EBME 359 final project 2003: Bidomain model of cardiac excitation

Niels F. Otani
Department of Biomedical Engineering
Case Western Reserve University, Cleveland, OH 44106

April 17, 2003

1 Introduction

For the final project in EBME 359 this year, you will be constructing a bidomain model of the heart, allowing you to see some of the basic electrical properties of the heart, including normal and abnormal action potential propagation, defibrillation, and electrocardiographic (ECG) measurement. Once the model is constructed, you will hopefully find that there are many fascinating topics within these three broad categories to investigate that can studied with relative ease. This guide will help you to organize the project, build the model, and then direct you towards some of the more interesting problems to which the model can be applied.

2 Logistics

Since this project is relatively more involved than other tasks you have tackled as part of this course, we have organized you into several groups. It will then be the responsibility of each group to produce a single working code, which will then be available to the various members of that group for their investigation of the various study topics.

Because the development of each of the simulation codes will be a group effort, the members of each group must very conscious of the importance of *teamwork* for the project to be a success. As is the case in the "real world" of industry, you and your group will probably have to deal with personality differences and differences in ability while working to develop and run the code, and prepare the report. It will be crucial for everyone to do their part, meaning that the various groups will need to help and motivate everyone in the group, so that all the tasks needed to complete the project get done. Don't forget to praise others in your group for their accomplishments. Use positive reinforcement whereever possible.

The goal of each group is to produce a report that will reflect all the accomplishments of the group and, more importantly, what members of the group have learned. Each member of the group must contribute to this report. Each section of the report will be headed by its title and its author's or authors' name(s). This will allow those of us doing the grading

to judge the contribution of each member of the group. As many as two people may appear as authors for a given section–we will then assume that each author did approximately half the work described in the section. If you find yourself paired with another group member for responsibility for a given section, you should probably try to do more topics to make up the difference. At the same time, we recognize that not all topics and specialty areas are equally difficult—for example, we believe the media specialty area (see below) is particularly difficult. Tackling a difficult specialty area or topic will be rewarded, as will doing a thorough job on any area or topic—especially when it results in new learning experiences. The report will also be graded as a whole, and, to some extent, will be compared to the reports of the other groups. This means that you should not help other groups, since doing so may lower your grade by comparison. You should also be a little careful with the security of your codes (needs to be said; sorry!). The reports will also be graded on an absolute scale, meaning that if the quality of reports is high overall, then the number of A's will also be correspondingly high.

This type of project has not been attempted in this course previously, so there are bound to be some midcourse corrections in our plans. However, here is how I envision the project will unfold.

(1) The first step will be to develop an organization for your group. To do this, the very first thing you should do is choose a CEO, a leader, for your group. For this position, you should choose someone who knows how to motivate people in a positive way to get things done. This person will also probably be the one who organizes and heads the meetings outside of the EBME 359 lab times, who will be generally responsible for the completion of the standard version of the code, and who will be responsible for the assembly and finalization of the report. Once the leader is chosen, s/he should then preside over the designation of the remainder of the members to the various specialties areas.

(2) Once each member of the group has been assigned a specialty, each member should complete the tasks described by the corresponding section in this guide. On completion of these tasks, each member will (a) have a working subroutine to contribute to her/his group's code, and (b) be knowledgeable about her/his specialty area. When all specialists have completed their tasks, the result should be a working, "standard" version of the code.

(3) The next step is to choose several of the suggested topics, described in this guide, for investigation. Of course, groups are free and in fact encouraged to study other ideas not covered here. You are also encouraged to go beyond what is described here, if you have new ideas and curiosities you wish to follow up on. Each of your topics, will, of course, require that modifications be made in portions of the code that were written by other members in your group. You may be able to make minor changes yourself. For more involved modifications, you should ask the corresponding specialist in your group to either help you or make the necessary modifications her/himself. Remember, you are a team, and will be graded on the overall quality of the report, in addition to your individual contributions!

(4) As you complete each of the tasks for your project, you should write, in as "final" a form as possible, a description of what you have done. This includes any tests you may have conducted while building your part of the code, results from the 'standard" version of the code, and hypotheses, results and conclusions from all the topics you researched. Include any plots which help you make your points. Please be careful and generous about giving credit to others in your group who helped you with the various modifications you required. If you

were helped by others in your group in interpreting your results, suggesting new computer experiments, etc., these contributions should be liberally acknowledged also. You should also keep all major versions of your code for inclusion in the appendix of the group report. Codes should be commented and should include the author's name(s). The CEO should also keep track of all organizing activities, describing what was accomplished at each, and also keep a brief log of the progression of the project through to completion. At some appropriate time near the end of the semester, your CEO should make sure that all these report fragments be assembled and finalized into the group's final report.

2.1 Form of the final report

Each group will submit one report that will describe all the work conducted by all the members of the group on the project. Important: keep a copy of your report, just in case!

The report should have three sections followed by appendices:

(1) *An introduction.* This section can be fairly brief (approximately one-half to one page). Describe the project as a whole, what was being modeled, how it was being modeled, and organization of the group doing the project. Highlight any activities unique to your group and/or any great results your group produced—especially activities and results that might set you apart and ahead of the other groups.

(2) *Construction of the standard version of the code. Note: each subsection within this section should include the subsections author(s) as part of the subsections' headings.* In this section, your group leader should write a subsection describing organizational aspects of putting together the standard version of the code, including any problems, and how they were overcome. Then each specialty area should write a subsection describing any difficulties in writing the standard version of the code for which they were responsible, the logic of how these problems were solved, and tests performed along the way to solve problems. Include application of EBME 309 material and EBME 359 lab material whereever possible! Each specialty area should also describe tests showing that their portion of the code works. The main code specialist should describe any difficulties encountered in assembling the contributions from the other specialists, how the difficulties were overcome, including any tests performed, and runs of the finished standard version of the code, showing that it works.

(3) *Project topics The title of each project topic description should include the name(s) of those principally involved in investigating the topic. Remember that, when more than N authors are listed, each will be assumed to have done 1/N-th of the work.* The author(s) of each topic should describe any problems encountered, and how those problems were tackled. Even if you were not successful in solving a problem, please describe your reasoning in trying to solve the problem. Many problems involved with these topics are very difficult—so the way you went about thinking about these problems (particularly in the context of EBME 309/359 material) is more important than their mere solution. Show and describe the key results and conclusions for your topic, using plots appropriate to demonstrating your claims.

(Appendices.) Include the standard version of your code, including all subroutines. Also, for each topic, include those portions of the code modified from the standard version critical for targeting the problem you were examining.

2.2 Grading of the project

EBME 359: The project grade will count as approximately 2 1/2 labs (33% of the course grade). The project grade will be composed of two parts (1) A group grade (1/3 of the project grade). This grade will be based on the overall quality of the entire group project. All students within a given group will receive the same group grade. (2) An individual grade (2/3 of the project grade). This grade will be based on the individual's contribution to both the standard version of the code and the project topics.

The following approximate standard for the individual grade will be used:

(1) The completion of a single topic rated as "Difficult" together with a good description of *why* what happens happens will be considered "A" work.

(2) The completion of a single topic rated as "Moderate" or "Medium" together with a good explanation of why what happens happens *plus* any sort of added insight or extra investigation beyond what was called for in this guide's description of the topic will be considered "A" work. One way to fit into the added insight/extra investigation catagory is to do an "Easy" topic in a way which answers some question you come across while doing the "Moderate" topic.

(3) The completion of two "Easy" topics, again with an adequate description will be considered "B" work. Significant added insight or extra investigation could raise this grade to an "A".

To calculate the group grade, we will first compare the group reports against one another and rank them in order (1st, 2nd, etc.). We will then calculate another grade for each report using absolute standard which is commensurate with the individual grade standard; that is, if all the individual grades making up a group's report were A's, then the absolute standard evaluation of the group grade would also be an "A". Qualitative consideration of both the rank and the absolute group grade will be used to determine the overall group grade.

EBME 309: The project will also be used to evaluate command of the material presented in EBME 309. Much of this understanding is reflected simply in a good working comprehension of how the code works, as demonstrated in your ability to solve problems to get the code to work, etc. I assume your knowledge of what the roles of the various equations are, understanding based on where these equations come from, etc., helps you to solve these problems. Simply getting many of these topics to work will, for me, be a good indicator of understanding of the EBME 309 material. However, it will probably help your grade if you mention explicitly these sorts of connections with EBME 309 material in your explanations of how you solved problems. Insightful descriptions of what you see happening in the simulations will also help the EBME 309 grade. Current projected weights for the EBME 309 final grade: Homework 40%, "Mid"-term exam 35%, Project grade, 25%.

3 Structure of the code

3.1 General computational considerations

As you have seen in the lectures, the equations for the bidomain model are,

$$\frac{\partial V_m}{\partial t} = -\frac{i_m}{c} + \nabla \cdot \mathbf{D_g} \cdot \nabla V_m + \nabla \cdot \mathbf{D_g} \cdot \nabla \Phi_e + \frac{i_{intracell}}{c} \tag{1}$$

and

$$\nabla \cdot (\mathbf{D_e} + \mathbf{D_g}) \cdot \nabla \Phi_e = -\nabla \cdot \mathbf{D_g} \cdot \nabla V_m - \frac{i_{intracell}}{c} - \frac{i_{extracell}}{c} \qquad \text{(System Equation)} \tag{2}$$

We will be solving these equations in *two dimensions* with spatial coordinates x and y.

For the ion channel model, we will be using a variant of the Fitzhugh-Nagumo equations; namely,

$$-\frac{i_m}{c} = \frac{1}{\epsilon_1}\left(V_m - \frac{V_m^3}{3} - W\right) \tag{3}$$

where the dynamics of W are given by,

$$\frac{\partial W}{\partial t} = \epsilon_2(V_m - \gamma W + \beta) \qquad \text{(System Equation)} \tag{4}$$

This form of the Fitzhugh-Nagumo equations differs slightly from those that you used in a previous lab in that there are two ϵ's, ϵ_1 and ϵ_2. One of the ϵ's, ϵ_1, controls how sharp the leading and trailing edges of the action potential are—the smaller ϵ_1 is, the more "vertical" the edges are. The sharper leading edges also makes the action potential travel faster, as you will see once the code is working. As you perhaps noticed from the quiz concerning these equations, the other ϵ, ϵ_2, controls the action potential duration (often abbreviated "APD"). The smaller ϵ_2 is, the longer the action potential lasts. Also, ϵ_2 controls recovery from refractoriness. The smaller ϵ_2 is, the longer it takes a cell to recover following the generation of an action potential.

Substituting Eq. (3) into Eq. (1) we have,

$$\frac{\partial V_m}{\partial t} = \frac{1}{\epsilon_1}\left(V_m - \frac{V_m^3}{3} - W\right) + \nabla \cdot \mathbf{D_g} \cdot \nabla (V_m + \Phi_e) + \frac{i_{intracell}}{c} \qquad \text{(System Equation)} \tag{5}$$

The three system equations are therefore Eqs. (2), (4) and (5). So how do we go about solving these? How should the code that solves these equations be structured? To answer this, the following computational considerations should be taken into account:

(1) Due to limited computing power, you will be solving this problem in two spatial dimensions rather than three. To solve the problem in two dimensions, you will set up a two-dimensional grid on which to define the variables. Notice that there are only three variables: V_m, W, and Φ_e. Thus, these variables will be defined in Matlab as

```
Vm2d = zeros(Nx,Ny);
W2d = zeros(Nx,Ny);
Phie2d = zeros(Nx,Ny);
```

where the parameters `Nx` and `Ny` define the number of nodes, or equivalently, gridpoints or simulation cells in the x and y directions, respectively. You can loosely think of each of the values of `Vm2d`, for example, `Vm2d(i,j)`, as being the membrane potential of the "cell" located at grid location (`i`,`j`) on the two-dimensional grid. Similarly, `W2d(i,j)` is the value of W in that cell, and `Phie2d(i,j)` is the extracellular potential at the corresponding position in the extracellular universe.

One way of envisioning this two-dimensional system with its two-dimensional grid of nodes is to think of it as actually representing a three-dimensional system with a three-dimensional grid. We then think of this 3-d grid as being composed of an infitine number of 2-d grids, lying one on top of the other, stacked up in the z direction. Let's assume that the spacing between any adjacent pair of these 2-d grids is Δz, while the grid spacings *within* each 2-d grid are Δx and Δy. Now here's the key: Suppose that exactly the same thing is happening on each of these 2-d grids; that is, suppose that the distribution of the values of all the variables (V_m, W and Φ_e in our case) on these grids are identical copies of one another at any time you care to look. Then there is no need to model all of these 2-d grids—we can just model one of them, and we know the rest of them will all be the same. In short, when we make a 2-d model of a 3-d siuation, we are assuming the same thing is happening in all planes sliced perpendicular to the z axis, and that our two-dimensional x-y simulation shows what's happening in any one of these slices. To make things even easier, in our case, we don't even have to worry about any effects from the other (identical) slices above or below the slice we are modeling. This is because the only way different slices (or more specifically, corresponding nodes on adjacent slices) can comunicate is through extracellular or gap junction resistors connecting the corresponding points on the different slices. But since the values of all variables are identical at corresponding points, it must be the case that both Φ_i and Φ_e are the same at the two points, so both the intracellular and extracellular potential drops are zero, so no currents flow between the planes, meaning that each plane operates as if none of the other planes is present. Thus we can think of our simlation as existing on a 1-by-Nx-Ny grid, with cells of size Δx by Δy by Δz. It will turn out that Δz will cancel out of all our expressions, and so can be set equal to 1 for all simulations.

