Appendix A
Matlab Fundamentals

This appendix provides an overview of Matlab[1] mathematical software, widely used in scientific computation applications to simulate physical systems, run computational algorithms, as well as perform comprehensive data analysis and visualisation. Optional toolboxes extend Matlab functionality to specialised applications including neural networks, signal processing, bioinformatics, system identification, image processing and systems biology.

A.1 Matlab Overview

Matlab provides an interpreter environment for executing an extensive library of in-built mathematical commands, as well as user-defined code scripts and functions. This section provides an overview of the Matlab interface and basic functionality.

A.1.1 User Interface

The default Matlab interface consists of several windows, as shown in Fig. A.1. These are the *command window* where commands are entered and executed by the Matlab interpreter, the *workspace* which lists variables defined in the current session, the *command history* which lists recent commands, and the *current folder* window and *file path* where user-defined scripts and functions are saved and accessed. Recent commands can be re-typed in the command window by using the up-arrow keyboard shortcut. Repeated use of the up-arrow will cycle through several commands, beginning from the most recent entered.

[1] The Mathworks Inc, Natick, Massachusetts, U.S.A.

S. Dokos, *Modelling Organs, Tissues, Cells and Devices*,
Lecture Notes in Bioengineering, DOI 10.1007/978-3-642-54801-7

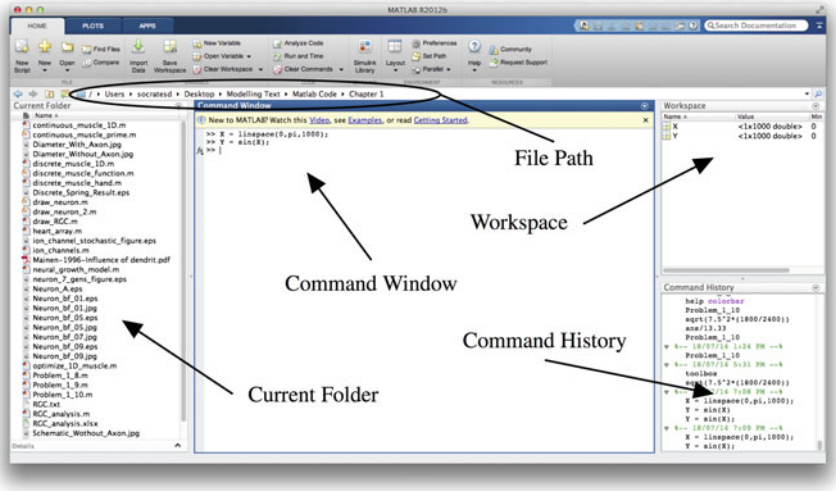


Fig. A.1 Default Matlab interface. The command window represents the main work area where commands are entered for execution by the Matlab interpreter. The workspace window displays current variables, the command history shows recent commands entered, and the current folder window specified by the file path lists local files

A.1.2 Working with Variables and Arrays

Variables do not need to be declared first: simply assign their value directly. For example, the following commands entered in the command window:

```
r = 2;
c = 2*pi*r;
```

will create the variables `r` and `c` in the current workspace and assign their values to 2 and $2 \times \pi \times 2 = 4\pi$ respectively (note that `pi` is the in-built Matlab constant for $\pi \approx 3.1416$). Note that each command ends with a semicolon. Although not strictly necessary, the semicolon prevents command results from being echoed in the command window.

To define an array, values can be entered element by element, as in the following:

```
x = [2,3,1,0,0];
y = [1;2;3;4;5];
A = [1,0;0,1];
```

which define `x` to be a five-element row array, `y` a five-element column array, and `A` as the 2×2 identity matrix. The subsequent command

```
d = x*y;
```

would perform array multiplication of a row and column vector, yielding the scalar `d` = 11. Reversing the order of the arrays in the command `E = y*x` would assign a 5×5 matrix to `E`.

Individual elements of the above arrays can be accessed using commands such as `x(1)` (returning a value of 2) and `A(1,2)` (returning a value of 0). To append elements to existing arrays, use commands like

```
z = [x,3];
w = [y;8];
```

which add extra elements of value 3 and 8 to the end of the above-defined arrays `x` and `y` respectively. Note that a comma appends to rows, whilst a semi-colon appends to columns. Similar principles apply when concatenating two-arrays. Thus, for example

```
z = [x,x];
w = [y;y];
```

would double the length of both `x` and `y` defined above.

Arrays can also be defined using

```
x = 0:0.001:1;
```

which creates a row array of 1001 uniformly-spaced elements, 0 as the first and 1 as the last, in increments of 0.001. An alternative is to use Matlab's `linspace` function:

```
x = linspace(0,1,1001);
```

which yields the same result. Note that this function takes three arguments, the first and last values of the array, and the total number of elements.

To square each element of `x`, use the `.^` exponent operator which acts on each element of `x` individually:

```
y = x.^2;
```

Analogous element by element array operators also defined for multiplication (`.*`) and division (`./`). Thus, for example, the following sequence of commands:

```
A = [1,2;3,4];
B = [5,6;7,8];
```

```
C = A*B;
D = A.*B;
E = A/B;
F = A./B;
```

would yield

$$\texttt{C} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}, \quad \texttt{D} = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}, \quad \texttt{E} = \begin{bmatrix} 3 & -2 \\ 2 & -1 \end{bmatrix}, \quad \texttt{F} = \begin{bmatrix} 0.2 & 0.3333 \\ 0.4286 & 0.5 \end{bmatrix}.$$

Note that the division operator (`/`) for calculating `E` denotes matrix division, such that `A/B` = `A*inv(B)` where `inv(B)` is the inverse of matrix `B`.

In addition to real number data types, Matlab allows other variable types including strings and complex numbers. For example, the commands

```
a = 'This is some text';
b = complex(1,2);
```

define `a` and `b` to be string and complex data types respectively.

A short list of basic Matlab operators and functions is given in Table A.1. More comprehensive documentation on Matlab operators, functions and advanced features can be found in the in-built documentation, which can be accessed from the command window using

```
doc matlab
```

Help on any command can be obtained in the command window by typing `help` followed by the command, e.g.

```
help linspace
```

Typing `doc` followed by the command will display html-formatted documentation instead:

```
doc linspace
```

A.1.3 Matlab Programming

Matlab provides an extensive set of high-level programming features for implementing complex automated numerical computations and algorithms.

Table A.1 List of basic Matlab operators and functions

Operator(s)	Description
`* + - /`	Basic arithmetic operators
`^`	Exponent operator e.g. `3^2` (= 9)
`.* ./ .^`	Element by element array operators
`mod(x,y)`	Modulus operator, yielding the remainder on division of `x` by `y`
`sin(x) cos(x) tan(x)`	Trigonometric functions
`exp(x)`	Exponential function e^x
`log(x)`	Natural logarithm
`\textbackslash{}`	Array division
`inv(A)`	Returns the inverse of square matrix `A`
`> < >= <= == ~=`	Comparison operators, returning a value of 1 if true, or 0 otherwise
`~ && ||`	NOT, AND, OR logical operators
`linspace(x1,x2,N)`	Generates row array of `N` equi-spaced values from `x1` to `x2`
`zeros(N,M)`	Returns an `N` × `M` matrix of zero elements
`ones(N,M)`	Returns an `N` × `M` matrix with all elements equal to 1
`rand(N,M)`	Returns an `N` × `M` matrix of random uniformly-distributed elements between 0 and 1
`randn(N,M)`	Returns an `N` × `M` matrix of normally-distributed random elements with mean 0 and standard deviation 1
`plot(x,y)`	Plots array `y` against `x`

A.1.3.1 Scripting

Matlab command sequences can also be saved as *scripts*; text files having a `.m` extension. Scripts can be written using the in-built Matlab *editor*, invoked from the Matlab File menu or from the Toolbar, depending on the version of Matlab. To execute the script, enter the name of the script (i.e. the filename without the `.m` extension) in the command window. For example, to generate a plot of $y = \sin(x) + 0.2\cos(2x)$, the following commands can be saved to a script named `my_waveform.m`:

```
% initialise x from 0 to 2*pi:
x = 0:2*pi/1000:2*pi;

% calculate waveform:
y = sin(x)+0.2*cos(2*x);
```

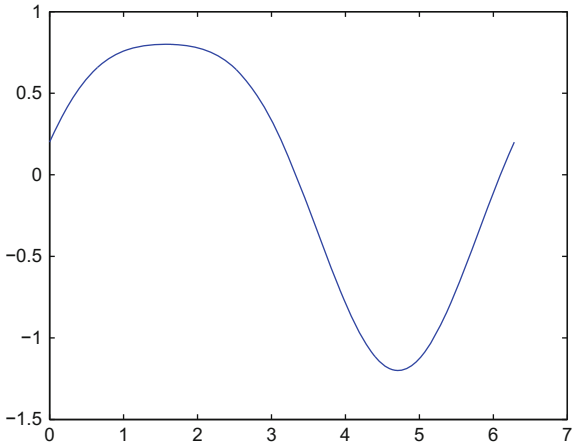


Fig. A.2 Plot of $y = sin(x) + 0.2cos(2x)$ using the `my_waveform` Matlab script

```
% plot graph:
plot(x,y);
```

Note that text following the `%` character in a line is a *comment*, useful for documenting code function, and is ignored by the Matlab interpreter. Entering `my_waveform` in the command window produces the plot shown in Fig. A.2.

A.1.3.2 Conditional Branching and Loops

As with all high-level programming languages, Matlab provides several conditional branching and loop structures, including `if... else` and `case` structures, as well as `for` and `while` loops. These can be used in scripts as well as user-defined functions (see Sect. A.1.5). For example, the following code generates, rather cumbersomely, a square-wave input stimulus current `I` from an array of time values, such that $\texttt{I} = \begin{cases} 50 & t \leq 10 \\ 0 & \text{otherwise} \end{cases}$:

```
t = 0:1:100;
I = zeros(1,101);
for i=1:101
    if (t(i)<=10)
        I(i)=50;
    else
        I(i)=0;
    end
end
```

Note that the same result could be generated using the far more compact code:

```
t = 0:1:100;
I = 50*(t<=10);
```

A.1.3.3 Code Debugging

The in-built Matlab editor provides continuous, automated code checking to alert the user to coding errors and warnings, as well as additional tools for code debugging. Fig. A.3 illustrates the editor view for the previous for-loop generating a square-wave stimulus, however this time with a missing `end` statement. An error indicator in the top right margin of the editor window alerts the user to a serious code error. The indicator colour can be red, orange or green and indicates either (1) a syntax error (red) in which the code will not run, (2) a warning (orange) in which the code will run but the user should heed the given suggestion, or (3) no error (green). Also shown in the right margin are line markers indicating the relevant location of errors and warnings.

It is also possible to insert breakpoints into the code by clicking in the left margin of the editor. When run, the code will pause execution at the breakpoint, allowing variables to be examined. In fact, any Matlab command can be executed from the command window whilst the code has paused, providing a very powerful debugging feature. Editor toolbuttons allow the user to subsequently step through the code, one line at a time, or continue execution until the next breakpoint.

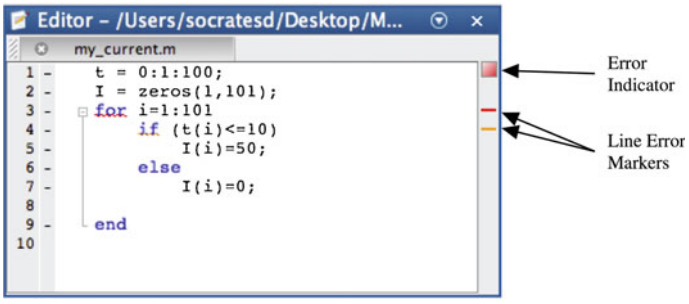


Fig. A.3 Matlab editor, with *red* indicator (*top right*) alerting the user to a syntax error, due here to a missing `end` statement. Also shown in the *right* margin are line markers indicating specific locations of code errors and warnings. In this case, a *red* marker at line 3 indicates that the `for` statement has a missing `end`, and the *orange* marker provides a warning that the `if` statement on line 4 does not have a matching `end`

A.1.4 Solving Linear Systems of Equations

The following linear system of equations:

$$\begin{aligned} 2x + 3y - 4z &= 7 \\ x + 5y - z &= 2 \\ x + y &= 1 \end{aligned}$$

can be represented by the equivalent array equation

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & -4 \\ 1 & 5 & -1 \\ 1 & 1 & 0 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 7 \\ 2 \\ 1 \end{bmatrix}$$

which has the solution

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}.$$

In Matlab, the above system can be solved for using the backslash (\) operator:

```
A = [2, 3, -4; 1, 5, -1; 1, 1, 0];
b = [7; 2; 1];
x = A\b;
```

which yields, correct to four decimal places,

$$\mathrm{x} = \begin{bmatrix} 1.0667 \\ -0.0667 \\ -1.2667 \end{bmatrix}.$$

Use of the backslash operator is equivalent to the Matlab command

```
x = inv(A)*b;
```

which inverts matrix **A** and multiplies by **b**. However, Matlab's backslash operator is more efficient and accurate than direct matrix inversion, particularly for large systems. Using this operator, Matlab can easily solve systems consisting of thousands of matrix elements, as in the following example:

```
A = rand(1000);
b = ones(1000,1);
x = A\b;
```

which only takes a fraction of a second to solve for on a current standard desktop or laptop computer! In the above code, `A` consists of a 1000×1000 matrix of uniformly-distributed random elements between 0 and 1, and `b` is a 1000-element column array consisting of 1's.

A.1.5 User-Defined Functions

In addition to hundreds of in-built mathematical functions, Matlab allows the user to define custom functions which can take multiple arguments, and produce multiple outputs. User-defined functions are saved in `.m` files whose first line contains the `function` reserved word. For example, to create a function to solve the system of equations $\mathbf{Ax} = \mathbf{b}$, the following code can be used:

```
function x = solve_my_system(A, b)
    x = A\b;
end
```

which must be saved in a `.m` file having the same name as the function: in this case, `solve_my_system.m`. Note that this function takes two arguments, `A` and `b`, and returns a single output `x`. The following command can then be invoked from the command window, or within other code:

```
C = [2, 3; 1, 4];
d = [3; 8];
z = solve_my_system(C, d);
```

To define a function with multiple outputs, use code such as:

```
function [x y] = solve_my_systems(A, b, c)
    x = A\b;
    y = A\c;
end
```

which would be invoked from the command window using

```
C = [2, 3; 1, 4];
d = [3; 8];
e = [1; 2];
[u v] = solve_my_systems(C, d, e);
```

A.1.6 Solving Systems of ODEs in Matlab

Matlab provides powerful functions for numerically solving systems of ordinary differential equations (ODEs). As an example, consider the following ODE system:

$$\begin{aligned}\frac{dx}{dt} &= -2x - 3y - 4z \\ \frac{dy}{dt} &= -x + 5z \\ \frac{dz}{dt} &= -x - 2y - 3z\end{aligned}$$

with initial values $x(0) = y(0) = z(0) = 1$. This can be written in matrix form as

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x}, \quad \mathbf{A} = \begin{bmatrix} -2 & -3 & -4 \\ -1 & 0 & 5 \\ -1 & -2 & -3 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \text{with} \quad \mathbf{x}(0) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

To solve such a system in Matlab, we write a function to output the time-derivative evaluations as a function of both t and $\mathbf{x}$:

```
function dxdt = derivs(t,x)
    A = [-2, -3, -4; -1, 0, 5; -1, -2, -3];
    dxdt = A*x;
end
```

The system can then be numerically-solved using Matlab's built-in ODE solver `ode15s` by coding the following in a separate script:

```
x_start = [1; 1; 1];
t_range = [0 5];
[t, y] = ode15s('derivs', t_range, x_start);
plot(t,y), legend('x','y','z');
```

Executing this code produces the plot shown in Fig. A.4. Note that the user-defined `derivs` function above included both `t` and `x` as arguments, even though only `x` was strictly required in this example (the time-derivatives of this system are functions only of $\mathbf{x}$). However, `ode15s` requires the user-specified derivative-evaluation function to include both `t` and `x` as arguments.

