plot');
177     xlabel('time (mSec)');
178     ylabel('Im (mA/sqcm)');
179
180 elseif displayflag == 2
181
182     plot(t,g_K,'b-');
183     axis([0 T_MAX ymin ymax]);
184     title('Potassium Conductance vs Time Plot');
185     xlabel('time (mSec)');
186     ylabel('g_K ');
187
188 end
```

```
182 % eval(sprintf('%s',plotstrcell{plotflag}));
183
184 hold on;
185
186 if plotflag == 1
187     delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188     hold(ax, 'off');
189
190 end
```

LLMs as Software Consultants.

As an experiment we will try different LLM publicly available to see how far away the technology is as a replacement for software consultants. We will try 4 models.

- OpenAI ChatGPT
- Google Gemini
- Mistral Lechat
- Microsoft Copilot

Except Copilot, all other models are free to use, although with a limited amount of queries. One of the interests of the experiment is to see how these different models with different natures explain the code. Mistrals model is “open source” while OpenAI’s and Google’s are proprietary and closed. Copilot is embedded into the IDE, having access to the whole file or even a bundle of files.

Understanding code through a simple prompt

The first try we will just feed the LLMs with the code of vclamp.m and the prompt of “Explain this code”.

ChatGPT: The model seemed to be able to reason what the code was for, explaining that it was an implementation of the Hodkin and Huxley model for neural function. The model then explains the code using what it determines to be ‘Key sections’. In the for loop it uses a general conception to explain the concepts of what happens inside. While brief it gives an okay explanation of the code, however it presumes too much of the reader, not explaining why parameters are changed or how the changes in the system affects them.

Gemini: This model was less organized but more thorough. If ChatGPT chose to explain the code in large sections Gemini chose to go in order and with a lower level of abstraction when generating the code segments. While ChatGPT had ignored headers and comments Gemini chose to give a brief explanation of the problem, identifying the need for vcgui to get parameters in the system. It also tackles loops and switches. However the explanations are mostly conceptual, not showing the explicit code. It does give some improvement recommendations, but only high level ones, so the person will need to take the suggestions and guess how to implement them.

Le Chat: The model gives the explanation on a high level of abstraction. Like the other it divides the code in sections, and briefly explains what the section does in the experiment.

```
182   % eval(sprintf('%s',plotstrcell{plotflag}));
183
184   hold on;
185
186   if plotflag == 1
187       delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188       hold(ax, 'off');
189
190   end
```

The problem with the explanation is that its very high level, not fully explaining the code, instead focusing on the experiment.

Copilot: The model explains the code on a more grounded approach. This model focuses more on the technical aspects of the code, recognizing that vclamp is for the experiment but only briefly talking about it. The code is separated by sections too, and while the explanation of what each section does is clear and with parts of the code, it does not give an idea of how each section works.

Understanding code through a more complex prompt

The LLMs will now be given a new prompt. “This code is for the Hodgkin-Huxley experiment. Explain the following code as if you were a software consultant working with a Biomedical Engineer and had to explain how this code operates. Keep in mind that the Biomedical Engineer has a brief knowledge of programming but none on software development, maintenance or good practices. One of the goals is that they comprehend how the code implements the experiment”.

ChatGPT: The model has decided with this new prompt to increase the level of abstraction when explaining the code. As in the previous case it explains sections of the program, although this time without showing any code. The level of detail explaining the loops and what is computed inside has increased. Another improvement is that it explains what each section does in the Hodgkin-Huxley experiment, making the Biomedical Engineer have an easier time understanding what that section does for the experiment. However, the lack of code explanation makes the implementation harder to follow for an inexperienced programmer.

Gemini: This model has also increased the level of abstraction when explaining the model. It focuses mainly on what the overall code has to do, explaining what the experiment is trying to accomplish and in what order. The explanation of each selected section is clear, although no code is visible and understanding where each section is implemented in the code is not clear.

Le Chat: The model has made a remarkably similar explanation to the simpler prompt. The explanations of the sections have been expanded but no code is shown and implementations are explained in a poor way compared to the other models.

Copilot: Like Mistrals models it expands briefly on the previous answer. The explanation of the model is still grounded on sections, increasing the explanation of what each code

```
182   % eval(sprintf('%s',plotstrcell{plotflag}));
183
184   hold on;
185
186   if plotflag == 1
187       delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188       hold(ax, 'off');
189
190   end
```

segment does. However the answers the model provides still seem to be directed to a user that has more of a programming background than a BME, despite the prompt given.

Conclusion of simple prompt

The difference in prompt length and complexity had an importance but not as significant as it was expected. While the models had no trouble understanding the purpose of the code, the explanations given were mostly vague. If our goal was to explain the code to a BME who already knew what the purpose was, but was trying to understand the implementation, this might have not been fully achieved. However, time spent understanding the code by a seasoned BME could decrease.
Of the models, it seems that, excluding Copilot, ChatGPT is the better one for explaining code to an unseasoned programmer, with Gemini behind and the open source one giving the least explanation. For someone with a better understanding Copilot would be better, explaining less about the experiment and instead focusing more in the code and how it works.
In conclusion, while these models might not be the best for BME to understand the code, for software consultants they could be very useful, aiding in understanding how the experiment is implemented and in what areas of the file.

