Introducing A Multi-Compartment Modeling Tool With the Potential to Analyze Branchpoint Propagation of Action Potentials

By

Yuxuan Wu

Astrid Prinz
Advisor

Biology

Astrid Prinz

Advisor

Shawn Hochman

Committee Member

Gordon Berman

Committee Member

Samuel J. Sober

Committee Member

2023

Introducing A Multi-Compartment Modeling Tool With the Potential to Analyze Branchpoint Propagation of Action Potentials

By

Yuxuan Wu

Astrid Prinz

Advisor

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Biology

2023

Abstract

Introducing A Multi-Compartment Modeling Tool With the Potential to Analyze Branchpoint Propagation of Action Potentials

By Yuxuan Wu

This thesis presents a multi-compartment modeling tool implemented in Python that provides the flexibility and configurability needed to simulate action potential propagation in complex axon structures involving branch points and electrical coupling. The model's development is motivated by the clinical significance of branch points in modulating sympathetic output and thus contributing to the pathophysiology of patients with spinal cord injuries (SCI). Sympathetic preganglionic neurons (SPNs) diverge to form axonal collaterals that synapse onto the sympathetic ganglionic chain with redundancy. The axonal branch points of such a complex network are susceptible to factors such as temperature undulations, axon geometry, and ion channel plasticity post-SCI, thereby causing conduction blocks in relaying action potentials through the sympathetic route. While similar simulation tools already exist, the significance of our model lies in its flexibility and specificity to our experimental studies on the topic. It allows for adjusting various parameters relating to axon dimensions and ion channel dynamics, which facilitates computational investigations of axonal modification of sympathetic signals. Using our modeling tool, we built two models–a Y-branch model simulating a main axon branch bifurcating into two daughter segments and a parallel-axon model with two axons coupled at the middle through gap junctions. Preliminary results were gathered to show the validity of our modeling tool and interesting observations of how axon geometry, temperature, and gGABA impact spike propagation across branch points.

Introducing A Multi-Compartment Modeling Tool With the Potential to Analyze Branchpoint Propagation of Action Potentials

By

Yuxuan Wu

Astrid Prinz

Advisor

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Biology

2023

Acknowledgments

I am grateful for Dr. Prinz's guidance and support throughout my honors thesis project. Her help with the mathematical modeling part of the project and willingness to invest time and energy in my work were invaluable. I could not have completed this project without her expertise and dedication.

Additionally, I would like to express my appreciation to the other members of my Honors committee, Dr. Berman, Dr. Sober, and Dr. Hochman. I am truly fortunate to have had such a talented and committed group of scholars supporting me throughout this process. Thank you for your time and effort in helping me achieve this important milestone.

Table of Contents

List of Figures

Introduction

Spinal cord injury (SCI) is a life-altering condition that often leads to sensorimotor and autonomic deficits (Partida et al., 2016). Due to pathologies such as traumatic injuries, inflammations, and tumors, SCI disrupts the neural conduit connecting higher control centers in the brain to the peripheral nervous system (PNS), thus resulting in a range of sensory, motor, and autonomic dysfunctions such as partial or complete paralysis, neuropathic pain, dysesthesia, cardiac dysrhythmia, and misregulated hemodynamics (Guest et al., 2022).

Compared with the abundance of studies on the sensorimotor impact of SCI, its autonomic symptoms are less popular targets for research. However, the clinical importance of understanding post-SCI autonomic malfunctions has been increasingly emphasized due to its prevalence. About 48%-60% of patients inflicted with SCI above the thoracic 6th vertebral level (T6) displayed symptoms associated with autonomic dysreflexia (AD) (Anjum et al., 2020). Specifically, damage to the descending autonomic pathways can result in deregulated or diminished sympathetic output from the spinal cord, thus leading to parasympathetic dominance and hemodynamic conditions such as hyper- or hypotension, reduced urinary function, and decreased gastrointestinal activity (Guest et al., 2022).

Aiming to unravel the physiological mechanism for sympathetic deregulation resulting from SCI, we are specifically interested in the axonal modulation of sympathetic output from sympathetic preganglionic neurons (SPNs), the last arbiters of CNS sympathetic signals leaving the spinal cord into the PNS (Deuchars & Lall, 2015). SPN axons tend to split at branch points to form extensive divergence before synapsing onto postganglionic cells in the sympathetic chain.

Although postganglionic neurons vastly outnumber SPNs, SPN axonal divergence amplifies CNS signaling by issuing multi-segmental projections with redundancy (approximately 200:1 ratio of SPN divergence vs. sympathetic ganglionic neurons in humans & 14:1 in mice) (McLachlan, 2003). As branch points have been observed to modulate signal conduction in somatosensory systems (Lucas-Osma et al., 2018; Wall, 1994, 1995; Wall & McMahon, 1994), SPN branch points might be involved in controlling signal conduction or failure to conduct to axonal extensions, thus making them exciting targets to study sympathetic control.

However, branch points have long been regarded as having little impact on the sympathetic outflow. Per existing studies on the *ex vivo* guinea pig thoracic paravertebral chain, spike propagations past SPN branch points have been assumed to be faithful without modification that could constitute a conduction block. Nonetheless, the observation was derived from experiments at room temperature (T), significantly lower than average body T (Blackman & Purves, 1969; Lichtman et al., 1979, 1980; Njå & Purves, 1977; Purves & Lichtman, 1980). This discrepancy may have led to wider neuronal spikes with larger magnitudes, thus masking the possibility of branch point failure (Hodgkin & Katz, 1949; Pekala et al., 2016). Therefore, our research collaborators in the Hochman Lab–Shawn Hochman, Ph.D., Yaqing Li, Ph.D., and Mallika Halder–decided to study spike conduction or failure at SPN branch points in adult mice *ex vivo* spinal cords.

Several hypotheses were proposed regarding the physiological mechanisms through which branch points could modify sympathetic gain control. Extrasynaptic α5-containing $GABA_A$ receptors ($GABA_ARs$) have been implicated in axonal signal modulations (Lucas-Osma et al., 2018; Trigo et al., 2008; Wall, 1995). As SPNs also have $GABA_A$Rs (Deuchars et al., 2005), such receptors, if present at branch points, can alter the neural excitability at branch points due to the chloride reversal potential being altered by post-SCI downregulation of potassium-chloride co-transporters (Huang et al., 2016; Lu et al., 2008), thus modifying the signal output to postganglionic neurons. Moreover, branch point conduction might be sensitive to changes in body T because slight T elevation can contribute to conduction failure (Pekala et al., 2016). This hypothesis is further supported by the widespread dysfunctional thermoregulation in SCI patients (Attia & Engel, 1983; Cabanac & Massonnet, 1977; Handrakis et al., 2015; *Hypothermia in Patients With Chronic Spinal Cord Injury - PMC*, n.d.).

Beyond branch point control, we are interested in axonal modulations in general. Ephaptic control, for instance, gives rise to synchronized spiking due to unmyelinated axons bundled together in a shared electrical environment (Debanne, 2004). This mechanism can impact axonal output, as mouse SPNs are predominantly unmyelinated axons with small diameters (mean is 0.4 μm and low is 0.1 μm in branching segments) (Lewis & Burton, 1977). Electrical coupling between axons, too, can lead to synchronized firing through gap junctions and, thus, modified signal output (Debanne, 2004). As pharmacological evidence demonstrates the expression of gap junctions, electrical coupling between tSPN axons is also worth investigating.

Due to the size of axonal branches limiting direct inquiries through measurements such as voltage recordings, we have developed a modeling tool that is both flexible and configurable, specifically designed to ascertain the modulation of signal propagation in axonal branches. To accommodate the complex geometry of axons involving branch points and gap junctions and to be able to simulate pharmacological interventions impacting various ion channel dynamics, we built the modeling tool in Python almost from scratch, besides some numerical analysis packages. Building upon the classic work by Wilfried Rall and others on passive cable theory (Rall, 1959), this model provides a unique opportunity to study axonal modulation and gain insights into the underlying mechanisms of neuronal communication.

Using our modeling tool, we created two multi-compartment model versions with different axonal configurations to complement some experimental hypotheses. The first one is a Y-branch model that simulates spike propagation past simple branch points where a main axon branches into two daughter collaterals. Here, features like gGABA, T, and axon diameter were manipulated, and spiking results were collected and visualized. The second model seeks to simulate two axons electrically coupled in the middle through gap junctions. Overall, we found some preliminary results from the Y-branch model regarding the effect of gGABA, temperature, and geometry on branch point failure. We also made exciting observations and tested the validity of our modeling tool through the electrical coupling model.

Method

Our modeling tool utilizes Python functions and class data structures to enhance modularization, thus making it easy to fulfill various simulation needs with minor modifications. Our model accommodates functionalities such as changing axon geometries, updating ion channel dynamics, changing integration parameters based on branching geometry, and much more.

The most prominent Python class in our modeling tool, “axoBranch,” constitutes a detailed template for creating an axon. With various axonal parameters embedded in the class, such as ion channel conductance, the membrane potential, and the axon length, an axon branch can be conveniently instantiated with parameters tailored explicitly to simulation goals by invoking the class's constructor.

Following is a complete set of parameterized features:

Axon geometry and electric properties:

axon length,

diameter,

compartment length,

membrane potential,

axial resistance

Ion channel gating variables:

mNa, (the activation variable of the Na channels)

hNa, (the inactivation variable of the Na channels)

nKd,

mA,

hA,

mH,

mM,

mCaL,

hCaL,

mKCa

Ion channel conductance and reversal potentials

Leak current:

G_leak

E_leak

Fast sodium current

G_Na

E_Na

Delayed-rectifier potassium current

G_Kd

E_K

GABA chloride current

G_GABA

E_GABA

Fast transient potassium current

G_A,

E_A,

Hyperpolarization-activated inward current

G_H,

E_H,

Slow and non-activating potassium current

G_M,

E_M

Persistent calcium current

G_CaL

E_CaL

Calcium-dependent potassium channels

G_KCa

E_KCa

All the ion channel conductance, reversal potentials, and gating variable dynamics are adapted from a single-compartment model of postganglionic cells except for the GABA chloride current (McKinnon et al., 2019). The following is a general description of our modeling tool and the two models created using it. Please refer to the Appendix for Python code files for anyone interested in implementation details.

Regarding structural representation, our modeling tool represents axons in a multi-segment approach achieved through Python list and Numpy array data structures, where each list or array stores a parameter of the axon, and the number of entries in each list or array corresponds to the total number of compartments in the axon branch. For instance, suppose axon a1 is appropriately created by calling the "axoBranch" class, then a1.V refers to the list of membrane voltage entries, one for each compartment indexed from 0 to a1.n_comp - 1 (a1.n_comp stores the number of compartments in axon a1). Similarly, any parameter associated with the compartment indexed x in axon a1 can be accessed through such syntactical structure: a1.<featureName>[x].

The challenging part of simulating signal propagation in complex axonal networks, such as branch divergence, is to find a way to incorporate axial currents from a variable number of compartments contiguous with a branch point segment. Therefore, we adopted the conductance-based multi-segment integration regime. Instead of updating currents, individual gating variables are updated for each simulation timestep, and change in voltage is calculated by incorporating conductance and axial currents from neighboring compartments into an intermediate set of integration variables labeled A, B, C, and D in the equation below (Dayan & Abbott, 2001).

Here is the general scheme for updating the voltage for the compartment μ at time t (this method enhances simulation speed and stability by estimating one timestep Δt later than the current timestamp):

$$\Delta V_{\mu} = (A_{\mu} V_{\mu-1}(t + z\Delta t) + B_{\mu} V_{\mu}(t + z\Delta t) + C_{\mu} V_{\mu+1}(t + z\Delta t) + D_{\mu}) * \Delta t \qquad (1)$$

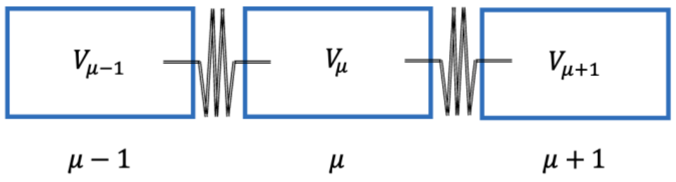

Figure 1. Illustration of integrating axial currents from neighboring compartments into the process of voltage update. The compartment in the middle, μ, has two neighboring segments and receives axial inputs from them.

Without explaining the mathematical details of the integration method, we will focus on how our model modularizes and thus simplifies the integration process. To calculate ΔV_{μ} for the middle compartment demonstrated in Figure 3, the integration function in our model "IntegParamUpdate" receives input that acknowledges each neighboring compartment ($\mu - 1$ and $\mu + 1$) and pairs up the trans-compartment conductance with the specific neighboring compartment's voltage at the previous timestep. For example, incorporating current from the $\mu - 1$ compartment, $(V_{\mu-1}, g_{\mu,\mu-1})$ is considered a conductance-voltage pair. These paired entries will be absorbed into integration parameters such as A and C (eq. 1) and used later to derive ΔV_{μ}. For compartments at axonal ends, there is only one neighboring compartment and, thus, one axial current; for any simple compartment without branching or electrical coupling, two axial currents are factored in from two neighboring compartments. However, when calculating ΔV for branch point compartments with various neighboring segments, the integration scheme above provides the flexibility needed to accommodate any additional axial or coupling current by adding a conductance-voltage entry without changing the integration protocol.

Below, two models are constructed using our modeling tool to analyze spike propagation across branch points and to visualize the effect of gap junctions on signal conduction. Figure 2 portrays the axonal structure simulated by the Y-branch model, in which the main axon (M) branches off into two daughter branches (b1, b2). Figure 3 illustrates two parallel axons electrically connected at the middle compartment through gap junctions.

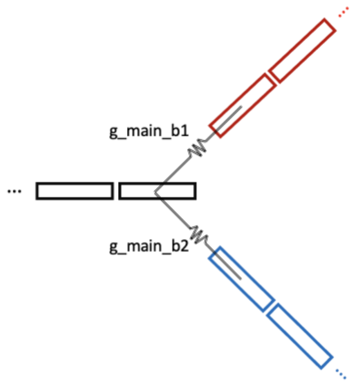

Figure 2. The Y-Branch model. Simulation of a simple branch point, including the mother axon (black) and two daughter branches (red and blue) connected to the mother branch at the same compartment to form a 3-way branch point. Labels "g_main_b1" and "g_main_b2" denote trans-compartment branch point conductance.

The conductance at the branch point from the main branch to the daughter branch b1, g_main_b1, is modeled such that its inverse, the resistance, is the combination of one-half of the main branch axial compartment resistance and one-half of the daughter axial compartment resistance.

Following is the derivation of the branch point conductance g_main_b1 (the conductance between the main branch compartment and the daughter b1 compartment adjacent to the branch point):

$$r\_main\_b1 = \frac{1}{2} * r_{main} + \frac{1}{2} * r_{b1}$$

Since conductance is the inverse of resistance,

$$g\_main\_b1 = \frac{1}{\frac{r_{main}}{2} + \frac{r_{b1}}{2}}$$

$$g\_main\_b1 = \frac{1}{\frac{1}{2 * g_{main}} + \frac{1}{2 * g_{b1}}}$$

$$g\_main\_b1 = \frac{2 * (g_{main} + g_{b1})}{g_{main} * g_{b1}}$$

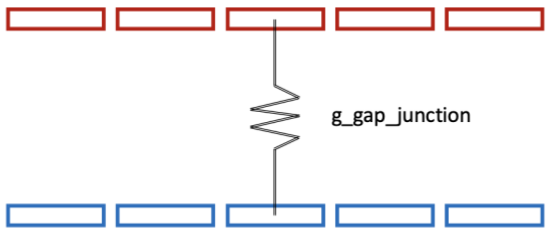

Figure 3. The parallel-axon model. The simulation includes two branches (red and blue) connected at the middle compartment through an electrical synapse. Label "g_gap_junction" denotes the cross-axon conductance modeled as gap junctions with a magnitude depending on the number of connexon channels present irrespective of axon geometry and axial conductance.

Unlike cross-branch conductance at the branch point, the electrical coupling of the two axons is constructed as an electrical synapse by a gap junction consisting of connexons. Therefore, the coupling strength, or the conductance between the two branches (g_gap_junction), does not depend on the axial conductance of the two connected middle compartments. Rather, g_gap_junction depends on the number of gap junctions and thus is manually entered.

Results

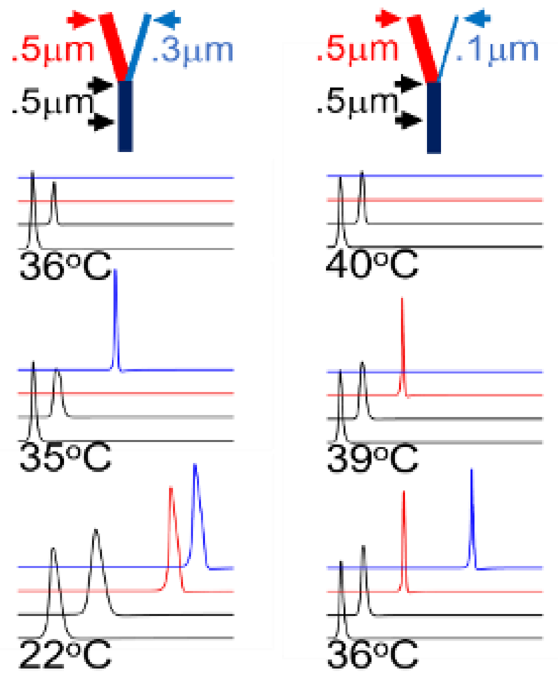

Figure 4. Influence of temperature on branch point failure. Two model configurations (top) differ in b2 diameter (blue branch). Voltage traces show membrane potential at different Ts for locations indicated by arrows. Both configurations transition from no failure to one branch failing to both branches failing as T increases. Left: the wider b1 branch fails first. Right: the narrower b2 branch fails first. (Figure adapted from Hochman/Prinz grant proposal, produced using code I generated in this honors project.)

Preliminary results from the y-branch Model demonstrate the effect of T and axon and branch point geometry on branch point failure (Figure 4). Among all the ionic currents available in our modeling tool, this model contains only fast sodium I_{Na}, delayed rectifier I_{Kd}, I_{GABA}, and the leak current (other ion channels turned off by setting their maximal conductance to zero). We have adapted the model to simulate two axonal network configurations that are identical, aside from the diameter of b2 being 0.3 μm (left) vs. 0.1μm (right), to analyze the impact of T and its interactions with branch point geometry. The main, b1, and b2 branches are each 1mm long, comprised of 100 compartments, respectively, and a spike stimulus is delivered to the initial compartment of the main branch. Temperature is modeled through its influence on ionic reversal potentials via the Nernst equation and ion channel gating time constants using a Q10 = 3, typical for ion channels (Hille, 2001). When T rises, both model configurations show a transition from faithful spike propagation in b1 and b2 to failure in one branch and eventually to failure in both. In the model at the left ($0.3\mu m$ b2), such transition in branch point conduction failure emerges between the T commonly utilized in most investigations (22°C) and mouse body T (36°C), supporting that past room-T experimental preparations might have masked the modulatory impact of branch points on action potential propagation. Moreover, the model at the left ($0.1\mu m$ b2) shifts from one branch failing to complete failure as T increases from 39C° to 40C°, which demonstrates how febrile symptoms or hyperthermia can impact the functioning of the sympathetic and other nervous systems. Curiously, in the left model, the wider b1 branch (red) fails first as T rises, while at the right, the narrower b1 branch (blue) fails first. This suggests two distinct failure mechanisms that are comprehensible through interactions between axon morphology and T-dependent spike duration and will be thoroughly investigated in subsequent work utilizing this model.

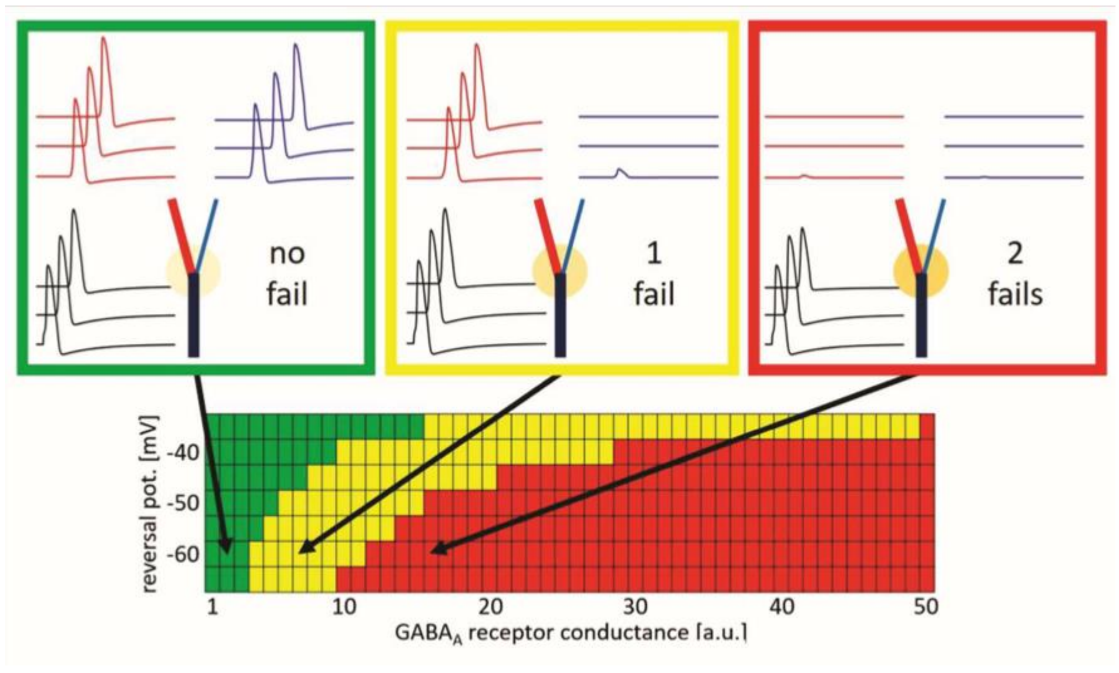

Figure 5. Impact of gGABA on branch point conduction block. Top, simulated action potential propagation in a branching axon with low (left), moderate (middle), and elevated (right) gGABA in axon compartments next to the branch point. Voltage traces shown are from the middle and the ends of the main branch (black), and the two daughter branches, b1 (red) and b2 (blue), respectively (blue). For tiny gGABA, a spike that originates in the main branch invades b1 and b2, but only b1 for intermediate gGABA, and neither branch for big gGABA. Bottom, branch point failure depends on gGABA and the chloride reversal potential E_{Cl-}. Green indicates no branch point failure; yellow indicates one branch fails; red shows both branches fail. (Figure adapted from Hochman/Prinz grant proposal, produced using code I generated in this honors project.)