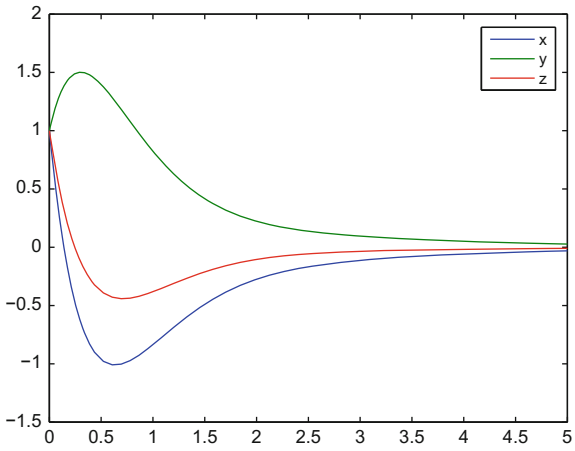


Fig. A.4 Numerical solution of ODE system using Matlab's `ode15s` function

Appendix B
Overview of COMSOL Multiphysics

COMSOL Multiphysics[2] is a versatile finite-element software package providing a convenient means for implementing a wide range of multiphysics models. These include standard physics modalities such as electromagnetism, structural and fluid mechanics, diffusion and heat transfer, as well as user-defined systems. Its multiphysics coupling capabilities render COMSOL an increasingly popular choice for bioengineering modelling. Optional add-on modules provide user-interfaces and functionality for additional physics implementations including electromagnetics, microelectromechanical systems (MEMS), heat transfer, nonlinear structural materials, fluid mechanics, microfluidics, as well as interfaces to other software such as the LiveLink for Matlab interface, which allows COMSOL models to be implemented from within Matlab.

B.1 COMSOL Basics

COMSOL has undergone several changes to its user-interface since early versions pre-2005. This section provides an overview of COMSOL v5.2, the most recent release at the time of writing.

B.1.1 User Interface

The default COMSOL user-interface consists of several windows, as shown in Fig. B.1. These include the *Model Builder Window*, which provides a tree representation of the current model, the *Settings Window* which reports associated model settings when clicking a node in the model tree, the *Graphics Window*, which displays the model geometry, mesh and results, and the *Information Windows* which provide

[2]COMSOL Inc., Burlington, MA.

S. Dokos, *Modelling Organs, Tissues, Cells and Devices*,
Lecture Notes in Bioengineering, DOI 10.1007/978-3-642-54801-7

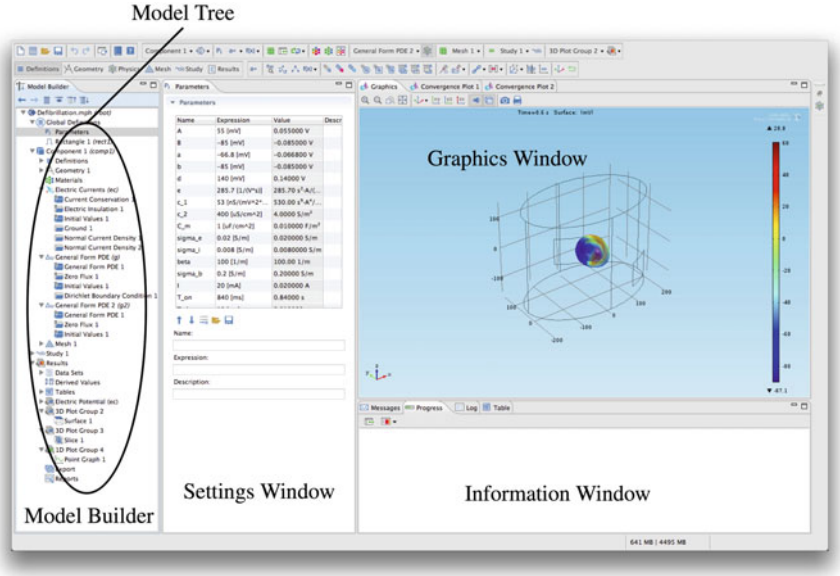


Fig. B.1 Default COMSOL interface (Mac OS X, version 5.2). From *left* to *right*, the model builder window displays the model tree, the settings window presents various model settings, the graphics window displays model geometry, mesh and results, and the information window displays non-graphical model information including solver progress details, error messages and post-processing evaluations. Across the *top* of the interface are various toolbars and menus

various model information including solution progress, solver logs, error messages, as well as the results of post-processing evaluations. Across the top of the interface are various toolbars and menus.

Central to the COMSOL interface is the *Model Tree* displayed in the Model Builder window. The model tree allows all aspects of a model to be specified and adjusted, including the model geometry, physics, equations, mesh and solver settings, as well as visualisation of results. When solving a model, it is useful to regard the model tree as being executed from top to bottom. Thus, settings in higher nodes in the tree will be visible to all subsequent nodes and sub-nodes.

The Model Builder, Settings and Graphics windows are fully-interactive. Thus, clicking on a node in the model tree will display its associated settings in the Settings window, allowing these to be specified. If the node pertains to the model geometry, mesh or results, the Graphics window will also be updated as appropriate. Right-clicking a node in the model tree will create a new sub-node associated with that node. To set model boundary conditions or domain properties, relevant domains, boundaries, edges or points can be specified by selecting these directly from the Graphics window.

Context-sensitive help can be obtained at any time by selecting the help button (?) at the top of the COMSOL interface. COMSOL also provides an extensive *Model*

Library containing a range of models with step by step instructions for implementation.

B.1.2 Specifying Models

COMSOL provides a series of tools and interfaces for implementing models from scratch, including the Model Wizard, geometry tools, physics and user-equation interfaces, mesh and solver settings, parameters, variables and model couplings, as well as post-processing analysis and visualisation.

B.1.2.1 The Model Wizard

The Model Wizard provides for rapid configuration of new models, and is accessed from the COMSOL start-up screen (or from the File|New menu) as shown in Fig. B.2. Clicking Model Wizard will bring up the Select Space Dimension panel, allowing a choice of 3D, 2D, 1D, 0D, as well as 2D and 1D axisymmetric space dimensions. Selecting the space dimension will then open the Select Physics panel, from which a number of physics interfaces can be added to the model, including the Mathematics

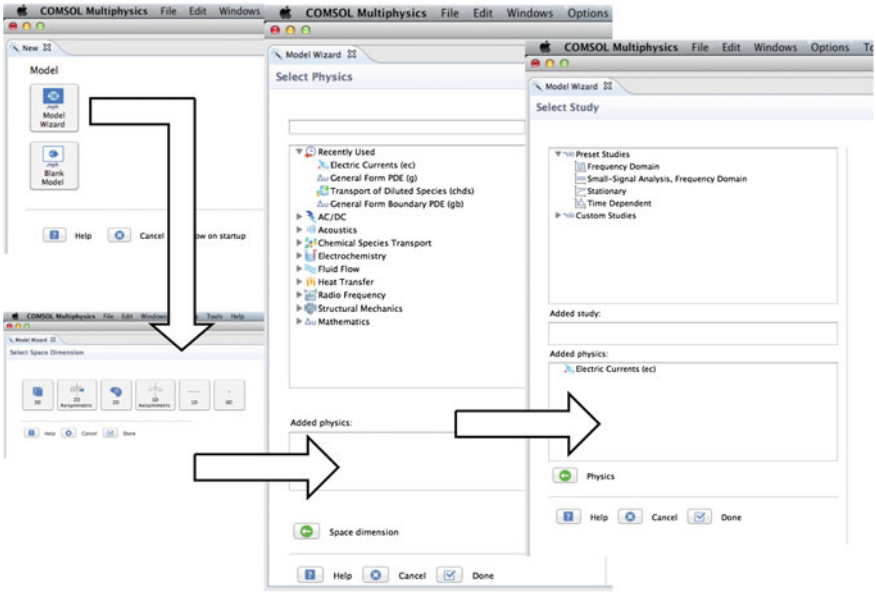


Fig. B.2 COMSOL model wizard. Shown at *top left* is the new model startup screen. Clicking model wizard will in turn bring up select space dimension, select physics and select study panels

interface for specifying user-defined equations. The list of physics interfaces displayed will depend on which optional COMSOL modules have been installed. Interfaces can be added to the model by clicking the "Add" button. Additional physics interfaces can also be added later from the model tree.

Once the required physics interfaces have been added, clicking the Study forward arrow button ( ) will open the Select Study panel for specifying a default study (i.e. solver) for the model. Depending on the physics interface(s) selected, the choice of solver can include Stationary, Time Dependent or Frequency Domain. Additional studies can also be added later to the same model. Clicking "Done" will exit the Model Wizard and display the main COMSOL interface with model tree configured according to the specified Model Wizard settings.

B.1.2.2 Creating a Geometry

Once the model tree has been initialised, the model geometry can be specified by right-clicking on the geometry node. Depending on the space dimension, the geometry interface allows basic geometric objects, known as *primitives*, to be defined and added to the model. Figure B.3(left) illustates some of the 3D geometric primitives available on right clicking the geometry node: analogous interfaces are available for other spatial dimensions. 3D primitives include blocks, cones, cylinders, spheres, ellipsoids, toroids, as well as parametric surfaces. On selecting a primitive, its dimensions and position can be specified from within the Settings window. Clicking the Build Selected ( ) or Build All Objects ( ) buttons in the Settings window will build the object, display the resulting geometry in the Graphics window. To automatically adjust the zoom and view the entire geometry, click the Zoom Extents button ( ) in the Graphics window.

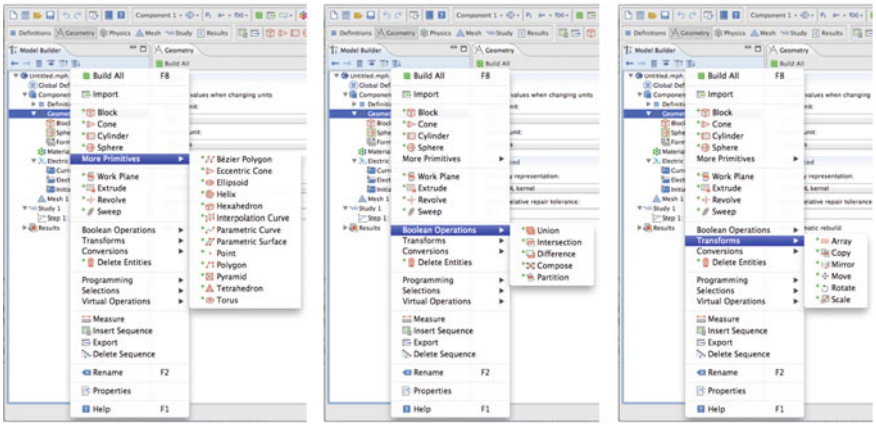


Fig. B.3 COMSOL 3D geometry tools available from the model tree geometry node. From *left* to *right* shows the geometry submenus for more primitives, Boolean operations and transforms

By right-clicking the geometry node, it is also possible to perform Boolean operations on geometric primitives, including subtracting objects from each other, forming unions and intersections, as well as custom combinations of these (Fig. B.3, middle). It is also possible to transform objects by moving, scaling, rotating, mirroring or copying (Fig. B.3, right). Geometric objects for Boolean operations and transformations can be selected using the Graphics window.

In 3D, it is also possible to define a *Work Plane*, a 2D plane embedded in the 3D geometry, again by right-clicking the geometry node. 2D primitives can be specified by right-clicking the associated plane geometry sub-node. These 2D objects can then be revolved or extruded into the 3D geometry.

Using a combination of COMSOL's in-built geometry tools, it is possible to construct fairly complex objects and shapes. For specifying more complex geometries, it is also possible to import CAD, STL and VRML files.

All geometric primitives, operations and transforms appear as geometry sub-nodes in the model tree. It is possible to click on an existing node in any order and modify its settings. Clicking the "Build All Objects" button in the Settings window will then update the geometry.

B.1.2.3 User-Defined Parameters, Functions and Variables

Right-clicking on the Global Definitions node in the model tree, just below the root node, and selecting Parameters allows global parameters to be defined, as shown in Fig. B.4. These parameters can be accessed and used anywhere in the model, including within the geometry settings (e.g. for specifying the size of objects). Parameters are entered in the Settings window by specifying their name, numerical value, and an optional brief description. It is also possible to enter expressions for values using Matlab-type syntax (without the semicolon), provided these

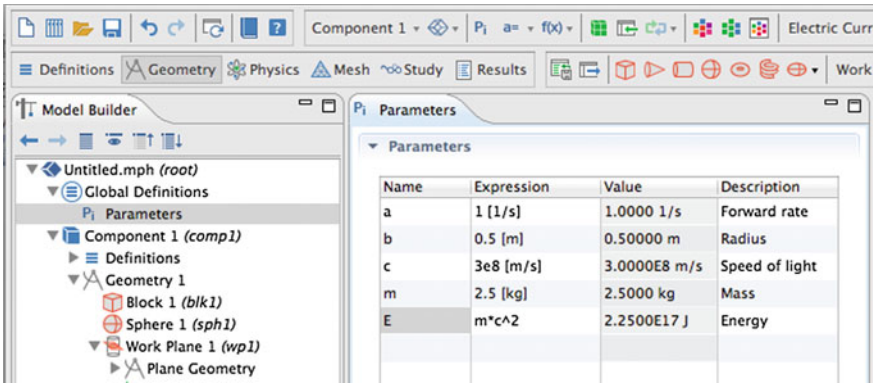


Fig. B.4 COMSOL global parameters interface

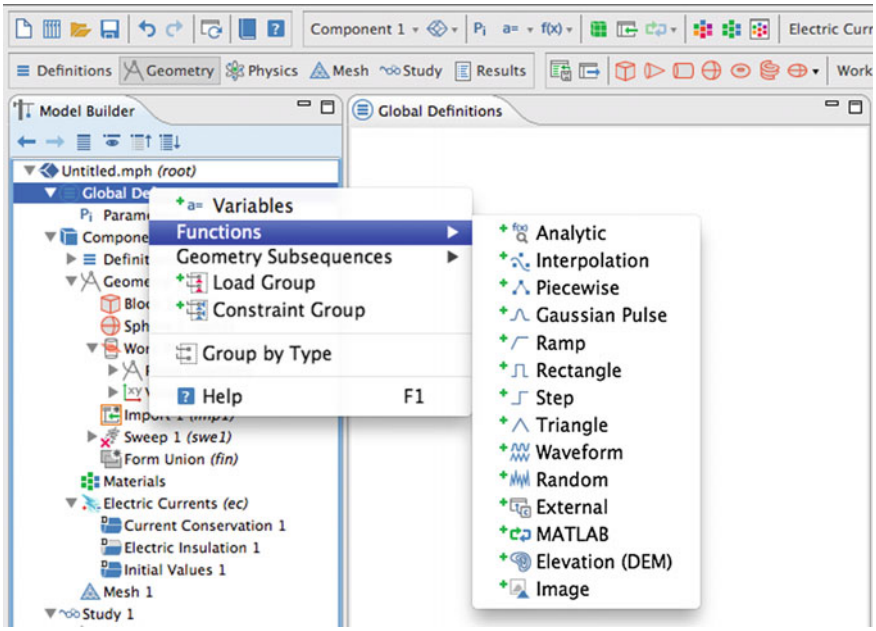


Fig. B.5 COMSOL user-defined function options

expressions yield constants. Examples, of allowable parameter expressions are `sqrt(2)`, `a*sin(pi/3)`, `a/b` etc., where `a` and `b` are themselves parameters.

COMSOL allows physical units for parameters to be specified using square brackets following the parameter value, as seen in Fig. B.4. COMSOL's units feature is very convenient, particularly for bioengineering models which typically involve multiple physical units. COMSOL supports standard SI unit nomenclature as well as other units, in addition to prefixes such as nano-, micro-, milli-, kilo-, or mega-. Thus for example, a value of 1 km would be expressed as `1[km]`, and 2 mA would be written as `2[mA]`. For expressions involving other terms with units, COMSOL automatically determines the unit of the dependent quantity. For example, a parameter value expression of `(2[m])*(1[1/s])` would have units of ms^{-1}.

In addition to parameters, user-defined functions can also be specified, with a number of function type options, as shown in Fig. B.5. These are accessible from anywhere in the model, and include Interpolation functions which interpolate a set of supplied data values, as well as Rectangle and Step functions. To specify a rectangle function for example, select this option and enter a function name along with upper and lower limits from the Settings window. The function will output a value equal to 1 if its argument lies between these limits, or 0 otherwise. The interface also allows the user to enter a 'smoothing factor', which defines the length of a transition zone region for continuous change from 0 to 1. This is a useful feature, since discontinuous function values can lead to model convergence issues. For user-defined functions,

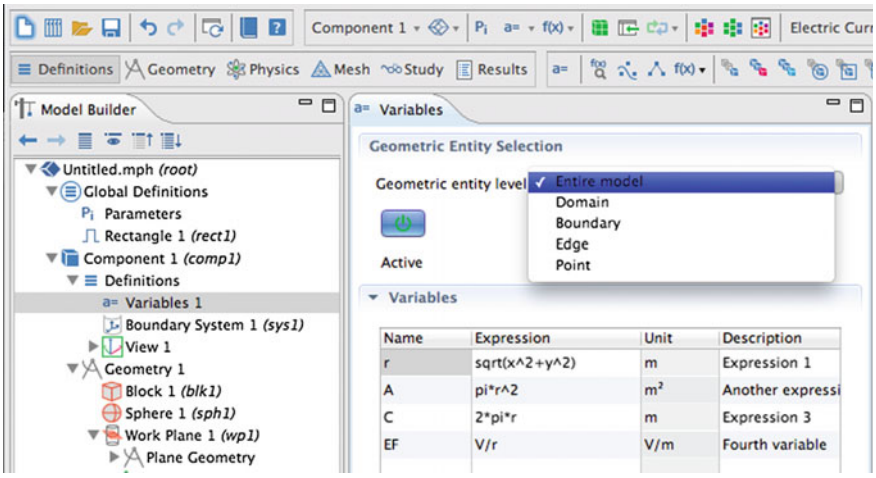


Fig. B.6 COMSOL user-defined variables interface. In the *top* part of the settings window, a geometric entity level for the variables can be specified

COMSOL assumes their inputs and outputs are dimensionless. The only exception is the Interpolation function type, where units for inputs and outputs can be specified.