```
182   % eval(sprintf('%s',plotstrcell{plotflag}));
183
184   hold on;
185
186   if plotflag == 1
187       delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188       hold(ax, 'off');
189
190   end
```

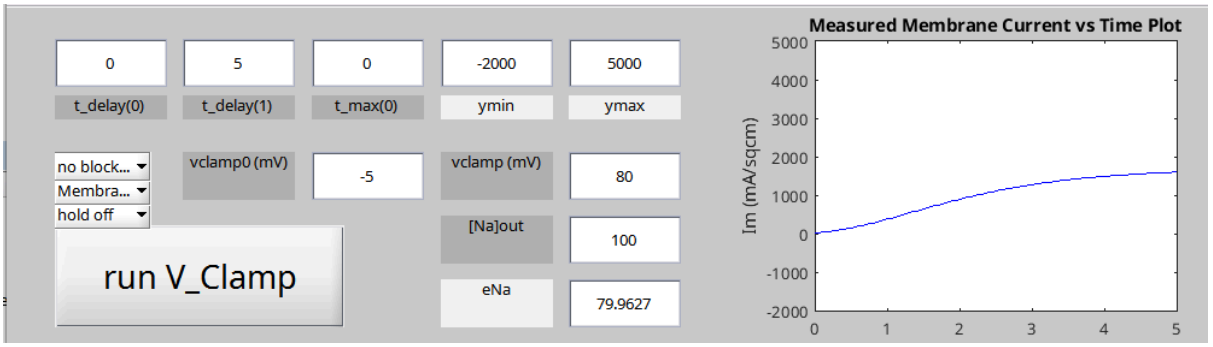
Part 2

Adding the Plot to the GUI

The GUI has been modified to include the plot generated by vclamp. To do this we made a global variable that was to be the figure() that generated the GUI in vcgui. In vclamp, we instantiated this variable, and when generating the plot, we gave the figure() the global variable as a parameter, that way the plot would be drawn on the GUI, after that we just needed to adjust the position of the plot and the GUI so that it wouldn't interfere with the other objects in the GUI