(2) The operators $\nabla \cdot \mathbf{D_g} \cdot \nabla$ and $\nabla \cdot \mathbf{D_e} \cdot \nabla$ will be represented in the code by matrices. These are same type of matrices you saw in Problem Set #7, when you looked at resistive networks. The matrices may be thought of as defining the electrical coupling of the various nodes in the circuit. Since resistors also couple together the extracellular nodes to form a resistive medium, it should not be surprising that this type of matrix is involved in describing the extracellular medium, too. Also, since the $\nabla \cdot \mathbf{D_e} \cdot \nabla$ came from these extracellular resistors when we applied Kirchhoff's current law to the extracellular nodes (see the bidomain notes), it should not be surprising that numerical representation of this operator is one of these resistive network matrices. A similar argument holds for the intracellular (gap junction) resistors, which together form the intracellular resistive medium. These two matrices, which are called admittance matrices by the electronic circuit community, will consequently take the place of the two nasty operators, $\nabla \cdot \mathbf{D_g} \cdot \nabla$ and $\nabla \cdot \mathbf{D_e} \cdot \nabla$, in the equations. A careful examination of the equations shows that the operator $\nabla \cdot \mathbf{D_g} \cdot \nabla$ may be replaced by multiplication by $-1/(c\Delta x \Delta y \Delta z)\mathbf{Y}_g$, while $\nabla \cdot \mathbf{D_e} \cdot \nabla$ may be replaced by multiplication by $-1/(c\Delta x \Delta y \Delta z)\mathbf{Y}_e$, where $\mathbf{Y}_g$ and $\mathbf{Y}_e$ are the matrices constructed by adding several terms

of the form $1/R$ to its various elements. Defining,

$$\mathbf{G}_g = \frac{1}{c\Delta x\Delta y\Delta z}\mathbf{Y}_g \tag{6}$$

$$\mathbf{G}_e = \frac{1}{c\Delta x\Delta y\Delta z}\mathbf{Y}_e \tag{7}$$

as our intracellular and extracellular admittance matrices, we have that the following replacements can be made:

$$\nabla \cdot \mathbf{D_g} \cdot \nabla \rightarrow -\mathbf{G}_g \tag{8}$$

and

$$\nabla \cdot \mathbf{D_e} \cdot \nabla \rightarrow -\mathbf{G}_e \tag{9}$$

(3) Remember from Problem Set #7 that these matrices expect to see the nodal voltages as column vectors. This means that we can't just feed these matrices the `Nx`-by-`Ny` arrays, `Vm2d` or `Phie2d`, as defined above. Instead, we have to somehow change these arrays into column vectors. Fortunately, Matlab has a convenient method for converting two-dimensional arrays into column vectors, and back again. If `Vm2d` is a 2-d array and `Vm` is a column vector; that is, if `Vm2d` and `Vm` are defined as,

```
Vm2d = zeros(Nx,Ny);
Vm = zeros(Nx*Ny,1);
```

then you can transfer the array data stored in `Vm2d` to the column vector `Vm` by using the syntax,

```
Vm(:)  = Vm2d;
```

You can put the data in `Vm` back into the 2-d array `Vm2d` by saying,

```
Vm2d(:)  = Vm;
```

Both of these are special Matlab syntaxes, designed specifically for this purpose, since it turns out to be a common operation.

At this point, we have reduced our system equations to,

$$\frac{\partial V_m}{\partial t} = \frac{1}{\epsilon_1}\left(V_m - \frac{V_m^3}{3} - W\right) - \mathbf{G}_g \cdot (V_m + \Phi_e) + \frac{i_{intracell}}{c} \tag{10}$$

$$\frac{\partial W}{\partial t} = \epsilon_2(V_m - \gamma W + \beta) \tag{11}$$

$$(\mathbf{G}_e + \mathbf{G}_g) \cdot \Phi_e = -\mathbf{G}_g \cdot V_m + \frac{i_{intracell}}{c} + \frac{i_{extracell}}{c} \tag{12}$$

Although these equations look very much like the original systems equations, please note that now the variables V_m, W, and Φ_e are now all (`Nx`*`Ny`)-by-1 column vectors, as are the quantities $i_{extracell}$ and $i_{intracell}$.

(4) Since both Eqs. (10) and (11) contain time derivatives, it is simplest to use these two equations to calculate V_m and W at the next timestep, using the forward Euler method. Thus,

$$V_m^{n+1} = V_m^n + \Delta t \left[\frac{1}{\epsilon_1} \left(V_m^n - \frac{(V_m^n)^3}{3} - W^n \right) - \mathbf{G}_g \cdot (V_m^n + \Phi_e^n) + \frac{i_{intracell}^n}{c} \right] \tag{13}$$

$$W^{n+1} = W^n + \epsilon_2 \Delta t (V_m^n - \gamma W^n + \beta) \tag{14}$$

where the superscripts n and $n+1$ designate column vectors defined at the nth and $(n+1)$st timesteps, respectively, and Δt is the timestep size. We note that, in order to start off both Eqs. (13) and (14), initial conditions must be defined for the V_m and W everywhere in the system. Another important consideration is that the value of Φ_e^n must be defined before these two equations can be used.

(5) The remaining equation, Eq. (12), can be used to solve for this extracellular field, Φ_e^n. This equation, however, does not have a time derivative, so a different approach is needed. The simplest procedure would normally be to invert the matrix $\mathbf{G}_e + \mathbf{G}_g$ to obtain,

$$\Phi_e^n = (\mathbf{G}_e + \mathbf{G}_g)^{-1} \cdot \left[-\mathbf{G}_g \cdot V_m^n + \frac{i_{intracell}^n}{c} + \frac{i_{extracell}^n}{c} \right] \tag{15}$$

There is, however, a minor problem which makes the matrix $\mathbf{G}_e + \mathbf{G}_g$ singular and therefore non-invertible. This difficulty is easily dealt with by defining one of the nodes to be the ground node. The resulting matrix, which is one column and row smaller, is invertible, and then Eq. (15) can be solved.

3.2 Bidomain algorithm flow chart

With these considerations in mind, we can develop a flow chart, illustrating a reasonable structure for the code. This flow chart is shown in Figure 1. In this subsection, we describe the overall organization of the flow chart in broad terms, reserving details for the "specialty" sections to follow.

The code begins by defining parameters, such as N_x, N_y, Δt, etc. It then defines the big dynamical variable arrays that contain the all the values of V_m, W, and Φ_e for all the cells in the simulation. In fact, two versions of each these big arrays will be defined: a two-dimensional array version (`Vm2d`, etc.) and a column version (`Vm`, etc.). There may are also other auxiliary arrays you wish to define here. For example, I use a couple of arrays that contain the x and y values of each cell, which I have found useful for plotting.

Once the arrays correponding to V_m and W are defined, the next step is to fill them with them with their initial values. This is necessary for V_m and W, whose algorithms require starting values (see above). The remaining variable, Φ_e, does not need to be initialized in this way, since its values at a given timestep are computed directly from V_m at the same timestep.

The last task to complete before entering the timestep is the construction of the gigantic intracellular and extracellular admittance arrays. On my computer, these arrays take several seconds to compute. Fortunately, the values of the resistances do not change with time in any of the project topics that I envision. This means that the construction of these arrays

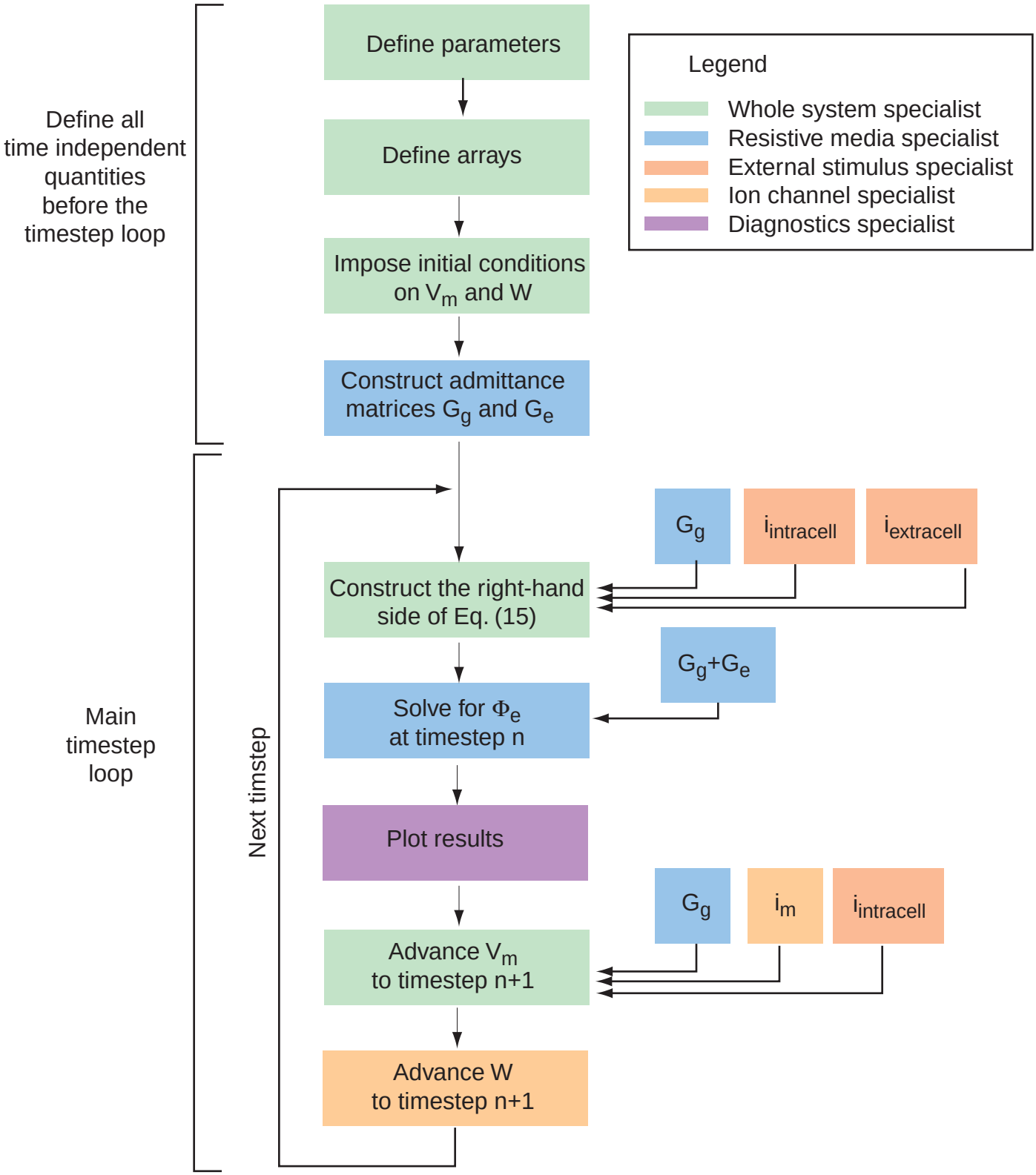

Figure 1: Suggested flow chart for the bidomain code.

need only be done once. Notice that it is very important to take advantage of the fact that these big arrays do not depend on time so should be computed *outside* the timestep loop.

Following these preparatory calculations, we enter the timestep loop for the first time, with parameters defined, the admittance matrices defined, V_m and W initialized, but Φ_e uninitialized. Thus, the first task is to calculate Φ_e from V_m and Φ_e. We do this by first computing the quantity in the brackets on the right-hand side of Eq. (15). This involves multiplying the V_m column matrix by the big admittance matrix $\mathbf{G}_g$, and then adding in the intracellular and extracellular stimulation currents, expressed as column vectors.

We can then find the extracellular potential by multiplying the (modified) inverse of $\mathbf{G}_e + \mathbf{G}_g$.

Now we have V_m, W and Φ_e all defined at the same time, at beginning of the first timestep. More generally, it will always be the case that all three of the variables will defined at the same time t at this point in the timestep loop. This is therefore a good time to make the plots. Plots will generally not be made on every timestep, so the plotting algorithm will contain code that causes it to swing into action only every so many timesteps.

The simultaneous definition of V_m, W and Φ_e at the same simulation time t is also what we need to advance V_m and W to the next timestep. This should be clear upon examination of Eqs. (13) and (14)—notice that everything on the right-hand side of these equations is defined at the same timestep n. The membrane potentials contained in the column vector V_m, for example, are advanced by first calculating the column vector defined by the expression in the square brackets in Eq. (13). One of the terms involved in obtaining this quantity requires us to add the two column vectors containing V_m and Φ_e, and then multiplying by the big matrix $\mathbf{G}_g$, a completely straightforward operation, since $\mathbf{G}_g$ was defined before the timestep loop was even entered. Calculation of the quantity in the square brackets also involves the addition of the column vector containing the present values of the intracellular stimulus current, if nonzero.

Once the square brackets quantity is defined, we can advance all the membrane potentials V_m to the next timestep by multiplying the square brackets column vector by Δt and adding it to the present value of V_m. The new values of V_m should be written on top of the old values in the same V_m column array, since there generally sufficient room to keep all the old values around.

We can the advance all the values in the W column vector in essentially the same way, using Eq. (14).

After W is updated, we are in the same situation as we were when we first entered the timestep loop: both V_m and W_m are defined at the same simulation time t, whereas Φ_e still needs to defined at the time t. Thus, this is the point at which we can return to the top of the timestep loop, whereupon the same procedure will occur again, this time one timestep further along in time.