GABA$_a$Rs and E_{Cl-} could impact branch point failure, as seen in Fig. 5. The model was set up with diameters of 0.5 µm (main), 0.4 µm (b1), and 0.2 µm (b2), and lengths of 1 mm for the main, b1, and b2. GABA$_a$Rs (modeled as a Cl- conductance gGABA) are present in all three axons (main, b1, and b2) in a 10 µm long axon segment adjacent to the branch point but not at the branch point itself or in the remaining parts of the axons, inspired by (Hari et al., 2021). When a spike is elicited at the far end of the main branch, the spike propagates to the branch point and invades both branches b1 and b2, only the (wider) b1 branch, or neither, depending on the magnitude of the gGABA and E_{Cl-}. Overall, failures are more likely for larger gGABA and more hyperpolarized E_{Cl-}.

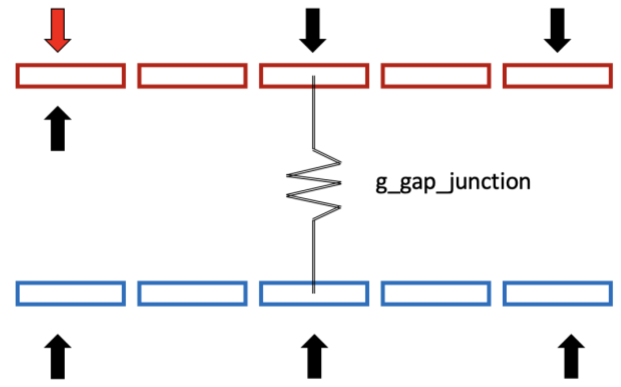

Figure 6. Parallel-axon model (same as Figure 3 except with arrows added). The red arrow points to where stimuli were delivered (first compartment in b1). Black arrows point to compartments (left to right, first, middle, and last compartments, respectively) where voltages were recorded and visualized.

To demonstrate that our modeling tool is amenable to different axon structures, a parallel-axon model was created with diameters of 0.4 μm (b1) and 0.13 μm (b2). Both axons are 0.2 mm long and consist of 20 compartments. Figure 6 illustrates the setup of our parallel-axon model, which, as the name suggests, consists of two axons that are connected in the middle through gap junctions and otherwise independent. Only fast sodium I_{Na}, delayed rectifier I_{Kd}, persistent calcium I_{CaL}, and the leak current are included in the model, with stimuli being delivered to the first compartment of the b1 branch only. Voltage recordings, however, were conducted throughout the b1 and the b2 branches at their first, middle, and last compartments. By varying the gap junction conductance g_gap_junction between b1 and b2, spiking propagation patterns varied in many ways, thus yielding interesting observations of spiking in our model and preliminary results to corroborate the validity of our model.

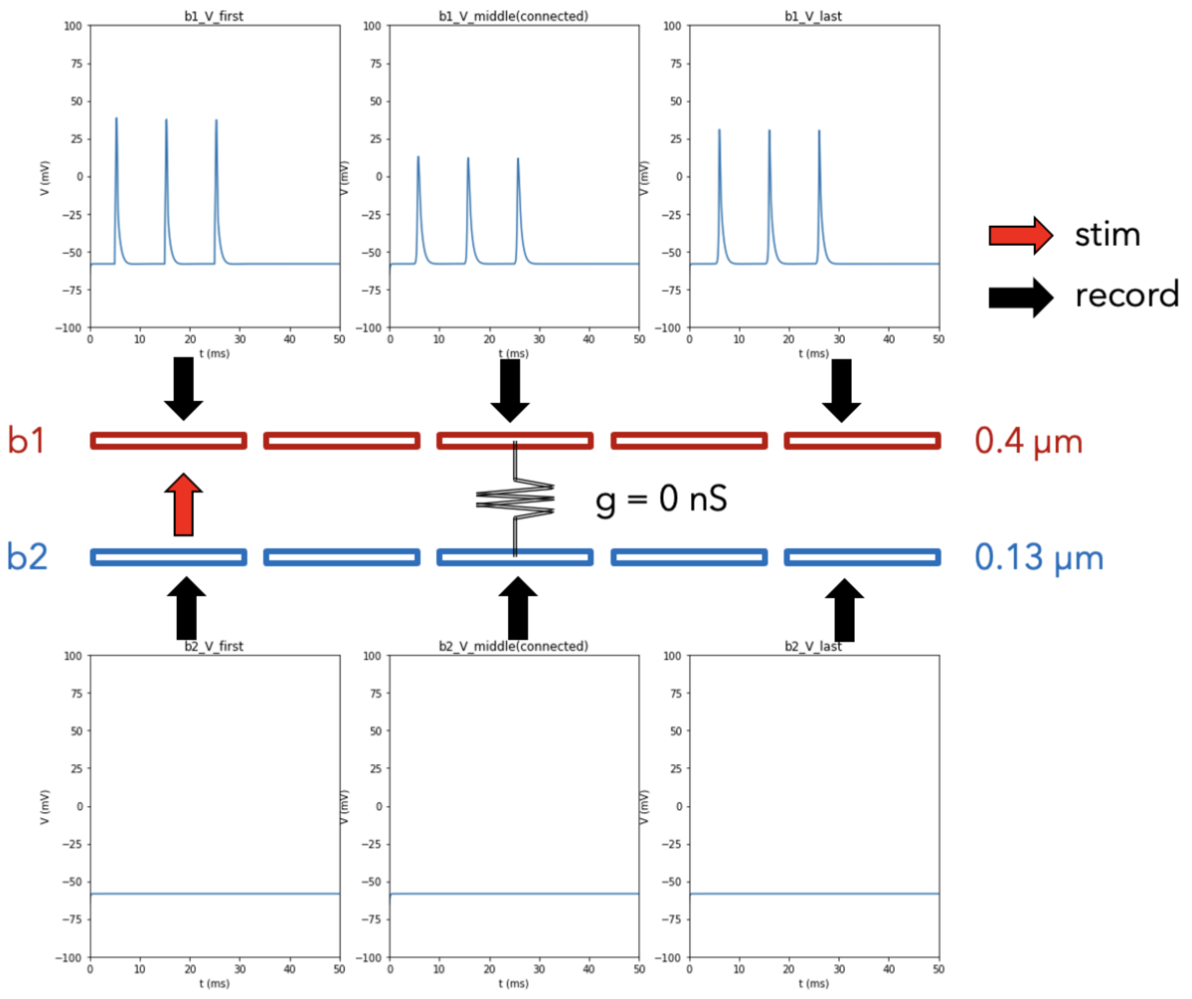

Figure 7. Spike propagation with no gap junction conductance. A series of 3 stimuli (100 Hz) were delivered to the first compartment of b1 (top left), causing spike propagation throughout b1, across the middle compartment (top middle) to the end (top right). No spike propagation from b1 to b2 was observed (bottom left, middle, and right).

Here, a train of three stimuli was delivered to the initial segment of b1 with a frequency of 100 Hz (once every 10 ms), causing three spikes. With the gap junction conductance g_gap_junction set to zero, axons b1 and b2 were electrically uncoupled, and Figure 5 illustrated through voltage recordings that all three spikes successfully crossed the whole length of b1. At the same time, b2 remained at resting potential without spiking. This spiking pattern aligns with our expectations because no spike should reach the b2 branch per our electric coupling scheme.

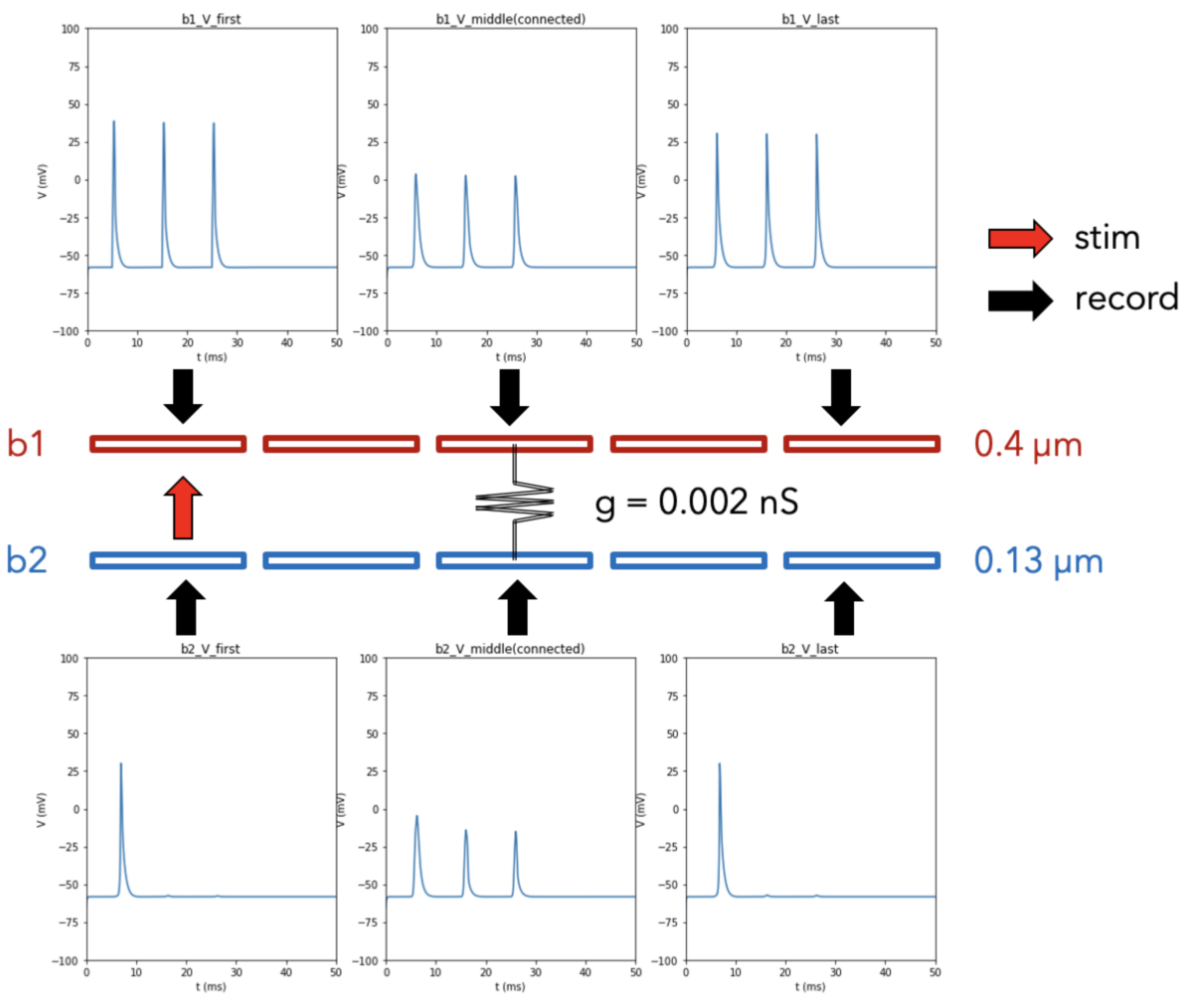

Figure 8. Spike propagation with low gap junction conductance. A series of 3 stimuli (100Hz) were delivered to the first compartment of b1 (top left), causing spike propagation throughout b1, across the middle compartment (top middle) to the end (top right). All 3 spikes propagated from b1's middle compartment to b2's through the electrical synapse (bottom middle). However, only the first spike propagated to both ends of b2 (bottom left, right)

Figure 8 demonstrates that as the gap junction conductance adopts a small yet non-zero value, only minor coupling strength exists to allow depolarizing current entering b2 from b1 as the spike propagates through the b1 branch. This electric synapse connecting the middle of both branches enabled all three spikes to pass into the b2 branch. However, only one spike pervaded all the compartments in b2, possibly due to the small g_gap_junction restricting the magnitude of current from b1 to b2. An alternative explanation is that the diameter of b2 (0.13 μm) resulted in a higher axial resistance than b1 (0.4 μm), which placed additional hurdles on spike propagation across b2 from the middle segment.

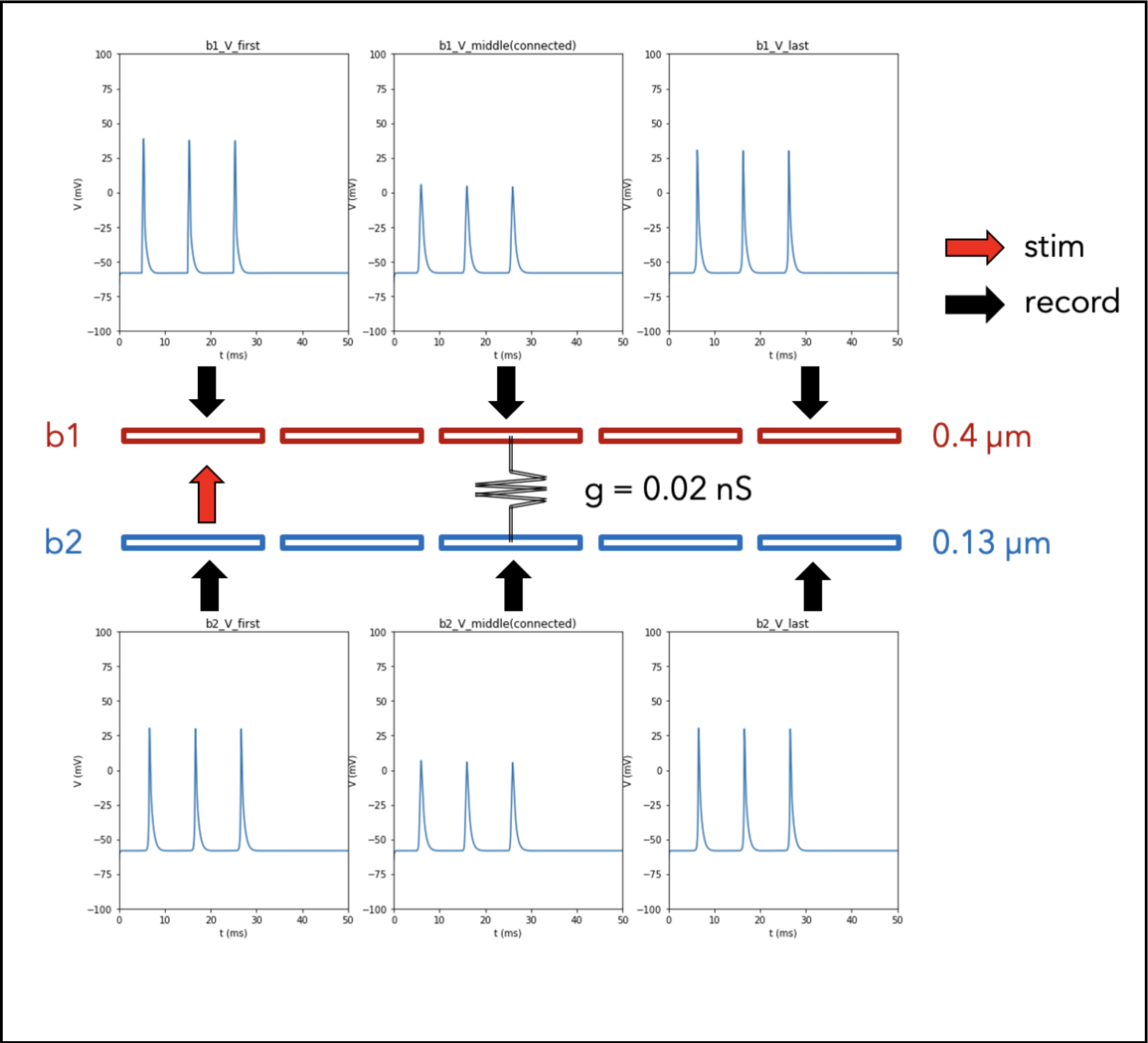

Figure 9. Spike propagation with moderate gap junction conductance. A series of 3 stimuli (100 Hz) were delivered to the first compartment of b1 (top left), causing spike propagation throughout b1, across the middle compartment (top middle) to the end (top right). All spikes propagated to both ends of b2–the first and the last compartment (bottom left, middle, and right)

Figure 9 shows voltage traces of spiking propagation in b1 and b2 with moderate electrical coupling. The depolarizing current into the b2 branch allows for successful spike propagation of all three action potentials into both ends of b2. Compared with voltage recordings in Figure 8, the successful propagation of three instead of one spike across b2 demonstrates that the magnitude of depolarization contributed to the previous partial conduction failure for elevated g_gap_junction, the only variable changed, enabled successful transmission of all spikes. Also, it is worth noting that spikes at both ends of an axon have higher amplitude than those recorded from the middle electrically coupled compartment. This phenomenon arises because compartments at axonal ends only have one neighboring compartment, while the middle compartment has two, plus the electrical synapse. Therefore, as a spike propagates to an end compartment, less axial current exists to diffuse the depolarized membrane potential, while two axial currents and one synaptic current bring down the spike magnitude at the middle compartments of b1 and b2. This difference in compartment voltage further supports the comprehensiveness of our models.

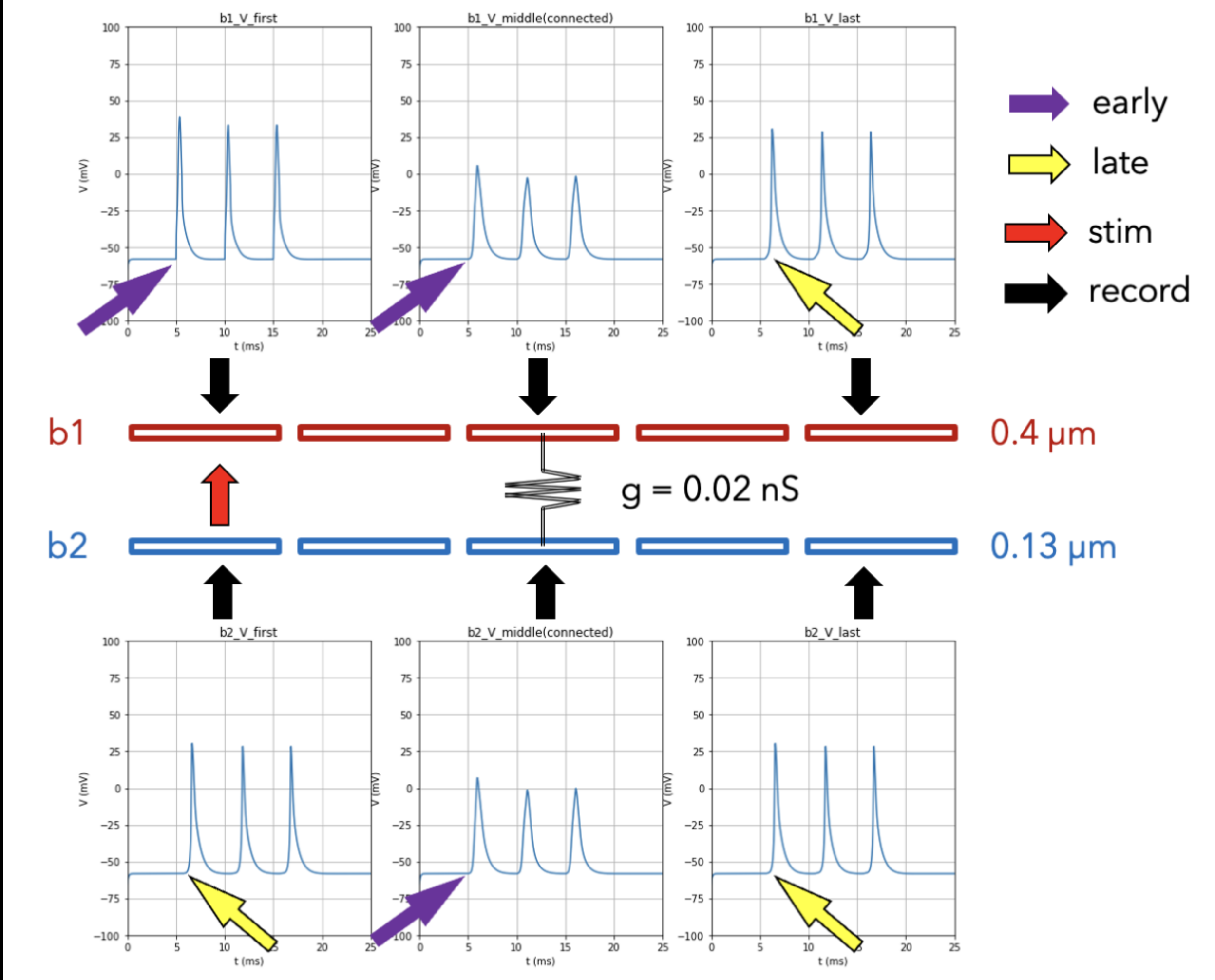

Figure 10. Timing of spike propagation. All spikes (200 Hz) propagate throughout b1 and b2. Yellow arrows indicate earlier initiation of spiking; purple arrows indicate action potentials that arise later. This clearly shows that the timing of spikes is correct, as the first spike in b2 first and b2 last compartments should be the last in time (bottom left, right), an apparent delay compared to the timing of the first spike across b1 compartments (top row).

Figure 10 shows voltage traces across axons b1 and b2 following a stimulation scheme with a higher frequency of 200 Hz. A close examination of the timing of spikes demonstrates that spikes initiate earlier in the compartments indicated by purple arrows (top left, top middle, bottom middle) than in those marked by yellow arrows (top right, bottom left, bottom right), thus revealing that spike propagation into b2’s both ends (bottom left and right) was indeed temporally later than b2’s middle compartment and all other compartments. Since only the first compartment of b1 receives stimuli, we expect action potentials to propagate to b1’s middle compartment and then into both b1’s far end through the axial pathway and b2’s middle compartment via the gap junction. Then, from b2’s middle compartment, action potentials would propagate to both ends of b2, thus causing a temporal discrepancy between spikes arising in both ends of b2 and other compartments. Therefore, the spike timing shown in Figure 10 aligns closely with our expectations and further supports our model’s validity.