When specifying physics interfaces, COMSOL will assign default names to the associated dependent variables. Thus, for example, the default variable for voltage in the AC/DC physics interface is V. These variable names can be overridden by the user. To specify additional user-defined variables, right-click the Definitions sub-node of the Component node in the model tree (typically named "Component 1") and select the Variables option.[3] This will create a table in the settings window, where new variables can be defined, as shown in Fig. B.6. Similar to the parameters interface, Matlab-like expressions can be entered to define values. Unlike parameters, however, the expressions don't have to yield constant values, but can include other variables that vary in space and/or time. Furthermore, user-defined variables can be associated with a *geometric entity level* to specify their *scope*. Depending on the spatial dimension of the model, this scope can include the entire model, specific domains, boundaries, edges or points. Multiple variable sub-nodes can be created within the Definitions node, to group variables into different scopes.

When naming user-defined parameters and variables, COMSOL is case-sensitive. A number of reserved names should be avoided, such as those shown in Table B.1.

[3] It is also possible to define variables in the Global Definitions node, but these can only be of global scope, unlike variables defined within a component node.

Table B.1 Examples of reserved COMSOL variables and parameters. Depending on the spatial dimension of the model and the study type, not all of these may be reserved in a given model. Variable `u` is a generic dependent variable defined in a physics interface, and should be replaced with the actual variable name

Name	Description	Name	Description
`t`	Time	`ut`	$\partial u/\partial t$
`x`, `y`, `z`, `r`, `X`, `Y`, `Z`, `R`	Position	`ux`, `uy`, `uz`	$\partial u/\partial x$, $\partial u/\partial y$, $\partial u/\partial z$
`freq`	Frequency	`utt`	$\partial^2 u/\partial t^2$
`lambda`	Eigenvalues	`uxx`, `uyy`, `uzz`	$\partial^2 u/\partial x^2$, $\partial^2 u/\partial y^2$, $\partial^2 u/\partial z^2$
`phase`	Phase angle	`uxy`, `uyz`, ... etc.	$\partial^2 u/\partial x \partial y$, $\partial^2 u/\partial y \partial z$, ...
`pi`	π (≈ 3.14159)	`uxt`, `uyt`, `uzt`	$\partial^2 u/\partial x \partial t$, $\partial^2 u/\partial y \partial t$, $\partial^2 u/\partial z \partial t$
`i`, `j`	$\sqrt{-1}$	`uxxt`, `uxyt`, ... etc.	$\partial^3 u/\partial^2 x \partial t$, $\partial^3 u/\partial x \partial y \partial t$, ...
`h`	Mesh size	`uxxtt`, `uxytt`, ...	$\partial^4 u/\partial^2 x \partial^2 t$, $\partial^4 u/\partial x \partial y \partial^2 t$, ...
`nx`, `ny`, `nz`	Normal vector components	`uTx`, `uTy`, `uTz`	Tangential derivatives

B.1.2.4 Assigning Materials

Right-clicking on the Materials node in the model tree allows the optional selection, definition and assignment of various physical materials to parts of the model. The materials include physical parameters utilised by COMSOL's physics interfaces such electrical conductivity, density, Young's modulus etc. To add a material from COMSOL's in-built material libraries, right-click the Materials node and select Add Material. Once the material is selected, click Add to Component to insert that material as a sub-node of the Materials node. The domains of the model in which the material is active can then be selected from the Graphics window, as shown in Fig. B.7.

It is not necessary to explicitly define materials in a model: material constants can manually be inserted into the appropriate physics settings by simply entering user-defined values. The latter can include global parameters defined under the Global Definitions node.

B.1.2.5 Physics and User-Defined Equation Settings

Physics interfaces added during the Model Wizard are visible as nodes in the model tree. If desired, additional physics nodes can be inserted by right-clicking on the Component node in the model tree (typically named "Component 1") and selecting "Add Physics". Right-clicking a physics node will then bring-up all available settings for that interface, including domain settings and boundary conditions. Figure B.8 lists

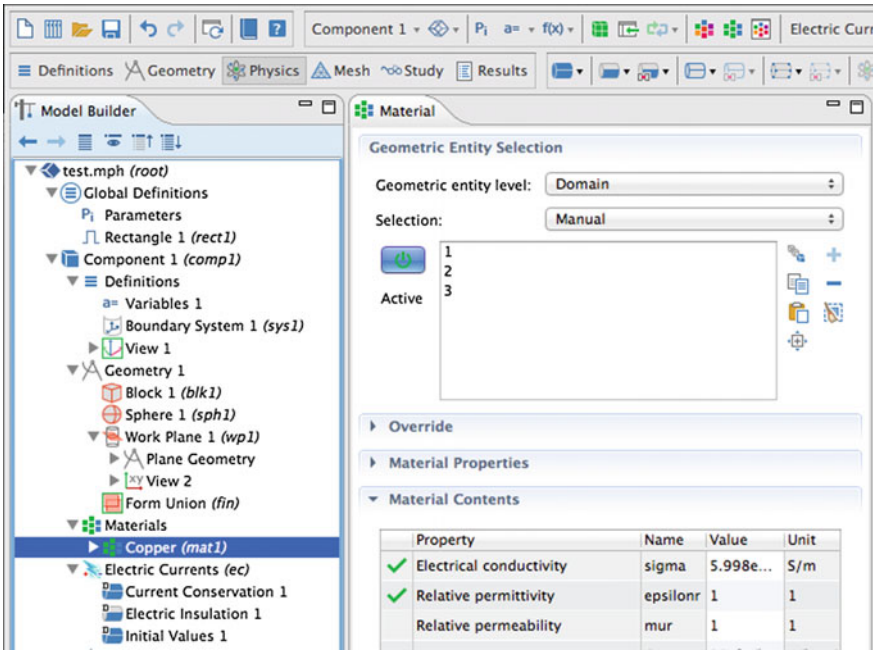


Fig. B.7 COMSOL materials interface. Added materials can be assigned to various parts of a model, and include material properties used by the physics interfaces

the options available when right-clicking the Electric Currents physics node, part of the AC/DC interface.

Physics options are mainly grouped into domain settings (shown for 3D with a solid-filled shape icon) and boundary conditions (shown for 3D as a boundary patch). Selecting any of these will allow specific settings to be entered in the Settings window. This includes specifying regions where that physics setting is applicable. Each selected setting will appear in the model tree as a sub-node of that physics node. Settings can be modified at any time by returning to that node.

To enter user-defined model equations, COMSOL provides several Mathematics interfaces for specifying a number of equation types, including the *Coefficient Form* and *General Form* PDE interfaces. The general form PDE is represented as

$$e_a\left(\frac{\partial^2 u}{\partial t^2}\right) + d_a\left(\frac{\partial u}{\partial t}\right) + \nabla \cdot \mathbf{\Gamma} = f$$

where u is the dependent variable, e_a is the mass coefficient, d_a is the damping coefficient, f is the source term, and $\mathbf{\Gamma}$ is the conservative flux associated with u. By default, as shown in Fig. B.9, $\mathbf{\Gamma}$ is set to the negative gradient of u, $-\nabla u$, with components (in 3D) given by:

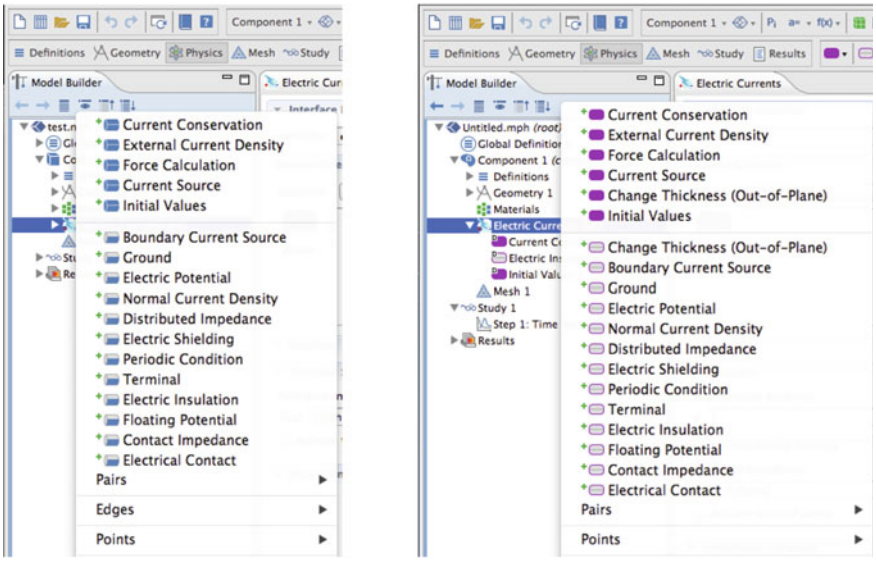


Fig. B.8 Example COMSOL physics options, in this case made available by right-clicking the electric currents node for 3D (*left*) and 2D (*right*) models. Physics options are mainly grouped into domain settings (upper options with solid-filled shape icons) and boundary conditions (lower options with highlighted boundary patch or edge)

$$\boldsymbol{\Gamma} = \begin{pmatrix} -\partial u/\partial x \\ -\partial u/\partial y \\ -\partial u/\partial z \end{pmatrix} = \begin{pmatrix} -\mathrm{ux} \\ -\mathrm{uy} \\ -\mathrm{uz} \end{pmatrix}$$

where the rightmost column-vector is written using COMSOL notation (see Table B.1). The e_a, d_a, f, and $\boldsymbol{\Gamma}$ terms can be modified by the user as required. Note that u can be replaced by any other variable name as required. Furthermore, u can consist of several variables, in which case both u and f are replaced by column-arrays, and e_a, d_a and $\boldsymbol{\Gamma}$ are matrices (the interface is updated accordingly).

As with the other physics interfaces, a range of domain settings and boundary conditions can be specified for the general PDE form. It is also possible (and recommended) to assign physical units to the dependant variable u and source term f.

B.1.2.6 Component Couplings

Right-clicking the component definitions sub-node allows a range of component couplings to be defined, as shown in Fig. B.10. Coupling operators evaluate expressions or integrals over one part of a model (i.e. component), and make these available globally, or to another part of the model. The integration coupling operators, for example,

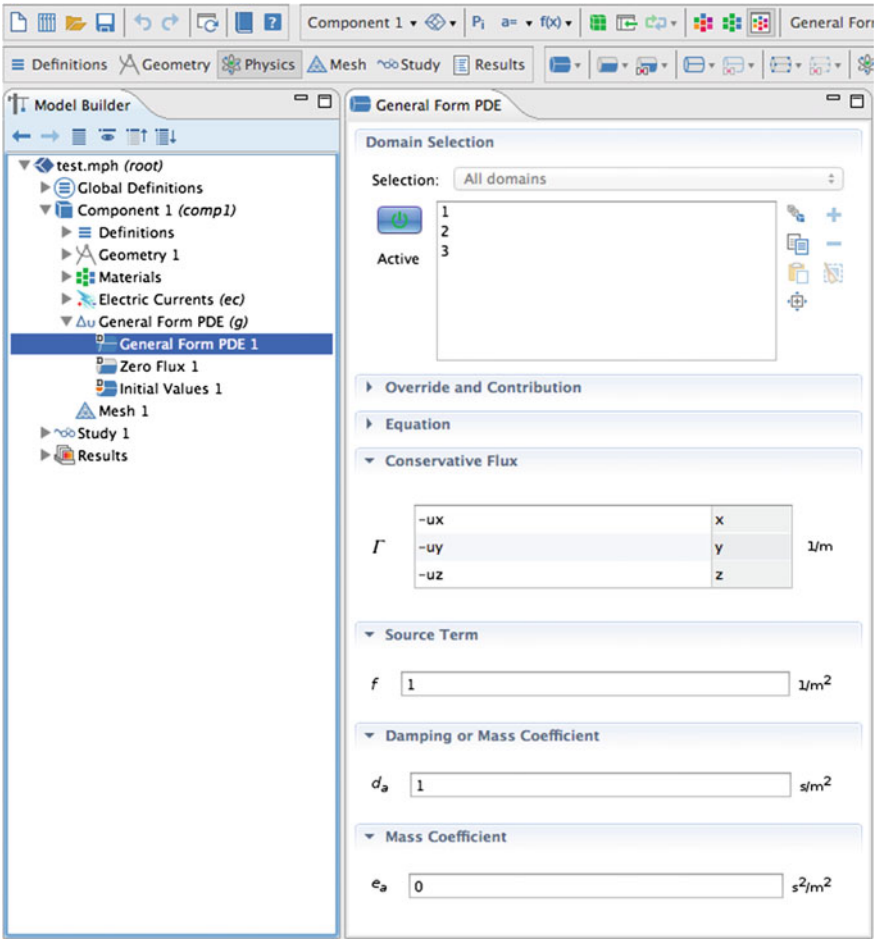


Fig. B.9 COMSOL general form PDE interface. Right-clicking the general form domain setting displays the general form terms in the settings window, which can be modified by the user

specify integration over one or more domains, boundaries, edges or points, and once defined, can be used in any COMSOL expression. Integration of an expression over a point simply returns the value at that point, and is useful for making pointwise values globally-available to other expressions.

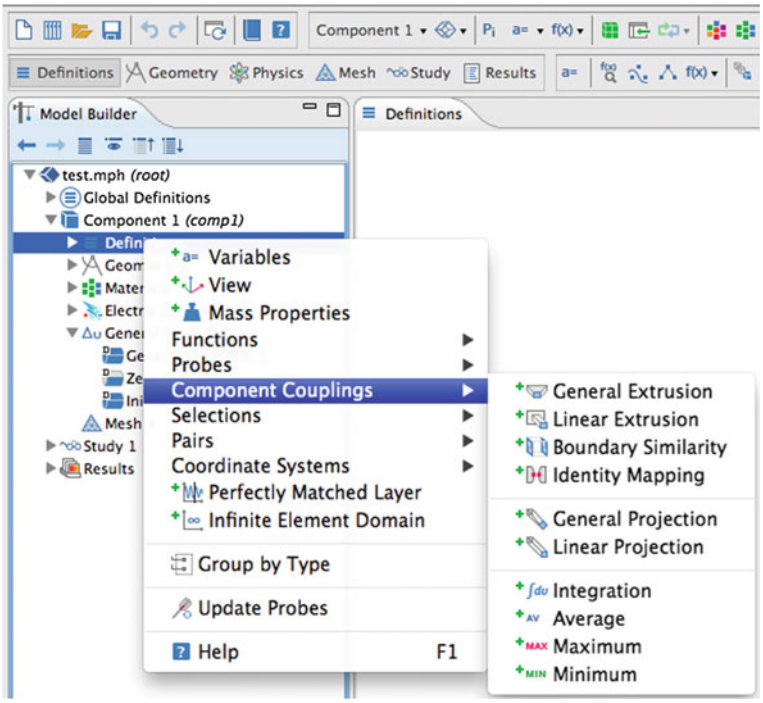


Fig. B.10 COMSOL component coupling options. These define coupling and integration operators that link parts of a model together or assign the evaluations to global scope

B.1.3 Solving and Visualisation

B.1.3.1 Mesh Settings

In order to solve a PDE model, COMSOL automatically generates a finite element mesh to spatially discretize the geometry. It consists of freely-generated tetrahedral elements (3D) or triangular elements (2D) according to default settings, but it is also possible to specify additional settings to mesh, for example, more finely over a given region or boundary. Mesh settings can be specified by right-clicking the mesh node and sub-nodes in the model tree, as shown in Fig. B.11. For example, to globally refine the mesh everywhere, select Size and choose from a number of predefined sizes including 'Fine', 'Finer', and 'Extra Fine'. To specify a custom mesh size over part of a model, choose the appropriate geometric entity level (e.g. edge, boundary, or entire model), select the region of interest, then choose "Custom" in the mesh size Settings window. This allows custom sizing to be applied to that part of the model. After custom sizes have been specified, select the "Free Tetrahedral" option to freely mesh the remaining geometry. Finally, click the Build All button ( ) to build and visualise the mesh in the Graphics window.