3.3 Specialty tasks

The description of the flow chart in the previous section shows that there are a number of specialty tasks involved in producing the code. The corresponding specialty areas are color coded in Fig. 1. There are five code writing specialties:

(1) *The whole system specialist.* This person is in charge of the main program, which contains the definition of parameters, definition and initialization of the dynamical variables, and the main timestep loop. The main program, of course, is also the program that calls all the other routines, written by all the other specialists in the group. This means that the whole system specialist must work closely with the other specialists, to ensure that their routines are compatible with the main program.

(2) *The ion channel specialist.* The person in charge of the ion channel dynamics is responsible for the Fitzhugh-Nagumo term that appears in Eq. (13) and is also responsible for advancing the W equation in time. Clearly, the ion channel dynamics are intimately linked to the timestep loop, since the latter is require to make the excitability of the cells come alive. Thus, the ion channel specialist should probably work very closely with the whole system specialist. In a way, the ion channel specialist has a bit of a head start, since much of what this specialist will be doing is based on the Fitzhugh-Nagumo model lab. Among the complications that this specialist will have to handle beyond what appeared in the previous lab are: the presence of Nx*Ny cells instead of one, and the possibility that different cells in different regions may have different properties (that is, different values of β, γ, and the ϵ's).

(3) *The resistive media specialist.* This person has two primary duties: (a) to construct both the extracellular and intracellular admittance matrices for different situations, and (b) to solve Eq. (15) for Φ_e. While these two tasks are not lengthy when measured in terms of the number of lines of code required, they are among the most difficult in this project, so the person chosen as this specialist should be your best and most resourceful computer coder. (Reminder: the upside to this is that this specialist will be looked upon favorably by the graders, especially if the code works!) For the first task, the specialist will develop automatic algorithms that place all the resistors making up the network for either the extracellular or intracellular space into the corresponding admittance matrix. For the second task, the main difficulty has to do with how to incorporate the ground node into the calculation by removing the corresponding column and row. Short in terms of the number of lines of code, but long in terms of conceptual difficulty.

(4) *The external stimulus specialist.* This person is in charge of supplying $i_{extracell}$ and $i_{intracell}$, the extracellular and intracellular stimuli respectively. The main difficulties here are (a) going back and forth between column vectors and 2-d arrays, (b) applying currents so as to ensure the sum of all externally applied currents is zero, and (c) knowing how to apply currents at certain times during the simulation for specified time intervals. One neat thing this specialist could develop (although it is not required) is some way for the user to interact with the simulation as it runs, by for example applying stimuli using the mouse to define the where the stimulus is applied, when, for what duration, and with what amplitude, etc.

(5) *The diagnostics specialist.* This person is responsible for generating all the flashy color plots and movies for the group. A good person for this specialty is a person that likes to look around at the various fancy routines that a language makes available. Matlab has a lot of neat plotting methods. Only a couple of new ones are actually necessary for this project beyond what you have already seen, but certainly use of some Matlab's other plotting methods could really improve your ability to see what's going on.

Additionally, there is a sixth specialist: *the group leader* whose responsibilities were described in the previous section. This person also may find it worthwhile to work closely

with (or even be the same person as) the whole systems specialist. The reason for this is that many of the decisions made at the level of the main program, which is the whole systems specialist's primary responsibility, also reflects the administrative decisions regarding the group, and vice versa.

One potential problem is that the number of specialty areas is slightly less than the average number of people in each group. If one or two people really end up with less responsibility, they should try to make up for it with increased activity on the available topics. Alternatively, it may make sense for your group to put two people on some of the most difficult programming tasks, for example, the resistive media specialty, or the whole code specialty area. The latter may be potentially difficult because it requires the organization of all the other specialists to write their code so that it will work in the main code.

4 Development of the "standard" version of the code

Following the organizational process, the next step for each of the groups is to develop a "standard version" of the code. This standard code will model stimulation of the tissue in a small region, which results in outward propagation of an action potential in all directions. The code will display membrane potential, transmembrane current j_m, and the extracellular potential it generates, as colorplot movies.

Each of the specialists will develop code that will become part of the standard version. In the process of developing this code, each specialist will hopefully become something of an expert on her/his portion of the code, so that later, when other members of the group ask that new features be added to portions of the specialist's code, the specialist will be able custom design these features quickly and with a minimum of effort. The routine(s) that each specialist develops for the standard code will also serve as the beginning point for other related routines, which will soon evolve into a small toolbox that the specialist will be able to call on to facilitate the incorporation of new features that others in the group may want.

In the following subsections, I am providing a detailed guide for each of the specialists that I hope will be helpful in developing the subroutines necessary for creating the standard version of the code. Each specialty guide is designed so that, if followed, should result in all the code required to produce the standard version.

I should insert a disclaimor here. While these guides will result in all the necessary routines, that doesn't mean that the various routines will understand each other! Each group should attempt to lay down guidelines as to what the inputs and outputs of each routine will be, and what their formats will be.

4.1 The whole system specialist

The whole system specialist—it's the easiest of jobs; it's the hardest of jobs! Here's the easy part: your job is to create a code that does absolutely nothing! Here's the hard part: The structure of this do-nothing code must be such that, when the other specialists drop their routines and code fragments into your code, the result will be a working, standard version of the code. You are in charge of the main program for the entire project! Therefore, you have two responsibilities in creating the standard version of the code: (1) Create a skeleton

(that is, a do-nothing) version of the main program, which will accommodate the other specialists' routines, and (2) *Communicate* with everyone in your group, to make sure their routines will work in your code. As some of the other specialists design their routines, they may find that they cannot make them fit into the plan you might have initially created; you then must work with that person to modify the main program. These changes may affect others in your group, so you may have to work with them also. Since there is a strong organizational component to this specialty, it might make sense for the CEO to also take on this responsibility.

While others in your group are happily working along, you may find the best way to keep yourself busy is to work with the ion channel specialist. Much more than the others in the group, this specialist is closely tied with the structure of the main program, and in fact cannot accomplish her/his portion of the standard code without use of the main program. Thus, you may find it convenient to work together with ion channel specialist, perhaps even sharing responsibilities between the two of you a bit. Perhaps the ion channel person and the whole system specialist should even be the same person (?).

In Figure 2, I have provided for you one possibility for the skeleton code. You should work with others in your group to define those portions I have left blurry. You may also find that this skeleton does not precisely suit the temperments of those in your group. Feel free to modify the structure of this code as you see fit. A couple of recommendations, however: (1) I strongly suggest you use subroutines, if not in the form I've suggested, then in some form. The use of code fragments when several developers are involved can lead to a main program that no one can understand or debug. You also run the risk that two people will use the same variable names for different purposes, overwriting them for their own purposes, thereby breaking the other's portion of the code, etc. One exception to this is the ion channel portion of the code, which is so intimately tied to the main program that subroutines may not be the best. (2) Make sure you think about the computational speed of the program when making decisions. Laura and I can help you with this. Generally, you should try to do things with as few loops as possible, especially loops over the grid indicies.

One tactic I believe you will find useful is to create dummy subroutines while you are waiting for the "real" routines to be written. For example, while waiting for the `solve_circuit` routine from your resistive media specialist, just create a routine like this:

```
function Phi_e = solve_circuit (Gg+Ge,right_hand_side,—parameters—);
Phi_e = 0;
```

One of the fun things about being the person in charge of this, is that you get watch your program spring to life as others in your group contribute their routines.

4.2 The ion channel specialist

As the ion channel specialist, your job is to bring Fitzhugh-Nagumo excitable dynamics to the project. For your portion of the standard version of the code, you should create a program that implements Fitzhugh-Nagumo dynamics in every cell, that is, on every gridpoint of the Nx-by-Ny computational grid. Since a model of Fitzhugh-Nagumo dynamics requires a timestep loop, and since that loop in this project resides in the main program, you will

```
% Define parameters:
Nx = 50; Ny = 40;
—Other parameters—
—Other specialists may also want to define some parameters here—

%Define and initialize main arrays:
—In this section, you should define Nx-by-Ny versions of Vm, W and Φe—
—It may also be convenient to initialize your Vm and W arrays at the same time you
initialize them—
—Then you should create the column arrays for Vm, W, and Φe and transfer the data in
the Nx-by-Ny arrays into them.—

% Construct admittance matrices:
Gg = construct_resistive_medium (—parameters—);
Ge = construct_resistive_medium (—different parameters—);

% Main timestep loop:

n = 0;

while (1)

    % Calculate stimuli
    i_intracell = stimulus_routine(–parameters—);
    i_extracell = ...;

    % Calculate right-hand side of Phi_e equation:
    right_hand_side = - Gg*Vm + i_intracell/c + i_extracell/c;

    % Solve for Phi_e:
    Phi_e = solve_circuit (Gg+Ge,right_hand_side,—parameters—);

    % Make plots:
    n
    makeplots (—arrays and parameters—);

    % Advance Vm to timestep n+1:
    Vm = Vm + Dt*(—stuff—);

    % Advance W to timestep n+1:
    W = W + Dt*(—more stuff—);

    n = n + 1;

end
```

Figure 2: Possible skeleton for the project main program.

have to work closely with the whole system specialist of your group to develop your code. Your best course of action might be to modify the main program directly, rather than create subroutines to drop into the main program.

You should probably develop the code in two phases. The first phase is a testing phase. For this purpose, you should use just a small number of cells by defining small values like 2 or 3 for `Nx` and `Ny`.

The specific details of what you need to do are perhaps most easily described by referring to the sample skeleton code shown in Fig. 2. In the parameters section, you should add the parameters required for Fitzhugh-Nagumo dynamics. For the standard version of the code, you can use the same values that were used in the Fitzhugh-Nagumo lab; namely, $\epsilon_1 = \epsilon_2 = 0.2$, $\beta = 0.7$ and $\gamma = 0.8$. Use a timestep $\Delta t = 0.01$.

For the initial conditions, you should work with the whole systems specialist to initialize some of the cells with resting state values—say $V_m = -1.2$ and $W = -0.62$, while the remainder should be initialized above threshold so that the cells fire, say $V_m = +1.0$ and $W = -0.62$.

You can set quantities coming from other specialty areas to zero (or use dummy versions of the corresponding functions to set the quantities to zero). For example, you can set $\mathbf{G}_g$ and $\mathbf{G}_e$ to zero, because you are trying to create model containing cells which displaying Fitzhugh-Nagumo dynamics completely independently of each other. Setting the two admittance matrices to zero implies that the resistances in the networks they represent are infinite, which effectively decouples the cells from one another. Similarly, you can set the intracellular and extracellular stimuli to zero. Cell firing is being produced by the initial conditions in this case.

The key to producing excitable dynamics, of course, is defining the V_m and W advances to the next timestep in the skeleton code in Fig. 2. You can do this simply by dropping the right-hand sides of Eqs. (3) and (4) into the appropriate spots in the skeleton code.

To check to see that your code is working, you should temporarily incorporate some simple plotting capability in the code. Before entering the main timestep loop, have the code define two history arrays (that is, arrays that will contain the values of two specified variables as functions of time). Once inside the loop, have the code write the values of V_m and W at single gridpoint of your choosing into the next element of the two history arrays. Immediately after the timestep loop but before the simulation ends, have your code plot these two arrays. (Note: the loop in Fig. 2 runs forever. Make temporary modifications that will stop the loop after a number timestep equal to the number of elements in your history arrays.) If you had chosen one of the gridpoints initiated above threshold to the gridpoint to record in the history arrays, you should see classic action potenial behavior in $V_m(t)$, while W should exhibit the rise of refractoriness during the action potential and then a fall back to the resting state value once the action potential is over. In other words, you should see the same traces as you saw in the Fitzhugh Nagumo lab. On the other hand, if you chose a cell at rest, nothing much should happen.

Once you feel confident your code is working, increase the size of your computational grid back to `Nx`=50 and `Ny`=40. Initialize the whole system at rest, with the exception of a group of cells close to the center of the left edge of the simulation. Start those cells above threshold. If a plotting algorithm is available from the diagnostics specialist, drop it into your code so that you can see how the membrane potentials of the various cells behave as a color movie. If

plotting routines yet exist, you can conduct this test by lifting the colorplot routine together with its support routines directly out of the phase resetting lab. You should see the firing of the cells you initiated above threshold, while the rest of the cells should remain in the resting state throughout the simulation.

If your code passes this test, you should then modify your code so that all the cells start at rest. Of course they should stay at rest. Pretty boring, except that this is the version of the code that forms the basis for the standard version.

Notes on what code-related items to look at once the standard version is working: (1) You should think about what would modifications would be necessary to your portion of the code if the cell properties (and therefore your parameters (e.g., ϵ_1, β, etc.) vary from one region in the simulation plane to another (This is termed *heterogeneity.*). This ability will be important for least some of the topics to be described later. (2) Since you are the ion channels expert, you may be asked by the other group members how parameters should be changed to produce desired changes in the morphology and functionality of the action potential. For this purpose, you may wish to reexamine the results you obtained from the Fizhugh-Nagumo lab, as well as run additional simulations to further familiarize yourself with how the various parameters affect the action potential. If you are ambitious, you could design a simulation in which, for example, the value of ϵ_1 varies from one end of the mycardium to the other. Since all the cells act indendently of each other, this would help clarify what the trend is as the parameter is varied.