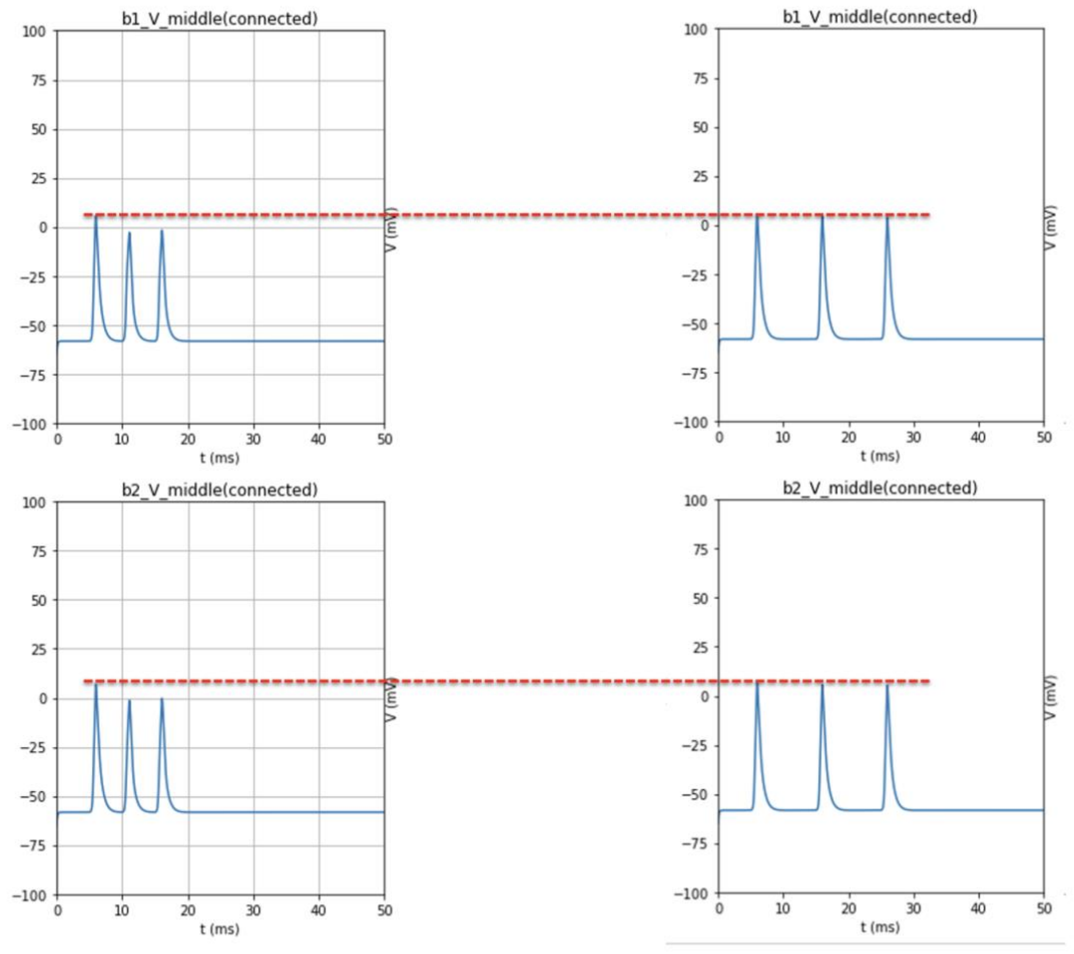

Figure 11. Effect of stimulus frequency on spike amplitude. All spikes propagate throughout b1 and b2. Only voltage traces of middle compartments are shown for comparison. Left (200 Hz) shows the second and the third spikes with decreased amplitude compared with three spikes of equally undiminished amplitude on the right (100 Hz), thus showing the effect of stimulation frequency on spiking.

Figure 11 compares voltage traces from two simulations that are identical with moderate coupling strength, except that the stimulation frequency here was different. Instead of stimulating at 100 Hz (once every 10 ms), as was the case on the right, the b1 branch for the simulation on the left was stimulated at a higher frequency of 200 Hz (once every 5 ms). For the panel on the left, although all three spikes were successfully conducted throughout b1 and b2, the two spikes following the first one at b1 and b2’s middle compartments (left top and bottom) displayed lower magnitude compared with equivalent voltage recordings on the right stimulated with a lower frequency (right top and bottom). A possible explanation for this depression regarding action potential amplitude relates to the relative refractory period that makes it harder to spike immediately following a previous spike.

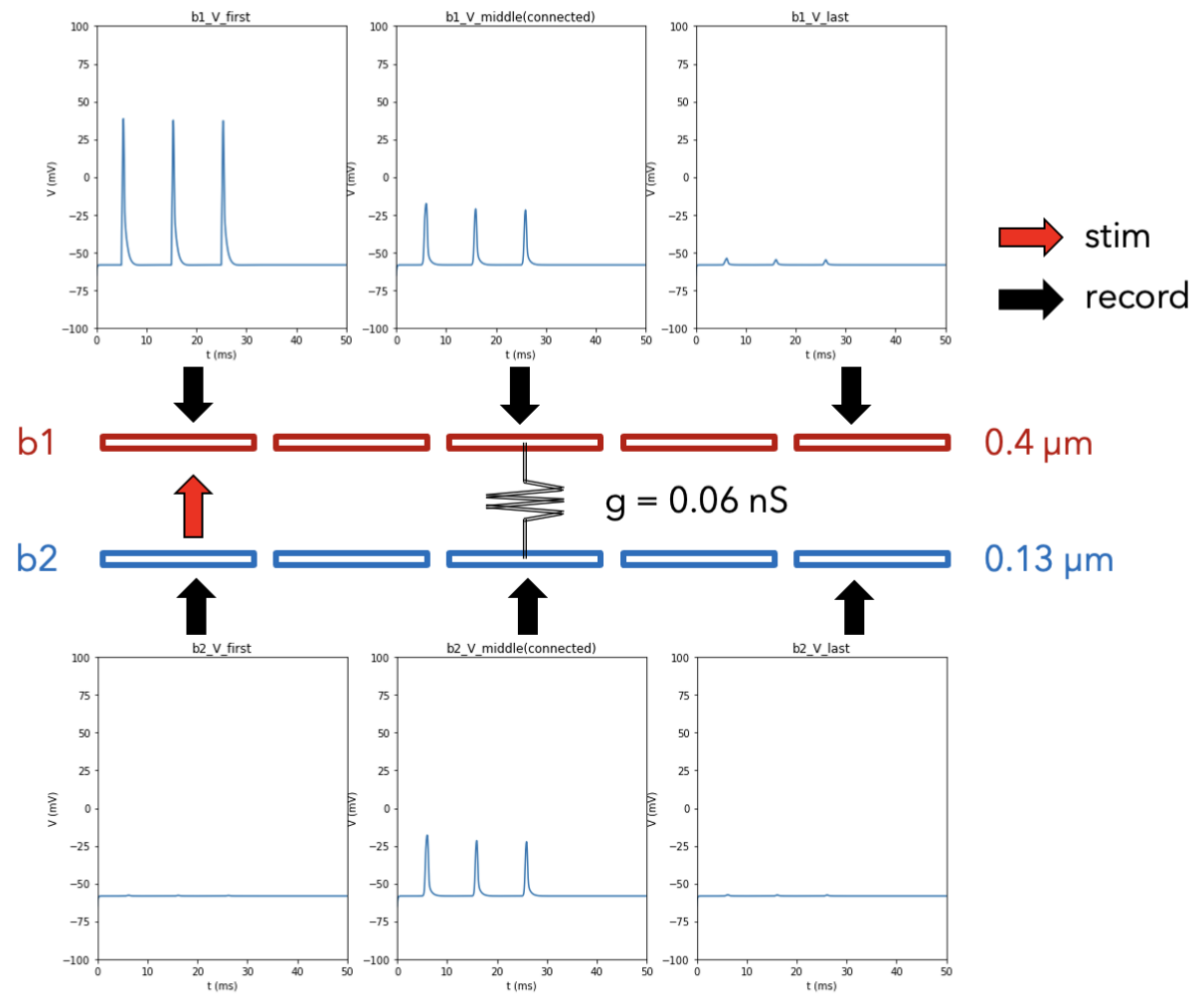

Figure 12. Impact of extremely large gap junction conductance on spike propagation. Post-stimulus voltage recordings in the first, the middle, and the last compartment in two parallel branches. Low-magnitude spikes in the middle of b1 (top middle) and b2 (bottom middle). No spike propagation reached the end of either axon (top right, bottom right, bottom left).

In this figure, as the gap junction conductance adopted an immense value (larger than axial conductance), b1 and b2 became isoelectric, which diffused the depolarization between the middle compartments of both branches. Therefore, none of the spikes were high enough in magnitude to propagate to the end of either b1 or b2.

Through a series of voltage traces across axons b1 and b2 in the parallel-axon model, we saw spiking propagation patterns that differed based on the coupling strength. However, these differences in conduction concurred with our expectations based on the electrophysiology of electric synapses through gap junctions, confirming that our model is ready for systematic studies of spike propagation in various electrophysiological configurations and under complete control of the researcher

Discussion

In collaboration with the Hochman lab, we created a modeling tool to complement experimental studies analyzing spike propagation in ex vivo mice parasympathetic chains. We demonstrated the versatility of our modeling tool by showing two model configurations – the Y-branch model simulating a simple branch point structure and the parallel-axon model exploring axonal modulations other than branch point conduction through axonal electric coupling. Although we were restrained by time and unable to query our models with systemically designed parameter sets, we garnered preliminary results demonstrating the validity of models created using our modeling tool and exciting observations of axonal control of signal conduction. The Y-branch model illustrated that temperature and axon diameters interact to display complex patterns regarding branch point failure. Moreover, gGABA and chloride reversal potentials showed systemic control of branch point failure and are worth further investigating. On the other hand, the parallel-axon model testified to our model's configuration functionality by showing that a different axon structure without branch points but electrically coupled could also be simulated using the template provided by our modeling tool. Moreover, voltage traces illustrated that the conductance magnitude across the electric synapse could indeed impact spike propagation, with moderate conductance allowing for successful propagation of all spikes. At the same time, conductance that was too large or too small resulted in partial or complete conduction failure at axon ends.

Having developed the modeling tool and tested its validity, our next steps are to let scientific inquiries guide the direction of simulations. For example, we could investigate the Y-branch model further by looping through a two-dimensional matrix of temperature and axon diameter parameters, thus yielding systematic observations of branch point failure predicted by these features. Similarly, the relationship between axon geometry and electrical synapse conductance can be further studied through extensive simulations. Most importantly, we aim to combine our simulations with electrophysiological data gathered experimentally from the Hochman lab. For example, it is hard for our model to mimic experimental conditions by simulating over long periods with a low-frequency stimulatory scheme due to limited algorithm efficiency and the lack of channel dynamics tuning. Therefore, another direction for refining our modeling tool is to find more efficient ways to conduct integration without sacrificing stability. With a more mature model capable of adopting experimental parameters, we hope to garner significant scientific insights into the axonal modulation of sympathetic control in the future.

References

Anjum, A., Yazid, M. D., Fauzi Daud, M., Idris, J., Ng, A. M. H., Selvi Naicker, A., Ismail, O. H. R., Athi Kumar, R. K., & Lokanathan, Y. (2020). Spinal Cord Injury: Pathophysiology, Multimolecular Interactions, and Underlying Recovery Mechanisms. *International Journal of Molecular Sciences*, *21*(20), 7533. https://doi.org/10.3390/ijms21207533https://doi.org/10.1098/rstb.1994.0022

Attia, M., & Engel, P. (1983). Thermoregulatory set point in patients with spinal cord injuries (spinal man). *Spinal Cord*, *21*(4), Article 4. https://doi.org/10.1038/sc.1983.37

Blackman, J. G., & Purves, R. D. (1969). Intracellular recordings from ganglia of the thoracic sympathetic chain of the guinea-pig. *The Journal of Physiology*, *203*(1), 173–198.

Cabanac, M., & Massonnet, B. (1977). Thermoregulatory responses as a function of core temperature in humans. *The Journal of Physiology*, *265*(3), 587–596.

Dayan, P., & Abbott, L. F. (2001). *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. Massachusetts Institute of Technology Press.

Debanne, D. (2004). Information processing in the axon. *Nature Reviews Neuroscience*, *5*(4), Article 4. https://doi.org/10.1038/nrn1397

Deuchars, S. A., & Lall, V. K. (2015). Sympathetic preganglionic neurons: Properties and inputs. *Comprehensive Physiology*, *5*(2), 829–869. https://doi.org/10.1002/cphy.c140020

Deuchars, S. A., Milligan, C. J., Stornetta, R. L., & Deuchars, J. (2005). GABAergic neurons in the central region of the spinal cord: A novel substrate for sympathetic inhibition. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, *25*(5), 1063–1070. https://doi.org/10.1523/JNEUROSCI.3740-04.2005

Guest, J., Datta, N., Jimsheleishvili, G., & Gater, D. R. (2022). Pathophysiology, Classification and Comorbidities after Traumatic Spinal Cord Injury. *Journal of Personalized Medicine*, *12*(7), Article 7. https://doi.org/10.3390/jpm12071126

Handrakis, J. P., Liu, S.-A., Rosado-Rivera, D., Krajewski, M., Spungen, A. M., Bang, C., Swonger, K., & Bauman, W. A. (2015). Effect of Mild Cold Exposure on Cognition in Persons with Tetraplegia. *Journal of Neurotrauma*, *32*(15), 1168–1175. https://doi.org/10.1089/neu.2014.3719

Hari, K., Lucas-Osma, A. M., Metz, K., Lin, S., Pardell, N., Roszko, D., Black, S., Minarik, A., Singla, R., Stephens, M. J., Fouad, K., Jones, K. E., Gorassini, M. A., Fenrich, K. K., Li, Y., & Bennett, D. J. (2021). *Nodal GABA facilitates axon spike transmission in the spinal cord* (p. 2021.01.20.427494). bioRxiv. https://doi.org/10.1101/2021.01.20.427494

Hille, B. (2001). *Ion channels of excitable membranes* (3rd ed). Sinauer. http://digitool.hbz-nrw.de:1801/webclient/DeliveryManager?pid=1043961&custom_att_2=simple_viewer

Hodgkin, A. L., & Katz, B. (1949). The effect of temperature on the electrical activity of the giant axon of the squid. *The Journal of Physiology*, *109*(1–2), 240–249.

Huang, Y.-J., Lee, K. H., Murphy, L., Garraway, S. M., & Grau, J. W. (2016). Acute spinal cord injury (SCI) transforms how GABA affects nociceptive sensitization. *Experimental Neurology*, *285*(Pt A), 82–95. https://doi.org/10.1016/j.expneurol.2016.09.005

Hypothermia in Patients With Chronic Spinal Cord Injury—PMC. (n.d.). Retrieved March 24, 2023, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2032005/

Lewis, J. C., & Burton, P. R. (1977). Ultrastructural studies of the superior cervical trunk of the mouse: Distribution, cytochemistry and stability of fibrous elements in preganglionic fibers. *The Journal of Comparative Neurology*, *171*(4), 605–618. https://doi.org/10.1002/cne.901710411

Lichtman, J. W., Purves, D., & Yip, J. W. (1979). On the purpose of selective innervation of guinea-pig superior cervical ganglion cells. *The Journal of Physiology*, *292*, 69–84.

Lichtman, J. W., Purves, D., & Yip, J. W. (1980). Innervation of sympathetic neurones in the guinea-pig thoracic chain. *The Journal of Physiology*, *298*, 285–299. https://doi.org/10.1113/jphysiol.1980.sp013081

Lu, Y., Zheng, J., Xiong, L., Zimmermann, M., & Yang, J. (2008). Spinal cord injury-induced attenuation of GABAergic inhibition in spinal dorsal horn circuits is associated with down-regulation of the chloride transporter KCC2 in rat. *The Journal of Physiology*, *586*(23), 5701–5715. https://doi.org/10.1113/jphysiol.2008.152348

Lucas-Osma, A. M., Li, Y., Lin, S., Black, S., Singla, R., Fouad, K., Fenrich, K. K., & Bennett, D. J. (2018). Extrasynaptic α5GABAA receptors on proprioceptive afferents produce a tonic depolarization that modulates sodium channel function in the rat spinal cord. *Journal of Neurophysiology*, *120*(6), 2953–2974. https://doi.org/10.1152/jn.00499.2018

McKinnon, M. L., Tian, K., Li, Y., Sokoloff, A. J., Galvin, M. L., Choi, M. H., Prinz, A., & Hochman, S. (2019). Dramatically Amplified Thoracic Sympathetic Postganglionic Excitability and Integrative Capacity Revealed with Whole-Cell Patch-Clamp Recordings. *ENeuro*, *6*(2), ENEURO.0433-18.2019. https://doi.org/10.1523/ENEURO.0433-18.2019

McLachlan, E. M. (2003). Transmission of signals through sympathetic ganglia—Modulation, integration or simply distribution? *Acta Physiologica Scandinavica*, *177*(3), 227–235. https://doi.org/10.1046/j.1365-201X.2003.01075.x

Njå, A., & Purves, D. (1977). Specific innervation of guinea-pig superior cervical ganglion cells by preganglionic fibres arising from different levels of the spinal cord. *The Journal of Physiology*, *264*(2), 565–583.

Partida, E., Mironets, E., Hou, S., & Tom, V. J. (2016). Cardiovascular dysfunction following spinal cord injury. *Neural Regeneration Research*, *11*(2), 189–194. https://doi.org/10.4103/1673-5374.177707

Pekala, D., Szkudlarek, H., & Raastad, M. (2016). Typical gray matter axons in mammalian brain fail to conduct action potentials faithfully at fever-like temperatures. *Physiological Reports*, *4*(19), e12981. https://doi.org/10.14814/phy2.12981

Purves, D., & Lichtman, J. W. (1980). Elimination of synapses in the developing nervous system. *Science (New York, N.Y.)*, *210*(4466), 153–157.

https://doi.org/10.1126/science.7414326

Rall, W. (1959). Branching dendritic trees and motoneuron membrane resistivity. *Experimental Neurology*, *1*(5), 491–527. https://doi.org/10.1016/0014-4886(59)90046-9

Trigo, F. F., Marty, A., & Stell, B. M. (2008). Axonal GABAA receptors. *The European Journal of Neuroscience*, *28*(5), 841–848. https://doi.org/10.1111/j.1460-9568.2008.06404.x

Wall, P. D. (1994). Control of impulse conduction in long range branches of afferents by increases and decreases of primary afferent depolarization in the rat. *The European Journal of Neuroscience*, *6*(7), 1136–1142. https://doi.org/10.1111/j.1460-9568.1994.tb00611.x

Wall, P. D. (1995). Do nerve impulses penetrate terminal arborizations? A pre-presynaptic control mechanism. *Trends in Neurosciences*, *18*(2), 99–103. https://doi.org/10.1016/0166-2236(95)93883-y

Wall, P. D., & McMahon, S. B. (1994). Long range afferents in rat spinal cord. III. Failure of impulse transmission in axons and relief of the failure after rhizotomy of dorsal roots. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, *343*(1304), 211–223. https://doi.org/10.1098/rstb.1994.0022

Appendix

File “segTest.py” containing class “segmentSpikeTest”

```
class segmentSpikeTest:
   #to avoid too much parameter inputs, use default values in the parameter
configuration
   def __init__(self, spikeThreshold):
       self.Seg_inSpike = False
       self.spikeThreshold = spikeThreshold
       self.Seg_spike_num = 0

   def findSpike(self, Seg_V):
       if(Seg_V >= self.spikeThreshold):
           if not self.Seg_inSpike:
               self.Seg_inSpike = True
               self.Seg_spike_num += 1
       else:
           if self.Seg_inSpike:
               self.Seg_inSpike = False
```