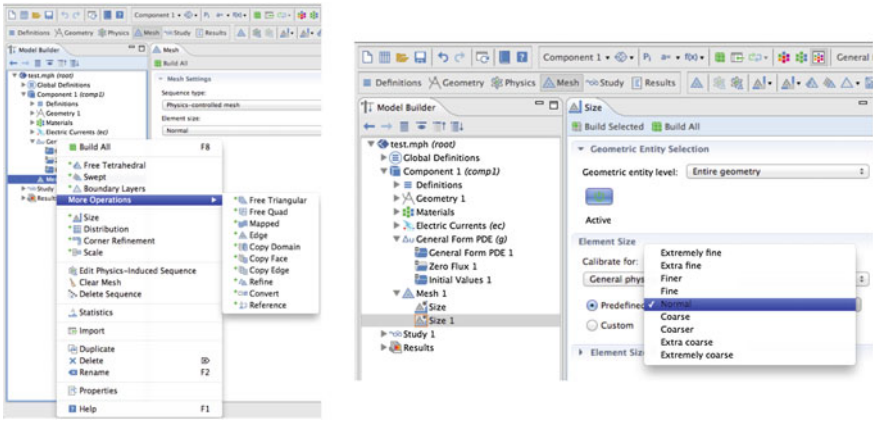


Fig. B.11 COMSOL meshing options. *Left* mesh options available on right-clicking the Mesh node of the model tree (typically named "Mesh 1"). *Right* selecting "size" provides options for predefined mesh sizes, as well as custom size settings

B.1.3.2 Solver Settings

Solver settings can be modified from the Study node in the model tree. Right clicking this node and selecting Show Default Solver will display basic solver settings which can be adjusted by the user (see Fig. B.12). For a time-dependent solver, for example, these settings include the output time steps as well as time stepping behaviour.

Once the solver settings have been specified, right-clicking the study node and selecting Compute (=) will solve the model. Solution progress can be examined in the Progress Information window.

COMSOL also allows parameter sweeps which generate multiple solutions for various values of one or more global parameters. Simply right-click the study node and select "Parametric Sweep" to specify the parameter(s) and their values. This feature is very useful and can be used to generate, for example, a mesh analysis plot by specifying a mesh size global parameter, assigning the maximum mesh size to this parameter, and then performing a parameter sweep to examine how the solution changes with mesh size.

B.1.3.3 Visualisation of Results

On solving a model, COMSOL will automatically generate a plot of the primary dependent variable in the Graphics window. Depending on the spatial dimension of the model, the default plot is either a multislice plot (3D model), a surface plot (2D model), or a line plot (1D model). Right-clicking the Results node in the model tree allows additional plots to be defined, including arrow and streamline plots, contour

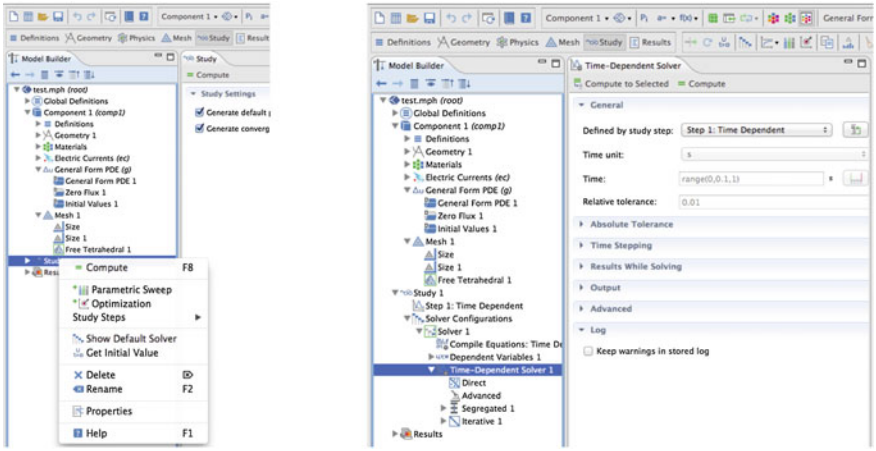


Fig. B.12 COMSOL solver options. *Left* options available on right-clicking the study node (typically named "study 1"). *Right* selecting "show default solver" provides options for the solver. Including time stepping behaviour and absolute tolerance

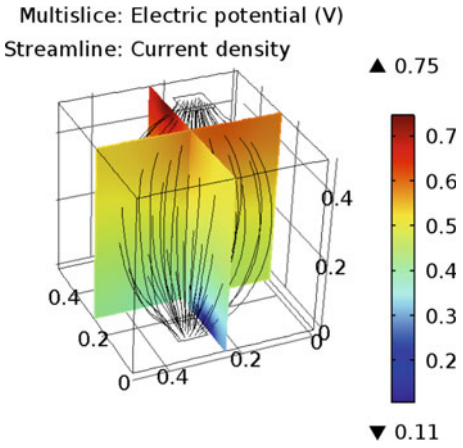


Fig. B.13 Example of COMSOL multislice plot combined with streamline plot. The plot shows the voltage distribution in a cubic volume conductor of sidelength 0.5 m. There are two electrodes at the *top* and *bottom* boundaries, with one electrode held at ground (0 V) and the other at a fixed potential (1 V). The two slices and colourbar show the electric potential (in V), and the streamlines show the direction of current flow

plots, as well as surface and line plots. It is possible to combine multiple plot types together into a single plot, as shown in Fig. B.13 for a slice and streamline plot.

On right-clicking the Results node, options may appear for 3D, 2D and 1D plot groups. Selecting one of these will create the appropriate sub-node in Results. Right-clicking on these sub-nodes provides further plot choices appropriate to the plot group. COMSOL allows for lower-dimensional plots than that of the model space-dimension. This allows, for example, plots along edges or surfaces in a 3D model. Using the 1D plot group, it is also possible to plot global variables or expressions against time or against a global parameter following a parametric sweep.

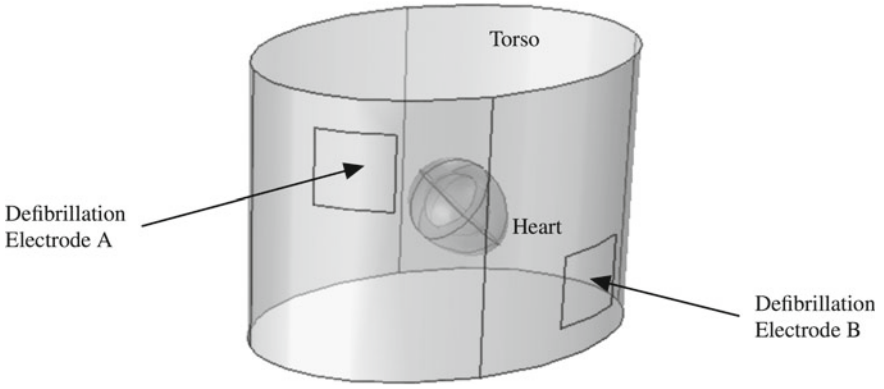


Fig. B.14 Cardiac defibrillation model geometry. The torso is an elliptic cylinder of height 300 mm, with two defibrillation electrodes, A and B, on its surface. The heart is embedded within the torso, and is electrically-active

B.2 Example Model: Cardiac Defibrillation

To illustrate many of COMSOL's features, this section will present a fully-implemented model of cardiac defibrillation, in which abnormal reentrant activation of the heart is "reset" by an external current applied to electrodes on the wall of the chest. The heart, torso and defibrillating electrodes are represented using the idealised geometry shown in Fig. B.14.

Outside the walls of the heart, the electric potential (V) is governed by

$$\nabla \cdot (-\sigma_b \nabla V) = 0$$

where σ_b is the electrical conductivity of the torso. Within the walls of the heart, extracellular (V_e) and intracellular (V_i) potentials are defined at every point using the *bidomain* formulation, coupled with modified Fitzhugh–Nagumo kinetics for the electrically-active tissue:

$$\beta C_m \left(\frac{\partial V_e}{\partial t} - \frac{\partial V_i}{\partial t} \right) + \nabla \cdot (-\sigma_e \nabla V_e) = \beta i_{ion}$$
$$\beta C_m \left(\frac{\partial V_i}{\partial t} - \frac{\partial V_e}{\partial t} \right) + \nabla \cdot (-\sigma_i \nabla V_i) = -\beta i_{ion}$$

with

$$i_{ion} = c_1 (V_m - a)(V_m - A)(V_m - B) + c_2 u (V_m - B)$$
$$V_m = V_i - V_e$$
$$\frac{\partial u}{\partial t} = e (V_m - du - b)$$

Table B.2 Parameter values of cardiac defibrillation model

Parameter	Value	Parameter	Value
A	55 mV	c_1	53 nS mV^{-2} cm^{-2}
B	–85 mV	c_2	400 μS cm^{-2}
a	–66.8 mV	C_m	1 μF
b	–85 mV	σ_e	0.02 S m^{-1}
d	140 mV	σ_i	0.008 S m^{-1}
e	285.7 V^{-1} s^{-1}	β	100 m^{-1}
σ_b	0.2 S m^{-1}	T_{ON}	840 ms
I_d	20 mA	T_{DUR}	10 ms

where u is an auxiliary 'recovery' variable, σ_e and σ_i are the extracellular and intracellular electrical conductivities within the heart, β is the surface to volume ratio, C_m is cell membrane capacitance per unit area, i_{ion} is the ionic current per unit cell membrane area, and A, B, a, b, d, e, c_1 and c_2 are parameters describing the active electrical activity of the heart.

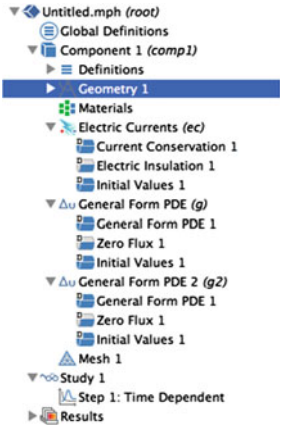
To defibrillate the heart, a rectangular current-pulse of amplitude I_d and duration T_{DUR} is applied to defibrillating electrode A (Fig. B.14) at time $t = T_{ON}$. Defibrillating electrode B is held at ground.

All external boundaries of the torso are electrically-insulating, except at the defibrillating electrodes. At the boundaries of the heart, the extracellular voltage equals the torso potential and the extracellular current density is continuous. For the intracellular potential, the boundaries of the heart are electrically-insulating. All model parameter values are given in Table B.2.

To implement this model in COMSOL, use the following steps:

Model Wizard

1. Open the Model Wizard and select the 3D spatial dimension.
2. In the Select Physics panel, choose AC/DC|Electric Currents. Click "Add".
3. Next, select Mathematics|PDE Interfaces|General Form PDE. Click "Add".
4. In the Review Physics panel at right, specify `U` as the Field name and 2 as the number of dependent variables. In the dependent variables list, enter the names of these variables as `Ve` and `Vi`. For the dependent variable quantity, specify the units as Electric potential (V), and the source term quantity as Current source (A/m^3).
5. Next, select again Mathematics|PDE Interfaces|General Form PDE, and click "Add". This will insert a second General Form PDE into the model. In the Review Physics panel, leave the field name as `u` and the number of dependent variables as 1. Leave the units of the dependent variable as Dimensionless, but enter the source term units manually as `1/s`.
6. Click the Study arrow to open the Select Study panel. Select Time Dependent, and click "Done". This will exit the Model Wizard, displaying the main COMSOL interface. The model tree will look like the following:

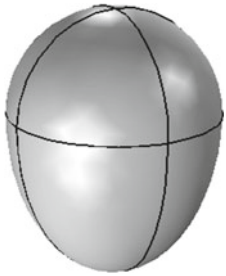


Geometry

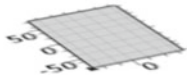
1. Select Geometry 1 in the model tree and specify the length unit as mm.
2. Right-click Geometry 1 and select Sphere. Specify the radius as 45 mm. Click Build Selected ( ).
3. Right-click Geometry 1 and select Sphere again. Specify the radius as 30 mm. Click Build Selected.
4. Right-click Geometry 1 and select Boolean Operations|Difference. In the Objects to add field, select the outer sphere (sph1). In the Objects to subtract field, turn on its Active button and select the inner sphere (sph2). To select this, you may need to hide the outer sphere first by clicking the Select and Hide button( ). This will make the inner sphere visible. Deselect the hide button select the inner sphere. Click the reset hiding button ( ) to make both spheres visible. Clicking Build Selected will subtract the inner sphere from the outer.
5. Next, right-click Geometry 1 and select Block. Specify the width, depth and height as 100, 100 and 50 mm respectively. Specify the corner coordinates as (–50, –50, –50 mm), and click Build Selected.
6. Right-click Geometry 1 again and select Boolean Operations|Difference. Select the hollowed-out sphere (dif1) as the object to add, and select the block (blk1) as the object to substract. Click Build Selected. This will result in the hollowed hemispherical 'shell' shown below:

7. Right-click Geometry 1 again and select More Primitives|Ellipsoid. Specify the a-, b- and c-semiaxes as 30, 30, and 46 mm respectively. Click Build Selected.

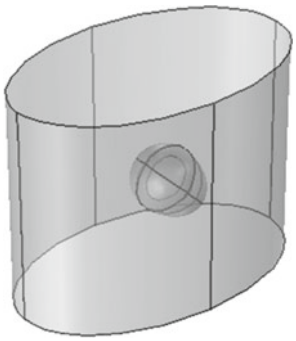
8. Right-click Geometry 1, selecting again More Primitives|Ellipsoid. This time, specify the a-, b- and c-semiaxes as 45, 45, and 69 mm respectively. Click Build Selected.
9. Right-click Geometry 1 and select Block. Specify the width, depth and height to all be 100 mm. Specify the corner of the block to be at (–50, –50, 0 mm). Click Build Selected.
10. Right-click Geometry 1 again and select Boolean Operations|Difference. Select the outer ellipsoid (elp1) as the object to add, and select the block (blk2) as the object to substract. Click Build Selected.
11. Now right-click Geometry 1 and select Boolean Operations|Difference one more time. Select the outer half-ellipsoid (dif3) as the object to add, and select the inner ellipsoid (elp1) as the object to substract. Click Build Selected. This will result in a hollowed-out 'heart' object:



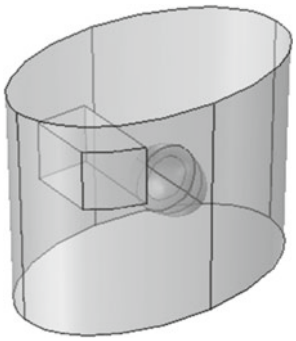
12. Next, rotate the heart by right-clicking Geometry 1 and selecting Transforms |Rotate. Select both halves of the heart (dif2 and dif4) and specify a rotation angle of −45°. Specify the axis of rotation as the y-axis. Click Build Selected. This completes the heart geometry.
13. We now proceed to specify the torso and defibrillating electrodes. Right-click Geometry 1 and select Work Plane. Define a quick xy-plane (the default setting), and specify the z-coordinate as –150 mm. Click Build Selected, followed by the Zoom Extents ( ) button in the Graphics window to view the whole geometry with work plane, as shown below.



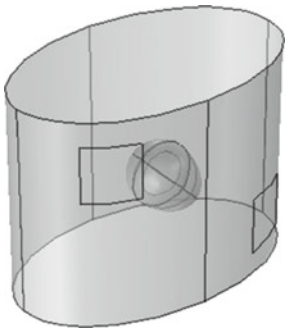
14. Now, right-click on the Plane Geometry subnode of Work Plane 1 and select Ellipse. Specify the a- and b-semiaxes as 200 and 125 mm respectively. Specify the (xw, yw) position of the centre to be (0, 30 mm) and click Build Selected. Click the Zoom Extents ( ) button to see the whole ellipse.
15. Next right-click Geometry 1 and select Extrude. By default, the input object of the extrude operation will be the work plane (wp1). Specify an extrude distance from the plane as 300 mm and click Build Selected. This will extrude the ellipse to construct the torso elliptic cylinder. Click Zoom Extents followed by the Transparency button ( ) to visualise the heart embedded in the torso as shown below.