4.3 The resistive media specialist

The resistive media specialist may have the most difficult code-writing job of all the members of the group. Fortunately, everything that is required can be boiled down into the construction of two subroutines, one to build admittance matrices, and one to solve the circuit equations. Once these routines have been tested, you will also need to work with your whole systems specialist to incorporate your routines into the main program, and define parameters for your routine.

While building your two routines, you should keep the following in mind:

(1) The admittance matrices you will be creating are huge and so would not fit into memory if defined in the conventional way (e.g., by saying something like `Gg = zeros(Nx*Ny,Nx*Ny);`). Fortunately, nearly all of the elements of these admittance matrices are zero, making them what is generally known as *sparse* matrices. Matlab is well-equipped to handle sparse matrices. By using the following method for defining matrices:

```
Gg = sparse(Nx*Ny,Nx*Ny);
```

instead of using the `zeros` or `ones` commands, Matlab will store the matrix by just keeping track of the row numbers, column numbers, and values of the non-zero elements, thereby greatly reducing the memory requirements. Therefore, please use the *sparse* command to define your admittance matrices. Further, in manipulating these matrices, make sure you only use commands that will allow Matlab to maintain these matrices as sparse. Many of the usual commands, like multiplying by constants, adding two sparse matrices, etc., will result in sparse matrices, so these commands are ok. However, the inverse of a sparse matrix is not usually sparse, so if you try to explicitly define a new matrix which is equal to a sparse

matrix, you may be in trouble. On the other hand, it is perfectly ok to do something like this:

```
b = M\x;
```

which is Matlab's way of expressing multiplying the inverse of the matrix **M** onto the column vector **x** to obtain the column vector **b**. If fact, if **M** is defined as a sparse matrix, Matlab will perform the operation much more quickly by not wasting time computing terms which involve multiplying by zero-valued matrix elements.

(2) Computing time is also a concern. You will only be building each admittance matrix once, so it's ok if that operation takes a little time, as long as it's not 10 minutes. (Even 30 seconds is probably too long, actually.) I've found that there some methods of building the matrix are faster than others, so you may want to play around a little, if you find your matrix building algorithm is too slow. The method I suggest below can build an admittance matrix for a 40x50 grid in about 5 seconds on my three-year old laptop computer.

On the other hand, your circuit solving routine will be called once per timestep, so it has to be as fast as possible. It definitely should not contain any loops. All operations should be vector or matrix operations, which work on the entire vector or matrix in one step. Even so, this routine is likely to be most time consuming part of the whole code.

Here are my recommendations on how to build these routines. These are only suggestions; you may design your routines as you see fit, as long as you keep the above considerations in mind. . ..

4.3.1 Routine to build the admittance matrix

Each resistor you will be adding to the admittance matrix is characterized by five numbers: the x and y grid indices of the one of the nodes to which the resistor is connected, the x and y indices of the other node, and the value of the resistor. On my computer, I obtained the best results by defining five vector arrays, one for each of these quantities. Each vector was defined to have a length equal to the total number of resistors I planned to add to the admittance matrix I was building. For both intracellular and extracellular media, the network of resistors was arranged in a rectangular lattice. I therefore wrote two loops: one to add the resistors that were connected from left to right in the lattice, and one to connect resistors running from top to bottom. Actually, each of these loops was a double loop. For the horizontal resistors, for example, the outer loop was over rows, and the inner loop was over resistors in that row.

After filling these five arrays, I next defined two additional vector arrays, each equal to the number of resistors. In one of these two arrays, I computed the index in the 1-d column vector associated with one of the nodes to which each resistor was connected. In the other array I computed the 1-d column vector index associated with the other node the resistors were connected to. In other words, two of the five vector arrays you have just defined contain the x and y indices of one of the nodes each of the resistors is connected to. These two indices must be converted into a single index for use with the column vectors that we are employing in this project. It's this index, one for each resistor, that should be stored in the first of these two vector arrays. The indices for the other node each resistor is connected to should be stored in the other array.

I'm doing a lot of talking here, but in fact this relationship between the two types of

array indices can be define in two lines of Matlab code. The tough part is figuring out a formula that relates this single index to the x and y nodal indices. To figure out what this relationship is, you should do a simple Matlab experiment. Define a small two-dimensional array, Nx by Ny in size, where Nx and Ny are small numbers, like 2 or 3. Then fill this array with numbers that are all different; for example,

$$\mathtt{M} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \tag{16}$$

Then define another array, `column_M` as an (Nx*Ny) by 1 column vector. Then use our trick to load the data in `M` into `column_M`:

```
column_M(:)  = M
```

Take note of where each of the elements of `M` end up inside of `column_M`. Use this figure out a rule for how Matlab stores two-dimensional data in a column vector, and then use that to come up with a general rule relating the two indices defining were a given element is stored in a 2-d array to the index defining where the same element is stored in the 1-d column vector. You should then be able to figure out what the two (vectorized!) lines of Matlab code should be.

These two new vector arrays now define the two rows and two columns in the admittance matrix that are relevant to adding the corresponding resistor to the matrix. (Remember from a previous problem that, in order to add a resistor the admittance matrix, you add terms that look like $\pm 1/R$, where R is the resistor value, to each of four elements in the matrix, arranged in two rows and two columns.) You can make a loop over the all the resistors, adding one resistor to the matrix per time through the loop.

I strongly suggest that you check your admittance matrix subroutine at this point using a small grid (e.g. $Nx = 2$, $Ny = 3$) and simple resistor values (perhaps $R = 1, 1/2, 1/3, \ldots$). Connect the resistors to the nodes any way you want, by loading them "by hand" in the five vector arrays. Then examine the matrix to see if the values ended up in the right places in the matrix. If you are using sparse matrices at this state, you may find it convenient to use the `full` command, (e.g., `full(Gg)`) to see the whole matrix in standard form.

Once you are confident your admittance matrix working, have your admittance matrix building routine connect up an Nx-by-Ny array in a rectangular array of resistors, with resistors running in the x direction all having value $\eta_x \Delta x/(\Delta y \Delta z)$, and resistors running in the y direction all having value $\eta_y \Delta y/(\Delta x \Delta z)$. This routine will generally be called twice by the main program, once for the extracellular resistive medium, characterized by $\eta_x = \eta_{ex}$ and $\eta_y = \eta_{ey}$, and once for the intracellular resistive medium, with generally different values, η_{ix} and η_{iy}, for η_x and η_y. Again, you should check out this version of the code using small values for Nx and Ny.

So that the admittance matrices defined by this routine match those used in our equations, they should be divided by $c\Delta x \Delta y \Delta z$ (see Eqs. (6–9)).

This is the version of the admittance matrix building routine that will be used in the standard version of the project code. It represents a homogeneous but anisotropic resistive medium (i.e., the resistor pattern looks the same everywhere (*homogeneous*), but the resistances are different for current flow in the two different directions x and y (*anisotropic*).

4.3.2 Building the circuit solving routine

The other routine you must devise calculates the nodal voltages for a circuit composed of a network of resistors defined by one your admittance matrices which is powered by one or more current sources.

This may sound complicated, but mathematically, it's very simple. All we are doing is solving the equation,

$$\mathbf{G} \cdot \mathbf{\Phi} = \mathbf{j} \tag{17}$$

for Φ. This is exactly the same equation you derived in Problem 2(b) of Problem Set #7. The matrix $\mathbf{G}$ is the admittance matrix, $\mathbf{\Phi}$ is a column vector containing the nodal voltages, and $\mathbf{j}$ is a column vector containing the injected currents. Furthermore, for the purposes of creating this subroutine, you get to assume that $\mathbf{G}$ and $\mathbf{j}$ are already known! Thus, if it weren't for a small problem, this circuit solving routine would be this simple:

```
function Phi = solve_circuit (G,j);
Phi = G\j;
```

In fact, your first test should be to just try this routine, just as written, using one of your homogeneous, anisotropic admittance matrices $\mathbf{G}$ from your previous routine. To make things simple, use $c = \Delta x = \Delta y = \Delta z = 1$ in defining your admittance matrix. Then your η's will just be the resistor values. For the column vector `j`, set one of the elements equal to some positive current value, and another element equal to minus the same current value. As you saw in Problem 2(c), P.S.#7, this corresponds to a current source connected between the corresponding two nodes. Do this for a very small circuit (again, maybe $Nx = 2$, $Ny = 3$). Check to see whether the nodal voltages you obtain in the Φ vector are correct for the circuit you have created. If you are clever about how you define your current sources, you can probably come up with circuits that you deduce the voltages for just by looking at the circuit. See if you can come up with one of those.

Well, depending on exactly what values and circuit configuration you use, Matlab may send you an warning message when you try to do this. (Or it may not!) The problem is that the matrix $\mathbf{G}$ turns out to be singular, which causes two complications (which are actually two manifestations of the same problem). One complication is that, $\mathbf{G}$ being singular means that our matrix equation actually has an infinite number of solutions. Matlab is often clever enough to figure out a way to give you one of these solutions, but of course you don't know which one of these solutions Matlab is choosing to give you. The other related difficulty is that $\mathbf{G}$ being singular means that $\mathbf{G}$ does not have an inverse, which means that, technically our equations, and in particular, Eq. (15), are incorrect.

The singularity is coming from a relatively simple problem—that is, if a given set of nodal voltages is a solution to the problem, then so is that same set of nodal voltages plus any constant. The reason for this ambiguity, of course, is that, for any circuit, it is only the *difference* of voltages between any pair of nodes that is important.

To remove the ambiguity, we simply declare one of the nodes to be electrical ground, defining its nodal voltage to be zero. All the other nodal voltages then become well defined. Now let's see how declaring a given node as electrical ground is implemented in the equations. If, say node 3 is defined to be ground, then the column vector representing the potential

must look like,

$$\mathbf{\Phi} = \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ 0 \\ \Phi_4 \\ \vdots \\ \Phi_{Nx*Ny} \end{bmatrix} \tag{18}$$

If we then write out Eq. (17) as the $Nx * Ny$ equations it represents, we see that none of the elements of the third column of the matrix $\mathbf{G}$ matter, because they are always multiplying the third element in $\mathbf{\Phi}$, which is zero. Therefore, if we eliminate the third column from the matrix $\mathbf{G}$ and eliminate the third element, 0, from $\mathbf{\Phi}$, we obtain exactly the same equations. Now, the other thing to notice is that, since the original matrix $\mathbf{G}$ was singular, it means that not all the component equations are independent—one of the equations can be expressed as a linear combination of the others. This means we can eliminate one equation without losing any information. The natural equation to eliminate is the third one, which corresponds to removing the third row from the matrix $\mathbf{G}$ and the third element from the right-hand side current vector $\mathbf{j}$. The new matrix equation, with the third row and column missing from $\mathbf{G}$ and the third element missing from both $\mathbf{\Phi}$ and $\mathbf{j}$ is then the equation we want. You can do these operations quite easily with Matlab. For example, a matrix `M_reduced` defined in the followng way:

```
M_reduced = M([1:2,4:(Nx*Ny)],[1:2,4:(Nx*Ny)]);
```

is just the matrix `M` with the third row and column missing.

Once you solve for $\mathbf{\Phi}$ with the third element missing, you must explicitly put 0 back in the third slot in the vector; otherwise other parts of the code will complain that $\mathbf{\Phi}$ only has $Nx * Ny - 1$ elements in it. Once you do this, you're done—the routine does what it's supposed to.

You should make these modifications, and then try it out on the same simple circuit as before. The correct nodal voltages should be returned by the routine, with the nodal voltage of the ground node you specified equal to zero.

4.3.3 Plugging your routines into the main program

Referring to Fig. 2, your routines will be used to define `Gg` and `Ge` prior to the beginning of the timestep loop, and will be used to solve for the extracellular field `Phi_e` inside the loop. For the standard version of the code, you should use the following parameters: $Nx = 50$, $Ny = 40$, $\eta_{ex} = 1.0$, $\eta_{ey} = 3.0$, $\eta_{ix} = 1.0$, $\eta_{iy} = 3.0$, $c = 1.0$, $\Delta x = 0.25$, $\Delta y = 0.25$, and $\Delta z = 1.0$.

4.4 The external stimulus specialist

As the external stimulus specialist, you are reponsible for creating the capability of delivering both intracellular and extracellular stimuli to the simulation system. For your contribution to the standard version of the code, you will develop a routine which will inject a current intracellularly in a rectangular area located roughly in the center of the simulation system

over a short time period at the beginning of the simulation. You will also be extracting the same amount of current from the same rectangular area from the extracellular space.

To apply a stimulus in a rectangular region for given interval of time, do the following. In your function, set up an `if-else-end` structure. Check to see whether either the timestep or the time (your choice) is within the time range in which the stimulus is to be applied. If it isn't just return 0 as your stimulus. If it is, define a two-dimensional array of dimension Nx by Ny with all the elements equal to zero. Then use the following Matlab construct:

```
M(ix1:ix2,iy1:iy2) = i0;
```

This will set all the elements of the matrix `M` whose x index falls between `ix1` and `ix2` and whose y index falls between `iy1` and `iy2` equal to `i0`—a rectangular region within the matrix. Finally convert this two-dimensional array into a (Nx*Ny)-by-1 column vector using the method described in the previous section. Return this column vector as the value of your function.

You can test your function by creating simple program that calls your function, changes the column vector it returns back into a two-dimensional vector, and then makes a colorplot of the result.