File “axonBranch.py” containing class “axoBranch”

```
import numpy as np

class axBranch:

   #constructor, what varaibles to use
   def __init__(self, L, d, l_comp, R_ax, V_init, \
                mNa_init, hNa_init, nKd_init, mA_init, hA_init, mH_init, mM_init,
mCaL_init, hCaL_init, mKCa_init, CaS_init, \
                G_leak_abs_uniform,E_leak_uniform, G_Na_abs_uniform, E_Na_uniform,
G_Kd_abs_uniform, E_K_uniform, \
                G_GABA_uniform, E_GABA_uniform, \
                G_A_abs_uniform, E_A_uniform, G_H_abs_uniform, E_H_uniform,
G_M_abs_uniform, E_M_uniform, \
                G_CaL_abs_uniform, E_CaL_uniform, G_KCa_abs_uniform, E_KCa_uniform):

       self.L = L #length of axon in um, distance between sympathetic ganglia is about
1mm
       self.d = d #diameter of axon in um
       self.l_comp = l_comp #length of each compartment in um
       self.n_comp = int(self.L / self.l_comp) #number of compartments
       self.comp_vol = np.pi * ((self.d/2) ** 2) * self.l_comp #um^3

       self.mid_ind = int(self.n_comp/2) #index of compartment roughly in middle of
cable
       self.A_mem_comp = np.pi*self.d*self.l_comp*1e-8 #membrane surface area of
compartment in square cm
       self.A_cross_comp = np.pi*self.d*self.d*1e-8/4 #axon cross-sectional in square
cm

       self.C_mem_comp = self.A_mem_comp*1e3 #membrane capacitace of individual
compartment in nF, assuming 1uF/cm2
       self.conductance_scaling_factor = 1e6*self.C_mem_comp/100 #factor used to scale
conductances because McKinnon et al model has 100pF capacitance

       # branch 1
       self.G_leak_abs = G_leak_abs_uniform #leak conductance in nS, based on McKinnon
Table 1, default is 1nS
       self.g_mem_leak_comp = self.conductance_scaling_factor*self.G_leak_abs/1e3
#membrane leak conductance per compartment in uS
       self.E_leak = E_leak_uniform #leak reversal potential in mV, according to
```

```
McKinnon, default is -55mV
       # Na:
       self.G_Na_abs = G_Na_abs_uniform #Na conductance in nS, based on McKinnon Table
1, default is 300nS
       self.g_mem_Na_comp = self.conductance_scaling_factor*self.G_Na_abs/1e3
#membrane Na conductance per compartment in uS
       self.E_Na = E_Na_uniform #Na reversal potential in mV, according to McKinnon,
default is 60mV
       # Kd:
       self.G_Kd_abs = G_Kd_abs_uniform #Kd conductance in nS, based on McKinnon Table
1, default is 2000nS
       self.g_mem_Kd_comp = self.conductance_scaling_factor*self.G_Kd_abs/1e3
#membrane Kd conductance per compartment in uS
       self.E_K = E_K_uniform #K reversal potential in mV, according to McKinnon,
default is -90mV
       # GABA conductance for compartments proximal to (but not at) branch point
       self.G_GABA_abs = G_GABA_uniform #GABA conductance in nS
       self.g_mem_GABA = self.conductance_scaling_factor*self.G_GABA_abs/1e3 #GABA
conductance per compartment in uS
       self.E_GABA = E_GABA_uniform #GABA (i.e., Cl-) reversal potential in mV, see
Prescott paper fig. 6, vary from -65mV (control) to -50mV (SCI)


       self.G_A_abs = G_A_abs_uniform #default 50 nS
       self.g_mem_A_comp = self.conductance_scaling_factor*self.G_A_abs/1e3
       self.E_A = E_A_uniform#  default of -90.0 in mV


       self.G_H_abs = G_H_abs_uniform #default 1nS
       self.g_mem_H_comp = self.conductance_scaling_factor*self.G_H_abs/1e3
       self.E_H = E_H_uniform   #  default of -32.0 in mV


       self.G_M_abs = G_M_abs_uniform #default 50 nS
       self.g_mem_M_comp = self.conductance_scaling_factor*self.G_M_abs/1e3
       self.E_M = E_M_uniform#  default of -90.0 in mV

       self.G_CaL_abs = G_CaL_abs_uniform #default 1.2 nS
       self.g_mem_CaL_comp = self.conductance_scaling_factor*self.G_CaL_abs/1e3
       self.E_CaL = E_CaL_uniform# default of 120.0 in mV
       ###########
```

```
        self.G_KCa_abs = G_KCa_abs_uniform #default 50 nS
        self.g_mem_KCa_comp = self.conductance_scaling_factor*self.G_KCa_abs/1e3
        self.E_KCa = E_KCa_uniform# at default already in mV


        self.R_ax = R_ax #axial resistivity in Ohm cm, from
https://www.frontiersin.org/articles/10.3389/fncel.2019.00413/full
        self.g_ax_comp = self.A_cross_comp*1e6/(self.R_ax*self.l_comp*1e-4) #axial
conductance between compartments in uS

        #Branch b1

        # compartmental voltage changes
        self.deltaV = np.full((self.n_comp,), 0.0)
        #self.deltaV = [0.0]

        #for i in range(1, self.n_comp, 1):
         #   self.deltaV.append(0.0)
        #self.deltaV = np.asarray(self.deltaV)

        # initial values for compartmental voltages, gating variables, conductances,
and currents
        self.V_init = V_init #initialize all voltages
        print(V_init)
        self.mNa_init = mNa_init #initialize all Na channels to deactivated
        self.hNa_init = hNa_init #initialize all Na channels to deinactivated
        self.nKd_init = nKd_init #initialize all Kd channels to deactivated


        self.mA_init = mA_init
        self.hA_init = hA_init
        self.mH_init = mH_init
        self.mM_init = mM_init
        self.mCaL_init = mCaL_init
        self.hCaL_init = hCaL_init
        self.mKCa_init = mKCa_init
        self.CaS_init = CaS_init



        self.gNa_init = self.g_mem_Na_comp * np.power(self.mNa_init, 2) * self.hNa_init
        self.gKd_init = self.g_mem_Kd_comp * np.power(self.nKd_init, 4)
        self.gleak_init = self.g_mem_leak_comp


        self.gA_init = self.g_mem_A_comp * np.power(self.mA_init, 3) * self.hA_init
        self.gH_init = self.g_mem_H_comp * self.mH_init
        self.gM_init = self.g_mem_M_comp * np.power(self.mM_init, 2)
        self.gCaL_init = self.g_mem_CaL_comp * self.mCaL_init * self.hCaL_init
        self.gKCa_init = self.g_mem_KCa_comp * self.mKCa_init


        #generate and fill in arrays of compartmental voltages, gating variables, and
currents
        self.V = [] #array of compartment voltages in mV
        self.mNa = [] #array of Na activation variables
        self.hNa = [] #array of Na inactivation variables
        self.nKd = [] #array of Kd activation variables



        self.mA = []
        self.hA = []
        self.mH = []
        self.mM = []
        self.mCaL = []
        self.hCaL = []
        self.mKCa= []
        self.CaS = []


        self.gNa = [] #array of Na conductances
        self.gKd = [] #array of Kd conductances
        self.gleak = [] #array of leak conductances
        self.gGABA = [] #array of GABA conductances, will be zeros except for
compartments proximal to branch point



        self.gA = []
        self.gH = []
        self.gM = []
        self.gCaL = []
        self.gKCa = []
        for i in range(0, self.n_comp):
            self.V.append(self.V_init) #initialize compartment voltage array
            self.mNa.append(self.mNa_init) #initialize compartment Na activation array
            self.hNa.append(self.hNa_init) #initialize compartment Na inactivation
array
            self.nKd.append(self.nKd_init) #initialize compartment Kd activation array



            self.mA.append(self.mA_init) #initialize compartment Kd activation array
            self.hA.append(self.hA_init) #initialize compartment Kd activation array
            self.mH.append(self.mH_init)
            self.mCaL.append(self.mCaL_init)
            self.hCaL.append(self.hCaL_init)
            self.mM.append(self.mM_init)
            self.mKCa.append(self.mKCa_init)
            self.CaS.append(self.CaS_init)


            self.gNa.append(self.gNa_init)
            self.gKd.append(self.gKd_init)
            self.gleak.append(self.gleak_init)
            self.gGABA.append(0.0)


            self.gA.append(self.gA_init)
            self.gH.append(self.gH_init)
            self.gM.append(self.gM_init)
            self.gCaL.append(self.gCaL_init)
            self.gKCa.append(self.gKCa_init)

        #this relates to conneciton and should be taken out right now
        self.gGABA[self.n_comp-2] = self.g_mem_GABA #put GABA conductance only in
compartment proximal to branch point (not at branch point)
```

File “integration.py” containing class “Integration”

```
import numpy as np


#axon c1 connects to axon c2 where c1 compartment 5 connects to c2 comparment 4
# c1 has a total compartment of 10, while c2 has a total compartment of 15
#originally thought about the whole process
#[(a[(value, v)], c[value, v])]
class Integration:



   def IntegParamInit(axo, ACsettingArray, dt, stimDict=None):
       axo.A = []
       axo.B = []
       axo.C = []
       axo.D = []
       axo.a = []
       axo.b = []
       axo.c = []
       axo.d = []

       for i in range(len(ACsettingArray)):
           temp_B = (-(axo.gNa[i]+axo.gKd[i]+axo.gleak[i]+axo.gGABA[i]+axo.gA[i] +
axo.gH[i] + axo.gM[i] + axo.gCaL[i] + axo.gKCa[i])/axo.C_mem_comp)#in units of uS/nF

           A_condVpairArr, C_condVpairArr = ACsettingArray[i]
           temp_A = 0.0
           temp_d = 0.0

           for aPair in A_condVpairArr:
               conductance, comp_voltage = aPair
               temp_A += conductance / axo.C_mem_comp
               temp_d += conductance / axo.C_mem_comp * comp_voltage
               temp_B += (-conductance/axo.C_mem_comp)



           temp_C = 0.0
           for cPair in C_condVpairArr:
               conductance, comp_voltage = cPair
               temp_C += conductance / axo.C_mem_comp
```

```
                temp_d += conductance / axo.C_mem_comp * comp_voltage
                temp_B += (-conductance/axo.C_mem_comp)

            axo.A.append(temp_A)
            axo.B.append(temp_B)
            axo.C.append(temp_C)

            if stimDict == None:
                stim_temp = 0.0
            else:
                stim_temp = stimDict.get(i, 0.0)


axo.D.append((axo.gNa[i]*axo.E_Na+axo.gKd[i]*axo.E_K+axo.gleak[i]*axo.E_leak+axo.gGABA
[i]*axo.E_GABA + stim_temp
                    +axo.gA[i]*axo.E_A + axo.gH[i]*axo.E_H + axo.gM[i]*axo.E_M +
axo.gCaL[i]*axo.E_CaL + axo.gKCa[i]*axo.E_KCa)/axo.C_mem_comp) #in units of nA/nF

            axo.a.append(temp_A * dt)
            axo.b.append(temp_B * dt)
            axo.c.append(temp_C * dt)

            axo.d.append((temp_d + axo.D[i] + axo.B[i] * axo.V[i]) * dt)


        axo.A = np.asarray(axo.A)
        axo.B = np.asarray(axo.B)
        axo.C = np.asarray(axo.C)
        axo.D = np.asarray(axo.D)
        axo.a = np.asarray(axo.a)
        axo.b = np.asarray(axo.b)
        axo.c = np.asarray(axo.c)
        axo.d = np.asarray(axo.d)

        axo.b_p = axo.b.copy()
        axo.d_p = axo.d.copy()
```