16. To build the defibrillation electrodes on the torso surface, we specify elongated blocks that intersect with the torso surface. Right-click Geometry 1 and select Block. Specify the width, depth and height to be 70, 150 and 80 mm respectively. Specify the centre (not corner) of the block to be at (–125, –30, 75 mm) and click Build Selected.
17. Next, right-click Geometry 1 and select Transforms|Copy. Select the torso (ext1) and click Build Selected. This creates a copy of the torso. Right-click Geometry 1 and select Boolean Operations|Intersection. Select both the torso (ext1) and the block (blk3) as input objects and click Build Selected. The resulting geometry will look like:



18. Now we repeat this procedure for the other electrode. Right-click Geometry 1 and select Block. Specify the width, depth and height to be 70, 150 and 80 mm respectively. Specify the centre of the block to now be at (125, –30, –75 mm) and click Build Selected.
19. As before, right-click Geometry 1 and select Transforms|Copy. Select the torso (copy1) and click Build Selected. Right-click Geometry 1 and select Boolean Operations|Intersection. Select both the torso (copy1) and the block (blk4) as input objects and click Build Selected.
20. Finally, right-click Geometry 1 and select Boolean Operations|Union. Specify the two intersection regions (int1 and int2) and the torso (copy2) as the three input objects. Deselect the "Keep interior boundaries" checkbox and click Build Selected. The final geometry obtained is shown below:



Global Definitions

1. Right-click Global Definitions and select Parameters. Enter the following details in the Parameters table of the Settings window:

Name	Expression	Description
A	55[mV]	Model parameter
B	-85[mV]	Model parameter
a	-66.8[mV]	Model parameter
b	-85[mV]	Model parameter
d	140[mV]	Model parameter
e	285.7[1/(V*s)]	Model parameter
c_1	53[nS/(mV^2*cm^2]	Model parameter
c_2	400[uS/cm^2]	Model parameter
C_m	1[uF/cm^2]	Membrane capacitance
sigma_e	0.02[S/m]	Extracellular conductivity
sigma_i	0.008[S/m]	Intracellular conductivity
beta	100[1/m]	Surface to volume ratio
sigma_b	0.2[S/m]	Torso bulk conductivity
I	100[mA]	Defibrillation amplitude
T_on	760[ms]	Defibrillation onset
T_dur	100[ms]	Defibrillation duration

2. Right-click Global Definitions and select Functions|Rectangle. Specify the lower limit as `T_on` and the upper limit as `T_on+T_dur`. In the Smoothing tab, specify the size of the transition zone as `T_dur/10`. Leave the function name to its default (rect1).

Component Definitions

1. Right-click the Definitions sub-node of Component 1 and select Component Couplings|Integration. Specify the geometric entity level as 'Boundary' and select boundary 5. This creates an integration operator for integrating expressions over this boundary. Leave the default operator name as intop1.
2. Right-click the Definitions sub-node again and select Variables. Leave the geometric entity level to its default as 'Entire model', and enter the following variables in the settings table:

Name	Expression	Description
`Area`	`intop1(1)`	`Electrode area`
`J_stim`	`(I/Area)*rect1(t[1/s])`	`Current density`

3. Again, right-click Definitions and select Variables to define a new variables group. Specify the geometric entity level as 'Domain' and select domains 2 and 3 corresponding to the heart. It may be easier to use the Select Box button ( ) in the Graphics window to drag a box around the heart to select these domains. Enter the following in the variables settings table:

Name	Expression
`Vm`	`Vi-Ve`
`i_ion`	`c_1*(Vm-a)*(Vm-A)*(Vm-B)+c_2*u*(Vm-B)`

Electric Currents

1. Select the Electric Currents node in the model tree. By default this physics is set to hold in all domains of the model (1–4). We wish to override this setting, since this physics should apply only to the torso and not the heart. Select domains 2 and 3 and click the Remove from Selection button ( ) to individually remove these domains.
2. Expand the Electric Currents node and select the Current Conservation sub-node. In the Settings window, specify the electrical conductivity to be user defined, and enter a value of `sigma_b`. Similarly, specify the relative permittivity to be user defined, and leave the default value of 1.
3. Right-click Electric Currents and select Ground. In the Settings window, select boundary 25 (defibrillating electrode B in Fig. B.14), to set this electrode to ground.

4. Right-click Electric Currents again and select Normal Current Density. Select boundary 5 (defibrillating electrode A in Fig. B.14) and specify a normal current density of `J_stim`.
5. Right-click Electric Currents again and select Normal Current Density. This time, select all the boundaries of the heart by dragging a select box around it. These correspond to boundaries 6, 7, 9–18, 20–23. For the inward normal current density, enter the expression `sigma_e*(Vex*nx+Vey*ny+Vex*nz)` to specify that the current density flowing into the torso from the heart is equal to the current density flowing out of the heart's extracellular domain.

General Form PDE

1. Select the General Form PDE node. Again by default, this PDE is set to apply to all domains of the model (1–4). However, since it should be applicable only within the heart, remove domains 1 and 4 using the Remove from Selection button (➖), leaving only domains 2 and 3.
2. Select the General Form PDE 1 sub-node of General Form PDE, and enter the following expressions for the conservative flux $\mathbf{\Gamma}$ components for variables V_e and V_i respectively:

Component	Expression
x	`-sigma_e*Vex`
y	`-sigma_e*Vey`
z	`-sigma_e*Vez`
x	`-sigma_i*Vix`
y	`-sigma_i*Viy`
z	`-sigma_i*Viz`

Enter the following expressions for the source term f corresponding to variables V_e and V_i respectively:

```
beta*i_ion+stim
-beta*i_ion
```

Finally, enter the following terms for the damping coefficient matrix d_a:

```
beta*C_m      -beta*C_m
-beta*C_m      beta*C_m
```

Leave the mass coefficient matrix e_a entries to their default value of 0.

3. Right-click General Form PDE again and select Dirichlet Boundary Condition. Using the select box, drag a selection around the heart to select all heart boundaries. Remove boundary 8 from the selection, which is the internal boundary between the top and bottom parts of the heart. Deselect the 'Prescribed value of Vi' checkbox, leaving only the 'Prescribed value of Ve' checkbox selected. Enter a value of V in the r_1 field. This constrains the value of Ve at the outer and inner boundaries of the heart to equal the torso potential.

4. Finally, select the Initial Values 1 sub-node of the General Form PDE node. Leave the initial value of Ve to its default value of 0. For Vi however, enter the initial value expression `-0.085 + 0.12*(y<0)*(z>0.02)`. This sets Vi to an initial value of $-0.085 + 0.12 = 0.035$ V for the sector of the heart corresponding to $y < 0$ and $z > 0.02$ m, representing an electrically-excited state. In all other heart regions, Vi is initially set to –0.085 V, corresponding to the resting, non-excited state.

General Form PDE 2

1. Select the General Form PDE 2 node. Select domains 1 and 4 and individually remove these using the Remove from Selection button (), leaving only domains 2 and 3.
2. Select the General Form PDE 1 sub-node of General Form PDE 2, and enter a value of 0 for each of the three components of conservative flux $\mathbf{\Gamma}$, since there are no spatial derivatives in the equation for u. For the source term f, enter the expression `e*(Vm-d*u-b)` and for the damping and mass coefficients d_a and e_a, leave their value as 1 and 0 respectively.
3. Finally, select the Initial Values 1 sub-node of the General Form PDE 2 node, and enter the initial value expression `5*(x>0.03)`. This sets variable u to an initial value of 5 when $x > 0.03$ m, corresponding to heart state that is temporarily inexcitable (i.e. refractory), and 0 elsewhere.

Mesh

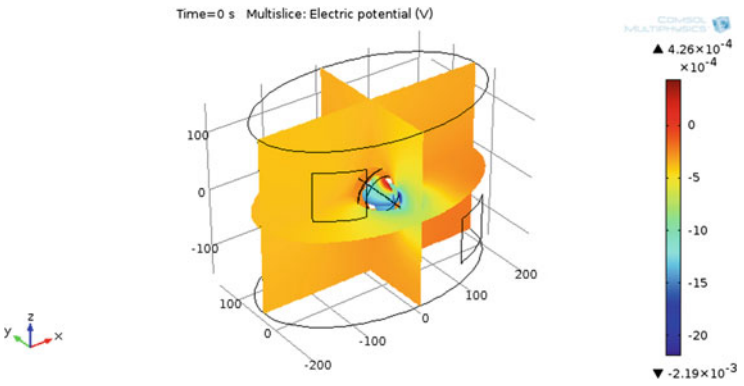
1. Right-click Mesh 1 and select Size. In the Settings window, specify 'Domain' as the geometric entity level, and select domains 2 and 3 corresponding to the heart. Under the Predefined Element Size option, select 'Finer' from the dropdown list.
2. Right-click Mesh 1 again and select Free Tetrahedral. Leave the default geometric entity level as 'Remaining'. This will mesh the remaining parts of the model with a free tetrahedral mesh.
3. Click Build All () in the Settings window to build and display the mesh.

Study

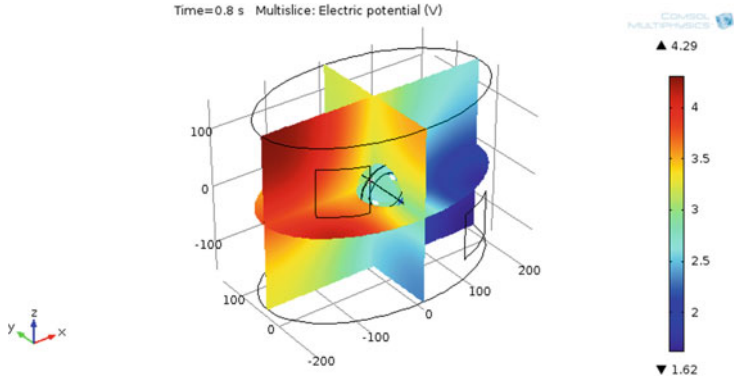
1. Select the Step1: Time Dependent sub-node of the Study 1 node. In the Settings window, Click the Range button ( ) adjacent to the Times field. Leave the entry method as 'Step' and enter Start, Step and Stop values of 0, 0.001 and 2 respectively. Click Replace. This will create a range of output time values from 0 to 2 s in time steps of 0.001 s.
2. Right-click the Study 1 node and select Show Default Solver. Select the Study 1|Solver Configurations|Solution 1|Time-Dependent Solver 1 node. In the Settings window, expand the Advanced tab, and for the 'Singular mass matrix' option, select 'Yes'. Under the Time Stepping tab, select 'Strict' for the Steps taken by solver option.
3. To solve the model, right-click Study 1 and select Compute (). Select the Progress tab in the Information window to see the solver progress.

Results

1. When the model has completed solving,[4] the Graphics window will display a default multislice plot of the torso potential (variable V) at $t = 0$, as shown below:

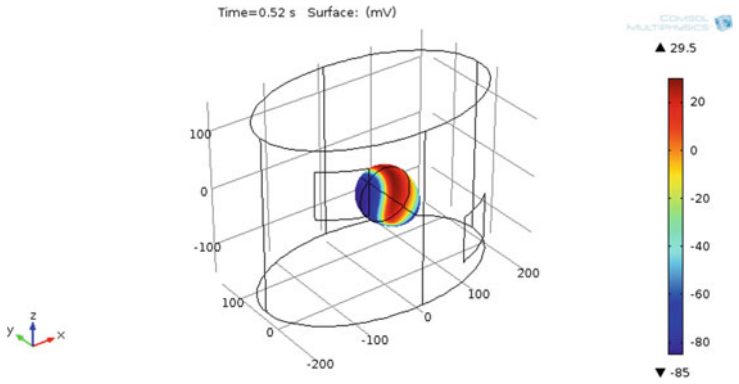


In the Settings window, select a time of 0.8 s from the Time dropdown list and click the Plot button ( ) to display the torso potential at this time, during the defibrillation stimulus:

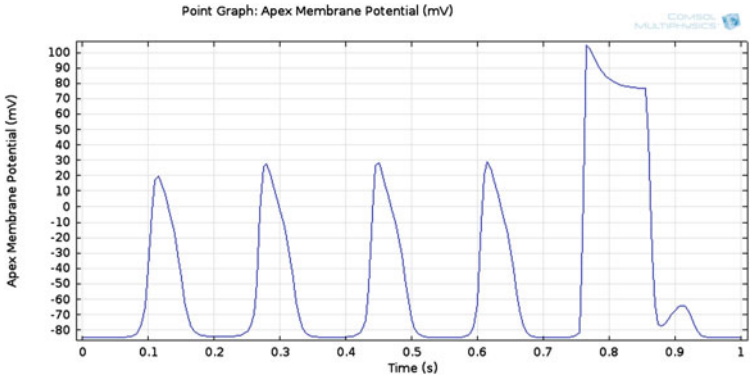


2. Right-click Results and select 3D Plot Group. Right-click this new plot group (3D Plot Group 4) and select Surface. In the Settings window, enter `Vm` as the expression to plot and select mV as the units from the Unit dropdown list. Left-click the 3D Plot Group 4 node again, and select a time of 0.52 s from the Time dropdown list. Clicking the Plot button ( ) will display the membrane potential on the surface of the heart at this time:

[4] Using my MacBook Air laptop with 8 GB RAM and OS X version 10.8.5, it took just under 5 min to solve this model.



3. Right-click Results and select 1D Plot Group. Right-click this new plot group (1D Plot Group 5) and select Point Graph. In the Settings window, enter `Vm` as the expression to plot and select mV as the units. In the Graphics window, select point 22 corresponding to a point at the apex of the heart. Click the Description checkbox, and type 'Apex Membrane Potential' in the Description field. By default, the x-axis data will be the time values. Clicking the Plot button ( ) will display the membrane potential at the apex against time:



This plot shows a repetitive sequence of action potentials (electrical excitations) at the surface of the heart prior to 0.7 s, due to re-entrant activation from a wave of excitation travelling continuously around the heart. Following application of an extracellular current between $t = 0.76$ and 0.86 s through the torso surface electrodes, these periodic self-excitations are terminated, indicating successful defibrillation.

Solutions

Problems of Chap. 1

1.1 (a) Let ϕ_1 and ϕ_2 be two distinct solutions satisfying the 1D diffusion equation, such that $\frac{\partial \phi_1}{\partial t} = D\frac{\partial^2 \phi_1}{\partial x^2}$ and $\frac{\partial \phi_2}{\partial t} = D\frac{\partial^2 \phi_2}{\partial x^2}$. We then form $u = c_1\phi_1 + c_2\phi_2$, to obtain:

$$\begin{aligned}\frac{\partial u}{\partial t} &= c_1\frac{\partial \phi_1}{\partial t} + c_2\frac{\partial \phi_2}{\partial t} \\ &= c_1 D\frac{\partial^2 \phi_1}{\partial x^2} + c_2 D\frac{\partial^2 \phi_2}{\partial x^2} \\ &= D\left[\frac{\partial^2 (c_1\phi_1 + c_2\phi_2)}{\partial x^2}\right] \\ &= D\frac{\partial^2 u}{\partial x^2}\end{aligned}$$

Hence u is also a solution, and the 1D diffusion equation is linear.
(b) Let ϕ_1 and ϕ_2 be two distinct solutions to the equation, and form $u = c_1\phi_1 + c_2\phi_2$, to obtain:

$$\begin{aligned}\frac{\mathrm{d}u}{\mathrm{d}t} &= c_1\frac{\mathrm{d}\phi_1}{\mathrm{d}t} + c_2\frac{\mathrm{d}\phi_2}{\mathrm{d}t} \\ &= c_1 k\phi_1\left(1 - \frac{\phi_1}{N_{max}}\right) + c_2 k\phi_2\left(1 - \frac{\phi_2}{N_{max}}\right) \\ &= k(c_1\phi_1 + c_2\phi_2) - \frac{k}{N_{max}}\left(c_1\phi_1^2 + c_2\phi_2^2\right)\end{aligned}$$

S. Dokos, *Modelling Organs, Tissues, Cells and Devices*,
Lecture Notes in Bioengineering, DOI 10.1007/978-3-642-54801-7

$$
\begin{aligned}
&= ku - \frac{k}{N_{max}}\left(u^2 - u^2 + c_1\phi_1^2 + c_2\phi_2^2\right) \\
&= ku - \frac{ku^2}{N_{max}} + \frac{k}{N_{max}}\left(u^2 + c_1\phi_1^2 + c_2\phi_2^2\right) \\
&= ku\left(1 - \frac{u}{N_{max}}\right) + \frac{k}{N_{max}}\left(u^2 + c_1\phi_1^2 + c_2\phi_2^2\right) \\
&\neq ku\left(1 - \frac{u}{N_{max}}\right)
\end{aligned}
$$

Hence, the equation is non-linear.

1.2 (a) MLT^{-2} (b) L^2T^{-1} (c) $M^{-1}L^{-2}T^4I^2$ (d) $ML^2T^{-3}I^{-2}$ (e) L^3T^{-1} (f) T^{-1} (g) The radian angle measure is defined as the circular arc length subtended divided by the radius. Hence its dimensions are $L/L = 1$, i.e. a dimensionless quantity.

1.3 $[B_0] = NL^{-3}T^{-1}$, $[k_1] = T^{-1}$, $[k_2] = T^{-1}$, $[k_3] = T^{-1}$, $[k_4] = L^3N^{-1}T^{-1}$, $[k_5] = T^{-1}$, $[k_6] = L^3N^{-1}T^{-1}$.