There is one constraint you must be careful of in all the stimuli you define. That is, that the total of all the intracellular and extracellular currents you deliver to the simulation must always sum to zero at all times. The reason for this has simply to do with a basic fact about all circuits: the total current flowing into a circuit has to equal the total return current flowing out. If you violate this constraint, the system will behave as if you had attached a point electrode to the ground node of the system. The location of this ground node is chosen by the media specialist. If the sum of all the currents you specify does not sum to zero, the residual current will return through this point electrode, and in so doing, will set up extracellular and instracellular fields near this electrode—generally not what you want.

It might be a good idea to write your code so that this constraint is always checked for or is somehow always automatically satisfied (although, to be honest, I don't take either of these precautions in my code). If you don't implement either of these procedures, then you must watch your output–look for unexpected fields around the ground node, which is the telltale sign of this problem.

To incorporate your routine into the main program, define whatever parameters your routine needs in the parameter section (see the skeleton code in Fig. 2) and then set `i_intracell`, the intracellular current column vector, equal to the output from your routine. Stimulate a region approximately 10 cells by 10 cells in size, approximately in the middle of the simulation, for ten timesteps, with current amplitude 10.0. During the same interval of time, extract exactly the same current from the same rectangular region from the extracellular space using the variable `i_extracell`.

This routine will serve as the starting point for the development of other stimulus routines that your group may need. Here are some of the stimulus patterns that may be necessary: current injected into a disk-shaped region, stimulus applied on one or more edges of the simulation region, extraction of current from the entire outside edge of the simulation region, and stimuli applied at regular intervals in a given region.

A really fun thing you can do as the stimulus specialist is to add functionality to the code that will allow users of your group to apply stimuli to the tissue interactively, as the simulation runs, when the user wants and where the user wants. You already know how

to do part of this—you know you can use Matlab's `ginput` routine to make the mouse an active player. The `ginput` function will allow the user to define the portion of the system over which s/he wants to apply the stimulus. The problem with the `ginput` function, if it's used by itself, is that the simulation will stop and wait for mouse input everytime `ginput` is encountered in the program. Clearly, you want the program to run without stopping unless you signal for it to stop. One way of doing this with Matlab is to use buttons. To see how to do this, type "`help uitools`". I personally use the `uicontrol` function to create buttons, but there are other possibilities. Calling `uicontrol` allows you to create a button in your figure. You can arrange it so that, if the button is pressed with your mouse, it changes the value of a variable. You can then check the value of that variable each timestep, to see whether the button has been pressed. If it has, your program can then call `ginput` and `input` to obtain the region and amplitude of the stimulus the user is requesting.

4.5 The diagnostics specialist

Your role as the diagnostics specialist is to create plotting and movie routines for your group, so that you and other members of your group can see the results of your simulations. The functions you create will take as inputs the column vectors generated by the code, such as V_m, W and Φ_e and produce various types of plots at regular time intervals which can be varied.

Your contribution to the standard version of the code is a routine that plots V_m, W and one other, unspecified variable as colorplots every so many timesteps. Your function should therefore take as arguments three column vectors, `Vm`, `W`, and a third column vector, along with any parameters it needs to know to produce the plots.

Since your routine only does something on plotting timesteps, it should look like one big if statement—if it isn't time to plot, do nothing. If it is, then do the following:

First create one or more two-dimensional arrays, of size Nx by Ny. Next transfer the data stored in the V_m, W and the third variable, assumed to be column arrays, into these two dimensional array(s). Finally make colorplots of these three two dimensional arrays. You can use the same `pcolor` "companion" functions that were used in the phase resetting lab, except that you will probably want to use a colormap that has contrasting colors at the ends of the colorscale—`colormap(jet)` for example, will do the trick. For the standard version of the code, I would like you to plot what I have been calling $\mathbf{j}_m/c = \partial V_m/\partial t + i_m/c$ as the third variable. As I have been describing in class, this quantity is the principal source for extracellular fields. The problem with plotting this quantity is that $\partial V_m/\partial t$ is not readily available. However, from Eq. (10), we see that $-\mathbf{G}_g \cdot (V_m + \Phi_e) + i_{intracell}/c$ is equal to $\partial V_m/\partial t + i_m/c$ and so can be used as a substitute for $\mathbf{j}_m/c$. Thus, the standard version of your graphics code should make colorplots of V_m, Φ_e, and $-\mathbf{G}_g \cdot (V_m + \Phi_e) + i_{intracell}/c$, each vs. x and y.

A couple of complications: First, you would like the x and y axes to be labeled in terms of the actual values of x and y, instead of simply the two-dimensional matrix indices. To do this, use the three argument version of the `pcolor` function. Type `help pcolor` inside Matlab to see how to do this. Second, in your colorplots, how do you know which direction corresponds to the x-axis, and which corresponds to the y axis? Also, as you move upward on the vertical axis in the `pcolor` plot, does y increase or decrease? You should think of

simple "computer" experiments to figure out how `pcolor` actually plots, and then use your Matlab prowess to figure out a solution, if `pcolor` doesn't plot x on the horizontal axis and y on the vertical axis, with y increasing from the bottom up.

Looking down the road a bit, you or your group may want the following types of plots:

(1) Plots of the variables associated with individual cells vs. *time*. These would be useful, for example, to compare to single cell behavior, or for comparing different cells in the same simulation.

(2) Contour plots of one variable superimposed on colorplots for another variable. Such plots would be useful, for example, in diagnosing where regions of inward and outward currents ($\mathbf{j}_m$) are relative to the action potential.

(3) Plots of dynamical variables vs. x for fixed value of y. These plots, which exhibit the variable in a given "slice" of the system, are sometimes more revealing, particularly when shown as a movie in time, than the corresponding colorplot movies.

(4) Plots of the direction of the electric field. You can use the Matlab functions `gradient` and `quiver` to plot electric field vectors at each point. As I've mentioned in class, there is a close connection between the charge-electric field relationship in electrostatics and the transmembrane current -electric field relationship here. We can apply whatever intuition we have about electric fields and charges to our problem if we have the electric field to look at. These plots would be particularly useful if superimposed on colorplots of $\mathbf{j}_m$.

(5) Fun plots. Matlab can also make 3-d surface plots using almost the same syntax as `pcolor`. Just for fun, or maybe you'll actually be able to see something new! (See the `surf` routine.)

4.6 Putting it all together

When the ion channel specialist, resistive media specialist, external stimulus specialist, and diagnostics specialist add their functions and/or code fragments to the skeleton main program written by the whole system specialist, the result should be a working program. The stimulus routine supplied by the stimulus specialist has been designed to raise a group of cells near the center of the system above the voltage threshold required for firing. This group of cells fires, since they, like all the cells in the system, have been equipped with Fitzhugh-Nagumo excitable dynamics provided the ion channel specialist. Once the these cells begin to fire, cells near the edge of this region will start to supply current to their outside neighbors through the gap junctions provided by the resistive media specialist. There should be enough current for these cells to fire as a result. This process will continue outward, creating a propagating action potential wave traveling outward in all directions. This presence of this traveling action potential should be abundently clear courtesy of the diagnostics specialist.

There are a few simple tests you can try and few interesting things to observe with this standard version of the code. First, the resistive media specialist has supplied extracellular and intracellular media which are anisotropic, with a resistivity three times higher in the y direction than the x direction. As a result propagation should be somewhat faster ($\sqrt{3}$ times faster, according to theory) in the x direction than the y direction. Thus the region of excitation should gradually grow more oblong in the x direction as the simulation progresses. Second, the theory says that the extracellular potential Φ_e should look just like the membrane potential V_m, and opposite in sign, anytime the ratio of η_{ex}/η_{ey} equals η_{ix}/η_{iy} as is the case

here. When this is true, $\mathbf{G}_e$ and $\mathbf{G}_g$ are just multiples of one another. You can see this from Eq. (12)...if $\mathbf{G}_e = k\mathbf{G}_g$ for some constant k, then $(k+1)\Phi_e = -V_m$ is going to be a possible solution as soon as the stimuli $i_{intracell}$ and $i_{extracell}$ are turned off. Another way of seeing this is from the electric charge analogy. As hopefully you see from the plot of $\mathbf{j}_m/c$, the effective charge distribution is just one charge shell inside another of opposite polarity. This type of charge distribution tends to produce very little field outside itself (for example, think of the case of two oppositely but equally charged spherical shells, one inside another). On the other hand, by making one change—by switching the values of η_{ex} and η_{ey}, you should then get a strong extracellular field, since the charge-field analogy no longer strictly applies for this case. In fact, this case, when $\eta_{ex}/\eta_{ey} \neq \eta_{ix}/\eta_{iy}$ (the unequal anisotropy ratio case) is currently an active research area right now in the study of cardiac defibrillation. So your standard version of the code is already standing close to the cutting edge of research.

5 Project topics

Once you have the standard version of the code running, there are many interesting topics to look at with the code. Many of these require only modest modification of the code; others require a little more work. The topics naturally divide themselves into three categories: (1) Dynamics of action potential propagation (in two dimensions), (3) Measurement of extracellular fields of action potentials (e.g., electrocardiographic (ECG) measurement), and (3) Defibrillation.

One of the most important things you can do when writing up your work on these topics is to describe any difficulties you encountered while trying to get your project to work, and what you did to try to solve the problems. This will allow us to see that you are using concepts taught in class to solve problems. You should do this even you do not actually solve the problem(s) you encounter—we are still interested in what concepts you invoke in trying to solve the problems.

Remember that providing reasons and explanations for what you see in your simulations are particularly important, since these descriptions are your chance to show your understanding of the material presented in EBME 309, and thus will serve as the basis for the corresponding portion of your grade in that course.

5.1 Dynamics of action potential propagation

The code you have written has the capability of modeling action potential propagation in two dimensions. Propagating action potentials have many properties, many of which are known, but still many of which are not well understood. Perhaps the most intriguing question that one can ask in the cardiac area is, under what circumstances are these wave self-sustaining? This question is important, because self-sustaining action potential activity underlies many of the most dangerous and lethal cardiac arrhythmias. There are two prominent types of self-sustaining waves: anatomical reentry, in which the wave travels around an obstacle, and functional reentry, an interesting rotating pattern that does not require an obstacle. The easies type of functional reentry for us to produce is so-called spiral wave reentry. Spiral waves have been studied for a long time now, and have been seen in the heart—still much

controversy exists about them.

For the topics in this section, you do not need to keep track of the behavior of Φ_e. Action potential dynamics, for the most part, does not require the presence of the extracellular potential, and conversely, the extracellular potential, for the most part, does not play a role in the dynamics of propagating action potentials. To take advantage of this simplification, set the Φ_e column vector equal to zero, initially, and then, in the timestep loop, comment out the line which solves for it so that it always stays equal to zero. When Φ_e is ignored in this fashion, the model is called *monodomain* model, since, with $\Phi_e = 0$, the entire extracellular space is at ground, and thus is being ignored. Once you think you have a working version of this simplified code, you should probably check its values at some random time during the simulation to make sure the really are zero. One reason for keeping the Φ_e column vector around, even though it is zero, is to facilitate its resurrection in the event we need it, either for defibrillating the pattern of action potentials we have produced, or to measure the ECG produced by the pattern, which is effectively a measure of the difference betwen extracellular potentials at two points.

The compelling reason to ignore Φ_e is that solving for Φ_e means solving a matrix equation, which is extremely time-consuming computationally once your system is larger than 50x50 nodes or so. The amount of computational time required to solve a matrix equation scales as the cube of the number of nodes, for the default method that Matlab uses, so that doubling the size of the computational grid (that is doubling Nx and/or Ny) can slow your code by as much as a factor of 8 when Φ_e is being solved for. In practical terms this means that you cannot use huge system sizes when you are solving for the extracellular potential, while, conversely, grid sizes of 200x200 or more are practical, depending on the speed of your computer, when you can afford to ignore Φ_e.

That said, here are some action potential propagation topics. This is by no means an exhaustive list. In fact, investigating phenomena you observe which extends what the original scope of the topics I present here is one of the best ways to obtain high grades for your portion of the project. Finding explanations for these phenomena further strengthens the case.

5.1.1 Development of spiral waves using cross-field stimulation

(Estimated difficulty: Easy) An convenient way to start spiral waves, both in experiments and in computer simulations, is to start a plane wave propagating across the system, say, from left to right, and then when the trailing edge of the first wave is in the middle of the system, a second plane initiated so it propagates at right angles to the first, as shown in Fig. 3. This is often how spiral waves are initiated in experiments. You can initiate each of these waves by applying short-duration intracellular currents to the two regions shown, while simulataneously extracting the same currents from the same regions extracellularly. You can generally get this to work using default values for the ion channel parameters and resistive media (see previous section). I will leave it to you to discover the current intensities and delay between the two stimuli required. In your report, you should describe the mechanism by which spiral waves form in your simulation, using plots from your simulations to illustrate your points. If some of your initial runs failed to produce spiral waves, describe what you thought was wrong, and what you did to fix it! Remember, showing your thinking in making

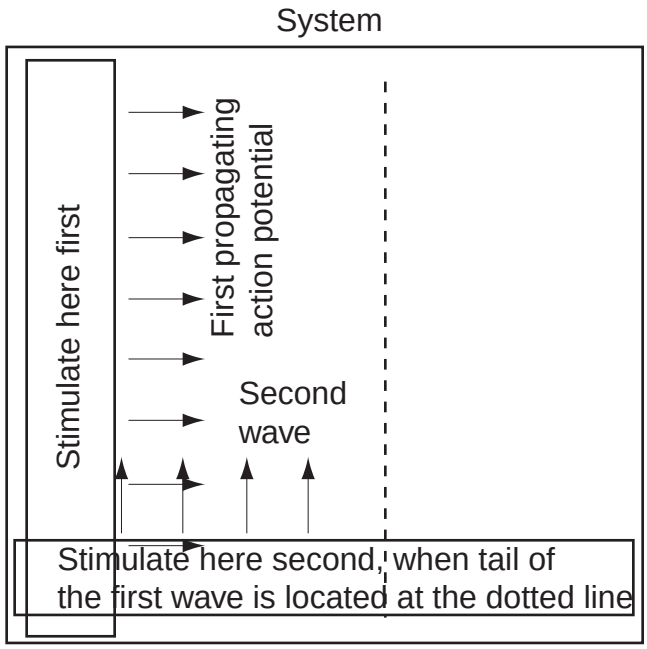

Figure 3: Illustration of the cross-field stimulation method, which can be used to create a rotating spiral wave.

appropriate adjustments is just as important as your description of the working model.