Y-branch Model

```
import pylab as plt
from scipy.integrate import odeint
import numpy as np
import pandas as pd
from scipy.signal import find_peaks
from scipy import signal
import csv
import math
from segTest import segmentSpikeTest
from axonBranch import axBranch
from integration import Integration






#This part uses segment SpikeTest from segTest.py to track spike conditions in each
segment

def mNaUpdate(Vold, mNaOld):
       alpha = 0.36 * (Vold + 33) / (1 - np.exp(-(Vold + 33) / 3))
       beta = - 0.4 * (Vold + 42) / (1 - np.exp((Vold + 42) / 20))
       vinf = alpha / (alpha + beta)
       tau = 2 / (alpha + beta)
       tau = tau * taufac
       new_mNa = vinf + (mNaOld- vinf) * np.exp(-dt / tau) if dt < tau else vinf
       return new_mNa

def hNaUpdate(Vold, hNaOld):
       alpha = - 0.1 * (Vold + 55) / (1 - np.exp((Vold + 55) / 6))
       beta = 4.5 / (1 + np.exp(-Vold / 10))
       vinf = alpha / (alpha + beta)
       tau = 2 / (alpha + beta)
       tau = tau * taufac
       new_hNa = vinf + (hNaOld - vinf) * np.exp(-dt / tau) if dt < tau else vinf
       return new_hNa

def nKdUpdate(Vold, nKdOld):
       alpha = 0.0047 * (Vold - 8) / (1 - np.exp(-(Vold - 8) / 12))
       beta = np.exp(-(Vold + 127) / 30)
       vinf = alpha / (alpha + beta)
       alpha = 0.0047 * (Vold + 12) / (1 - np.exp(-(Vold + 12) / 12))
       beta = np.exp(-(Vold + 147) / 30)
       tau = 1 / (alpha + beta)
       tau = tau * taufac
       new_nKd = vinf + (nKdOld - vinf) * np.exp(-dt / tau) if dt < tau else vinf
       return new_nKd
def mCaLUpdate(Vold, mCaLOld):
       alpha_mCaL = 7.5 / (1 + np.exp((13 - Vold) / 7))
       beta_mCaL = 1.65 / (1 + np.exp((Vold - 14) / 4))
       mCaL_inf = alpha_mCaL / (alpha_mCaL + beta_mCaL)
       tau_mCaL = 1 / (alpha_mCaL + beta_mCaL)
       new_mCaL = mCaL_inf + (mCaLOld - mCaL_inf) * np.exp(-dt / tau_mCaL) if dt <
tau_mCaL else mCaL_inf
       return new_mCaL
def hCaLUpdate(Vold, hCaLOld):
       alpha_hCaL = 0.0068 / (1 + np.exp((Vold + 30) / 12))
       beta_hCaL = 0.06 / (1 + np.exp(-Vold / 11))
       hCaL_inf = alpha_hCaL / (alpha_hCaL + beta_hCaL)
       tau_hCaL = 1 / (alpha_hCaL + beta_hCaL)
       new_hCaL = hCaL_inf + (hCaLOld - hCaL_inf) * np.exp(-dt / tau_hCaL) if dt <
tau_hCaL else hCaL_inf
       return new_hCaL
def mMUpdate(Vold, mMOld):
       mM_inf = 1 / (1 + np.exp(-(Vold + 35) / 10))
       tau_mM = 2000 / (3.3 * (np.exp((Vold + 35) / 40) + np.exp(-(Vold + 35) / 20)))
       new_mM = mM_inf + (mMOld - mM_inf) * np.exp(-dt / tau_mM) if dt < tau_mM else
mM_inf
       return new_mM

"""
CaS needs input and so does SCa, also no need of Vold
"""
def mKCaUpdate(mKCaOld, CaS_old):
       mKCa_inf = CaS_old ** 2 / (CaS_old ** 2 + SCa ** 2)
       tau_mKCa = tauKCa_0 / (1 + (CaS_old / SCa) ** 2)
       new_mKCa = mKCa_inf + (mKCaOld - mKCa_inf) * np.exp(-dt / tau_mKCa) if dt <
tau_mKCa else mKCa_inf
       return new_mKCa

def mAUpdate(Vold, mAOld):
       mA_inf = (0.0761 * np.exp((Vold + 94.22) / 31.84) / (1 + np.exp((Vold + 1.17) /
28.93))) ** (1/3)
       tau_mA = 0.3632 + 1.158 / (1 + np.exp((Vold + 55.96) / 20.12))
       new_mA = mA_inf + (mAOld - mA_inf) * np.exp(-dt / tau_mA) if dt < tau_mA else
mA_inf
       return new_mA
def hAUpdate(Vold, hAOld):
       hA_inf = (1 / (1 + np.exp(0.069 * (Vold + 53.3)))) ** 4
       tau_hA = (0.124 + 2.678 / (1 + np.exp((Vold + 50) / 16.027))) * tau_hA_scale
       new_hA = hA_inf + (hAOld - hA_inf) * np.exp(-dt / tau_hA) if dt < tau_hA else
hA_inf
       return new_hA

def mHUpdate(Vold, mHOld):
       mh_inf = 1 / (1 + np.exp((Vold + 87.6) / 11.7))
       tau_mh_activ = 53.5 + 67.7 * np.exp(-(Vold + 120) / 22.4)
       tau_mh_deactiv = 40.9 - 0.45 * Vold
       tau_mh = tau_mh_activ if mh_inf > mHOld else tau_mh_deactiv
       new_mH = mh_inf + (mHOld - mh_inf) * np.exp(-dt / tau_mh)
       return new_mH

def CaSUpdate(CaSold, ICaL_old, comp_vol):
   #print (ICaL_old)
   #print(comp_vol)
   alpha_CaS = dt * 1e6 / (comp_vol * 2 * 96485) #uM        1mM/1000uM
   #alpha_CaS = 0.2
   #print(alpha_CaS)

   return  CaSold * np.exp(-f * kCaS * dt) - alpha_CaS / kCaS * ICaL_old * (1 -
np.exp(-f * kCaS * dt))

#fire function
def fireSlots(start, end, freq, duration):
       stimTime = end - start
       numStim = math.floor(stimTime * freq)
       wholeDura = 1 / freq
       if(wholeDura <= duration):
           print ("Too large of a duration!")
           return



       cur = start
       result = []
```

```
        for i in range(numStim):
            pair = (cur, cur + duration)
            cur = cur + wholeDura
            result.append(pair)
        return result

def integCopy(axo):
   axo.V_old = axo.V.copy() #array of previous time step's compartment voltages in mV
   axo.mNa_old = axo.mNa.copy() #array of previous time step's compartment Na
activations
   axo.hNa_old = axo.hNa.copy() #array of previous time step's compartment Na
inactivations
   axo.nKd_old = axo.nKd.copy() #array of previous time step's compartment Kd
activations

   axo.mA_old = axo.mA.copy()
   axo.hA_old = axo.hA.copy()
   axo.mH_old = axo.mH.copy()
   axo.mCaL_old = axo.mCaL.copy()
   axo.hCaL_old = axo.hCaL.copy()
   axo.mM_old = axo.mM.copy()
   axo.mKCa_old = axo.mKCa.copy()
   axo.CaS_old = axo.CaS.copy()

def gatingUpdate(axo, i):
   axo.mNa[i] = mNaUpdate(axo.V_old[i], axo.mNa_old[i])
   axo.hNa[i] = hNaUpdate(axo.V_old[i], axo.hNa_old[i])
   axo.gNa[i] = axo.g_mem_Na_comp * np.power(axo.mNa[i], 2) * axo.hNa[i]

   # Kd
   axo.nKd[i] = nKdUpdate(axo.V_old[i], axo.nKd_old[i])
   axo.gKd[i] = axo.g_mem_Kd_comp * np.power(axo.nKd[i], 4)
   # leak
   axo.gleak[i] = axo.g_mem_leak_comp

   axo.mA[i] = mAUpdate(axo.V_old[i], axo.mA_old[i])
   axo.hA[i] = hAUpdate(axo.V_old[i], axo.hA_old[i])
   axo.gA[i] = axo.g_mem_A_comp * np.power(axo.mA[i], 3) * axo.hA[i]

   axo.mH[i] = mHUpdate(axo.V_old[i], axo.mH_old[i])
   axo.gH[i] = axo.g_mem_H_comp * axo.mH[i]
```

```
    axo.mM[i] = mMUpdate(axo.V_old[i], axo.mM_old[i])
    axo.gM[i] = axo.g_mem_M_comp * np.power(axo.mM[i], 2)

    axo.mCaL[i] = mCaLUpdate(axo.V_old[i], axo.mCaL_old[i])
    axo.hCaL[i] = hCaLUpdate(axo.V_old[i], axo.hCaL_old[i])
    axo.gCaL[i] = axo.g_mem_CaL_comp * axo.mCaL[i] * axo.hCaL[i]

    axo.mKCa[i] = mKCaUpdate(axo.mKCa_old[i], axo.CaS_old[i])
    axo.gKCa[i] = axo.g_mem_KCa_comp * axo.mKCa[i]

    axo.CaS[i] = CaSUpdate(axo.CaS_old[i], axo.gCaL[i] * (axo.V_old[i] - axo.E_CaL),
axo.comp_vol) #

def IntegParamUpdate(axo, ACsettingArray, dt):
    axo.A = []
    axo.B = []
    axo.C = []
    axo.D = []
    axo.a = []
    axo.b = []
    axo.c = []
    axo.d = []

    for i in range(len(ACsettingArray)):
        ###attention
        gatingUpdate(axo, i)
        temp_B = (-(axo.gNa[i]+axo.gKd[i]+axo.gleak[i]+axo.gGABA[i]+axo.gA[i] +
axo.gH[i] + axo.gM[i] + axo.gCaL[i] + axo.gKCa[i])/axo.C_mem_comp)#in units of uS/nF

        A_condVpairArr, C_condVpairArr = ACsettingArray[i]
        temp_A = 0.0
        temp_d = 0.0

        for aPair in A_condVpairArr:
            conductance, comp_voltage = aPair
            temp_A += conductance / axo.C_mem_comp
            temp_d += conductance / axo.C_mem_comp * comp_voltage
            temp_B += (-conductance/axo.C_mem_comp)


        temp_C = 0.0
        for cPair in C_condVpairArr:
            conductance, comp_voltage = cPair
            temp_C += conductance / axo.C_mem_comp
            temp_d += conductance / axo.C_mem_comp * comp_voltage
            temp_B += (-conductance/axo.C_mem_comp)

        axo.A.append(temp_A)
        axo.B.append(temp_B)
        axo.C.append(temp_C)

axo.D.append((axo.gNa[i]*axo.E_Na+axo.gKd[i]*axo.E_K+axo.gleak[i]*axo.E_leak+axo.gGABA
[i]*axo.E_GABA
                +axo.gA[i]*axo.E_A + axo.gH[i]*axo.E_H + axo.gM[i]*axo.E_M +
axo.gCaL[i]*axo.E_CaL + axo.gKCa[i]*axo.E_KCa)/axo.C_mem_comp) #in units of nA/nF

        axo.a.append(temp_A * dt)
        axo.b.append(temp_B * dt)
        axo.c.append(temp_C * dt)

        axo.d.append((temp_d + axo.D[i] + axo.B[i] * axo.V[i]) * dt)
        if i == 0:
            axo.b_p[0] = axo.b[0] # _p stands for prime as in Dayan and Abbott appendix
6
            axo.d_p[0] = axo.d[0]
        else:
            axo.b_p[i] = axo.b[i] + axo.a[i]*axo.c[i-1]/(1-axo.b_p[i-1]) #equation 6.54
in D&A
            axo.d_p[i] = axo.d[i] + axo.a[i]*axo.d_p[i-1]/(1-axo.b_p[i-1]) #equation
6.55 in D&A

def IntegParamUpdate(axo, ACsettingArray, dt):


    for i in range(len(ACsettingArray)):
        gatingUpdate(axo, i)
        temp_B = (-(axo.gNa[i]+axo.gKd[i]+axo.gleak[i]+axo.gGABA[i]+axo.gA[i] +
axo.gH[i] + axo.gM[i] + axo.gCaL[i] + axo.gKCa[i])/axo.C_mem_comp)#in units of uS/nF

        A_condVpairArr, C_condVpairArr = ACsettingArray[i]
        temp_A = 0.0
        temp_d = 0.0

        for aPair in A_condVpairArr:
            conductance, comp_voltage = aPair
            temp_A += conductance / axo.C_mem_comp
            temp_d += conductance / axo.C_mem_comp * comp_voltage
            temp_B += (-conductance/axo.C_mem_comp)


        temp_C = 0.0
        for cPair in C_condVpairArr:
            conductance, comp_voltage = cPair
            temp_C += conductance / axo.C_mem_comp
            temp_d += conductance / axo.C_mem_comp * comp_voltage
            temp_B += (-conductance/axo.C_mem_comp)

        axo.A[i] = (temp_A)
        axo.B[i] = (temp_B)
        axo.C[i] = (temp_C)
        axo.D[i] =
((axo.gNa[i]*axo.E_Na+axo.gKd[i]*axo.E_K+axo.gleak[i]*axo.E_leak+axo.gGABA[i]*axo.E_GA
BA
                +axo.gA[i]*axo.E_A + axo.gH[i]*axo.E_H + axo.gM[i]*axo.E_M +
axo.gCaL[i]*axo.E_CaL + axo.gKCa[i]*axo.E_KCa)/axo.C_mem_comp) #in units of nA/nF

        axo.a[i] = (temp_A * dt)
        axo.b[i] = (temp_B * dt)
        axo.c[i] = (temp_C * dt)

        axo.d[i] = ((temp_d + axo.D[i] + axo.B[i] * axo.V[i]) * dt)
        if i == 0:
            axo.b_p[0] = axo.b[0] # _p stands for prime as in Dayan and Abbott appendix
6
            axo.d_p[0] = axo.d[0]
        else:
            axo.b_p[i] = axo.b[i] + axo.a[i]*axo.c[i-1]/(1-axo.b_p[i-1]) #equation 6.54
in D&A
            axo.d_p[i] = axo.d[i] + axo.a[i]*axo.d_p[i-1]/(1-axo.b_p[i-1]) #equation
6.55 in D&A

# Plot I_stim (stimulus current in first compartment) vs time
def plotI_stim(time, stimulus, name, total_time):
   plt.figure(figsize=(12,9))
   plt.plot(time, stimulus)
   plt.title(name + ': I_stim vs time')
   plt.xlabel('t (ms)')
   plt.ylabel('I (nA)')
   plt.xlim(0, total_time)
   plt.ylim(-1, 5)
   #axes = plt.gca()
   #axes.yaxis.grid()
   plt.show()

# Plot V_rec_first (recorded membrane voltage in first compartment) vs time
def plotVfirst(time, firstVoltage, name, total_time):
   plt.figure(figsize=(12,9))
   plt.plot(time, firstVoltage)
   plt.title(name + ': V_first vs time')
   plt.xlabel('t (ms)')
   plt.ylabel('V (mV)')
   plt.xlim(0, total_time)
   #plt.xlim(0.9, 1.2)
   plt.ylim(-100, 100)
   #axes = plt.gca()
   #axes.yaxis.grid()
   plt.show()

# Plot V_rec_second (recorded membrane voltage in second compartment) vs time
def plotVsecond(time, secondVoltage, name, total_time):
   plt.figure(figsize=(12,9))
   plt.plot(time, secondVoltage)
   plt.title(name + ': V_second vs time')
   plt.xlabel('t (ms)')
   plt.ylabel('V (mV)')
   plt.xlim(0, total_time)
   #plt.xlim(0.9, 1.2)
   plt.ylim(-100, 100)
   #axes = plt.gca()
   #axes.yaxis.grid()
   plt.show()

# Plot V_rec_middle (recorded membrane voltage in middle compartment) vs time
def plotVmiddle(time, middleVoltage, name, total_time):
   plt.figure(figsize=(12,9))
   plt.plot(time, middleVoltage)
   plt.title(name + ': V_middle vs time')
   plt.xlabel('t (ms)')
```

```
    plt.ylabel('V (mV)')
    plt.xlim(0, total_time)
    #plt.xlim(0.9, 1.2)
    plt.ylim(-100, 100)
    #axes = plt.gca()
    #axes.yaxis.grid()
    plt.show()

# Plot V_rec_last (recorded membrane voltage in last compartment) vs time
def plotVlast(time, lastVoltage, name, total_time):
    plt.figure(figsize=(12,9))
    plt.plot(time, lastVoltage)
    plt.title(name + ': V_last vs time')
    plt.xlabel('t (ms)')
    plt.ylabel('V (mV)')
    plt.xlim(0, total_time)
    #plt.xlim(0.9, 1.2)
    plt.ylim(-100, 100)
    #axes = plt.gca()
    #axes.yaxis.grid()
    plt.show()

# Plot V_rec_nexttoGABA (recorded membrane voltage in compartment next to GABA) vs
time
def plotVnexttoGABA(time, nexttoGABAVoltage, name, total_time):
    plt.figure(figsize=(12,9))
    plt.plot(time, nexttoGABAVoltage)
    plt.title(name + ': V_nexttoGABA vs time')
    plt.xlabel('t (ms)')
    plt.ylabel('V (mV)')
    plt.xlim(0, total_time)
    #plt.xlim(0.9, 1.2)
    plt.ylim(-100, 100)
    #axes = plt.gca()
    #axes.yaxis.grid()
    plt.show()
### ALL ARRAYS ACROSS COMPARTMENTS WILL USE INDICES 0 THROUGH n_comp-1
### THIS IS DIFFERENT FROM DAYAN & ABBOTT, which uses 1 THROUGH N

def plotCaS(time, CaS, name, total_time):
    plt.figure(figsize=(12,9))
    plt.plot(time, CaS)
```

```
   plt.title(name + ': Ca2+ Concentration vs time')
   plt.xlabel('t (ms)')
   plt.ylabel('CaS (uM)')
   plt.xlim(0, total_time)
   #plt.xlim(0.9, 1.2)
   plt.ylim(0.001, 0.002)
   #axes = plt.gca()
   #axes.yaxis.grid()
   plt.show()


index2 = 0
b1d_ini = 0.35
b2d_ini = 0.12
twoDarray = []

while index2 < 1:
   oneDarray = []
   index1 = 0
   index2+=1
   while index1 < 1:

       index1 +=1



       # time parameters
       dt = 0.01 #numerical integration time step
       t_total = 20.0 #total simulation time in ms
       t_now = 0.0 #time right now in ms
       t = [t_now] #time array in ms

       # stimulus parameters (stimulus is current pulse injected in first compartment)
       t_stimstart = 5.0 #stimulation start time
       t_stimend = 5.5 #stimulation end time
       I_stim_amp = 1.0 #stimulation current amplitude in nA, 0.15 is good
       I_stim_now = 0.0 #stimulus current in nA
       I_stim = [I_stim_now] #stimulus current time course

       # temperature
       default_temp_C = 22 #default temperature in Celcius. Hochman lab exp mostly at
room temperature, 22C. Assumption is that default model parameters are tuned to this
```

```
temperature.
        default_temp_K = default_temp_C + 273.15 #conversion to Kelvin


        # ADJUST THIS TO CHANGE TEMPERATURE THROUGHOUT:
        temp_C = 39.6 #temperature used in simulation in Celsius


        temp_K = temp_C + 273.15 #conversion to Kelvin
        Q10 = 3.0 #Q10 for adjusting activation and inactivation rates, typical range
for ion channels is 2.4 - 4, see
https://link.springer.com/referenceworkentry/10.1007%2F978-1-4614-7320-6_236-1
        taufac = np.power(Q10, (default_temp_K-temp_K)/10) #factor to multiply
activation and inactivation time constants to adjust for temperature dependence of
gating dynamics

        # 'uniform' parameters set here will apply throughout the model, so they don't
have to be changed for each branch individually
        # 'default' means not adjusted for temperature relative to the McKinnon
parameters
        E_Na_uniform_default = 60.0 #Na reversal potential in mV, according to
McKinnon, default is 60mV
        E_Na_uniform = E_Na_uniform_default * temp_K / default_temp_K #adjust according
to Nernst equation
        E_leak_uniform_default = -55.0 #leak reversal potential in mV, according to
McKinnon, default is -55mV
        E_leak_uniform = E_leak_uniform_default * temp_K / default_temp_K #adjust
according to Nernst equation
        E_K_uniform_default = -90.0 #K reversal potential in mV, according to McKinnon,
default is -90mV
        E_K_uniform = E_K_uniform_default * temp_K / default_temp_K #adjust according
to Nernst equation
        E_GABA_uniform_default = -60.0 #GABA (i.e., Cl-) reversal potential in mV, see
Prescott paper fig. 6, vary from -65mV (control) to -50mV (SCI)
        E_GABA_uniform = E_GABA_uniform_default * temp_K / default_temp_K
        #####
        E_A_uniform_default = -90.0 #default -90
        E_A_uniform = E_A_uniform_default* temp_K  /default_temp_K

        E_H_uniform_default = -32.0 #default -32
        E_H_uniform = E_H_uniform_default * temp_K /default_temp_K
        E_M_uniform_default = -90.0 #default -90
        E_M_uniform = E_M_uniform_default * temp_K /default_temp_K

        E_CaL_uniform_default = 120.0 #default 120
        E_CaL_uniform = E_CaL_uniform_default * temp_K /default_temp_K

        E_KCa_uniform_default = -90.0 #default -90
        E_KCa_uniform = E_KCa_uniform_default * temp_K /default_temp_K




        #####
        """
        ADD E_OtherChannel_Uniform
        """
        G_Na_abs_uniform = 7.0 #Na conductance in nS, based on McKinnon Table 1,
default is 300nS
        G_Kd_abs_uniform = 100.0 #Kd conductance in nS, based on McKinnon Table 1,
default is 2000nS
        G_leak_abs_uniform = 1.0 #leak conductance in nS, based on McKinnon Table 1,
default is 1nS
        G_GABA_uniform = 0.0 #GABA conductance in nS

        G_A_abs_uniform  = 0 #default 50 nS, used 1, high impact, hyperpolarize
        G_H_abs_uniform  = 0 #default 1nS, used 1, low impact, depolarize

        G_M_abs_uniform  = 0 #default 50 nS,used 1, decrease the number spikes,
        G_CaL_abs_uniform  = 1.2 #default 1.2 nS, used 1.2
        G_KCa_abs_uniform  = 0 #default 50 nS, used 1

        ##########
        f = 0.01                   # percent of free to bound Ca2+
        alpha_CaS = 0.2              # uM/pA; convertion factor from current to
concentration 0.002 original
        kCaS = 0.024               # /ms; Ca2+ removal rate, kCaS is proportional to
1/tau_removal; 0.008 - 0.025

        SCa = 1                    # uM; half-saturation of [Ca2+]; 25uM in Ermentrount
book, 0.2uM in Kurian et al. 2011
        tauKCa_0 = 50              # ms