1.4 (a) $[c_0] = ML^{-1}T^{-2}$, SI units: $\mathrm{N\,m^{-2}} = \mathrm{Pa}$, $[c_1] = ML^8T$, SI units: $\mathrm{kg\,m^8\,s}$, $[c_2] = ML^{-1}$, SI units: $\mathrm{Pa\,s^2\,rad^{-2}}$
(b) a: $\mathrm{mV^{-1}\,s^{-1}}$, b: mV, c: mV, A: $\mathrm{s^{-1}}$, B: mV, C: mV.

1.5 (a) $ML^3T^{-3}I^{-2}$
(b) Physical quantities are R_a (access resistance), D and ρ. From these, we form the products

$$
\begin{aligned}
\pi_i &= {R_a}^a D^b \rho^c \\
[\pi_i] &= \left(ML^2T^{-3}I^{-2}\right)^a L^b \left(ML^3T^{-3}I^{-2}\right)^c \\
&= M^{(a+c)} L^{(2a+b+3c)} T^{(-3a-3c)} I^{(-2a-2c)}
\end{aligned}
$$

For these to be dimensionless, we require

$$
\begin{aligned}
a + c &= 0 \\
2a + b + 3c &= 0 \\
-3a - 3c &= 0 \\
-2a - 2c &= 0
\end{aligned}
$$

which has infinitely many solutions of the form

$$
\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \alpha
$$

where α may be freely chosen. Choosing $\alpha = 1$, we have $\pi_1 = R_a D\rho^{-1}$. Hence,

$$R_a = \frac{c\rho}{D}$$

where c is a dimensionless constant.

1.6 We form the dimensionless variables $u^* = u/V$, $x^* = x/L$, $t^* = \omega t$, $p^* = p/(\rho V^2)$. This leads to the scaled equation:

$$\frac{\partial u^*}{\partial t^*} + \left(\frac{V}{\omega L}\right) u^* \frac{\partial u^*}{\partial x^*} = -\left(\frac{V}{\omega L}\right) \frac{\partial p^*}{\partial x^*} + \left(\frac{\mu}{\rho \omega L^2}\right) \frac{\partial^2 u^*}{\partial x^{*2}}$$

with two dimensionless parameters characterising this system:

$$p_1 = \frac{V}{\omega L}$$
$$p_2 = \frac{\mu}{\rho \omega L^2}$$

1.7 The scaled form of the Hodgkin–Huxley equations may be written as:

$$\frac{\mathrm{d}V^*}{\mathrm{d}t^*} = p_1(V^* - 1) + p_2 V^* + p_3(V^* - p_4)$$
$$\frac{\mathrm{d}n}{\mathrm{d}t^*} = \alpha_n^*(1 - n) - \beta_n^* n$$
$$\frac{\mathrm{d}m}{\mathrm{d}t^*} = \alpha_m^*(1 - m) - \beta_m^* m$$
$$\frac{\mathrm{d}h}{\mathrm{d}t^*} = \alpha_h^*(1 - h) - \beta_h^* h$$

with

$$\alpha_n^* = \frac{p_5(V^* + p_6)}{1 - \exp\left[\frac{-(V^* + p_6)}{p_7}\right]} \qquad \beta_n^* = \exp\left[\frac{-(V^* + p_8)}{p_9}\right]$$
$$\alpha_m^* = \frac{p_{10}(V^* + p_{11})}{1 - \exp\left[\frac{-(V^* + p_{11})}{p_{12}}\right]} \qquad \beta_m^* = p_{13}\exp\left[\frac{-(V^* + p_{14})}{p_{15}}\right]$$
$$\alpha_h^* = p_{16}\exp\left[\frac{-(V^* + p_{17})}{p_{18}}\right] \qquad \beta_h^* = \frac{p_{19}}{1 + \exp\left[\frac{-(V^* + p_{20})}{p_{21}}\right]}$$

where 21 parameters characterise the system:

$$
\begin{array}{lll}
p_1 = -\frac{g_{Na}}{CB_n} & p_2 = -\frac{g_K}{CB_n} & p_3 = -\frac{g_L}{CB_n} \\
p_4 = \frac{V_L - V_K}{V_{Na} - V_K} & p_5 = \frac{A_n}{B_n}(V_{Na} - V_K) & p_6 = \frac{V_K + V_{an}}{V_{Na} - V_K} \\
p_7 = S_{an} & p_8 = \frac{V_K + V_{bn}}{V_{Na} - V_K} & p_9 = \frac{s_{bn}}{V_{Na} - V_K} \\
p_{10} = \frac{A_m}{B_n}(V_{Na} - V_K) & p_{11} = \frac{V_K + V_{am}}{V_{Na} - V_K} & p_{12} = \frac{s_{am}}{V_{Na} - V_K} \\
p_{13} = \frac{B_m}{B_n} & p_{14} = \frac{V_K + V_{bm}}{V_{Na} - V_K} & p_{15} = \frac{s_{bm}}{V_{Na} - V_K} \\
p_{16} = \frac{A_h}{B_n} & p_{17} = \frac{V_K + V_{ah}}{V_{Na} - V_K} & p_{18} = \frac{s_{ah}}{V_{Na} - V_K} \\
p_{19} = \frac{B_h}{B_n} & p_{20} = \frac{V_K + V_{bh}}{V_{Na} - V_K} & p_{21} = \frac{s_{bh}}{V_{Na} - V_K}
\end{array}
$$

1.8 Assume there are three compounds U, V, and W in the hydrogel, each of which can enter into a chemical reaction with the analyte to produce distinct complexes that fluoresce with wavelengths λ_1, λ_2 and λ_3 respectively. Furthermore, these reactions are given by:

$$
\begin{aligned}
\mathrm{U} + \mathrm{n_1A} &\underset{k_2}{\overset{k_1}{\rightleftharpoons}} \mathrm{C_1} \\
\mathrm{V} + \mathrm{n_2A} &\underset{k_4}{\overset{k_3}{\rightleftharpoons}} \mathrm{C_2} \\
\mathrm{W} + \mathrm{n_3A} &\underset{k_6}{\overset{k_5}{\rightleftharpoons}} \mathrm{C_3}
\end{aligned}
$$

where n_1, n_2, n_3 are the number of molecules of analyte A needed for each reaction, and $\mathrm{C_1}$, $\mathrm{C_2}$, $\mathrm{C_3}$ are the resultant complexes formed. If the total concentration of each compound/complex is denoted by T_1, T_2 and T_3, then

$$
\begin{aligned}
[\mathrm{U}] + [\mathrm{C_1}] &= T_1 \\
[\mathrm{V}] + [\mathrm{C_2}] &= T_2 \\
[\mathrm{W}] + [\mathrm{C_3}] &= T_3
\end{aligned}
$$

Furthermore, if $\lambda_U, \lambda_V, \lambda_W$ are the fluorescence wavelengths of U, V, and W respectively, then the mean wavelength of the hydrogel-analyte system will be given by:

$$
\begin{aligned}
\bar{\lambda} &= \frac{\lambda_U [\mathrm{U}] + \lambda_1 [\mathrm{C_1}] + \lambda_V [\mathrm{V}] + \lambda_2 [\mathrm{C_2}] + \lambda_W [\mathrm{W}] + \lambda_3 [\mathrm{C_3}]}{T_1 + T_2 + T_3} \\
&= \frac{\lambda_U (T_1 - [\mathrm{C_1}]) + \lambda_V (T_2 - [\mathrm{C_2}]) + \lambda_W (T_3 - [\mathrm{C_3}]) + \lambda_1 [\mathrm{C_1}] + \lambda_2 [\mathrm{C_2}] + \lambda_3 [\mathrm{C_3}]}{T_1 + T_2 + T_3} \\
&= \frac{\lambda_U T_1 + \lambda_V T_2 + \lambda_W T_3 + (\lambda_1 - \lambda_U)[\mathrm{C_1}] + (\lambda_2 - \lambda_V)[\mathrm{C_2}] + (\lambda_3 - \lambda_W)[\mathrm{C_3}]}{T_1 + T_2 + T_3}
\end{aligned}
$$

Denoting the concentration of analyte A with c, the above reaction scheme can be modelled as the following system of differential equations:

$$\frac{\mathrm{d}\,[\mathrm{C}_1]}{\mathrm{d}t} = k_1\,[\mathrm{U}]\,c^{n_1} - k_2\,[\mathrm{C}_1] = k_1\,(T_1 - [\mathrm{C}_1])\,c^{n_1} - k_2\,[\mathrm{C}_1]$$
$$\frac{\mathrm{d}\,[\mathrm{C}_2]}{\mathrm{d}t} = k_3\,[\mathrm{V}]\,c^{n_2} - k_4\,[\mathrm{C}_2] = k_3\,(T_2 - [\mathrm{C}_2])\,c^{n_1} - k_4\,[\mathrm{C}_2]$$
$$\frac{\mathrm{d}\,[\mathrm{C}_3]}{\mathrm{d}t} = k_5\,[\mathrm{W}]\,c^{n_3} - k_6\,[\mathrm{C}_3] = k_5\,(T_3 - [\mathrm{C}_3])\,c^{n_1} - k_6\,[\mathrm{C}_3]$$

The steady-state concentrations of C_1, C_2, C_3 can be found by equating the above derivatives to zero, to obtain:

$$[\mathrm{C}_1] = \frac{T_1 c^{n_1}}{\left[c^{n_1} + \frac{k_2}{k_1}\right]}, \qquad [\mathrm{C}_2] = \frac{T_2 c^{n_2}}{\left[c^{n_2} + \frac{k_4}{k_3}\right]}, \qquad [\mathrm{C}_3] = \frac{T_3 c^{n_3}}{\left[c^{n_3} + \frac{k_6}{k_5}\right]}$$

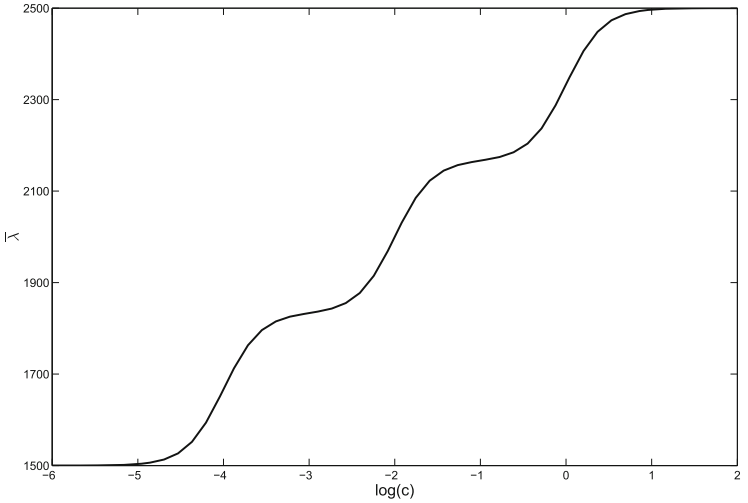
with the steady-state fluorescent wavelength given by:

$$\bar{\lambda} = \lambda_0 + \frac{F_1(\lambda_1 - \lambda_U)c^{n_1}}{\left[c^{n_1} + \frac{k_2}{k_1}\right]} + \frac{F_2(\lambda_2 - \lambda_V)c^{n_2}}{\left[c^{n_2} + \frac{k_4}{k_3}\right]} + \frac{F_3(\lambda_3 - \lambda_W)c^{n_3}}{\left[c^{n_3} + \frac{k_6}{k_5}\right]}$$

where

$$\lambda_0 = \frac{\lambda_U T_1 + \lambda_V T_2 + \lambda_W T_3}{T_1 + T_2 + T_3},\ F_1 = \frac{T_1}{T_1 + T_2 + T_3},\ F_2 = \frac{T_2}{T_1 + T_2 + T_3},$$
$$F_3 = \frac{T_3}{T_1 + T_2 + T_3}$$

Choosing, for example, the following parameter values (all in arbitrary units): $T_1 = T_2 = T_3 = 1$, $n_1 = n_2 = n_3 = 2$, $\lambda_U = 1000$, $\lambda_V = 1500$, $\lambda_W = 2000$, $\lambda_1 = 2000$, $\lambda_2 = 2500$, $\lambda_3 = 3000$, $k_1 = 10^8$, $k_3 = 10^4$ and $k_2 = k_4 = k_5 = k_6 = 1$, the following graph for steady-state mean wavelength against log analyte concentration is readily obtained.



1.9 For a cylindrical axon of length L, radius r, axoplasmic resistivity ρ_i, and membrane resistance r_m, the axon can be discretized into N discrete elements, each of length $\Delta x = L/N$. Values of the R_i and R_m components are then given by:

$$R_i = \frac{\rho_i \Delta x}{\pi r^2}, \qquad R_m = \frac{r_m}{2\pi r \Delta x}$$

Applying Kirchhoff's current law to each node, the following system of equations can be obtained:

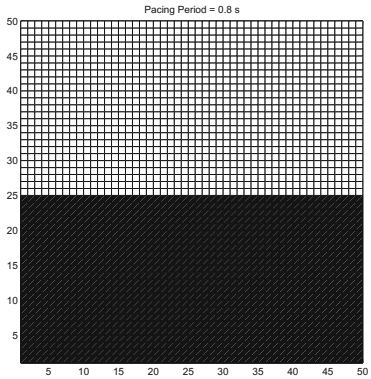
$$\begin{bmatrix} \left(\frac{1}{R_m} + \frac{1}{R_i}\right) & -\frac{1}{R_i} & 0 & \cdots & 0 \\ -\frac{1}{R_i} & \left(\frac{1}{R_m} + \frac{2}{R_i}\right) & -\frac{1}{R_i} & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & & -\frac{1}{R_i} & \left(\frac{1}{R_m} + \frac{1}{R_i}\right) \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_N \end{bmatrix} = \begin{bmatrix} I \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where V_1, V_2, …, V_N are the membrane voltages at the nodes, and I is the current injected into the axon at the first node. Using Matlab's `interp1` interpolation function, the length constant at which the membrane voltage has fallen to $V_1 e^{-1}$ can then be determined. This will depend on the number of nodes N as follows:

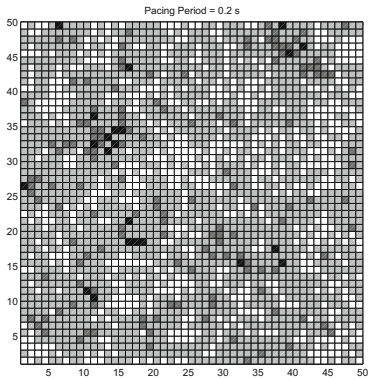
N	Length Constant (mm)
5	2.733
10	2.570
20	2.510
40	2.506
80	2.506
160	2.505

$N = 20$ is sufficient to yield a length constant accuracy of 1%.

1.10 (a) For a heart pacing period of 0.8 s, the plot at $t = 1.65$ s is shown below, where black regions denote state 0 (i.e. quiescent), and white regions denote state 3 (excited):



(b) For a heart pacing period of 0.2 s, the plot at $t = 1.65$ s is shown below, where black regions denote state 0 (i.e. quiescent), dark grey regions denote state 1 (absolute refractory), light grey regions denote state 2 (relative refractory), and white regions denote state 3 (excited):



Some suggestions for implementing this model in Matlab are as follows:

– Define a 50 × 50 array `H_state` to store the current state of each region in the grid. Update the values of the array during each time increment.
– Define a 50 × 50 array `S_duration` to store the time elapsed for the current state in each grid region. Whenever a region changes state, its elapsed time should be reset to zero. Whilst remaining in its current state, the time elapsed should be incremented by `dT` on every time step.
– To plot the grid states, use the command

```
pcolor(H_state), caxis([0 3]), axis('square');
```

– To generate a square array of random total refractory periods, use

```
RP = MRP + SD*randn(size(H_state));
```

– To determine the total number of excited neighbours at any instant, use the code:

```
padded_exc = [zeros(1,52); (H_state(:,50) == 3), ...
(H_state == 3), (H_state(:,1) == 3); zeros(1,52)];

EN = padded_exc(1:50,1:50) + padded_exc(1:50,2:51) + ...
padded_exc(1:50,3:52) + padded_exc(2:51,1:50) + ...
padded_exc(2:51,3:52) + padded_exc(3:52,1:50) + ...
padded_exc(3:52,2:51) + padded_exc(3:52,3:52);
```

This provides an extra "padded" row and column around all edges of `H_state`, allowing the direct summation of the "eight" neighbours.