5.1.2 Creation of spiral waves by stimulating on the tail of an action potential wave

(Estimated difficulty: Easy) A second way to create a spiral wave (actually two spiral waves) is to stimulate on the tail of a propagating action potential, as shown in Fig. 4. In this case, you create a plane wave stimulus by stimulating in a narrow rectangular area on the left edge of the simulation, and then apply a stimulus, appropriately timed, in a circular area, on the trailing edge of the first action potential. Some researchers believe that dangerous rotating waves are produced when a group of renegade cells fire spontaneously and with unfortunate timing on the tail of a normal action potential. This simulation is meant to model this mechanism, where the renegade cells (called an "ectopic" node by some) is represented by the stimulated cells in the circular region.

5.1.3 Onset of reentry around dead tissue.

(Estimated difficulty: medium) It is also possible to develop rotating waves with the help of an obstacle. An important example of this is what some researchers believe can happen following the onset of a heart attack. The term "heart attack" generally refers to the death of cardiac tissue caused by the cessation of blood supply produced by coronary blockage. One theory is that death following a heart attack is sometimes caused by the development of a rotating or "reentrant" wave around the infarct, the dead tissue. This rotating wave does not pump blood efficiently to the body, leading through a series of events, to death. We can

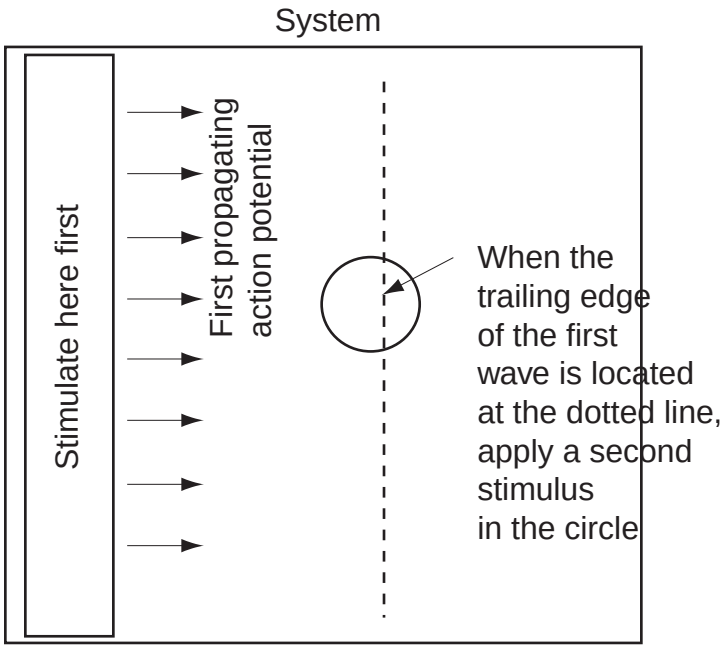

Figure 4: A pair of spiral waves can also be produced by stimulating in a circular area of the trailing (i.e., the "tail") of an action potential

model this situation by representing the dead tissue as cells that have lost their gap junction connections to adjacent large cells. In other words, we represent the dead tissue as a circular region from which all gap junction resistors have been removed. Physiologically, we know this to be true—the heart tries to protect itself following a heart attack by breaking gap junction connection with the dead tissue. The other important ingredient is to modify the tissue surrounding the infarct so that a portion of it displays prolonged action potential duration, once excited. These variations in cardiac tissue properties are common—for example, the action potential is known to be longer in the midmyocardium, but shorter on either the epi- or endocardial surfaces. Normally, these variations are harmless, but when combined with gap junction decoupling, a real danger may exist.

To see this effect, design your system with a "hole" in the intracellular resistive medium; that is, build the medium so that it has a circular region with no gap junction resistors. Allow the medium outside this circular region to have its normal complement of gap junction resistors. Next, modify the ion channel model so that ϵ_2 is abnormally small to the left of the "hole," as shown in Fig. 5, while it takes its normal value to the right. When the simulation is run, apply two pacing stimuli from the bottom of the system. The wave from the first stimulus establishes the pattern of refractoriness around the hole. When the second wave encounters this pattern, it blocks (i.e., fails to propagate) on one side of the hole, but propagates on the other side. By the time the propagating wave spreads above the obstacle to the side on which the blockage occurred, the tissue has recovered, allowing the action potential to propagate downwards back towards the pacing source. The inability of the second wave to propagate upwards on one side of the obstacle combined with its ability later to propagate downwards on the same side is an example of what is widely referred to as "unidirectional block." The presence of this feature is often regarded as either important or

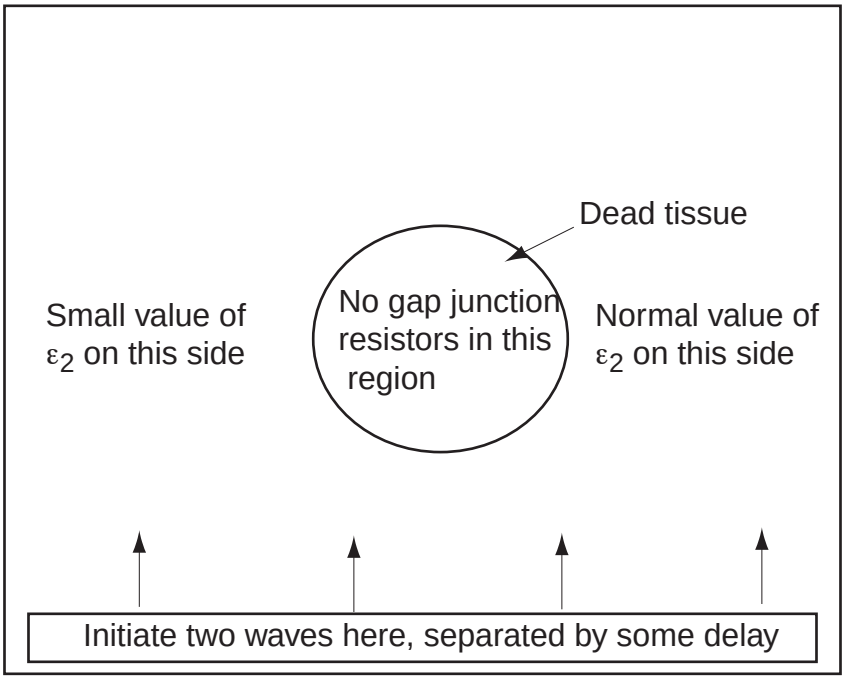

Figure 5: System configuration designed to illustrate how the presence of an antomical feature (dead cardiac tissue in this case) can combine with heterogeneity of repolarization to produce reentry.

essential for establishing rotating waves in the heart.

5.1.4 "Source-sink" characteristics of action potential propagation.

(Estimated difficulty: Part (1): Easy, Part (2): Moderate to easy) Is it possible for a cell which is well-connected via gap junctions to a firing cell to not fire? The surprising answer is yes. The reason is that, if the firing cell is connected to many cells at rest, the former may not have enough current to raise all, or indeed, any of the cells above the firing threshold. Try the following two experiments. (1) Apply the current per unit area i_0 you usually use to start an action potential wave to a very small region—perhaps to a region only containing one cell or a few cells. Depending on conditions, you may find that surrounding cells do not fire. Just for verification, try applying the same i_0 to a larger region. Now do you get firing? Try to explain this in terms of the number of cells initially firing to the number of unfired cells connected to firing cells. (2) Next try building the system shown in Fig. 6. Remove all gap junctions connecting the nodes falling on the two lines shown. Doing this effectively establishes two walls through which the action potential cannot travel. The two walls thus form a narrow corridor. Suppose now that an action potential is intiated at location A. Will this action potential be able to escape from the corridor? Next stimulate at location B. Will the action potential be able to enter the corridor? Why or why not?

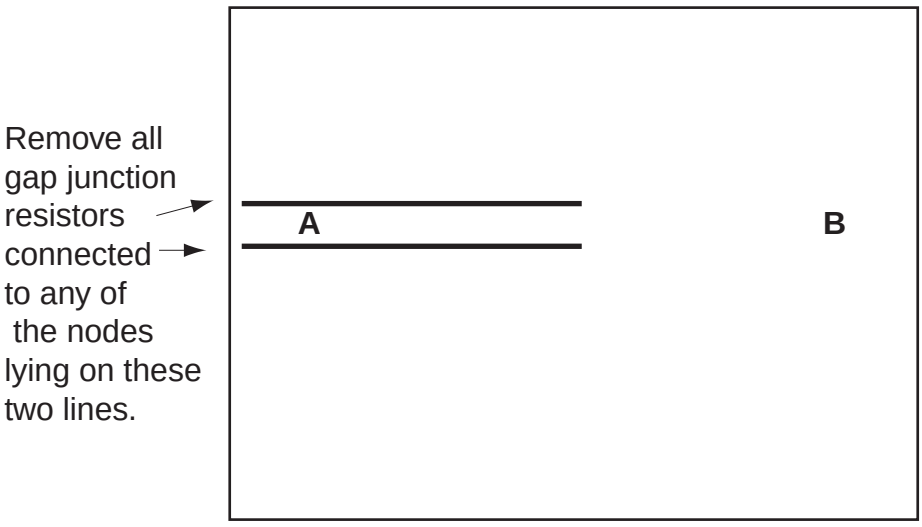

Figure 6: System configuration to study the source-sink properties of action potential propagation.

5.1.5 Propagation velocity of action potentials

(Estimated difficulty: Very easy) It is well-known that the propagation speed of action potentials is inversely proportional to the square root of the resistivity seen by the wave in the direction of propagation. You can see this by setting η's to different values then measuring the major and minor axes of the expanding elliptical wavefront.

5.1.6 The two dimensional heart

(Estimated difficulty: Moderate to as difficult as you want) There is a great deal of interest in making a realistic, three-dimensional computer model of the electrical system of the heart. These models are created using technology not unlike that used in the monodomain model you have created here. In Fig. 7, I illustrate one simple example for a model of the heart in two spatial dimensions. It effectively only has two chambers; also the dependence of the action potential duration (through variation in ϵ_2) more describes what is happening in the ventricular free wall, rather than in the entirety of the ventricles; also the His-Purkinje system is missing, etc.—so there is lots of room for improvement. One advantage of this model is that it produces an ECG roughly resembling the normal ECG measured clinically (see the topic below entitled, "How action potential propagation in the heart generates P, R, and T-waves"). You might also use this model as a starting point for examining in a simple way other cardiac rhythm disorders, such as AV nodal block, or, if combined with some of the strategies used in some other monodomain topics in this subsection, atrial and ventricular tachycardia and fibrillation. Another possibility: By creating a second hole (the AV node being the first) in the barrier between atrium and ventricle, you create the possibility of anatomical reentry, a condition known as Wolff-Parkinson-White (WPW) syndrome. In this syndrome, a wave is established that travels down through the AV node into the ventricles, then up through the second hole back into the atria, then back down through the AV node, etc. The tough part here is getting this condition going from just the waves being produced