        tau_hA_scale = 100          # scaling factor for tau_hA
        """
        ADD G_OtherChannel_Uniform
        """
        # model geometry settings
        # main axon
        L = 200.0 #length of axon in um, distance between sympathetic ganglia is about
1mm
        d = 0.5 #diameter of axon in um
        l_comp = 10.0 #length of each compartment in um
        n_comp = int(L/l_comp) #number of compartments





        comp_vol = np.pi * ((d/2) ** 2) * l_comp #um^3





        # geometry-related
        mid_ind = int(n_comp/2) #index of compartment roughly in middle of cable
        A_mem_comp = np.pi*d*l_comp*1e-8 #membrane surface area of compartment in
square cm
        A_cross_comp = np.pi*d*d*1e-8/4 #axon cross-sectional in square cm


        # capacitance related
        C_mem_comp = A_mem_comp*1e3 #membrane capacitace of individual compartment in
nF, assuming 1uF/cm2
        conductance_scaling_factor = 1e6*C_mem_comp/100 #factor used to scale
conductances because McKinnon et al model has 100pF capacitance

        # membrane conductances and reversals
        # main branch
        G_leak_abs = G_leak_abs_uniform #leak conductance in nS, based on McKinnon
Table 1, default is 1nS
        g_mem_leak_comp = conductance_scaling_factor*G_leak_abs/1e3 #membrane leak
```

```
conductance per compartment in uS
        E_leak = E_leak_uniform #leak reversal potential in mV, according to McKinnon,
default is -55mV
        # Na:
        G_Na_abs = G_Na_abs_uniform #Na conductance in nS, based on McKinnon Table 1,
default is 300nS
        g_mem_Na_comp = conductance_scaling_factor*G_Na_abs/1e3 #membrane Na
conductance per compartment in uS
        E_Na = E_Na_uniform #Na reversal potential in mV, according to McKinnon,
default is 60mV
        # Kd:
        G_Kd_abs = G_Kd_abs_uniform #Kd conductance in nS, based on McKinnon Table 1,
default is 2000nS
        g_mem_Kd_comp = conductance_scaling_factor*G_Kd_abs/1e3 #membrane Kd
conductance per compartment in uS
        E_K = E_K_uniform #K reversal potential in mV, according to McKinnon, default
is -90mV
        # GABA conductance for compartments proximal to (but not at) branch point
        G_GABA_abs = G_GABA_uniform #GABA conductance in nS
        g_mem_GABA = conductance_scaling_factor*G_GABA_abs/1e3 #GABA conductance per
compartment in uS
        E_GABA = E_GABA_uniform #GABA (i.e., Cl-) reversal potential in mV, see
Prescott paper fig. 6, vary from -65mV (control) to -50mV (SCI)



        G_A_abs = G_A_abs_uniform #default 50 nS
        g_mem_A_comp = conductance_scaling_factor*G_A_abs/1e3
        E_A = E_A_uniform#  default of -90.0 in mV


        G_H_abs = G_H_abs_uniform #default 1nS
        g_mem_H_comp = conductance_scaling_factor*G_H_abs/1e3
        E_H = E_H_uniform   #  default of -32.0 in mV


        G_M_abs = G_M_abs_uniform #default 50 nS
        g_mem_M_comp = conductance_scaling_factor*G_M_abs/1e3
        E_M = E_M_uniform#  default of -90.0 in mV

        G_CaL_abs = G_CaL_abs_uniform #default 1.2 nS
        g_mem_CaL_comp = conductance_scaling_factor*G_CaL_abs/1e3
        E_CaL = E_CaL_uniform# default of 120.0 in mV
        ###########

        G_KCa_abs = G_KCa_abs_uniform #default 50 nS
        g_mem_KCa_comp = conductance_scaling_factor*G_KCa_abs/1e3
        E_KCa = E_KCa_uniform# at default already in mV










        """
        ADD OtherChannel for both main and other two branches
        """

        # axial conductance related
        # main axon
        R_ax = 100.0 #axial resistivity in Ohm cm, from
https://www.frontiersin.org/articles/10.3389/fncel.2019.00413/full
        g_ax_comp = A_cross_comp*1e6/(R_ax*l_comp*1e-4) #axial conductance between
compartments in uS
        #g_ax_comp = 0.0 #uncouple compartments, for testing




        # compartmental voltage changes
        deltaV = [0.0]
        for i in range(1, n_comp, 1):
            deltaV.append(0.0)
        deltaV = np.asarray(deltaV)

        # initial values for compartmental voltages, gating variables, conductances,
and currents
        V_init = -65.0 #initialize all voltages
        mNa_init = 0.0 #initialize all Na channels to deactivated
        hNa_init = 1.0 #initialize all Na channels to deinactivated
        nKd_init = 0.0 #initialize all Kd channels to deactivated
        mA_init = 0.0
        hA_init = 1.0
        mH_init = 0.0
        mM_init = 0.0
        mCaL_init = 0.0
        hCaL_init = 1.0
        mKCa_init = 0.0
        CaS_init = 0.001



        gNa_init = g_mem_Na_comp * np.power(mNa_init, 2) * hNa_init
        gKd_init = g_mem_Kd_comp * np.power(nKd_init, 4)
        gleak_init = g_mem_leak_comp


        gA_init = g_mem_A_comp * np.power(mA_init, 3) * hA_init
        gH_init = g_mem_H_comp * mH_init
        gM_init = g_mem_M_comp * np.power(mM_init, 2)
        gCaL_init = g_mem_CaL_comp * mCaL_init * hCaL_init
        gKCa_init = g_mem_KCa_comp * mKCa_init


        INa_init = gNa_init * (V_init - E_Na) #initialize Na currents
        IKd_init = gKd_init * (V_init - E_K) #initialize Kd currents
        Ileak_init = gleak_init * (V_init - E_leak) #initialize leak currents

        IA_init = gA_init * (V_init - E_A)
        IH_init = gH_init * (V_init - E_H)
        IM_init = gM_init * (V_init - E_M)
        ICaL_init = gCaL_init * (V_init - E_CaL)
        IKCa_init = gKCa_init * (V_init - E_KCa)


        #generate and fill in arrays of compartmental voltages, gating variables, and
currents
        V = [V_init] #array of compartment voltages in mV
        mNa = [mNa_init] #array of Na activation variables
        hNa = [hNa_init] #array of Na inactivation variables
        nKd = [nKd_init] #array of Kd activation variables
        mA = [mA_init]
        hA = [hA_init]
        mH = [mH_init]
        mM = [mM_init]
        mCaL = [mCaL_init]
        hCaL = [hCaL_init]
        mKCa= [mKCa_init]
        CaS = [CaS_init]

        gNa = [gNa_init] #array of Na conductances
        gKd = [gKd_init] #array of Kd conductances
        gleak = [gleak_init] #array of leak conductances
        gGABA = [0.0] #array of GABA conductances, will be zeros except for
compartments proximal to branch point

        gA = [gA_init]
        gH = [gH_init]
        gM = [gM_init]
        gCaL = [gCaL_init]
        gKCa = [gKCa_init]



        for i in range(1, n_comp):
            V.append(V_init) #initialize compartment voltage array
            mNa.append(mNa_init) #initialize compartment Na activation array
            hNa.append(hNa_init) #initialize compartment Na inactivation array
            nKd.append(nKd_init) #initialize compartment Kd activation array

            mA.append(mA_init) #initialize compartment Kd activation array
            hA.append(hA_init) #initialize compartment Kd activation array
            mH.append(mH_init)
            mCaL.append(mCaL_init)
            hCaL.append(hCaL_init)
            mM.append(mM_init)
            mKCa.append(mKCa_init)
            CaS.append(CaS_init)


            gNa.append(gNa_init)
            gKd.append(gKd_init)
            gleak.append(gleak_init)
            gGABA.append(0.0)


            gA.append(gA_init)
            gH.append(gH_init)
            gM.append(gM_init)
            gCaL.append(gCaL_init)
            gKCa.append(gKCa_init)



        gGABA[1] = g_mem_GABA #put GABA conductance only in compartment proximal to
branch point (not at branch point)

        #Branch b1
        b1 = axBranch(L=200.0, d = b1d_ini + index1 * -0.03, l_comp =10.0 , R_ax =
100.0, V_init = -65.0, \
                    mNa_init = mNa_init, hNa_init = hNa_init, nKd_init = nKd_init,
mA_init = mA_init, hA_init = hA_init, mH_init = mH_init, \
                    mM_init = mM_init, mCaL_init = mCaL_init, hCaL_init = hCaL_init,
mKCa_init = mKCa_init, CaS_init = CaS_init, \
                    G_leak_abs_uniform = G_leak_abs_uniform, E_leak_uniform =
E_leak_uniform, G_Na_abs_uniform = G_Na_abs_uniform, \
                    E_Na_uniform = E_Na_uniform, G_Kd_abs_uniform = G_Kd_abs_uniform,
E_K_uniform = E_K_uniform, \
                    G_GABA_uniform = G_GABA_uniform, E_GABA_uniform = E_GABA_uniform,
\
                    G_A_abs_uniform = G_A_abs_uniform, E_A_uniform = E_A_uniform,
G_H_abs_uniform = G_H_abs_uniform,\
                    E_H_uniform = E_H_uniform, G_M_abs_uniform = G_M_abs_uniform,
E_M_uniform = E_M_uniform, \
                    G_CaL_abs_uniform = G_CaL_abs_uniform, E_CaL_uniform =
E_CaL_uniform, G_KCa_abs_uniform = G_KCa_abs_uniform, E_KCa_uniform = E_KCa_uniform)
        #Branch b2
        b2 = axBranch(L=200.0, d = b2d_ini + index2 * 0.03, l_comp =10.0 , R_ax =
100.0, V_init = -65.0, \
                    mNa_init = mNa_init, hNa_init = hNa_init, nKd_init =  nKd_init,
mA_init = mA_init, hA_init = hA_init, mH_init = mH_init, \
                    mM_init = mM_init, mCaL_init = mCaL_init, hCaL_init = hCaL_init,
mKCa_init = mKCa_init, CaS_init = CaS_init, \
                    G_leak_abs_uniform = G_leak_abs_uniform, E_leak_uniform =
E_leak_uniform, G_Na_abs_uniform = G_Na_abs_uniform, \
                    E_Na_uniform = E_Na_uniform, G_Kd_abs_uniform = G_Kd_abs_uniform,
E_K_uniform = E_K_uniform, \
                    G_GABA_uniform = G_GABA_uniform, E_GABA_uniform = E_GABA_uniform,
\
                    G_A_abs_uniform = G_A_abs_uniform, E_A_uniform = E_A_uniform,
G_H_abs_uniform = G_H_abs_uniform,\
                    E_H_uniform = E_H_uniform, G_M_abs_uniform = G_M_abs_uniform,
E_M_uniform = E_M_uniform, \
                    G_CaL_abs_uniform = G_CaL_abs_uniform, E_CaL_uniform =
E_CaL_uniform, G_KCa_abs_uniform = G_KCa_abs_uniform, E_KCa_uniform = E_KCa_uniform)
       b3 = axBranch(L=200.0, d = b1d_ini + index1 * -0.03, l_comp =10.0 , R_ax =
100.0, V_init = -65.0, \
                    mNa_init = mNa_init, hNa_init = hNa_init, nKd_init = nKd_init,
mA_init = mA_init, hA_init = hA_init, mH_init = mH_init, \
                    mM_init = mM_init, mCaL_init = mCaL_init, hCaL_init = hCaL_init,
mKCa_init = mKCa_init, CaS_init = CaS_init, \
                    G_leak_abs_uniform = G_leak_abs_uniform, E_leak_uniform =
E_leak_uniform, G_Na_abs_uniform = G_Na_abs_uniform, \
                    E_Na_uniform = E_Na_uniform, G_Kd_abs_uniform = G_Kd_abs_uniform,
E_K_uniform = E_K_uniform, \
                    G_GABA_uniform = G_GABA_uniform, E_GABA_uniform = E_GABA_uniform,
\
                    G_A_abs_uniform = G_A_abs_uniform, E_A_uniform = E_A_uniform,
G_H_abs_uniform = G_H_abs_uniform,\
                    E_H_uniform = E_H_uniform, G_M_abs_uniform = G_M_abs_uniform,
E_M_uniform = E_M_uniform, \
                    G_CaL_abs_uniform = G_CaL_abs_uniform, E_CaL_uniform =
E_CaL_uniform, G_KCa_abs_uniform = G_KCa_abs_uniform, E_KCa_uniform = E_KCa_uniform)



       main_First = segmentSpikeTest( -30 )
       main_Middle = segmentSpikeTest( -30 )
       main_Last = segmentSpikeTest( -30 )
       b1.First = segmentSpikeTest( -30 )
       b1.Middle = segmentSpikeTest( -30 )
       b1.Last = segmentSpikeTest( -30 )
       b2.First  = segmentSpikeTest( -30 )
       b2.Middle  = segmentSpikeTest( -30 )
       b2.Last = segmentSpikeTest( -30 )

       # coupling between main axon and branches, see Dayan & Abbott page 219, fig
```

```
6.16
        g_main_b1 = 2.0*g_ax_comp*b1.g_ax_comp/(g_ax_comp+b1.g_ax_comp)
        g_main_b2 = 2.0*g_ax_comp*b2.g_ax_comp/(g_ax_comp+b2.g_ax_comp)

        # initialize compartmental integration parameters
        A = [(g_main_b1+g_main_b2)/C_mem_comp] #initialize branch compartment, in units
of uS/nF
        #####
        B = [-(gNa[0]+gKd[0]+gleak[0]+gGABA[0] + gA[0] + gH[0] + gM[0] + gCaL[0] +
gKCa[0]
              +g_ax_comp+g_main_b1+g_main_b2)/C_mem_comp] #in units of uS/nF
        #####
        C = [g_ax_comp/C_mem_comp] #in units of uS/nF
        #####
        D = [(gNa[0]*E_Na+gKd[0]*E_K+gleak[0]*E_leak+gGABA[0]*E_GABA
             +gA[0]*E_Na + gH[0]*E_H + gM[0]*E_Na + gCaL[0]*E_CaL +
gKCa[0]*E_KCa)/C_mem_comp] #in units of nA/nF
        ######
        a = [A[0]*dt] #in units of uS*ms/nF
        b = [B[0]*dt] #in units of uS*ms/nF
        c = [C[0]*dt] #in units of uS*ms/nF
        d = [(D[0]+(g_main_b1*b1.V[b1.n_comp-1]+g_main_b2*b2.V[b2.n_comp-
1])/C_mem_comp+B[0]*V[0]+C[0]*V[1])*dt] #in units of nA*ms/nF

        for i in range(1, n_comp-1, 1): #initialize compartment integration parameter
arrays, for middle compartments
            A.append(g_ax_comp/C_mem_comp) #in units of uS/nF
            B.append(-(gNa[i]+gKd[i]+gleak[i]+gGABA[i]+gA[i] + gH[i] + gM[i] + gCaL[i]
+ gKCa[i]+2.0*g_ax_comp)/C_mem_comp) #in units of uS/nF
            C.append(g_ax_comp/C_mem_comp) #in units of uS/nF
            D.append((gNa[i]*E_Na+gKd[i]*E_K+gleak[i]*E_leak+gGABA[i]*E_GABA +
                      gA[i]*E_A + gH[i]*E_H + gM[i]*E_M + gCaL[i]*E_CaL +
gKCa[i]*E_KCa)/C_mem_comp) #in units of nA/nF
            a.append(A[i]*dt)
            b.append(B[i]*dt)
            c.append(C[i]*dt)
            d.append((D[i]+A[i]*V[i-1]+B[i]*V[i]+C[i]*V[i+1])*dt)

        A.append(g_ax_comp/C_mem_comp) #in units of uS/nF, for last compartment
        B.append(-(gNa[n_comp-1]+gKd[n_comp-1]+gleak[n_comp-1]+gGABA[n_comp-1] +
                  gA[n_comp-1] + gH[n_comp-1] + gM[n_comp-1] + gCaL[n_comp-1] +
gKCa[n_comp-1]+g_ax_comp)/C_mem_comp) #in units of uS/nF
```

```
       C.append(0.0) #in units of uS/nF
       D.append((gNa[n_comp-1]*E_Na+gKd[n_comp-1]*E_K+gleak[n_comp-
1]*E_leak+gGABA[n_comp-1]*E_GABA+I_stim_now
                 +gA[n_comp-1]*E_A + gH[n_comp-1]*E_H + gM[n_comp-1]*E_M +
gCaL[n_comp-1]*E_CaL + gKCa[n_comp-1]*E_KCa)/C_mem_comp) #in units of nA/nF
       a.append(A[n_comp-1]*dt)
       b.append(B[n_comp-1]*dt)
       c.append(C[n_comp-1]*dt)
       d.append((D[n_comp-1]+A[n_comp-1]*V[n_comp-2]+B[n_comp-1]*V[n_comp-1])*dt)

       A = np.asarray(A)
       B = np.asarray(B)
       C = np.asarray(C)
       D = np.asarray(D)
       a = np.asarray(a)
       b = np.asarray(b)
       c = np.asarray(c)
       d = np.asarray(d)

       b_p = b.copy()
       d_p = d.copy()
       Cas_monitor_main = [CaS_init] #the same compartment as v_rec_middle
       # recording electrodes (located in first and last compartment)# _p stands for
prime as in Dayan and Abbott appendix 6
       V_rec_first = [V_init] #recorded voltage time course in first compartment in mV
       V_rec_second = [V_init] #recorded voltage time course in second compartment in
mV
       V_rec_middle = [V_init] #recorded voltage time course in middle compartment in
mV
       V_rec_nexttoGABA = [V_init] #recorded voltage time course in compartment next
to GABA in mV
       V_rec_last = [V_init] #recorded voltage time course in last compartment in mV


       b1.ACArr = [([(0.0,0.0)],[(b1.g_ax_comp, b1.V[1])])]




       for i in range(1, b1.n_comp-1, 1): #initialize compartment integration
parameter arrays, for middle compartments
           b1.ACArr.append(([(b1.g_ax_comp, b1.V[i-1])], [(b1.g_ax_comp,b1.V[i+1])]))
```

```
        b1.ACArr.append(([(b1.g_ax_comp, b1.V[b1.n_comp-2])], [(g_main_b1,V[0])]))



        Integration.IntegParamInit(b1, b1.ACArr, dt)


        # recording electrodes (located in first and last compartment)# _p stands for
prime as in Dayan and Abbott appendix 6
        b1.V_rec_first = [b1.V_init] #recorded voltage time course in first compartment
in mV
        b1.V_rec_second = [b1.V_init] #recorded voltage time course in second
compartment in mV
        b1.V_rec_middle = [b1.V_init] #recorded voltage time course in middle
compartment in mV
        b1.V_rec_last = [b1.V_init] #recorded voltage time course in last compartment
in mV
        b1.V_rec_nexttoGABA = [V_init] #recorded voltage time course in compartment
next to GABA in mV

        b2.ACArr = [([(0.0,0.0)],[(b2.g_ax_comp, b2.V[1])])]



        for i in range(1, b2.n_comp-1, 1): #initialize compartment integration
parameter arrays, for middle compartments
            b2.ACArr.append(([(b2.g_ax_comp, b2.V[i-1])], [(b2.g_ax_comp,b2.V[i+1])]))



        b2.ACArr.append(([(b2.g_ax_comp, b2.V[b2.n_comp-2])], [(g_main_b2,V[0])]))



        Integration.IntegParamInit(b2, b2.ACArr, dt)

        # recording electrodes (located in first and last compartment)# _p stands for
prime as in Dayan and Abbott appendix 6
        b2.V_rec_first = [b2.V_init] #recorded voltage time course in first compartment
in mV
```

```
        b2.V_rec_second = [b2.V_init] #recorded voltage time course in second
compartment in mV
        b2.V_rec_middle = [b2.V_init] #recorded voltage time course in middle
compartment in mV
        b2.V_rec_last = [b2.V_init] #recorded voltage time course in last compartment
in mV
        b2.V_rec_nexttoGABA = [V_init] #recorded voltage time course in compartment
next to GABA in mV



        ##############################################################################
        ##############################################################################
        ##############################################################################



        # BEGIN SIMULATION
        slotList = fireSlots(5, 20, 1/4, 0.5)         #original slotList = fireSlots(5,
28, 1/5, 1)
        while t_now+dt < t_total:
                t_now += dt
                t.append(t_now)

                stimulatedOrNot = False
                for pair in slotList:
                    s, e = pair
                    if(t_now >= s and t_now < e):
                        stimulatedOrNot = True
                        break
                if stimulatedOrNot:

               # if t_now>=t_stimstart and t_now<t_stimend:

                    I_stim_now=I_stim_amp #apply stimulus current
                else:
                    I_stim_now=0.0;

                # store previous time step values in _old arrays
                V_old = V.copy() #array of previous time step's compartment voltages in
mV
                mNa_old = mNa.copy() #array of previous time step's compartment Na
```

```
activations
               hNa_old = hNa.copy() #array of previous time step's compartment Na
inactivations
               nKd_old = nKd.copy() #array of previous time step's compartment Kd
activations
               mA_old = mA.copy()
               hA_old = hA.copy()
               mH_old = mH.copy()
               mCaL_old = mCaL.copy()
               hCaL_old = hCaL.copy()
               mM_old = mM.copy()
               mKCa_old = mKCa.copy()
               CaS_old = CaS.copy()






               b1.V_old = b1.V.copy() #array of previous time step's compartment
voltages in mV
               b1.mNa_old = b1.mNa.copy() #array of previous time step's compartment
Na activations
               b1.hNa_old = b1.hNa.copy() #array of previous time step's compartment
Na inactivations
               b1.nKd_old = b1.nKd.copy() #array of previous time step's compartment
Kd activations

               b1.mA_old = b1.mA.copy()
               b1.hA_old = b1.hA.copy()
               b1.mH_old = b1.mH.copy()
               b1.mCaL_old = b1.mCaL.copy()
               b1.hCaL_old = b1.hCaL.copy()
               b1.mM_old = b1.mM.copy()
               b1.mKCa_old = b1.mKCa.copy()
               b1.CaS_old = b1.CaS.copy()


               b2.V_old = b2.V.copy() #array of previous time step's compartment
voltages in mV
               b2.mNa_old = b2.mNa.copy() #array of previous time step's compartment
Na activations
               b2.hNa_old = b2.hNa.copy() #array of previous time step's compartment
```

```
Na inactivations
               b2.nKd_old = b2.nKd.copy() #array of previous time step's compartment
Kd activations

               b2.mA_old = b2.mA.copy()
               b2.hA_old = b2.hA.copy()
               b2.mH_old = b2.mH.copy()
               b2.mCaL_old = b2.mCaL.copy()
               b2.hCaL_old = b2.hCaL.copy()
               b2.mM_old = b2.mM.copy()
               b2.mKCa_old = b2.mKCa.copy()
               b2.CaS_old = b2.CaS.copy()




               # integratinng branch 1 dynamic variables
               # first compartment
               # Na
               b1.ACArr = [([(0.0,0.0)],[(b1.g_ax_comp, b1.V[1])])]

               for i in range(1, b1.n_comp-1, 1): #initialize compartment integration
parameter arrays, for middle compartments
                   b1.ACArr.append(([(b1.g_ax_comp, b1.V[i-1])],
[(b1.g_ax_comp,b1.V[i+1])]))

               b1.ACArr.append(([(b1.g_ax_comp, b1.V[b1.n_comp-2])],
[(g_main_b1,V[0])]))

               IntegParamUpdate(b1, b1.ACArr, dt)


               b2.ACArr = [([(0.0,0.0)],[(b2.g_ax_comp, b2.V[1])])]



               for i in range(1, b2.n_comp-1, 1): #initialize compartment integration
parameter arrays, for middle compartments
                   b2.ACArr.append(([(b2.g_ax_comp, b2.V[i-1])],
[(b2.g_ax_comp,b2.V[i+1])]))


               b2.ACArr.append(([(b2.g_ax_comp, b2.V[b2.n_comp-2])],
[(g_main_b2,V[0])]))
               IntegParamUpdate(b2, b2.ACArr, dt)


               ###last branch compartments get axial current from main axon
compartment 0

               # main axon
               # compartment 0 gets axial current from branch compartments n-1
               # Na



               gNa[0] = g_mem_Na_comp * np.power(mNa[0], 2) * hNa[0]
               mNa[0] = mNaUpdate(V_old[0], mNa_old[0])
               hNa[0] = hNaUpdate(V_old[0], hNa_old[0])
               gNa[0] = g_mem_Na_comp * np.power(mNa[0], 2) * hNa[0]

               # Kd
               nKd[0] = nKdUpdate(V_old[0], nKd_old[0])
               gKd[0] = g_mem_Kd_comp * np.power(nKd[0], 4)
               # leak
               gleak[0] = g_mem_leak_comp



               mA[0] = mAUpdate(V_old[0], mA_old[0])
               hA[0] = hAUpdate(V_old[0], hA_old[0])
               gA[0] = g_mem_A_comp * np.power(mA[0], 3) * hA[0]

               mH[0] = mHUpdate(V_old[0], mH_old[0])
               gH[0] = g_mem_H_comp * mH[0]

               mM[0] = mMUpdate(V_old[0], mM_old[0])
               gM[0] = g_mem_M_comp * np.power(mM[0], 2)

               mCaL[0] = mCaLUpdate(V_old[0], mCaL_old[0])
               hCaL[0] = hCaLUpdate(V_old[0], hCaL_old[0])
               gCaL[0] = g_mem_CaL_comp * mCaL[0] * hCaL[0]

               mKCa[0] = mKCaUpdate(mKCa_old[0], CaS_old[0])
               gKCa[0] = g_mem_KCa_comp * mKCa[0]
```

```
               CaS[0] = CaSUpdate(CaS_old[0], gCaL[0] * (V_old[0] - E_CaL), comp_vol)



               # integration parameters
               A[0] = (g_main_b1+g_main_b2)/C_mem_comp #in units of uS/nF
               ####
               B[0] = -(gNa[0]+gKd[0]+gleak[0]+gGABA[0]+gA[0]

+gH[0]+gM[0]+gCaL[0]+gKCa[0]+g_ax_comp+g_main_b1+g_main_b2)/C_mem_comp #in units of
uS/nF
               ####
               C[0] = g_ax_comp/C_mem_comp #in units of uS/nF
               ####
               D[0] = (gNa[0]*E_Na+gKd[0]*E_K+gleak[0]*E_leak+gGABA[0]*E_GABA
                       + gA[0]*E_A + gH[0]*E_H + gM[0]*E_M + gCaL[0]*E_CaL +
gKCa[0]*E_KCa)/C_mem_comp #in units of nA/nF
               ####
               a[0] = A[0]*dt #in units of uS*ms/nF
               b[0] = B[0]*dt #in units of uS*ms/nF
               c[0] = C[0]*dt #in units of uS*ms/nF
               d[0] = (D[0]+(g_main_b1*b1.V[b1.n_comp-1]+g_main_b2*b2.V[b2.n_comp-
1])/C_mem_comp+B[0]*V[0]+C[0]*V[1])*dt #in units of nA*ms/nF

               b_p[0] = b[0] # _p stands for prime as in Dayan and Abbott appendix 6
               d_p[0] = d[0]

               for i in range(1, n_comp-1, 1): #middle compartments
                   # Na
                   mNa[i] = mNaUpdate(V_old[i], mNa_old[i])
                   hNa[i] = hNaUpdate(V_old[i], hNa_old[i])
                   gNa[i] = g_mem_Na_comp * np.power(mNa[i], 2) * hNa[i]

                   # Kd
                   nKd[i] = nKdUpdate(V_old[i], nKd_old[i])

                   gKd[i] = g_mem_Kd_comp * np.power(nKd[i], 4)

                   # leak
                   gleak[i] = g_mem_leak_comp

                   mA[i] = mAUpdate(V_old[i], mA_old[i])
                   hA[i] = hAUpdate(V_old[i], hA_old[i])
```

```
                    gA[i] = g_mem_A_comp * np.power(mA[i], 3) * hA[i]

                    mH[i] = mHUpdate(V_old[i], mH_old[i])
                    gH[i] = g_mem_H_comp * mH[i]

                    mM[i] = mMUpdate(V_old[i], mM_old[i])
                    gM[i] = g_mem_M_comp * np.power(mM[i], 2)

                    mCaL[i] = mCaLUpdate(V_old[i], mCaL_old[i])
                    hCaL[i] = hCaLUpdate(V_old[i], hCaL_old[i])
                    gCaL[i] = g_mem_CaL_comp * mCaL[i] * hCaL[i]

                    mKCa[i] = mKCaUpdate(mKCa_old[i], CaS_old[i])
                    gKCa[i] = g_mem_KCa_comp * mKCa[i]
                    CaS[i] = CaSUpdate(CaS_old[i], gCaL[i] * (V_old[i] -
E_CaL),comp_vol)#gCaL[i] * (V_old[i] - E_CaL)
                    #if i == mid_ind:
                        #print(f'I_CaL: {gCaL[i] * (V_old[i] - E_CaL)}')
                        #print(f'[Ca]: {CaS[i]}')



                    # integration parameters
                    A[i] = g_ax_comp/C_mem_comp #in units of uS/nF
                    #####
                    B[i] = -(gNa[i]+gKd[i]+gleak[i]+gGABA[i]+gA[i]
                                +gH[i]+gM[i]+gCaL[i]+gKCa[i]+2.0*g_ax_comp)/C_mem_comp
                    #####
                    C[i] = g_ax_comp/C_mem_comp #in units of uS/nF
                    #####
                    D[i] = (gNa[i]*E_Na+gKd[i]*E_K+gleak[i]*E_leak+gGABA[i]*E_GABA
                            + gA[i]*E_A + gH[i]*E_H + gM[i]*E_M + gCaL[i]*E_CaL +
gKCa[i]*E_KCa)/C_mem_comp
                    #####
                    a[i] = A[i]*dt
                    b[i] = B[i]*dt
                    c[i] = C[i]*dt
                    d[i] = (D[i]+A[i]*V[i-1]+B[i]*V[i]+C[i]*V[i+1])*dt

                    b_p[i] = b[i] + a[i]*c[i-1]/(1-b_p[i-1]) #equation 6.54 in D&A
                    d_p[i] = d[i] + a[i]*d_p[i-1]/(1-b_p[i-1]) #equation 6.55 in D&A

                #compartment n_comp-1
                # Na
                mNa[n_comp-1] = mNaUpdate(V_old[n_comp-1], mNa_old[n_comp-1])
                hNa[n_comp-1] = hNaUpdate(V_old[n_comp-1], hNa_old[n_comp-1])
                gNa[n_comp-1] = g_mem_Na_comp * np.power(mNa[n_comp-1], 2) *
hNa[n_comp-1]

                # Kd
                nKd[n_comp-1] = nKdUpdate(V_old[n_comp-1], nKd_old[n_comp-1])
                gKd[n_comp-1] = g_mem_Kd_comp * np.power(nKd[n_comp-1], 4)
                        # leak
                gleak[n_comp-1] = g_mem_leak_comp
                mA[n_comp-1] = mAUpdate(V_old[n_comp-1], mA_old[n_comp-1])
                hA[n_comp-1] = hAUpdate(V_old[n_comp-1], hA_old[n_comp-1])
                gA[n_comp-1] = g_mem_A_comp * np.power(mA[n_comp-1], 3) * hA[n_comp-1]

                mH[n_comp-1] = mHUpdate(V_old[n_comp-1], mH_old[n_comp-1])
                gH[n_comp-1] = g_mem_H_comp * mH[n_comp-1]

                mM[n_comp-1] = mMUpdate(V_old[n_comp-1], mM_old[n_comp-1])
                gM[n_comp-1] = g_mem_M_comp * np.power(mM[n_comp-1], 2)

                mCaL[n_comp-1] = mCaLUpdate(V_old[n_comp-1], mCaL_old[n_comp-1])
                hCaL[n_comp-1] = hCaLUpdate(V_old[n_comp-1], hCaL_old[n_comp-1])
                gCaL[n_comp-1] = g_mem_CaL_comp * mCaL[n_comp-1] * hCaL[n_comp-1]

                mKCa[n_comp-1] = mKCaUpdate(mKCa_old[n_comp-1], CaS_old[n_comp-1])
                gKCa[n_comp-1] = g_mem_KCa_comp * mKCa[n_comp-1]
                CaS[n_comp-1] = CaSUpdate(CaS_old[n_comp-1], gCaL[n_comp-1] *
(V_old[n_comp-1] - E_CaL),comp_vol)


                # integration parameters
                A[n_comp-1] = g_ax_comp/C_mem_comp #in units of uS/nF
                ####
                B[n_comp-1] = -(gNa[n_comp-1]+gKd[n_comp-1]+gleak[n_comp-
1]+gGABA[n_comp-1]
                                +gA[n_comp-1]+gH[n_comp-1]+gM[n_comp-1]+gCaL[n_comp-
1]+gKCa[n_comp-1]
                                +g_ax_comp)/C_mem_comp
                ####
                C[n_comp-1] = 0.0 #in units of uS/nF
                ####
```

```
               D[n_comp-1] = (gNa[n_comp-1]*E_Na+gKd[n_comp-1]*E_K+gleak[n_comp-
1]*E_leak+gGABA[n_comp-1]*E_GABA+I_stim_now
                              + gA[n_comp-1]*E_A + gH[n_comp-1]*E_H + gM[n_comp-1]*E_M
+ gCaL[n_comp-1]*E_CaL + gKCa[n_comp-1]*E_KCa)/C_mem_comp
               ####
               a[n_comp-1] = A[n_comp-1]*dt
               b[n_comp-1] = B[n_comp-1]*dt
               c[n_comp-1] = C[n_comp-1]*dt
               d[n_comp-1] = (D[n_comp-1]+A[n_comp-1]*V[n_comp-2]+B[n_comp-
1]*V[n_comp-1])*dt

               b_p[n_comp-1] = b[n_comp-1] + a[n_comp-1]*c[n_comp-2]/(1-b_p[n_comp-2])
#equation 6.54 in D&A
               d_p[n_comp-1] = d[n_comp-1] + a[n_comp-1]*d_p[n_comp-2]/(1-b_p[n_comp-
2]) #equation 6.55 in D&A



              ###############
               deltaV[n_comp-1] = d_p[n_comp-1]/(1-b_p[n_comp-1]) #equation 6.56 in
D&A, update voltage for last compartment
               V[n_comp-1] = V_old[n_comp-1] + deltaV[n_comp-1]

               for i in range(n_comp-1, 0, -1): # step through the middle compartments
backward to update voltages
                   deltaV[i-1] = (c[i-1]*deltaV[i]+d_p[i-1])/(1-b_p[i-1])
                   V[i-1] = V_old[i-1] + deltaV[i-1]

               # branch 1
               b1.deltaV[b1.n_comp-1] = b1.d_p[b1.n_comp-1]/(1-b1.b_p[b1.n_comp-1])
#equation 6.56 in D&A, update voltage for last compartment
               b1.V[b1.n_comp-1] = b1.V_old[b1.n_comp-1] + b1.deltaV[b1.n_comp-1]

               for i in range(b1.n_comp-1, 0, -1): # step through the middle
compartments backward to update voltages
                   b1.deltaV[i-1] = (b1.c[i-1]*b1.deltaV[i]+b1.d_p[i-1])/(1-b1.b_p[i-
1])
                   b1.V[i-1] = b1.V_old[i-1] + b1.deltaV[i-1]

               # branch 2
               b2.deltaV[b2.n_comp-1] = b2.d_p[b2.n_comp-1]/(1-b2.b_p[b2.n_comp-1])
#equation 6.56 in D&A, update voltage for last compartment
               b2.V[b2.n_comp-1] = b2.V_old[b2.n_comp-1] + b2.deltaV[b2.n_comp-1]
```

```
            for i in range(b2.n_comp-1, 0, -1): # step through the middle
compartments backward to update voltages
                b2.deltaV[i-1] = (b2.c[i-1]*b2.deltaV[i]+b2.d_p[i-1])/(1-b2.b_p[i-
1])
                b2.V[i-1] = b2.V_old[i-1] + b2.deltaV[i-1]
        ######################

            ###branch compartments n-1 updated based on main axon compartment 0
voltage



            I_stim.append(I_stim_now) #record stimulus current
            Cas_monitor_main.append(CaS[mid_ind])
            V_rec_first.append(V[n_comp-1]) #record voltage in stimulated
compartment
            V_rec_second.append(V[n_comp-2]) #record voltage in compartment next to
stimulus
            V_rec_middle.append(V[mid_ind]) #record voltage in middle compartment
            V_rec_last.append(V[0]) #record voltage in branch compartment
            V_rec_nexttoGABA.append(V[2]) #recorded voltage time course in
compartment next to GABA in mV

            b1.V_rec_first.append(b1.V[b1.n_comp-1]) #record voltage in first
compartment
            b1.V_rec_second.append(b1.V[b1.n_comp-2]) #record voltage in second
compartment
            b1.V_rec_middle.append(b1.V[b1.mid_ind]) #record voltage in middle
compartment
            b1.V_rec_last.append(b1.V[0]) #record voltage in last compartment
            b1.V_rec_nexttoGABA.append(b1.V[b1.n_comp-3]) #recorded voltage time
course in compartment next to GABA in mV

            b2.V_rec_first.append(b2.V[b2.n_comp-1]) #record voltage in first
compartment
            b2.V_rec_second.append(b2.V[b2.n_comp-2]) #record voltage in second
compartment
            b2.V_rec_middle.append(b2.V[b2.mid_ind]) #record voltage in middle
compartment
            b2.V_rec_last.append(b2.V[0]) #record voltage in last compartment
            b2.V_rec_nexttoGABA.append(b2.V[b2.n_comp-3]) #recorded voltage time
```

```
course in compartment next to GABA in mV
               main_First.findSpike(V_rec_first[-1])
               main_Middle.findSpike(V_rec_middle[-1])
               main_Last.findSpike(V_rec_last[-1])
               b1.First.findSpike(b1.V_rec_first[-1])
               b1.Middle.findSpike(b1.V_rec_middle[-1])
               b1.Last.findSpike(b1.V_rec_last[-1])
               b2.First.findSpike(b2.V_rec_first[-1])
               b2.Middle.findSpike(b2.V_rec_middle[-1])
               b2.Last.findSpike(b2.V_rec_last[-1])



       # END SIMULATION



       plotI_stim(t, I_stim, 'Main', t_total) #Comment out if you don't want to output
the graph
       plotVfirst(t, V_rec_first, 'Main', t_total) #Comment out if you don't want to
output the graph
       #plotVsecond(t, V_rec_second, 'Main', t_total) #Comment out if you don't want
to output the graph
       plotVmiddle(t, V_rec_middle, 'Main', t_total) #Comment out if you don't want to
output the graph
       #plotVnexttoGABA(t, V_rec_nexttoGABA, 'Main', t_total) #Comment out if you
don't want to output the graph
       plotVlast(t, V_rec_last, 'Main', t_total) #Comment out if you don't want to
output the graph

       plotVfirst(t, b1.V_rec_first, 'b1', t_total) #Comment out if you don't want to
output the graph
       #plotVsecond(t, b1.V_rec_second, 'b1', t_total) #Comment out if you don't want
to output the graph
       #plotVnexttoGABA(t, b1.V_rec_nexttoGABA, 'Main', t_total) #Comment out if you
don't want to output the graph
       plotVmiddle(t, b1.V_rec_middle, 'b1', t_total) #Comment out if you don't want
to output the graph
       plotVlast(t, b1.V_rec_last, 'b1', t_total) #Comment out if you don't want to
output the graph

       plotVfirst(t, b2.V_rec_first, 'b2', t_total) #Comment out if you don't want to
output the graph
```

```
        #plotVsecond(t, b2.V_rec_second, 'b2', t_total) #Comment out if you don't want
to output the graph
        #plotVnexttoGABA(t, b2.V_rec_nexttoGABA, 'Main', t_total) #Comment out if you
don't want to output the graph
        plotVmiddle(t, b2.V_rec_middle, 'b2', t_total) #Comment out if you don't want
to output the graph
        plotVlast(t, b2.V_rec_last, 'b2', t_total) #Comment out if you don't want to
output the graph
        plotCaS(t, Cas_monitor_main, 'Main_middle', t_total)
        print("Your simulations are successfully completed!")


        print("main_First_Spike_Number: %d" % main_First.Seg_spike_num)
        print("main_Middle_Spike_Number: %d" % main_Middle.Seg_spike_num)
        print("main_Last_Spike_Number: %d" % main_Last.Seg_spike_num)
        print("b1.First_Spike_Number: %d" % b1.First.Seg_spike_num)
        print("b1.Middle_Spike_Number: %d" % b1.Middle.Seg_spike_num)
        print("b1.Last_Spike_Number: %d" % b1.Last.Seg_spike_num)
        print("b2.First_Spike_Number: %d" % b2.First.Seg_spike_num)
        print("b2.Middle_Spike_Number: %d" % b2.Middle.Seg_spike_num)
        print("b2.Last_Spike_Number: %d" % b2.Last.Seg_spike_num)
        #arrayR = [("mf", main_First.Seg_spike_num), ("mm", main_Middle.Seg_spike_num),
("ml", main_Last.Seg_spike_num), ("b1f", b1.First.Seg_spike_num), ("b1m",
b1.Middle.Seg_spike_num),
         #          ("b1l", b1.Last.Seg_spike_num), ("b2f", b2.First.Seg_spike_num),
("b2m", b2.Middle.Seg_spike_num), ("b2l", b2.Last.Seg_spike_num)]
        #a = "b2.dim: " + str(round(index1 * 0.05, 2))
        #dict[a] = arrayR
        index1 += 1
        oneDarray.append(b2.Last.Seg_spike_num)
    twoDarray.append(oneDarray)
    index2 += 1
```