The full Matlab code listing that generated the above plots (in this instance, for the rapid pacing case) is given below:

```
% Solves a cellular automata model of cardiac electrical
% activation, based on Mitchell et al. (1992), "Cellular
% Automaton Model of Ventricular Fibrillation", IEEE
% Transactions on Biomedical Engineering, 39:253-259.

N = 50;                     % number of cells in each column
M = 50;                     % number of cells in each row
H_state = zeros(N,M);       % state of all cells of the heart
                            % 3 = excited
                            % 2 = absolute refractory
                            % 1 = relative refractory
                            % 0 = quiescent
S_duration =...             % time each cell has spent in its
  zeros(size(H_state));     % current state
Dt = 0.002;                 % time step (in seconds)
t_end = 1.7;                % final time (in seconds)
MRP = 0.25;                 % mean refractory period (in seconds)
SD = 0.1;                   % standard deviation of refractory
                            % period (in seconds)
ES = 0.07;                  % Excited-state duration
RP = MRP + ...              % Total refractory period
SD*randn(size(H_state));    % (abs. + rel.) for each cell (in seconds)
T = 0.2;                    % Heartbeat period (in seconds)

% begin simulation
```

```
for t = 0:Dt:t_end
   % determine number of excited neighbours for each cell
   padded_exc = [zeros(1,M+2); (H_state(:,M) == 3), ...
      (H_state == 3), (H_state(:,1) == 3); zeros(1,M+2)];
   EN = padded_exc(1:N,1:M) + padded_exc(1:N,2:M+1) + ...
      padded_exc(1:N,3:M+2) + padded_exc(2:N+1,1:M) + ...
      padded_exc(2:N+1,3:M+2) + padded_exc(3:N+2,1:M) + ...
      padded_exc(3:N+2,2:M+1) + padded_exc(3:N+2,3:M+2);

   % calculate spread of activation
   for i = 1:M
      for j = 1:N
         if (H_state(i,j) == 0)      % if quiescent
            if (EN(i,j) > 0)
               H_state(i,j) = 3;
               S_duration(i,j) = 0;
            else
               S_duration(i,j) = S_duration(i,j) + Dt;
            end;
         elseif (H_state(i,j) == 3)  % if excited
            if (S_duration(i,j) >= ES)
               H_state(i,j) = 2;
               S_duration(i,j) = 0;
            else
               S_duration(i,j) = S_duration(i,j) + Dt;
            end;
         elseif (H_state(i,j) == 2)  % if absolute refractory
            if (S_duration(i,j) >= RP(i,j)-0.05)
               H_state(i,j) = 1;
               S_duration(i,j) = 0;
            else
               S_duration(i,j) = S_duration(i,j) + Dt;
            end;
         else                        % if relative refractory
            if ((S_duration(i,j) <= 0.002)&&(EN(i,j) == 8))
               H_state(i,j) = 3;
               S_duration(i,j) = 0;
            elseif ((S_duration(i,j) <= 0.004)&&(EN(i,j) >= 7))
               H_state(i,j) = 3;
               S_duration(i,j) = 0;
            elseif ((S_duration(i,j) <= 0.006)&&(EN(i,j) >= 6))
               H_state(i,j) = 3;
               S_duration(i,j) = 0;
            elseif ((S_duration(i,j) <= 0.008)&&(EN(i,j) >= 5))
               H_state(i,j) = 3;
               S_duration(i,j) = 0;
            elseif ((S_duration(i,j) <= 0.012)&&(EN(i,j) >= 4))
               H_state(i,j) = 3;
               S_duration(i,j) = 0;
            elseif ((S_duration(i,j) <= 0.02)&&(EN(i,j) >= 3))
               H_state(i,j) = 3;
               S_duration(i,j) = 0;
            elseif ((S_duration(i,j) <= 0.05)&&(EN(i,j) >= 2))
```

```
                H_state(i,j) = 3;
                S_duration(i,j) = 0;
            elseif ((S_duration(i,j) > 0.05)&&(EN(i,j) >= 1))
                H_state(i,j) = 3;
                S_duration(i,j) = 0;
            elseif (S_duration(i,j) > 0.05)
                H_state(i,j) = 0;
                S_duration(i,j) = 0;
            else
                S_duration(i,j) = S_duration(i,j) + Dt;
            end;
        end;
    end;    % j loop
end;        % i loop
% if necessary, excite the pacemaker cell
if (mod(t,T) < Dt)
    if (H_state(N,round(M/4)) ~= 2)
        H_state(N,round(M/4)) = 3;
        S_duration(N,round(M/4)) = 0;
    end;
end;

% plot states
pause(0.1);
pcolor(H_state), caxis([0 3]), axis('square'), ...
    title('Simulation of Cardiac Electrical Activity ');
end;            % t loop
```

Problems of Chap. 2

2.1 (a) To solve $\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha_x(1-x) - \beta_x x$, we re-write it as the non-homogeneous linear equation:

$$\frac{\mathrm{d}x}{\mathrm{d}t} + (\alpha_x + \beta_x)x = \alpha_x$$

To solve this ODE, we first solve the homogeneous form

$$\frac{\mathrm{d}x}{\mathrm{d}t} + (\alpha_x + \beta_x)x = 0$$

with characteristic equation

$$m + (\alpha_x + \beta_x) = 0$$
$$\therefore \quad m = -(\alpha_x + \beta_x)$$

Hence, the homogeneous solution is

$$x_h = C\mathrm{e}^{-(\alpha_x+\beta_x)t}$$

where C is a constant.
Next, we solve for a particular solution, which we can conveniently take as the steady-state value of x. This steady-state value can be obtained from the original non-homogeneous ODE by simply setting $\frac{\mathrm{d}x}{\mathrm{d}t} = 0$,

$$\Rightarrow \quad (\alpha_x + \beta_x)x = \alpha_x$$

Hence, a particular solution is

$$x_p = \frac{\alpha_x}{\alpha_x + \beta_x}$$

The general solution is the sum of the homogeneous and particular solutions:

$$x = C\mathrm{e}^{-(\alpha_x+\beta_x)t} + \frac{\alpha_x}{\alpha_x + \beta_x}$$

When $t = 0$, $x = x_0$

$$\therefore \quad x_0 = C + \frac{\alpha_x}{\alpha_x + \beta_x}$$
$$\Rightarrow \quad C = x_0 - \frac{\alpha_x}{\alpha_x + \beta_x}$$

Thus, the solution of the ODE is

$$x = \left[x_0 - \frac{\alpha_x}{\alpha_x + \beta_x}\right]\mathrm{e}^{-(\alpha_x+\beta_x)t} + \frac{\alpha_x}{\alpha_x + \beta_x}$$

(b) From above, the steady-state solution, x_∞, is given by

$$x_\infty = \frac{\alpha_x}{\alpha_x + \beta_x}$$

and since α_x, β_x are functions of the membrane potential, which equals V_{clamp} during the voltage-clamp, this steady state value is more accurately expressed as

$$x_\infty = \frac{\alpha_x(V_{clamp})}{\alpha_x(V_{clamp}) + \beta_x(V_{clamp})}$$

Hence, a reasonable estimate for x_0 would be the corresponding steady-state value at the holding potential V_{hold}, prior to the onset of the voltage-clamp step, or

$$x_0 = \frac{\alpha_x(V_{hold})}{\alpha_x(V_{hold}) + \beta_x(V_{hold})}$$

2.2 (a) The total force applied to the muscle is given by

$$F = k_1 x_1 = b\frac{\mathrm{d}x_2}{\mathrm{d}t} + k_2 x_2$$

and since the total length x is held fixed at X_m, then $x_1 = X_m - x_2$. Substituting this value for x_1 into the right-most equality above, we have:

$$b\frac{\mathrm{d}x_2}{\mathrm{d}t} + k_2 x_2 = k_1 [X_m - x_2]$$

$$b\frac{\mathrm{d}x_2}{\mathrm{d}t} + (k_1 + k_2)x_2 = k_1 X_m$$

$$\text{or} \qquad \frac{\mathrm{d}x_2}{\mathrm{d}t} + \left(\frac{k_1 + k_2}{b}\right) x_2 = \frac{k_1}{b} X_m$$

To solve this ODE, the corresponding homogeneous equation is

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} + \left(\frac{k_1 + k_2}{b}\right) x_2 = 0$$

with homogeneous solution

$$x_{2,h} = C\mathrm{e}^{-\left(\frac{k_1+k_2}{b}\right)t}$$

where C is a constant. A particular solution can be found for the steady-state value of x_2 by setting $\frac{\mathrm{d}x_2}{\mathrm{d}t} = 0$ in the non-homogeneous ODE. This yields the particular solution

$$x_{2,p} = \left(\frac{k_1}{k_1 + k_2}\right) X_m$$

Hence, the general solution is the sum of the homogeneous and particular solutions:

$$x_2 = C\mathrm{e}^{-\left(\frac{k_1+k_2}{b}\right)t} + \left(\frac{k_1}{k_1 + k_2}\right) X_m$$

When $t = 0$, $x_2 = 0$, which implies $C = -\left(\frac{k_1}{k_1+k_2}\right) X_m$. Hence

$$x_2 = \left(\frac{k_1}{k_1 + k_2}\right) X_m \left[1 - \mathrm{e}^{-\left(\frac{k_1+k_2}{b}\right)t}\right]$$

Knowing x_2, we can readily obtain x_1:

$$\begin{aligned} x_1 &= X_m - x_2 \\ &= \left(\frac{k_2}{k_1 + k_2}\right) X_m + \left(\frac{k_1}{k_1 + k_2}\right) X_m \mathrm{e}^{-\left(\frac{k_1+k_2}{b}\right)t} \\ &= \left(\frac{X_m}{k_1 + k_2}\right) \left[k_2 + k_1 \mathrm{e}^{-\left(\frac{k_1+k_2}{b}\right)t}\right] \end{aligned}$$

Since $F = k_1 x_1$, the applied force is readily determined as

$$F = \left(\frac{k_1 X_m}{k_1 + k_2}\right)\left[k_2 + k_1 \mathrm{e}^{-\left(\frac{k_1+k_2}{b}\right)t}\right]$$

(b) The fixed applied force, F_m, satisfies

$$F_m = k_1 x_1 = b\frac{\mathrm{d}x_2}{\mathrm{d}t} + k_2 x_2$$

and working with variable x_2, we can rewrite the above as

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} + \frac{k_2}{b}x_2 = \frac{F_m}{b}$$

The homogeneous ODE is

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} + \frac{k_2}{b}x_2 = 0$$

with solution

$$x_{2,h} = C\mathrm{e}^{-\left(\frac{k_2}{b}\right)t}$$

where C is a constant. The particular solution can be found from the steady-state value of x_2 in the non-homogeneous ODE by setting $\frac{\mathrm{d}x_2}{\mathrm{d}t} = 0$. We therefore obtain the particular solution

$$x_{2,p} = \frac{F_m}{k_2}$$

and the general solution

$$x_2 = C\mathrm{e}^{-\left(\frac{k_2}{b}\right)t} + \frac{F_m}{k_2}$$

When $t = 0$, $x_2 = 0$, which implies $C = -\frac{F_m}{k_2}$. Hence

$$x_2 = \frac{F_m}{k_2}\left[1 - \mathrm{e}^{-\left(\frac{k_2}{b}\right)t}\right]$$

Furthermore, since $F_m = k_1 x_1$, we have $x_1 = F_m/k_1$. Hence, the total length of the muscle is given by

$$x = x_1 + x_2 = F_m\left[\frac{1}{k_1} + \frac{1}{k_2} - \frac{1}{k_2}\mathrm{e}^{-\left(\frac{k_2}{b}\right)t}\right]$$

2.3 (a) The total current flowing through the parallel resistive and capacitive branches is equal to the applied stimulus current. Hence, the ODE for this system is given by

$$C\frac{\mathrm{d}V}{\mathrm{d}t} + \frac{V}{R} = I$$

or

$$\frac{\mathrm{d}V}{\mathrm{d}t} + \frac{V}{RC} = \frac{I}{C}$$

The homogeneous ODE for this system is

$$\frac{\mathrm{d}V}{\mathrm{d}t} + \frac{V}{RC} = 0$$

with homogeneous solution $V_h = C\mathrm{e}^{-\frac{t}{RC}}$, where C is a constant. The particular solution, $V_{2,p}$, can be obtained by setting $\frac{\mathrm{d}V}{\mathrm{d}t} = 0$ in the original ODE to obtain:

$$V_{2,p} = IR$$

Hence, the general solution, given by the sum of the homogeneous and particular solutions, is

$$V = IR + C\mathrm{e}^{-\frac{t}{RC}}$$

When $t = 0$, $V = 0$, which implies $C = -IR$. Hence,

$$V = IR\left(1 - \mathrm{e}^{-\frac{t}{RC}}\right)$$

Now, when $V = V_{th}$, the required stimulus duration T must therefore satisfy

$$V_{th} = IR\left(1 - \mathrm{e}^{-\frac{T}{RC}}\right)$$

Hence, the strength-duration characteristic is

$$I = \frac{V_{th}}{R\left(1 - \mathrm{e}^{-\frac{T}{RC}}\right)}$$

(b) From the strength-duration characteristic above, the rheobase may be found from the stimulus current I as $T \to \infty$, or simply

$$I_{rheobase} = \frac{V_{th}}{R}$$

The chronaxie can be found by solving for stimulus duration T corresponding to an applied current of $2I_{rheobase}$. From the previous strength-duration characteristic, we have:

$$
\begin{aligned}
2\frac{V_{th}}{R} &= \frac{V_{th}}{R\left(1-\mathrm{e}^{-\frac{T}{RC}}\right)} \\
2 &= \frac{1}{1-\mathrm{e}^{-\frac{T}{RC}}} \\
1-\mathrm{e}^{-\frac{T}{RC}} &= \frac{1}{2} \\
\mathrm{e}^{-\frac{T}{RC}} &= \frac{1}{2} \\
\frac{T}{RC} &= \ln 2 \\
\therefore \quad T_{chronaxie} &= RC\ln 2 \\
&\approx 0.69RC
\end{aligned}
$$

2.4 (a) The pair of ODEs describing the coupled spring masses is

$$
\begin{aligned}
M\ddot{x_1} &= -kx_1 + k(x_2 - x_1) \\
M\ddot{x_2} &= k(x_1 - x_2) - kx_2
\end{aligned}
$$

or

$$
\begin{aligned}
M\ddot{x_1} &= -2kx_1 + kx_2 \\
M\ddot{x_2} &= kx_1 - 2kx_2
\end{aligned}
$$

(b) Adding the above ODEs together produces

$$
M\ddot{x_1} + M\ddot{x_2} = -kx_1 - kx_2
$$

or simply

$$
M\ddot{y_1} = -ky_1
$$

where we have used the substitution $y_1 = x_1 + x_2$. To solve this ODE, we can write its characteristic equation as

$$
\begin{aligned}
Mm^2 + k &= 0 \\
\Rightarrow \quad m &= \pm i\sqrt{\frac{k}{M}}
\end{aligned}
$$

Hence,

$$
\begin{aligned}
y_1 &= C_1\mathrm{e}^{it\sqrt{\frac{k}{M}}} + C_2\mathrm{e}^{-it\sqrt{\frac{k}{M}}} \\
&= C_1\cos\left(t\sqrt{\frac{k}{M}}\right) + C_1 i\sin\left(t\sqrt{\frac{k}{M}}\right) + C_2\cos\left(t\sqrt{\frac{k}{M}}\right) - C_2 i\sin\left(t\sqrt{\frac{k}{M}}\right)
\end{aligned}
$$

where C_1 and C_2 are constants. Differentiating this expression, we obtain

$$\begin{aligned}\dot{y_1} = &- C_1\sqrt{\frac{k}{M}}\sin\left(t\sqrt{\frac{k}{M}}\right) + C_1\sqrt{\frac{k}{M}}i\cos\left(t\sqrt{\frac{k}{M}}\right)\\ &- C_2\sqrt{\frac{k}{M}}\sin\left(t\sqrt{\frac{k}{M}}\right) - C_2\sqrt{\frac{k}{M}}i\cos\left(t\sqrt{\frac{k}{M}}\right)\end{aligned}$$

When $t = 0$, $y_1 = u_1 + u_2$ and $\dot{y_1} = \dot{x_1} + \dot{x_2} = 0$. Substituting these into the above equations for y_1 and $\dot{y_1}$ yields

$$\begin{aligned}u_1 + u_2 &= C_1 + C_2\\ 0 &= C_1\sqrt{\frac{k}{M}}i - C_2\sqrt{\frac{k}{M}}i\end{aligned}$$

or

$$C_1 = C_2 = \frac{1}{2}(u_1 + u_2)$$

Hence,

$$y_1 = (u_1 + u_2)\cos\left(t\sqrt{\frac{k}{M}}\right)$$

Similarly, we can subtract the two original ODEs in x_1 and x_2 to obtain:

$$M\ddot{x_1} - M\ddot{x_2} = -3kx_1 + 3kx_2$$

or

$$M\ddot{y_2} = -3ky_2$$

where we have used the substitution $y_2 = x_1 - x_2$. The characteristic equation of this ODE is

$$\begin{aligned}Mm^2 + 3k &= 0\\ \Rightarrow \quad m &= \pm i\sqrt{\frac{3k}{M}}\end{aligned}$$