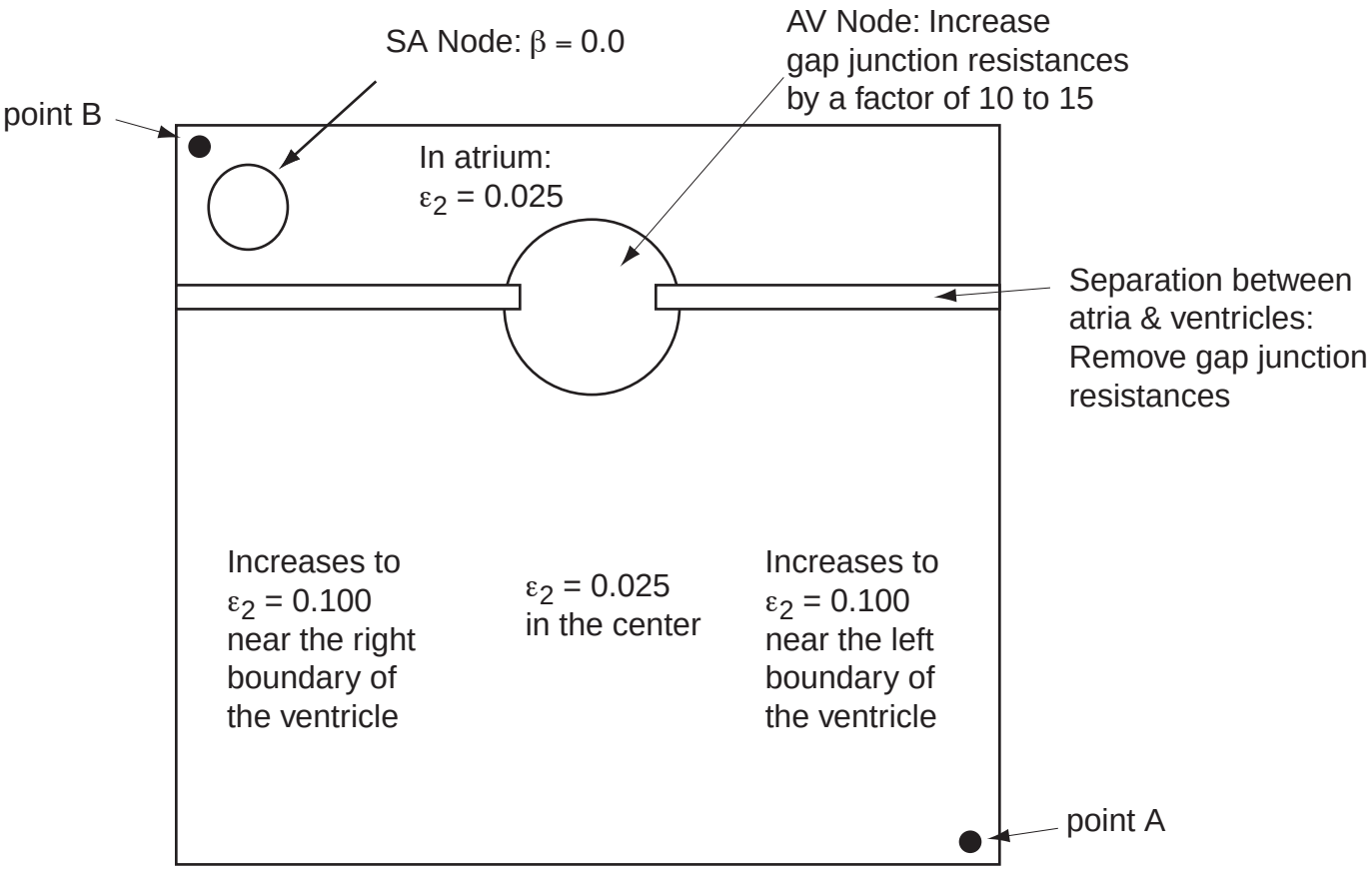

Figure 7: One possible two-dimensional model of the electrical system of the heart. Different and/or improved models are also possible.

by the SA node. You can see that there is a strong similarity here to the previous topic on anatomical reentry (see above). In particular, it can be difficult to create the conditions that will produce unidirectional block. Each of these add-on topics counts as a separate topic, each in the "Difficult" category.

5.1.7 Fibrillation created by tissue heterogeneity

(Estimated difficulty: Moderate). A leading theory for how fibrillation is created is based on the presence of heterogeneity of the cardiac tissue. Heterogeneity simply means that the properties are different at different locations on the tissue; that is, the values of ϵ_1, ϵ_2, β, η_x, η_y, etc. are different in different places. Try using Matlab's random number generator, `rand`, to add "noise" to one of these parameters at a time. For example, adding noise to β should start to break up the expanding wavefronts once they have traveled some distance from the pacing site. Then, for each case, repeatedly stimulate (that is, pace) from a small region. Alternatively, you can start a spiral wave spinning in one corner or put some pacemaker cells in one corner. Either will spew out a rapid train of action potential waves. For this simulation, you may need a fairly large grid perhaps 100x100 to 200x200 to give the waves enough room to break up. See if can see what exactly it is that is breaking up the wave for at least one of your noisy parameters. At one time, this mechanism was the leading theory for how atrial fibrillation is produced. Recently, however, other theories have been advanced.

5.1.8 Spiral wave drift

(Estimated difficulty: Running the simulations: easy. Figuring out why the drift is in the direction it is: Moderate to difficult.) Spiral waves, once created, do not always stay in one place. If one of the ion channel parameters varies slowly and smoothly from one side of the system to the other, the spiral wave will drift in some direction. This may be a good thing—a drifting spiral will usually run into some nonconducting anatomical structure and die. To the clinician this looks like a nonsustained tachycardia—not as dangerous as a tachycardia that won't stop. Here's a key question: given a smooth variation of a parameter in one direction, why does the spiral wave drift in the direction that you observe? Can you figure it out from the dynamics? (Sometimes this is a difficult question, sometimes less difficult.)

5.2 Extracellular fields of action potentials

The behavior of action potentials in the heart is often measured through the extracellular fields they produce, in the form of ECGs or other measuring methods, using catheters, or other devices. It is therefore important to understand the mechanism by which action potentials produce extracellular fields, so that we can then reconstruct what the action potential(s) must have looked like, given the measured extracellular field. Here are a few computer experiments that exhibit this process:

5.2.1 Fields of an action potential propagating in a fiber

(Estimated difficulty: Moderate.) As has been discussed in class, the field of a propagating action potential has a characteristic quadrapole shape: positive ahead and behind the action

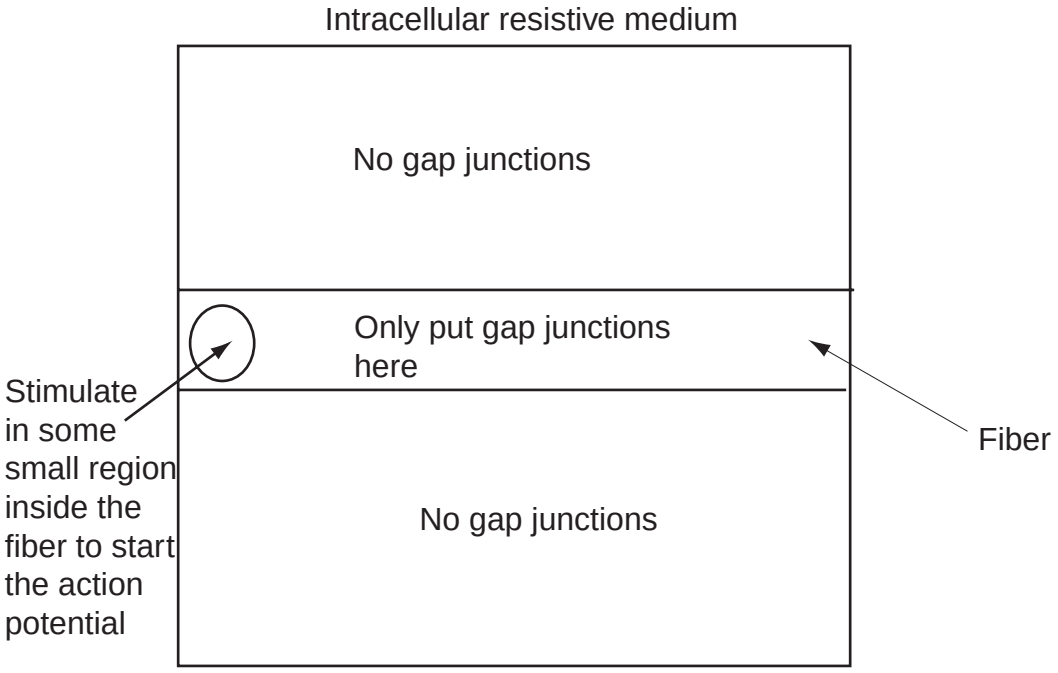

Figure 8: One possible two-dimensional model of the electrical system of the heart. Different and/or improved models are also possible.

potential, and negative on the flanks. To see this extracellular potential field, first create an intracellular resistive medium that represents the fiber, as shown in Fig. 8. Modify your resistive medium construction code so that it only puts gap junction resistors in the region that represents the fiber. Technically, excitable cells should also only be loaded in the fiber region; however, you may find it easier to lay down ion channel dynamics everywhere. (This is simplest; otherwise you will have to figure out what V_m means in a region where there are no cells, and therefore no cell membranes, etc.) Be mindful that, by doing things this way, you have a bunch of excitable cells the region outside the fiber. They will normally be invisible, since they are not connected to each other or the fiber. However, if you stimulate in the region outside the fiber, these cells will fire. Because of this, you should probably restrict your stimuli to regions inside the fiber. In contrast to the intracellular medium, the resistive extracellular medium exists everywhere, both "under" the fiber and away from it. You can therefore use the standard version of your resistive medium code to model the extracellular space. You can start an action potential in the fiber by stimulating in a small region, as shown in Fig. 8. There are several useful plots you can make. These are colorplots of V_m, Φ_e, and the current through the membrane, $\partial V_m/\partial t + i_m/c$, which you can more easily plot by plotting $-\mathbf{G}_g \cdot (V_m + \Phi_e) + i_{intracell}/c$, as should be clear from Eq. (10). The relevance of plotting the membrane current is that it serves as the source for the extracellular fields, just as charge is the source for electric field, as described in the bidomain notes. You may also find it enlightening to plot the $-\nabla V_m$, since, as shown in class, this derivative of V_m is the source of dipole fields. You can easily calculate this quantity by using Matlab's `gradient` function, which in turn is easily plotted as a vector field using Matlab's `quiver` function. You read about these functions typing `help gradient` or `help quiver` on Matlab's command

line.

5.2.2 How action potential propagation in the heart generates P, R, and T-waves

(Estimated difficulty: Very difficult.) You can model how the standard ECG trace, complete with the P wave, the R wave portion of the QRS complex, and the T wave by reinstating the extracellular field Φ_e to the 2-d heart simulation topic described in the previous subsection. The lead-2 ECG (which is the one most widely displayed) may be obtained by subtracting the Φ_e measured at point B (see Fig. 7) from Φ_e at point A. What makes this topic difficult is that the signal from atrial repolarization is supposed to fall within the QRS complex, and is thereby hidden. In my model, at least, getting this timing correct was tricky. A substantial delay was required in the AV node, but when this was attempted, the gap junction resistances required were so high that the action potential often blocked (i.e., failed to propagate), and thus never made it into the ventricles (AV nodal block). The reason for block was probably a source-sink problem (see the topic by this name in the action potential propagation subsection). Even when the timing was set up properly, the signal produced by action potential propagation in the atria was much larger than it is clinically, making it hard to hide in the QRS complex. In the real life situation, this signal is small, because the mass of tissue in the atria is small (mostly due to thin atrial walls). It's not so obvious how to model this in a 2-d model. You can minimize the signal by making the atrial region small (e.g., it's only about one-fourth the size of the total system in Fig. 7) but even then the signal is quite large. Since this topic is difficult in its details, I would not expect perfect results if you were to tackle it. Give it a try, and if you get something reasonably close, with good written explanations of what you observed and what you tried, that will result in a good grade!

5.2.3 ECGs from abnormal action potential propagation patterns

(Estimated difficulty: Difficult) I would like to encourage you to try measuring ECGs from any of the interesting patterns you encountered from topics in the previous subsection on action potential propagation. There are therefore actually several topics here, each of which counts as a separate project topic. Just reactivate the extracellular portion of the code, and, as long as the system size is not too large (which will slow down the code enormously), you should be in business. What makes these topics difficult is the interpretation of what you see. You will have to apply your understanding of the bidomain model to action potential patterns I have not covered in class, which means you will have to do some reasoning on your own. However, please be encouraged that this type of reasoning will be rewarded in the grade!

5.3 Defibrillation

The third major use of this code is to study the process of defibrillation; that is the application of a large stimulus to the extracellular space in order to generate membrane potentials large enough to disrupt unwanted action potential propagation patterns, such as spiral waves

and anatomical reentry. The topics in this category naturaly divide themselves into two subcategories: (1) How the defibrillating field gets into the tissue, and (2) What the field's effect is on abnormal action potential propagation patterns. With respect to subcatagory (1), the intriguing aspect of these topics is that fields do not penetrate as readily as one might expect. For example, if you apply a large current to the extracellular medium using two plane electrodes, one on either side of the system, you will indeed generate a huge extracellular potential drop. Paradoxically, however, when the tissue properties are homogeneous, the *membrane potential* this field generates is confined to two narrow layers on the two surfaces of the tissue closest to the two electrodes—the membrane potential deep in the tissue is not affected at all. What's strange is that the extracellular field deep within the tissue is huge—you would have expected that this large extracellular field would have done something to create a significant membrane potential in the same location, but such is not the case. Now, in the real world, a substantial membrane potential is generated deep within the tissue (or probably is, since we have no good direct methods for measuring it), which in turn interacts with the "bad" waves, sometimes killing them. So how do these fields get in? There are two schools of thought on this. One is, that small-scale heterogeneity at level of individual cells allows the fields to penetrate. You can model this by adding and subtracting random values to the individual gap junction resistances. The other is, larger scale variations in either defibrillating fields or the resistive properties of the media combine with unequal anisotropy ratios in the resistivities to allow the fields in. You can see this by applying current unevenly on the edges of the system, while simultaneously adjusting the values of the resistivities so that $\eta_{ex}/\eta_{ey} \neq \eta_{ix}/\eta_{iy}$. In view of the importance of this question of how the fields get in, I would recommend that anyone looking topics 2,3,4 or 5, below, should also look at topic 1. The remaining topics also benefit from a broad understanding of when the fields will penetrate and when they won't, so for those topics, I would recommend that you at least read the descriptions of topics 1,2,3,4 and 5.