Parallel-axon Model

```
import pylab as plt
from scipy.integrate import odeint
import numpy as np
import pandas as pd
from scipy.signal import find_peaks
from scipy import signal
import csv
import math
from segTest import segmentSpikeTest
from axonBranch import axBranch
from integration import Integration






#This part uses segment SpikeTest from segTest.py to track spike conditions in each
segment

def mNaUpdate(Vold, mNaOld):
       alpha = 0.36 * (Vold + 33) / (1 - np.exp(-(Vold + 33) / 3))
       beta = - 0.4 * (Vold + 42) / (1 - np.exp((Vold + 42) / 20))
       vinf = alpha / (alpha + beta)
       tau = 2 / (alpha + beta)
       tau = tau * taufac
       new_mNa = vinf + (mNaOld- vinf) * np.exp(-dt / tau) if dt < tau else vinf
       return new_mNa

def hNaUpdate(Vold, hNaOld):
       alpha = - 0.1 * (Vold + 55) / (1 - np.exp((Vold + 55) / 6))
       beta = 4.5 / (1 + np.exp(-Vold / 10))
       vinf = alpha / (alpha + beta)
       tau = 2 / (alpha + beta)
       tau = tau * taufac
       new_hNa = vinf + (hNaOld - vinf) * np.exp(-dt / tau) if dt < tau else vinf
       return new_hNa

def nKdUpdate(Vold, nKdOld):
       alpha = 0.0047 * (Vold - 8) / (1 - np.exp(-(Vold - 8) / 12))
       beta = np.exp(-(Vold + 127) / 30)
       vinf = alpha / (alpha + beta)
       alpha = 0.0047 * (Vold + 12) / (1 - np.exp(-(Vold + 12) / 12))
       beta = np.exp(-(Vold + 147) / 30)
       tau = 1 / (alpha + beta)
       tau = tau * taufac
       new_nKd = vinf + (nKdOld - vinf) * np.exp(-dt / tau) if dt < tau else vinf
       return new_nKd
def mCaLUpdate(Vold, mCaLOld):
       alpha_mCaL = 7.5 / (1 + np.exp((13 - Vold) / 7))
       beta_mCaL = 1.65 / (1 + np.exp((Vold - 14) / 4))
       mCaL_inf = alpha_mCaL / (alpha_mCaL + beta_mCaL)
       tau_mCaL = 1 / (alpha_mCaL + beta_mCaL)
       new_mCaL = mCaL_inf + (mCaLOld - mCaL_inf) * np.exp(-dt / tau_mCaL) if dt <
tau_mCaL else mCaL_inf
       return new_mCaL
def hCaLUpdate(Vold, hCaLOld):
       alpha_hCaL = 0.0068 / (1 + np.exp((Vold + 30) / 12))
       beta_hCaL = 0.06 / (1 + np.exp(-Vold / 11))
       hCaL_inf = alpha_hCaL / (alpha_hCaL + beta_hCaL)
       tau_hCaL = 1 / (alpha_hCaL + beta_hCaL)
       new_hCaL = hCaL_inf + (hCaLOld - hCaL_inf) * np.exp(-dt / tau_hCaL) if dt <
tau_hCaL else hCaL_inf
       return new_hCaL
def mMUpdate(Vold, mMOld):
       mM_inf = 1 / (1 + np.exp(-(Vold + 35) / 10))
       tau_mM = 2000 / (3.3 * (np.exp((Vold + 35) / 40) + np.exp(-(Vold + 35) / 20)))
       new_mM = mM_inf + (mMOld - mM_inf) * np.exp(-dt / tau_mM) if dt < tau_mM else
mM_inf
       return new_mM

"""
CaS needs input and so does SCa, also no need of Vold
"""
def mKCaUpdate(mKCaOld, CaS_old):
       mKCa_inf = CaS_old ** 2 / (CaS_old ** 2 + SCa ** 2)
       tau_mKCa = tauKCa_0 / (1 + (CaS_old / SCa) ** 2)
       new_mKCa = mKCa_inf + (mKCaOld - mKCa_inf) * np.exp(-dt / tau_mKCa) if dt <
tau_mKCa else mKCa_inf
       return new_mKCa

def mAUpdate(Vold, mAOld):
       mA_inf = (0.0761 * np.exp((Vold + 94.22) / 31.84) / (1 + np.exp((Vold + 1.17) /
28.93))) ** (1/3)
       tau_mA = 0.3632 + 1.158 / (1 + np.exp((Vold + 55.96) / 20.12))
       new_mA = mA_inf + (mAOld - mA_inf) * np.exp(-dt / tau_mA) if dt < tau_mA else
mA_inf
       return new_mA
def hAUpdate(Vold, hAOld):
       hA_inf = (1 / (1 + np.exp(0.069 * (Vold + 53.3)))) ** 4
       tau_hA = (0.124 + 2.678 / (1 + np.exp((Vold + 50) / 16.027))) * tau_hA_scale
       new_hA = hA_inf + (hAOld - hA_inf) * np.exp(-dt / tau_hA) if dt < tau_hA else
hA_inf
       return new_hA

def mHUpdate(Vold, mHOld):
       mh_inf = 1 / (1 + np.exp((Vold + 87.6) / 11.7))
       tau_mh_activ = 53.5 + 67.7 * np.exp(-(Vold + 120) / 22.4)
       tau_mh_deactiv = 40.9 - 0.45 * Vold
       tau_mh = tau_mh_activ if mh_inf > mHOld else tau_mh_deactiv
       new_mH = mh_inf + (mHOld - mh_inf) * np.exp(-dt / tau_mh)
       return new_mH

def CaSUpdate(CaSold, ICaL_old, comp_vol):
   #print (ICaL_old)
   #print(comp_vol)
   alpha_CaS = dt * 1e6 / (comp_vol * 2 * 96485) #uM        1mM/1000uM
   #alpha_CaS = 0.2
   #print(alpha_CaS)

   return  CaSold * np.exp(-f * kCaS * dt) - alpha_CaS / kCaS * ICaL_old * (1 -
np.exp(-f * kCaS * dt))

#fire function
def fireSlots(start, end, freq, duration):
       stimTime = end - start
       numStim = math.floor(stimTime * freq)
       wholeDura = 1 / freq
       if(wholeDura <= duration):
           print ("Too large of a duration!")
           return



       cur = start
       result = []
        for i in range(numStim):
            pair = (cur, cur + duration)
            cur = cur + wholeDura
            result.append(pair)
        return result

def integCopy(axo):
   axo.V_old = axo.V.copy() #array of previous time step's compartment voltages in mV
   axo.mNa_old = axo.mNa.copy() #array of previous time step's compartment Na
activations
   axo.hNa_old = axo.hNa.copy() #array of previous time step's compartment Na
inactivations
   axo.nKd_old = axo.nKd.copy() #array of previous time step's compartment Kd
activations

   axo.mA_old = axo.mA.copy()
   axo.hA_old = axo.hA.copy()
   axo.mH_old = axo.mH.copy()
   axo.mCaL_old = axo.mCaL.copy()
   axo.hCaL_old = axo.hCaL.copy()
   axo.mM_old = axo.mM.copy()
   axo.mKCa_old = axo.mKCa.copy()
   axo.CaS_old = axo.CaS.copy()

def gatingUpdate(axo, i):
   axo.mNa[i] = mNaUpdate(axo.V_old[i], axo.mNa_old[i])
   axo.hNa[i] = hNaUpdate(axo.V_old[i], axo.hNa_old[i])
   axo.gNa[i] = axo.g_mem_Na_comp * np.power(axo.mNa[i], 2) * axo.hNa[i]

   # Kd
   axo.nKd[i] = nKdUpdate(axo.V_old[i], axo.nKd_old[i])
   axo.gKd[i] = axo.g_mem_Kd_comp * np.power(axo.nKd[i], 4)
   # leak
   axo.gleak[i] = axo.g_mem_leak_comp

   axo.mA[i] = mAUpdate(axo.V_old[i], axo.mA_old[i])
   axo.hA[i] = hAUpdate(axo.V_old[i], axo.hA_old[i])
   axo.gA[i] = axo.g_mem_A_comp * np.power(axo.mA[i], 3) * axo.hA[i]

   axo.mH[i] = mHUpdate(axo.V_old[i], axo.mH_old[i])
   axo.gH[i] = axo.g_mem_H_comp * axo.mH[i]
   axo.mM[i] = mMUpdate(axo.V_old[i], axo.mM_old[i])
   axo.gM[i] = axo.g_mem_M_comp * np.power(axo.mM[i], 2)

   axo.mCaL[i] = mCaLUpdate(axo.V_old[i], axo.mCaL_old[i])
   axo.hCaL[i] = hCaLUpdate(axo.V_old[i], axo.hCaL_old[i])
   axo.gCaL[i] = axo.g_mem_CaL_comp * axo.mCaL[i] * axo.hCaL[i]

   axo.mKCa[i] = mKCaUpdate(axo.mKCa_old[i], axo.CaS_old[i])
   axo.gKCa[i] = axo.g_mem_KCa_comp * axo.mKCa[i]

   axo.CaS[i] = CaSUpdate(axo.CaS_old[i], axo.gCaL[i] * (axo.V_old[i] - axo.E_CaL),
axo.comp_vol) #

#stimulus {index:current, }
def IntegParamUpdate(axo, ACsettingArray,dt, stimDict=None):
   for i in range(len(ACsettingArray)):
       gatingUpdate(axo, i)
       temp_B = (-(axo.gNa[i]+axo.gKd[i]+axo.gleak[i]+axo.gGABA[i]+axo.gA[i] +
axo.gH[i] + axo.gM[i] + axo.gCaL[i] + axo.gKCa[i])/axo.C_mem_comp)#in units of uS/nF

       A_condVpairArr, C_condVpairArr = ACsettingArray[i]
       temp_A = 0.0
       temp_d = 0.0

       for aPair in A_condVpairArr:
           conductance, comp_voltage = aPair
           temp_A += conductance / axo.C_mem_comp
           temp_d += conductance / axo.C_mem_comp * comp_voltage
           temp_B += (-conductance/axo.C_mem_comp)


       temp_C = 0.0
       for cPair in C_condVpairArr:
           conductance, comp_voltage = cPair
           temp_C += conductance / axo.C_mem_comp
           temp_d += conductance / axo.C_mem_comp * comp_voltage
           temp_B += (-conductance/axo.C_mem_comp)

       axo.A[i] = (temp_A)
       axo.B[i] = (temp_B)
       axo.C[i] = (temp_C)
```

```
        if stimDict == None:
            temp_stim = 0.0
        else:
            temp_stim = stimDict.get(i, 0.0)

        axo.D[i] =
((axo.gNa[i]*axo.E_Na+axo.gKd[i]*axo.E_K+axo.gleak[i]*axo.E_leak+axo.gGABA[i]*axo.E_GA
BA + temp_stim
                +axo.gA[i]*axo.E_A + axo.gH[i]*axo.E_H + axo.gM[i]*axo.E_M +
axo.gCaL[i]*axo.E_CaL + axo.gKCa[i]*axo.E_KCa)/axo.C_mem_comp) #in units of nA/nF
        axo.a[i] = (temp_A * dt)
        axo.b[i] = (temp_B * dt)
        axo.c[i] = (temp_C * dt)

        axo.d[i] = ((temp_d + axo.D[i] + axo.B[i] * axo.V[i]) * dt)
        if i == 0:
            axo.b_p[0] = axo.b[0] # _p stands for prime as in Dayan and Abbott appendix
6
            axo.d_p[0] = axo.d[0]
        else:
            axo.b_p[i] = axo.b[i] + axo.a[i]*axo.c[i-1]/(1-axo.b_p[i-1]) #equation 6.54
in D&A
            axo.d_p[i] = axo.d[i] + axo.a[i]*axo.d_p[i-1]/(1-axo.b_p[i-1]) #equation
6.55 in D&A

# Plot I_stim (stimulus current in first compartment) vs time
def plotI_stim(time, stimulus, name, total_time):
   plt.figure(figsize=(12,9))
   plt.plot(time, stimulus)
   plt.title(name + ': I_stim vs time')
   plt.xlabel('t (ms)')
   plt.ylabel('I (nA)')
   plt.xlim(0, total_time)
   plt.ylim(-1, 5)
   #axes = plt.gca()
   #axes.yaxis.grid()
   plt.show()

# Plot V_rec_first (recorded membrane voltage in first compartment) vs time
def plotVfirst(time, firstVoltage, name, total_time):
   plt.figure(figsize=(12,9))
   plt.plot(time, firstVoltage)
   plt.title(name + ': V_first vs time')
   plt.xlabel('t (ms)')
   plt.ylabel('V (mV)')
   plt.xlim(0, total_time)
   #plt.xlim(0.9, 1.2)
   plt.ylim(-100, 100)
   #axes = plt.gca()
   #axes.yaxis.grid()
   plt.show()

# Plot V_rec_second (recorded membrane voltage in second compartment) vs time
def plotVsecond(time, secondVoltage, name, total_time):
   plt.figure(figsize=(12,9))
   plt.plot(time, secondVoltage)
   plt.title(name + ': V_second vs time')
   plt.xlabel('t (ms)')
   plt.ylabel('V (mV)')
   plt.xlim(0, total_time)
   #plt.xlim(0.9, 1.2)
   plt.ylim(-100, 100)
   #axes = plt.gca()
   #axes.yaxis.grid()
   plt.show()

# Plot V_rec_middle (recorded membrane voltage in middle compartment) vs time
def plotVmiddle(time, middleVoltage, name, total_time):
   plt.figure(figsize=(12,9))
   plt.plot(time, middleVoltage)
   plt.title(name + ': V_middle vs time')
   plt.xlabel('t (ms)')
   plt.ylabel('V (mV)')
   plt.xlim(0, total_time)
   #plt.xlim(0.9, 1.2)
   plt.ylim(-100, 100)
   #axes = plt.gca()
   #axes.yaxis.grid()
   plt.show()

# Plot V_rec_last (recorded membrane voltage in last compartment) vs time
def plotVlast(time, lastVoltage, name, total_time):
   plt.figure(figsize=(12,9))
   plt.plot(time, lastVoltage)
```

```
    plt.title(name + ': V_last vs time')
    plt.xlabel('t (ms)')
    plt.ylabel('V (mV)')
    plt.xlim(0, total_time)
    #plt.xlim(0.9, 1.2)
    plt.ylim(-100, 100)
    #axes = plt.gca()
    #axes.yaxis.grid()
    plt.show()

# Plot V_rec_nexttoGABA (recorded membrane voltage in compartment next to GABA) vs
time
def plotVnexttoGABA(time, nexttoGABAVoltage, name, total_time):
    plt.figure(figsize=(12,9))
    plt.plot(time, nexttoGABAVoltage)
    plt.title(name + ': V_nexttoGABA vs time')
    plt.xlabel('t (ms)')
    plt.ylabel('V (mV)')
    plt.xlim(0, total_time)
    #plt.xlim(0.9, 1.2)
    plt.ylim(-100, 100)
    #axes = plt.gca()
    #axes.yaxis.grid()
    plt.show()
### ALL ARRAYS ACROSS COMPARTMENTS WILL USE INDICES 0 THROUGH n_comp-1
### THIS IS DIFFERENT FROM DAYAN & ABBOTT, which uses 1 THROUGH N

def plotCaS(time, CaS, name, total_time):
    plt.figure(figsize=(12,9))
    plt.plot(time, CaS)
    plt.title(name + ': Ca2+ Concentration vs time')
    plt.xlabel('t (ms)')
    plt.ylabel('CaS (uM)')
    plt.xlim(0, total_time)
    #plt.xlim(0.9, 1.2)
    plt.ylim(0.001, 0.002)
    #axes = plt.gca()
    #axes.yaxis.grid()
    plt.show()

def plotVgeneral(time, voltage, title, total_time, subplotNum):
    plt.subplot(subplotNum)
```

```
   plt.plot(time, voltage)
   plt.title(title)
   plt.xlabel('t (ms)')
   #plt.grid()
   plt.ylabel('V (mV)')
   plt.xlim(0, total_time)
   #plt.xlim(0.9, 1.2)
   plt.ylim(-100, 100)
   #axes = plt.gca()
   #axes.yaxis.grid()


index2 = 0
b1d_ini = 0.4
b2d_ini = 0.13
twoDarray = []

while index2 < 1:
   oneDarray = []
   index1 = 0
   index2+=1
   while index1 < 1:

       index1 +=1




       # time parameters
       dt = 0.01 #numerical integration time step
       t_total = 25.0 #total simulation time in ms
       t_now = 0.0 #time right now in ms
       t = [t_now] #time array in ms

       # stimulus parameters (stimulus is current pulse injected in first compartment)
       t_stimstart = 5.0 #stimulation start time
       t_stimend = 5.5 #stimulation end time
       I_stim_amp = 0.15 #stimulation current amplitude in nA, 0.15 is good
       I_stim_now = 0.0 #stimulus current in nA
       I_stim = [I_stim_now] #stimulus current time course

       # temperature
       default_temp_C = 22 #default temperature in Celcius. Hochman lab exp mostly at
```

```
room temperature, 22C. Assumption is that default model parameters are tuned to this
temperature.
        default_temp_K = default_temp_C + 273.15 #conversion to Kelvin


        # ADJUST THIS TO CHANGE TEMPERATURE THROUGHOUT:
        temp_C = 39.6 #temperature used in simulation in Celsius


        temp_K = temp_C + 273.15 #conversion to Kelvin
        Q10 = 3.0 #Q10 for adjusting activation and inactivation rates, typical range
for ion channels is 2.4 - 4, see
https://link.springer.com/referenceworkentry/10.1007%2F978-1-4614-7320-6_236-1
        taufac = np.power(Q10, (default_temp_K-temp_K)/10) #factor to multiply
activation and inactivation time constants to adjust for temperature dependence of
gating dynamics

        # 'uniform' parameters set here will apply throughout the model, so they don't
have to be changed for each branch individually
        # 'default' means not adjusted for temperature relative to the McKinnon
parameters
        E_Na_uniform_default = 60.0 #Na reversal potential in mV, according to
McKinnon, default is 60mV
        E_Na_uniform = E_Na_uniform_default * temp_K / default_temp_K #adjust according
to Nernst equation
        E_leak_uniform_default = -55.0 #leak reversal potential in mV, according to
McKinnon, default is -55mV
        E_leak_uniform = E_leak_uniform_default * temp_K / default_temp_K #adjust
according to Nernst equation
        E_K_uniform_default = -90.0 #K reversal potential in mV, according to McKinnon,
default is -90mV
        E_K_uniform = E_K_uniform_default * temp_K / default_temp_K #adjust according
to Nernst equation
        E_GABA_uniform_default = -60.0 #GABA (i.e., Cl-) reversal potential in mV, see
Prescott paper fig. 6, vary from -65mV (control) to -50mV (SCI)
        E_GABA_uniform = E_GABA_uniform_default * temp_K / default_temp_K
        #####
        E_A_uniform_default = -90.0 #default -90
        E_A_uniform = E_A_uniform_default* temp_K  /default_temp_K

        E_H_uniform_default = -32.0 #default -32
        E_H_uniform = E_H_uniform_default * temp_K /default_temp_K
```

```
       E_M_uniform_default = -90.0 #default -90
       E_M_uniform = E_M_uniform_default * temp_K /default_temp_K

       E_CaL_uniform_default = 120.0 #default 120
       E_CaL_uniform = E_CaL_uniform_default * temp_K /default_temp_K

       E_KCa_uniform_default = -90.0 #default -90
       E_KCa_uniform = E_KCa_uniform_default * temp_K /default_temp_K




       #####
       """
       ADD E_OtherChannel_Uniform
       """
       G_Na_abs_uniform = 7.0 #Na conductance in nS, based on McKinnon Table 1,
default is 300nS
       G_Kd_abs_uniform = 100.0 #Kd conductance in nS, based on McKinnon Table 1,
default is 2000nS
       G_leak_abs_uniform = 1.0 #leak conductance in nS, based on McKinnon Table 1,
default is 1nS
       G_GABA_uniform = 0.0 #GABA conductance in nS

       G_A_abs_uniform  = 0 #default 50 nS, used 1, high impact, hyperpolarize
       G_H_abs_uniform  = 0 #default 1nS, used 1, low impact, depolarize

       G_M_abs_uniform  = 0 #default 50 nS,used 1, decrease the number spikes,
       G_CaL_abs_uniform  = 1.2 #default 1.2 nS, used 1.2
       G_KCa_abs_uniform  = 0 #default 50 nS, used 1

       G_gap_junction = 0.02#nS
       #sparse firing
       #0,06 too large
       #0.02 middle
       #0.002 weak
       #0, no connection

       #0.04 too large
       #0.02 middle
       #0.002 weak
       #0, no connection
```

```
        ###########
        f = 0.01                    # percent of free to bound Ca2+
        alpha_CaS = 0.2               # uM/pA; convertion factor from current to
concentration 0.002 original
        kCaS = 0.024                # /ms; Ca2+ removal rate, kCaS is proportional to
1/tau_removal; 0.008 - 0.025

        SCa = 1                     # uM; half-saturation of [Ca2+]; 25uM in Ermentrount
book, 0.2uM in Kurian et al. 2011
        tauKCa_0 = 50               # ms
        tau_hA_scale = 100           # scaling factor for tau_hA

        """
        ADD G_OtherChannel_Uniform
        """
        # model geometry settings
        # main axon
        L = 200.0 #length of axon in um, distance between sympathetic ganglia is about
1mm
        d = 0.5 #diameter of axon in um
        l_comp = 10.0 #length of each compartment in um
        n_comp = int(L/l_comp) #number of compartments





        comp_vol = np.pi * ((d/2) ** 2) * l_comp #um^3





        # geometry-related
        mid_ind = int(n_comp/2) #index of compartment roughly in middle of cable
        A_mem_comp = np.pi*d*l_comp*1e-8 #membrane surface area of compartment in
square cm
        A_cross_comp = np.pi*d*d*1e-8/4 #axon cross-sectional in square cm
```

```
	# capacitance related
	C_mem_comp = A_mem_comp*1e3 #membrane capacitace of individual compartment in
nF, assuming 1uF/cm2
	conductance_scaling_factor = 1e6*C_mem_comp/100 #factor used to scale
conductances because McKinnon et al model has 100pF capacitance

	# membrane conductances and reversals
	# main branch
	G_leak_abs = G_leak_abs_uniform #leak conductance in nS, based on McKinnon
Table 1, default is 1nS
	g_mem_leak_comp = conductance_scaling_factor*G_leak_abs/1e3 #membrane leak
conductance per compartment in uS
	E_leak = E_leak_uniform #leak reversal potential in mV, according to McKinnon,
default is -55mV
	# Na:
	G_Na_abs = G_Na_abs_uniform #Na conductance in nS, based on McKinnon Table 1,
default is 300nS
	g_mem_Na_comp = conductance_scaling_factor*G_Na_abs/1e3 #membrane Na
conductance per compartment in uS
	E_Na = E_Na_uniform #Na reversal potential in mV, according to McKinnon,
default is 60mV
	# Kd:
	G_Kd_abs = G_Kd_abs_uniform #Kd conductance in nS, based on McKinnon Table 1,
default is 2000nS
	g_mem_Kd_comp = conductance_scaling_factor*G_Kd_abs/1e3 #membrane Kd
conductance per compartment in uS
	E_K = E_K_uniform #K reversal potential in mV, according to McKinnon, default
is -90mV
	# GABA conductance for compartments proximal to (but not at) branch point
	G_GABA_abs = G_GABA_uniform #GABA conductance in nS
	g_mem_GABA = conductance_scaling_factor*G_GABA_abs/1e3 #GABA conductance per
compartment in uS
	E_GABA = E_GABA_uniform #GABA (i.e., Cl-) reversal potential in mV, see
Prescott paper fig. 6, vary from -65mV (control) to -50mV (SCI)



	G_A_abs = G_A_abs_uniform #default 50 nS
	g_mem_A_comp = conductance_scaling_factor*G_A_abs/1e3
	E_A = E_A_uniform#  default of -90.0 in mV
```

```
        G_H_abs = G_H_abs_uniform #default 1nS
        g_mem_H_comp = conductance_scaling_factor*G_H_abs/1e3
        E_H = E_H_uniform   #  default of -32.0 in mV


        G_M_abs = G_M_abs_uniform #default 50 nS
        g_mem_M_comp = conductance_scaling_factor*G_M_abs/1e3
        E_M = E_M_uniform#  default of -90.0 in mV

        G_CaL_abs = G_CaL_abs_uniform #default 1.2 nS
        g_mem_CaL_comp = conductance_scaling_factor*G_CaL_abs/1e3
        E_CaL = E_CaL_uniform# default of 120.0 in mV
        ###########

        G_KCa_abs = G_KCa_abs_uniform #default 50 nS
        g_mem_KCa_comp = conductance_scaling_factor*G_KCa_abs/1e3
        E_KCa = E_KCa_uniform# at default already in mV










        """
        ADD OtherChannel for both main and other two branches
        """

        # axial conductance related
        # main axon
        R_ax = 100.0 #axial resistivity in Ohm cm, from
https://www.frontiersin.org/articles/10.3389/fncel.2019.00413/full
        g_ax_comp = A_cross_comp*1e6/(R_ax*l_comp*1e-4) #axial conductance between
compartments in uS
        #g_ax_comp = 0.0 #uncouple compartments, for testing




        # compartmental voltage changes
        deltaV = [0.0]
        for i in range(1, n_comp, 1):
            deltaV.append(0.0)
        deltaV = np.asarray(deltaV)

        # initial values for compartmental voltages, gating variables, conductances,
and currents
        V_init = -65.0 #initialize all voltages
        mNa_init = 0.0 #initialize all Na channels to deactivated
        hNa_init = 1.0 #initialize all Na channels to deinactivated
        nKd_init = 0.0 #initialize all Kd channels to deactivated

        mA_init = 0.0
        hA_init = 1.0
        mH_init = 0.0
        mM_init = 0.0
        mCaL_init = 0.0
        hCaL_init = 1.0
        mKCa_init = 0.0
        CaS_init = 0.001




        gNa_init = g_mem_Na_comp * np.power(mNa_init, 2) * hNa_init
        gKd_init = g_mem_Kd_comp * np.power(nKd_init, 4)
        gleak_init = g_mem_leak_comp


        gA_init = g_mem_A_comp * np.power(mA_init, 3) * hA_init
        gH_init = g_mem_H_comp * mH_init
        gM_init = g_mem_M_comp * np.power(mM_init, 2)
        gCaL_init = g_mem_CaL_comp * mCaL_init * hCaL_init
        gKCa_init = g_mem_KCa_comp * mKCa_init


        INa_init = gNa_init * (V_init - E_Na) #initialize Na currents
        IKd_init = gKd_init * (V_init - E_K) #initialize Kd currents
        Ileak_init = gleak_init * (V_init - E_leak) #initialize leak currents

        IA_init = gA_init * (V_init - E_A)
        IH_init = gH_init * (V_init - E_H)
        IM_init = gM_init * (V_init - E_M)
```

```
       ICaL_init = gCaL_init * (V_init - E_CaL)
       IKCa_init = gKCa_init * (V_init - E_KCa)

       b1 = axBranch(L=200.0, d = b1d_ini + index1 * -0.03, l_comp =10.0 , R_ax =
100.0, V_init = -65.0, \
                   mNa_init = mNa_init, hNa_init = hNa_init, nKd_init = nKd_init,
mA_init = mA_init, hA_init = hA_init, mH_init = mH_init, \
                   mM_init = mM_init, mCaL_init = mCaL_init, hCaL_init = hCaL_init,
mKCa_init = mKCa_init, CaS_init = CaS_init, \
                   G_leak_abs_uniform = G_leak_abs_uniform, E_leak_uniform =
E_leak_uniform, G_Na_abs_uniform = G_Na_abs_uniform, \
                   E_Na_uniform = E_Na_uniform, G_Kd_abs_uniform = G_Kd_abs_uniform,
E_K_uniform = E_K_uniform, \
                   G_GABA_uniform = G_GABA_uniform, E_GABA_uniform = E_GABA_uniform,
\
                   G_A_abs_uniform = G_A_abs_uniform, E_A_uniform = E_A_uniform,
G_H_abs_uniform = G_H_abs_uniform,\
                   E_H_uniform = E_H_uniform, G_M_abs_uniform = G_M_abs_uniform,
E_M_uniform = E_M_uniform, \
                   G_CaL_abs_uniform = G_CaL_abs_uniform, E_CaL_uniform =
E_CaL_uniform, G_KCa_abs_uniform = G_KCa_abs_uniform, E_KCa_uniform = E_KCa_uniform)
       #Branch b2
       b2 = axBranch(L=200.0, d = b2d_ini + index2 * 0.03, l_comp =10.0 , R_ax =
100.0, V_init = -65.0, \
                   mNa_init = mNa_init, hNa_init = hNa_init, nKd_init =  nKd_init,
mA_init = mA_init, hA_init = hA_init, mH_init = mH_init, \
                   mM_init = mM_init, mCaL_init = mCaL_init, hCaL_init = hCaL_init,
mKCa_init = mKCa_init, CaS_init = CaS_init, \
                   G_leak_abs_uniform = G_leak_abs_uniform, E_leak_uniform =
E_leak_uniform, G_Na_abs_uniform = G_Na_abs_uniform, \
                   E_Na_uniform = E_Na_uniform, G_Kd_abs_uniform = G_Kd_abs_uniform,
E_K_uniform = E_K_uniform, \
                   G_GABA_uniform = G_GABA_uniform, E_GABA_uniform = E_GABA_uniform,
\
                   G_A_abs_uniform = G_A_abs_uniform, E_A_uniform = E_A_uniform,
G_H_abs_uniform = G_H_abs_uniform,\
                   E_H_uniform = E_H_uniform, G_M_abs_uniform = G_M_abs_uniform,
E_M_uniform = E_M_uniform, \
                   G_CaL_abs_uniform = G_CaL_abs_uniform, E_CaL_uniform =
E_CaL_uniform, G_KCa_abs_uniform = G_KCa_abs_uniform, E_KCa_uniform = E_KCa_uniform)

#A(0) -> C(n_comp -1), crosspoint = b1.n_comp // 2 connected to b2.n_comp // 2,
stimulate at compartment 0.
#---------
        #-
        #-
#--------
       b1.Cas_monitor_main = [CaS_init] #the same compartment as v_rec_middle
       b2.Cas_monitor_main = [CaS_init]
       b1.connIndex = b1.n_comp // 2
       b2.connIndex = b2.n_comp // 2

       b1.First = segmentSpikeTest( -30 )
       b1.Middle = segmentSpikeTest( -30 )
       b1.Last = segmentSpikeTest( -30 )
       b2.First  = segmentSpikeTest( -30 )
       b2.Middle  = segmentSpikeTest( -30 )
       b2.Last = segmentSpikeTest( -30 )

       # coupling between main axon and branches, see Dayan & Abbott page 219, fig
6.16
       g_middle_junction = G_gap_junction
       #2.0*b1.g_ax_comp*b2.g_ax_comp/(b1.g_ax_comp+b2.g_ax_comp)


       b1.ACArr = [([(0.0,0.0)],[(b1.g_ax_comp, b1.V[1])])]


       for i in range(1, b1.n_comp-1, 1): #initialize compartment integration
parameter arrays, for middle compartments
           if i == b1.connIndex:
               b1.ACArr.append(([(b1.g_ax_comp, b1.V[i-1])],
[(b1.g_ax_comp,b1.V[i+1]), (g_middle_junction, b2.V[b2.connIndex])]))

           else:
               b1.ACArr.append(([(b1.g_ax_comp, b1.V[i-1])],
[(b1.g_ax_comp,b1.V[i+1])]))


       b1.ACArr.append(([(b1.g_ax_comp, b1.V[b1.n_comp-2])], [(0.0,0.0)]))

       stimDict1 = {}
       stimDict1[0] = I_stim_now
       Integration.IntegParamInit(b1, b1.ACArr, dt, stimDict1)
```

```
        b1.V_rec_first = [b1.V_init] #recorded voltage time course in first compartment
in mV
        b1.V_rec_second = [b1.V_init] #recorded voltage time course in second
compartment in mV
        b1.V_rec_middle = [b1.V_init] #recorded voltage time course in middle
compartment in mV
        b1.V_rec_last = [b1.V_init] #recorded voltage time course in last compartment
in mV
        b1.V_rec_nexttoGABA = [V_init] #recorded voltage time course in compartment
next to GABA in mV







        b2.ACArr = [([(0.0,0.0)],[(b2.g_ax_comp, b2.V[1])])]


        for i in range(1, b2.n_comp-1, 1): #initialize compartment integration
parameter arrays, for middle compartments
            if i == b2.connIndex:
                b2.ACArr.append(([(b2.g_ax_comp, b2.V[i-1])],
[(b2.g_ax_comp,b2.V[i+1]), (g_middle_junction, b1.V[b1.connIndex])]))
            else:
                b2.ACArr.append(([(b2.g_ax_comp, b2.V[i-1])],
[(b2.g_ax_comp,b2.V[i+1])]))



        b2.ACArr.append(([(b2.g_ax_comp, b2.V[b2.n_comp-2])], [(0.0,0.0)]))

        Integration.IntegParamInit(b2, b2.ACArr, dt, stimDict = None)

        # recording electrodes (located in first and last compartment)# _p stands for
prime as in Dayan and Abbott appendix 6
        b2.V_rec_first = [b2.V_init] #recorded voltage time course in first compartment
in mV
        b2.V_rec_second = [b2.V_init] #recorded voltage time course in second
compartment in mV
        b2.V_rec_middle = [b2.V_init] #recorded voltage time course in middle
```

```
compartment in mV
       b2.V_rec_last = [b2.V_init] #recorded voltage time course in last compartment
in mV
       b2.V_rec_nexttoGABA = [V_init] #recorded voltage time course in compartment
next to GABA in mV

       # BEGIN SIMULATION
       #slotList = fireSlots(5, 40, 1/10, 0.6)         #original1 slotList =
fireSlots(5, 28, 1/5, 1)
       #used 2
       slotList = fireSlots(5, 15, 1/3, 0.6)
       while t_now+dt < t_total:
               t_now += dt
               t.append(t_now)

               stimulatedOrNot = False
               for pair in slotList:
                   s, e = pair
                   if(t_now >= s and t_now < e):
                       stimulatedOrNot = True
                       break
               if stimulatedOrNot:

              # if t_now>=t_stimstart and t_now<t_stimend:

                   I_stim_now=I_stim_amp #apply stimulus current
               else:
                   I_stim_now=0.0;

               # store previous time step values in _old arrays
               integCopy(b1)
               integCopy(b2)

               b1.ACArr = [([(0.0,0.0)],[(b1.g_ax_comp, b1.V[1])])]

               for i in range(1, b1.n_comp-1, 1): #initialize compartment integration
parameter arrays, for middle compartments
                   if i == b1.connIndex:
                       b1.ACArr.append(([(b1.g_ax_comp, b1.V[i-1])],
[(b1.g_ax_comp,b1.V[i+1]), (g_middle_junction, b2.V[b2.connIndex])]))
                   else:
                       b1.ACArr.append(([(b1.g_ax_comp, b1.V[i-1])],
[(b1.g_ax_comp,b1.V[i+1])]))



               b1.ACArr.append(([(b1.g_ax_comp, b1.V[b1.n_comp-2])], [(0.0,0.0)]))
               stimDict2 = {}
               stimDict2[0] = I_stim_now

               IntegParamUpdate(b1, b1.ACArr,dt, stimDict2)





               b2.ACArr = [([(0.0,0.0)],[(b2.g_ax_comp, b2.V[1])])]
               for i in range(1, b2.n_comp-1, 1): #initialize compartment integration
parameter arrays, for middle compartments
                   if i == b2.connIndex:
                       b2.ACArr.append(([(b2.g_ax_comp, b2.V[i-1])],
[(b2.g_ax_comp,b2.V[i+1]), (g_middle_junction, b1.V[b1.connIndex])]))
                   else:
                       b2.ACArr.append(([(b2.g_ax_comp, b2.V[i-1])],
[(b2.g_ax_comp,b2.V[i+1])]))
               b2.ACArr.append(([(b2.g_ax_comp, b2.V[b2.n_comp-2])], [(0.0,0.0)]))
               IntegParamUpdate(b2, b2.ACArr, dt, stimDict = None)


               ################



                # branch 1
               b1.deltaV[b1.n_comp-1] = b1.d_p[b1.n_comp-1]/(1-b1.b_p[b1.n_comp-1])
#equation 6.56 in D&A, update voltage for last compartment
               b1.V[b1.n_comp-1] = b1.V_old[b1.n_comp-1] + b1.deltaV[b1.n_comp-1]

               for i in range(b1.n_comp-1, 0, -1): # step through the middle
compartments backward to update voltages
                   b1.deltaV[i-1] = (b1.c[i-1]*b1.deltaV[i]+b1.d_p[i-1])/(1-b1.b_p[i-
1])
                   b1.V[i-1] = b1.V_old[i-1] + b1.deltaV[i-1]

                # branch 2
               b2.deltaV[b2.n_comp-1] = b2.d_p[b2.n_comp-1]/(1-b2.b_p[b2.n_comp-1])
#equation 6.56 in D&A, update voltage for last compartment
```

```
               b2.V[b2.n_comp-1] = b2.V_old[b2.n_comp-1] + b2.deltaV[b2.n_comp-1]

               for i in range(b2.n_comp-1, 0, -1): # step through the middle
compartments backward to update voltages
                   b2.deltaV[i-1] = (b2.c[i-1]*b2.deltaV[i]+b2.d_p[i-1])/(1-b2.b_p[i-
1])
                   b2.V[i-1] = b2.V_old[i-1] + b2.deltaV[i-1]
            ######################

               I_stim.append(I_stim_now) #record stimulus current
               b1.Cas_monitor_main.append(b1.CaS[b1.mid_ind])
               b2.Cas_monitor_main.append(b2.CaS[b2.mid_ind])




               b1.V_rec_first.append(b1.V[0]) #record voltage in first compartment
               b1.V_rec_middle.append(b1.V[b1.connIndex]) #record voltage in middle
compartment
               b1.V_rec_last.append(b1.V[b1.n_comp-1]) #record voltage in last
compartment

               b2.V_rec_first.append(b2.V[0]) #record voltage in first compartment
               b2.V_rec_middle.append(b2.V[b2.connIndex]) #record voltage in middle
compartment
               b2.V_rec_last.append(b2.V[b2.n_comp-1]) #record voltage in last
compartment


       #plotI_stim(t, I_stim, 'Main', t_total) #Comment out if you don't want to
output the graph

       """
       plotVfirst(t, b1.V_rec_first, 'b1', t_total) #Comment out if you don't want to
output the graph
       #plotVsecond(t, b1.V_rec_second, 'b1', t_total) #Comment out if you don't want
to output the graph
       #plotVnexttoGABA(t, b1.V_rec_nexttoGABA, 'Main', t_total) #Comment out if you
don't want to output the graph
       plotVmiddle(t, b1.V_rec_middle, 'b1', t_total) #Comment out if you don't want
to output the graph
       plotVlast(t, b1.V_rec_last, 'b1', t_total) #Comment out if you don't want to
```

```
output the graph

       plotVfirst(t, b2.V_rec_first, 'b2', t_total) #Comment out if you don't want to
output the graph

       #plotVsecond(t, b2.V_rec_second, 'b2', t_total) #Comment out if you don't want
to output the graph
       #plotVnexttoGABA(t, b2.V_rec_nexttoGABA, 'Main', t_total) #Comment out if you
don't want to output the graph
       plotVmiddle(t, b2.V_rec_middle, 'b2', t_total) #Comment out if you don't want
to output the graph
       plotVlast(t, b2.V_rec_last, 'b2', t_total) #Comment out if you don't want to
output the graph
       plotCaS(t, b1.Cas_monitor_main, 'b1_Middle', t_total)
       """
       plt.figure(figsize=(15,12))
       plotVgeneral(t, b1.V_rec_first, 'b1_V_first', t_total, 231) #Comment out if you
don't want to output the graph
       plotVgeneral(t, b1.V_rec_middle, 'b1_V_middle(connected)', t_total,232)
#Comment out if you don't want to output the graph
       plotVgeneral(t, b1.V_rec_last, 'b1_V_last', t_total, 233) #Comment out if you
don't want to output the graph
       plotVgeneral(t, b2.V_rec_first, 'b2_V_first', t_total, 234)
       plotVgeneral(t, b2.V_rec_middle, 'b2_V_middle(connected)', t_total, 235)
#Comment out if you don't want to output the graph
       plotVgeneral(t, b2.V_rec_last, 'b2_V_last', t_total, 236) #Comment out if you
don't want to output the graph
       plt.show()
```