```
5   global a;
```

```
167  %% BME 445/545 - LAB_A Voltage Clamp Experiment Plotting
168  figure(a)
169
170  axes('Position', [(350 + (80*4))/1000, 0.1, 0.3, 0.8]);
```

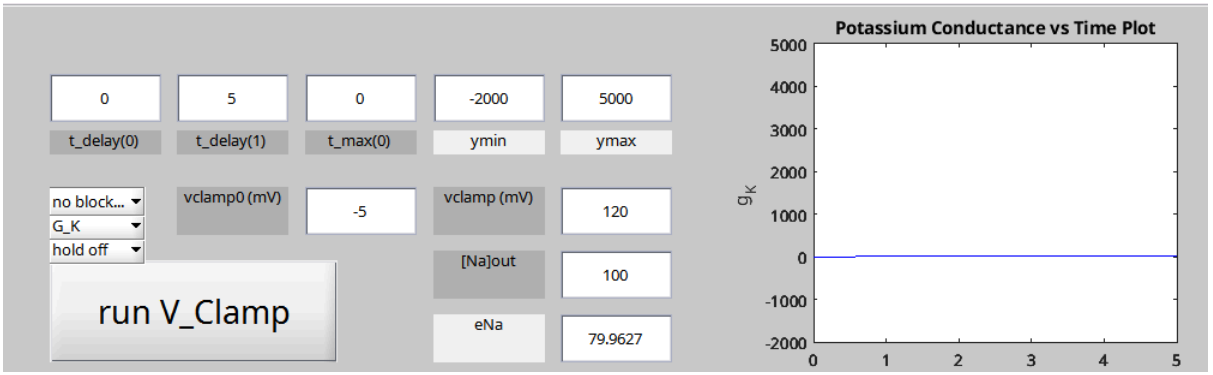


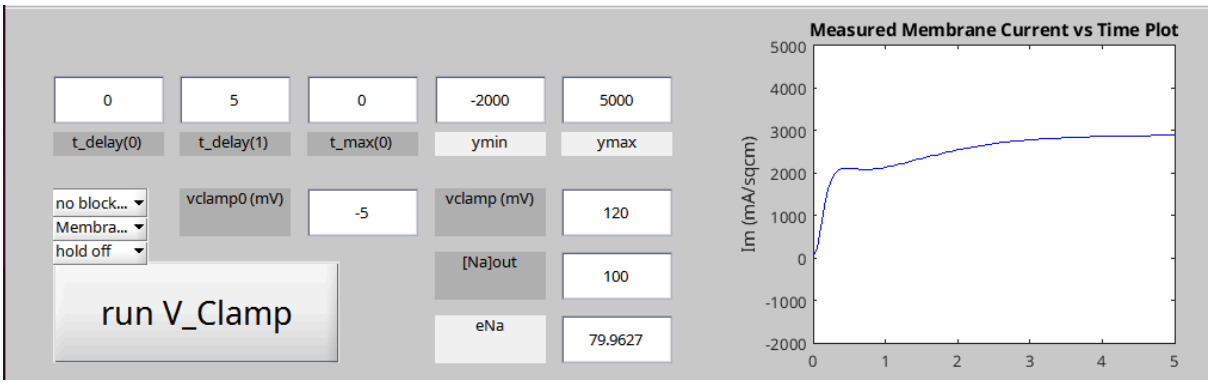
FEBLAB A Task 2

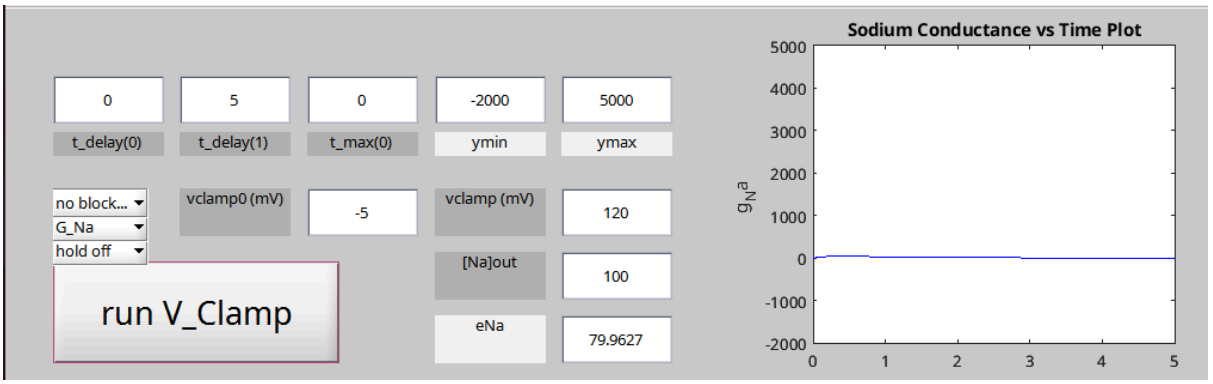
For this lab our task was to first, implement a way to visualize the plot with Potassium Conductance and Sodium Conductance, and secondly to visualize the graph with multiple mV.

The plotting of the conductances was fairly easy to realize. We added the two choices to the popup menu and modified vclamp to change the plot after getting the data from the vcgui.

```
182  % eval(sprintf('%s',plotstrcell{plotflag}));
183
184  hold on;
185
186  if plotflag == 1
187      delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188      hold(ax, 'off');
189
190  end
```







The visualization of the graph however was unsuccessful. We tried to use the hold on functions but it didnt seem to work. It seems that maybe because of the matlab version there might have been some problems in the implementation

```
182   % eval(sprintf('%s',plotstrcell{plotflag}));
183
184   hold on;
185
186   if plotflag == 1
187       delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188       hold(ax, 'off');
189
190   end
```

Software Dev Considerations

Windows vs linux

When developing software cross platform issues are an important matter to take into consideration. While in linux we have the dexterity and flexibility that comes with the system, windows comes with a large user base, supported software and easy integration.
Taking into account the person's profile that will work on these projects, a biomedical engineer with some background in software development but probably not a large affinity, the use of Windows might be preferred.
Another advantage of Windows is the ease of use when trying to use hardware specific acceleration, since some of the drivers required might not be available for Linux and the installation might require an experienced user.

Version Control

If the used system were windows and the team developing the software were composed of several users the easiest implementation of a version control would be through the use of Git with Github. Each user could do a clone of the main code. There would be two branches, the production branch and the development. All users would work with the development branch, doing commits and pull requests when changes are implemented, allowing for easy version control. There would be a software manager ensuring that when significant changes were made the development and production branch merged without problems.

```
182   % eval(sprintf('%s',plotstrcell{plotflag}));
183
184   hold on;
185
186   if plotflag == 1
187       delete(findall(findall(gcf,'Type','axe'),'Type','text'));
188       hold(ax, 'off');
189
190   end
```