5.3.1 The defibrillation paradox: how do defibrillating fields penetrate deep into cardiac tissue?

(Estimated difficulty: Easy) To understand what all the fuss is about, first try this computer experiment. Modify the standard version of your code so that a huge, uniform current is applied extracellularly on one edge of the system, and is uniformly extracted from the opposite edge, as shown in Fig. 9. Eliminate the machinery in the standard version that was responsible for generating an expanding, propagating elliptical wave. Make colorplots of Φ_e and V_m. You should find a huge gradient is produced in the extracellular field, and that huge membrane potentials appear on both surfaces, but nothing happens to the membrane potential inside. Sure, one of the surface membrane potential regions launches an action potential wave into the system, but creating waves is generally not effective in killing rotating waves. What we want is to generate large membrane potentials over a large percentage of the tissue at once. Try this computer experiment once using equal anisotropy ratios and once using unequal ratios. It shouldn't make a difference, because all the current, both intracellular and extracellular, flows in one direction from one electrode to the other, so the resistivity in the other direction doesn't matter. Thus, the system doesn't even "see" the inequality in the ratios.

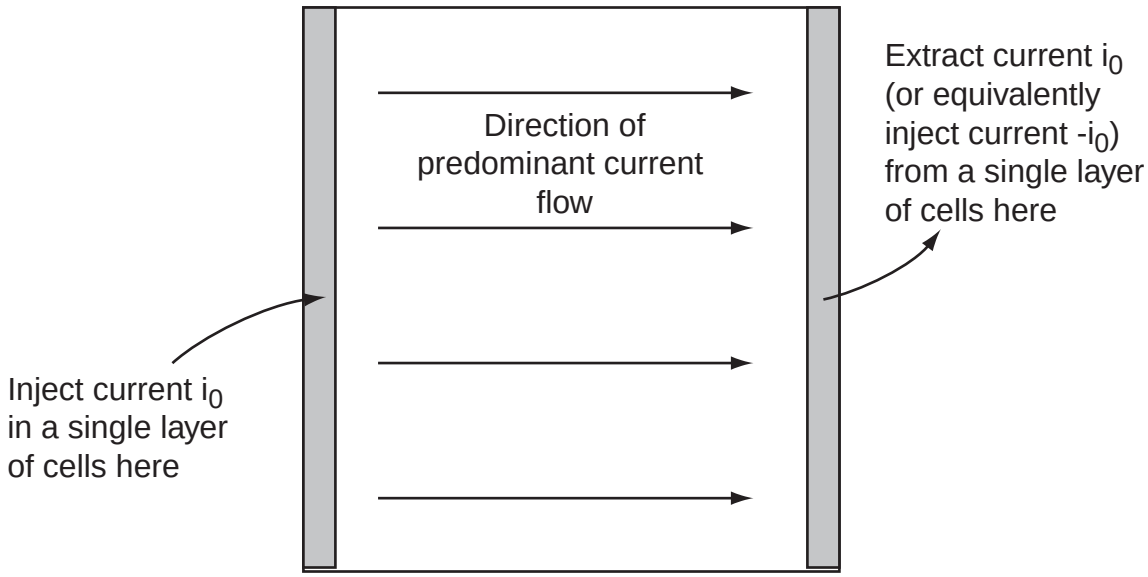

Figure 9: The defibrillation paradox: Why doesn't the application of current through uniform cardiac tissue using two plane electrodes produce any significant defibrillating effect deep within the tissue?

5.3.2 The role of tissue heterogeneity in producing defibrillation effects deep in cardiac tissue

(Estimated difficulty: Easy to moderate) So one way of getting the field in is to add heterogeneity to the intracellular resistivity. You can do this by adding and subtracting random numbers from the values of gap junction resistances. Make sure none of the resistance values end up negative!

5.3.3 Effect of different anisotropy ratios on the membrane potential

(Estimated difficulty: Moderate) Another way to generate substantial membrane potentials deep in the tissue relies on the inequality of the anistropy ratios, inside and out. Before showing how this inequality can produce this deep penetration, first we should learn something about what effects unequal anisotropic ratios produce. To see this, start with the standard version of the code, and remove those portions of the code that produce the expanding action potential wave. Instead, stimulate intracellularly (no extracellular stimulation!) in the center of the system. For this computer experiment, ideally, we would like the current to flow away from this stimulus point to infinity, never to return. We can't do this, of course, so we do the next best thing: we collect the outward-flowing current at the boundary of the system. To implement this, we inject a current into the extracellular medium all the way around the boundary that is equal and opposite to the current injected in the center of the system. This is the same as collecting the current, if you think about it.... So, if the current density of the injected current at the center is i_0, then the current density to inject along the boundary is $-i_0/N$, where N is the number of cells on the boundary. (Each cell collects $1/N$th of the current; hence the factor of $1/N$.) In order to do this, you will need to write a modified version of your stimulus routine that can inject identical currents into every cell

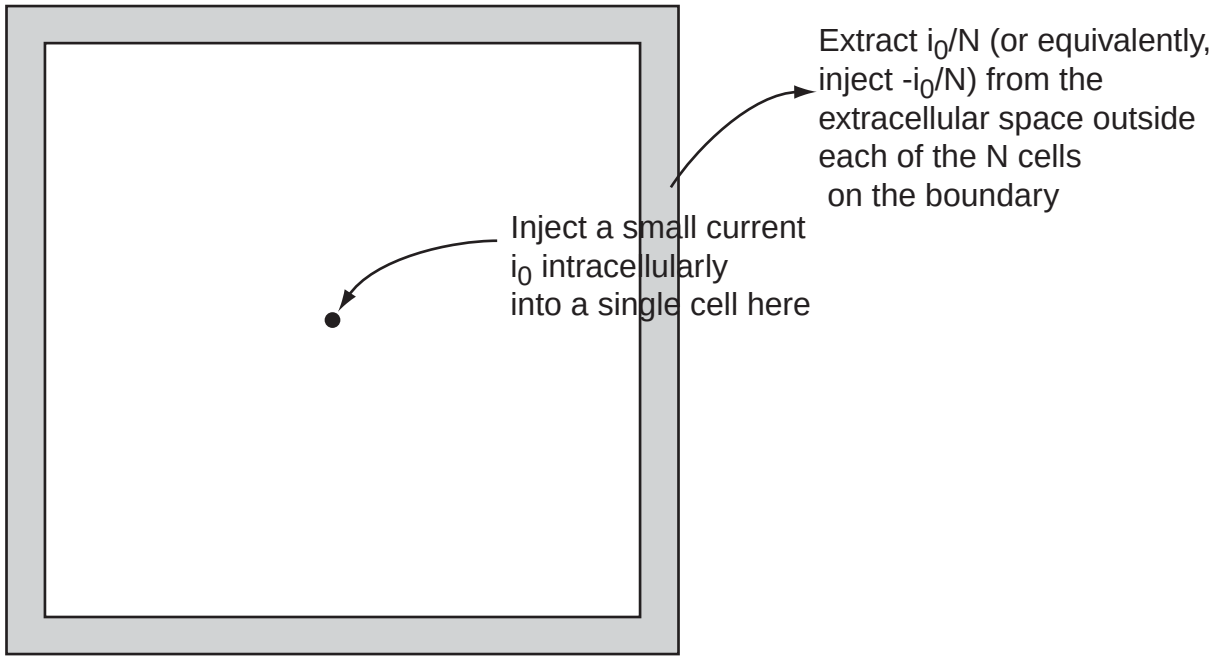

Figure 10: Placement of the electrodes for the study of the field produced by a point electrode in a medium with unequal resistivity anistropic ratios.

on the edge of the system. The arrangement of electrodes is shown in Fig. 10. Our goal in this case is to discover the pattern of membrane potentials a current injected at a point likes to produce; therefore, you should use a very small current i_0 to avoid creating action potentials, which would be considered the product of the excitable medium, not the probe current. For one case, choose values for η_{ex}, η_{ey}, η_{ix} and η_{iy} such that the anisotropy ratios, η_{ex}/η_{ey} and η_{ix}/η_{iy}, are equal. Then do another case in which the resistivities so that ratios are unequal. Make one ratio substantially greater than one, and the other significantly less than one.

For each case, wait for the system to settle down, then look at the membrane potential near the center electrode. For the equal anisotropy case, the membrane potential should only be substantially nonzero close to the electrode. In contrast, the unequal case exhibits more of a quadrapole-like field, with lobes of it extending quite some distance away from the electrode. It is this ability of the unequal-ratio system to maintain significant membrane potentials far away from the source of current that allows the deep penetration of defibrillating fields.

5.3.4 Use of unequal anisotropy ratios in the resistivities to get the defibrillating field in

(Estimated difficulty: Moderate) From the previous topic, we see that media with unequal resistivity anisotropy ratios like to support deep penetrating membrane potential fields. However, as we see from the first topic of this subsection, this potential (pardon the pun) of unequal anisotropy cannot be realized with large plane electrodes. So, the secret is to use smaller electrodes, or electrodes that inject current in a spatially dependent way. Figure 11

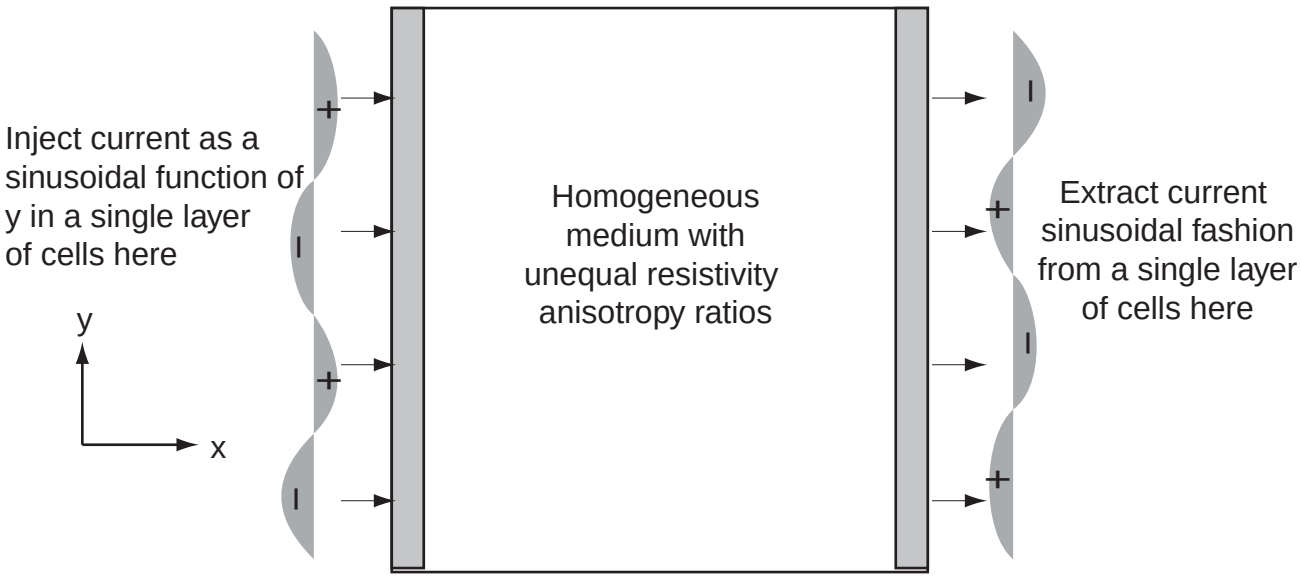

Figure 11: The use of electrodes that inject a current whose amplitude is spatially dependent allows the generation of a substantial membrane potential deep in the tissue.

illustrates the concept. If the injected current is not uniformly distributed, but instead is positive at some locations of the electrode, and negative at others, or more specifically, is sinusoidal in the y-direction, then there exists the possibility of current flowing in both the x and y directions, which then brings the intracellular and extracellular resistivities in both directions into play. This allows the unequal anisotropies in resistivity to express themselves, which makes the possibility of long-range generation of the membrane potential, deep in the tissue, at a significant distance from the electrodes, possible. See if you can see this effect by modifying your electrodes to deliver current in this sinusoidal pattern. Then compare the membrane potential colorplots you get with those from an equal anisotropy run.

5.3.5 Interaction of defibrillation fields with action potentials

(Estimated difficulty: Difficult) Of course, the goal of all this is to stop these rotating waves. Pick your favorite rotating wave from the Propagating Action Potentials subsection and try to wipe it out using your favorite defibrillating method from this subsection. There are several possible combinations here, each of which counts as a separate topic. Who among your group would you most like to see in the virtual ER?

5.3.6 Adverse effects of virtual electrodes: reinitiating ventricle tachycardia

(Estimated difficulty: Difficult to Very Difficult) It seems that one of the themes of cardiology is that many of the supposed cures are also the progenitors of new problems. Many antiarrhythmic drugs actually cause rhythm disorders when used improperly. Now it seems that the same electrodes that can deliver life-saving defibrillating shocks can also start new rotating waves, perpetuating the problem. Similarly, the presence of unequal anisotropy ratios which may be what allows defibrillating fields to penetrate, also may be the agent

which makes possible the development of these new rotating waves. We can see this by turning up the intensity of the currents in the computer experiment illustrated in Fig. 10. The right values for the size of the central stimulus region and current intensity and the correct polarity for the current can initiate *quadrafoil reentry*. In this type of reentry, no less than *four* rotating waves are formed. This topic is difficult because it can be a little tricky to find the right values for these parameters. Once you get the four spiral waves, figuring out the mechanism by which they occur is relatively easy. This simulation is another example of the usefulness of computer modeling: the simulations suggest new directions in which to look, making the discovery process so much easier in many cases. Who would have suspected that four spiral waves could form as the result of unequal resistivity anisotropy ratios?