Hence,

$$\begin{aligned}y_2 &= C_3 e^{it\sqrt{\frac{3k}{M}}} + C_4 e^{-it\sqrt{\frac{3k}{M}}}\\ &= C_3\cos\left(t\sqrt{\frac{3k}{M}}\right) + C_3 i\sin\left(t\sqrt{\frac{3k}{M}}\right) + C_4\cos\left(t\sqrt{\frac{3k}{M}}\right) - C_4 i\sin\left(t\sqrt{\frac{3k}{M}}\right)\end{aligned}$$

where C_3 and C_4 are constants. Differentiating this expression, we obtain

$$\dot{y_2} = -C_3\sqrt{\frac{3k}{M}}\sin\left(t\sqrt{\frac{3k}{M}}\right) + C_3\sqrt{\frac{3k}{M}}i\cos\left(t\sqrt{\frac{3k}{M}}\right) - C_4\sqrt{\frac{3k}{M}}\sin\left(t\sqrt{\frac{3k}{M}}\right) - C_4\sqrt{\frac{3k}{M}}i\cos\left(t\sqrt{\frac{3k}{M}}\right)$$

When $t = 0$, $y_2 = u_1 - u_2$ and $\dot{y_2} = \dot{x_1} - \dot{x_2} = 0$. Substituting these into the above equations for y_2 and $\dot{y_2}$ yields

$$u_1 - u_2 = C_3 + C_4$$
$$0 = C_3\sqrt{\frac{3k}{M}}i - C_4\sqrt{\frac{3k}{M}}i$$

or

$$C_3 = C_4 = \frac{1}{2}(u_1 - u_2)$$

Hence,

$$y_2 = (u_1 - u_2)\cos\left(t\sqrt{\frac{3k}{M}}\right)$$

To recover x_1 and x_2 from y_1 and y_2, we can use the expressions

$$x_1 = \frac{1}{2}(y_1 + y_2)$$
$$x_2 = \frac{1}{2}(y_1 - y_2)$$

to finally obtain

$$x_1 = \frac{1}{2}(u_1 + u_2)\cos\left(t\sqrt{\frac{k}{M}}\right) + \frac{1}{2}(u_1 - u_2)\cos\left(t\sqrt{\frac{3k}{M}}\right)$$
$$x_2 = \frac{1}{2}(u_1 + u_2)\cos\left(t\sqrt{\frac{k}{M}}\right) - \frac{1}{2}(u_1 - u_2)\cos\left(t\sqrt{\frac{3k}{M}}\right)$$

2.5 The ODEs for the glucose-insulin model are

$$\frac{\mathrm{d}I}{\mathrm{d}t} = k_2 I_p(t) - k_3 I$$
$$\frac{\mathrm{d}G}{\mathrm{d}t} = B_0 - (k_1 + k_4 I + k_5 + k_6 I)$$

The following Matlab code (GI_solve.m and GI_prime.m) will numerically-integrate these and display the solution:

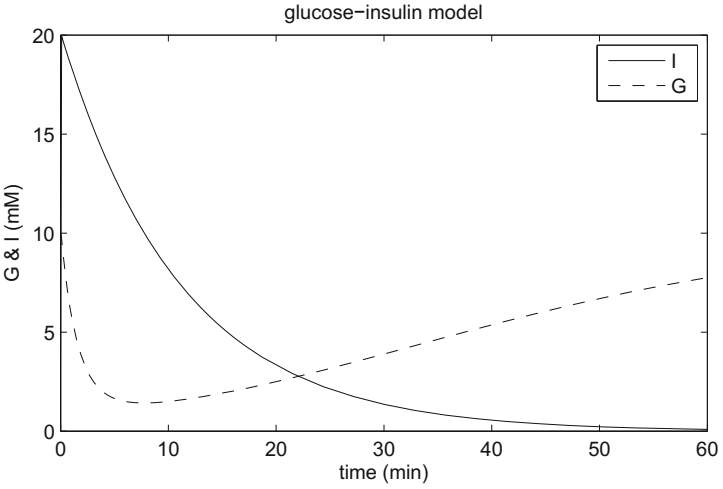
GI_solve.m

```
[time, y_out] = ode15s('GI_prime', [0 60], [0 10]);

plot(time, y_out(:,1),'k', time, y_out(:,2),'k--'), ...
    xlabel ('time (min)'), ylabel('G & I (mM)'), ...
    title('glucose-insulin model'), legend('I', 'G');
```

GI_prime.m

```
function y_prime = GI_prime(t,y)
    y_prime = zeros(2,1);
    k1 = 0.015;
    k2 = 1;
    k3 = 0.09;
    k4 = 0.01;
    k5 = 0.035;
    k6 = 0.02;
    B0 = 0.5;
    I = y(1);
    G = y(2);
    if (t < 0.1)
        Ip = 200;
    else
        Ip = 0;
    end;
    y_prime(1) = k2*Ip - k3*I;
    y_prime(2) = B0 - (k1+k4*I+k5+k6*I)*G;
```

producing the plot:



2.6 (a) Denoting the total outflow from the ventricle as Q, we can write

$$Q = C_s \frac{\mathrm{d}P_s}{\mathrm{d}t} + \frac{P_s}{R_s}$$

To evaluate Q, there are two cases to consider:
(1) $P_v \leq P_s$. In this case, the aortic valve (diode) prevents any backflow and $Q = 0$.
(2) $P_v > P_s$. Denoting the flows through elements R_o and L_o by Q_R and Q_L respectively, we have

$$\begin{aligned} Q &= Q_L + Q_R \\ &= Q_L + \frac{P_v - P_s}{R_o} \end{aligned}$$

Combining both cases above:

$$C_s \frac{\mathrm{d}P_s}{\mathrm{d}t} + \frac{P_s}{R_s} = \begin{cases} Q_L + \frac{P_v - P_s}{R_o} & P_v > P_s \\ 0 & P_v \leq Ps \end{cases}$$

which can be re-arranged to

$$\frac{\mathrm{d}P_s}{\mathrm{d}t} = \begin{cases} \frac{Q_L}{C_s} + \frac{P_v - P_s}{R_o C_s} - \frac{P_s}{R_s C_s} & P_v > P_s \\ -\frac{P_s}{R_s} & P_v \leq P_s \end{cases}$$

To evaluate Q_L, we can consider the same two cases:
(1) $P_v \leq P_s$. In this case, $Q = 0$ and $Q_R = -Q_L$. The pressure drop across L_o is

$$Q_R R_o = -R_o Q_L = L_o \frac{\mathrm{d}Q_L}{\mathrm{d}t}$$

Hence,

$$\frac{\mathrm{d}Q_L}{\mathrm{d}t} = -\frac{R_o}{L_o}Q_L$$

(2) $P_v > P_s$. In this case, the pressure drop across L_o is given by:

$$P_v - P_s = L_o\frac{\mathrm{d}Q_L}{\mathrm{d}t}$$
$$\therefore \quad \frac{\mathrm{d}Q_L}{\mathrm{d}t} = \frac{P_v - P_s}{L_o}$$

Hence, the pair of ODEs characterising the system is

$$\frac{\mathrm{d}P_s}{\mathrm{d}t} = \begin{cases} \frac{Q_L}{C_s} + \frac{P_v - P_s}{R_o C_s} - \frac{P_s}{R_s C_s} & P_v > P_s \\ -\frac{P_s}{R_s} & P_v \leq P_s \end{cases}$$
$$\frac{\mathrm{d}Q_L}{\mathrm{d}t} = \begin{cases} \frac{P_v - P_s}{L_o} & P_v > P_s \\ -\frac{R_o}{L_o}Q_L & P_v \leq P_s \end{cases}$$

(b) The following Matlab code (Windkessel_solve.m and Windkessel_prime.m) will solve these ODEs over a time interval of $100T$ (i.e. 100 heartbeats) to obtain a sufficient steady-state P_s waveform, then use the final state variable values as subsequent initial values to solve for a further time interval of T. The initial values for P_s and Q_L are both zero at the start of the $100T$ simulation:

Windkessel_solve.m

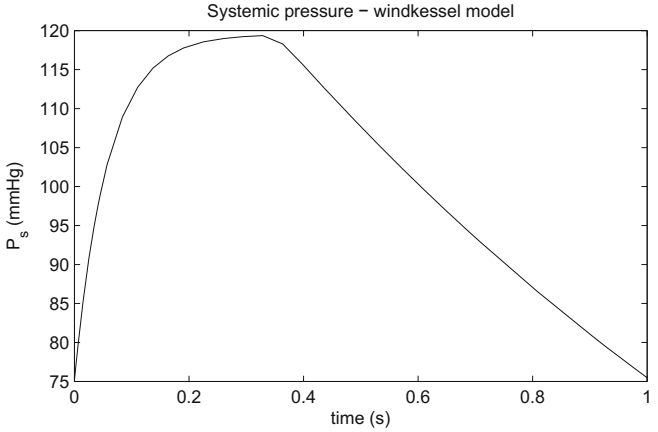
```
% define global parameters
global R0 L0 Cs Rs P T tc
R0 = 0.06;      % mmHg.s/cm^3
L0 = 0.2;       % mmHg.s^2/cm^3
Cs = 1;         % cm^3/mmHg
Rs = 1.4;       % mmHg.s/cm^3
P = 120;        % mmHg
T = 1;          % s
tc = 0.35;      % s

% solve first for 100 heartbeats
Ps_init = 0;
QL_init = 0;
[time, y_out] = ode15s('Windkessel_prime', ...
    [0 100*T], [Ps_init QL_init]);
```

```
% solve for one additional heartbeat for steady-state
Ps_init = y_out(end,1);
QL_init = y_out(end,2);
[time, y_out] = ode15s('Windkessel_prime', ...
    [0 T], [Ps_init QL_init]);

plot(time, y_out(:,1),'k'), ...
    xlabel ('time (s)'), ylabel('P_s (mmHg)'), ...
    title('Systemic pressure - windkessel model');
```

Windkessel_prime.m

```
function y_prime = Windkessel_prime(t,y)
global R0 L0 Cs Rs P T tc
    y_prime = zeros(2,1);
    Ps = y(1);
    QL = y(2);
    tt = mod(t,T);    % to produce a periodic waveform
    if (tt < tc)
        Pv = P;
    else
        Pv = 0;
    end;
    if (Pv > Ps)
        y_prime(1) = QL/Cs + (Pv-Ps)/(R0*Cs) - Ps/(Rs*Cs);
        y_prime(2) = (Pv-Ps)/L0;
    else
        y_prime(1) = -Ps/Rs;
        y_prime(2) = -R0*QL/L0;
    end;
```

producing the following plot:



2.7 The following Matlab code (ECG_solve.m and ECG_prime.m) will solve the ECG model over a time interval of 1 s:

ECG_solve.m

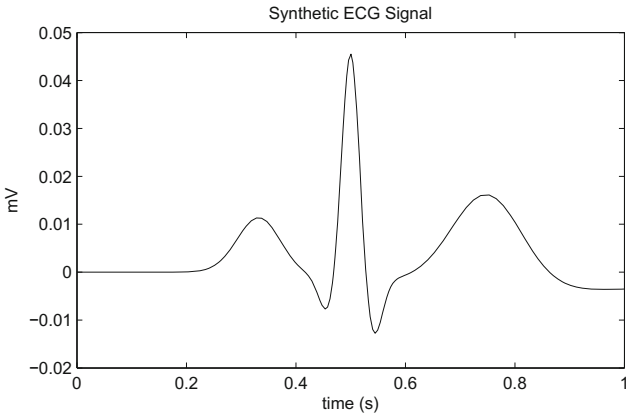
```
% define global parameters
global A B TH w z0;
A = [1.2, -5, 30, -7.5, 0.75];
B = [0.25, 0.1, 0.1, 0.1, 0.4];
TH = [-1/3, -1/12, 0, 1/12, 1/2]*pi;
w = 2*pi;
z0 = 0;

[time, y_out] = ode15s('ECG_prime', [0 1], [-1 0 0]);

plot(time, y_out(:,3),'k'), ...
    xlabel ('time (s)'), ylabel('mV'), ...
    title('Synthetic ECG Signal');
```

ECG_prime.m

```
function Y_prime = ECG_prime(t,Y)
global A B TH w z0;
    Y_prime = zeros(3,1);
    x = Y(1);
    y = Y(2);
    z = Y(3);
    alpha = 1-sqrt(x^2+y^2);
    th = atan2(y,x);
    Y_prime(1) = alpha*x - w*y;
```

```
Y_prime(2) = alpha*y + w*x;

Y_prime(3) = -(z-z0);
for ii = 1:5
    Y_prime(3) = Y_prime(3)-...
        A(ii)*(th-TH(ii))*exp(-(th-TH(ii))^2/(2*B(ii)^2));
end;
```

to produce the plot



2.8 The Frankenhaeuser–Huxley neural model is a typical example of pitfalls typically encountered when modelling biological systems. A chief difficulty arises from the choice of appropriate units. Since (1) the membrane capacitance is in units of $\mu\mathrm{F\,cm^{-2}}$, (2) the transmembrane potential is in units of mV and (3) time is in ms, the membrane ionic current i_{ion} in the equation

$$\frac{\mathrm{d}V}{\mathrm{d}t} = -\frac{i_{ion}}{C_m}$$

should be in units of $\mu\mathrm{A\,cm^{-2}}$. Naive use of the supplied parameters for i_{Na}, i_K and i_P would result in units of $\mathrm{mA\,cm^{-2}}$, so these evaluated currents must be multiplied by a factor of 1000 (note that i_L is already in the correct units of $\mu\mathrm{A\,cm^{-2}}$). Another difficulty is that the stimulus current duration is only 0.12 ms, so that naive use of any of of Matlab's ODE solvers such as `ode15s` would likely result in too large a step size, bypassing the stimulus. It is therefore necessary to restrict the step size, which can be achieved by specifying a maximum step with Matlab's `odeset` command. These 'tricks' have been implemented in the following Matlab code (FH_solve.m and FH_prime.m):

FH_solve.m

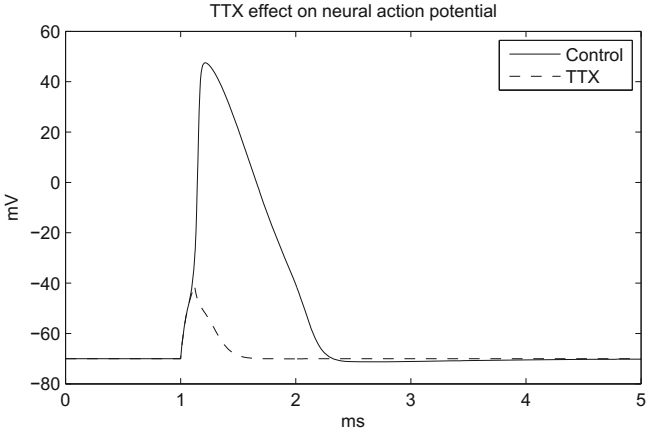
```
% solves the Frankenhaeuser--Huxley neural model
global Cm P_Na P_K P_P g_L V_L Na_o;
global Na_i K_o K_i I_s t_on t_dur;
global Er F R T;

Cm = 2;          % uC/cm^2
P_Na = 0.008;    % cm/s
P_K = 0.0012;    % cm/s
P_P = 0.00054;   % cm/s
g_L = 30.3;      % mS/cm^2
V_L = 0.026;     % mV
Na_o = 114.5;    % mM
Na_i = 13.74;    % mM
K_o = 2.5;       % mM
K_i = 120;       % mM
I_s = 1;         % mA/cm^2
t_on = 1;        % ms
t_dur = 0.12;    % ms
Er = -70;        % mV
F = 96.49;       % C/mmol
R = 8.31;        % J/(mol*K)
T = 310;         % K

Y_init = [0, 0.0005, 0.8249, 0.0268, 0.0049];
options = odeset('MaxStep', 0.01);
% solve first for control case (no TTX)
[time_1, Y_out_1] = ode15s('FH_prime', [0 5], Y_init, options);
% next, solve for presence of TTX
P_Na = 0.2*P_Na;
[time_2, Y_out_2] = ode15s('FH_prime', [0 5], Y_init, options);
plot(time_1, Y_out_1(:,1)+Er, 'k', time_2, Y_out_2(:,1)+Er, 'k--'), ...
    xlabel('ms'), ylabel('mV'), legend('Control', 'TTX'), ...
    title('TTX effect on neural action potential');
```

FH_prime.m

```
function y_out = FH_prime(t,y)
% calculates derivatives for solving the
% Frankenhaeuser--Huxley neural model.
global Cm P_Na P_K P_P g_L V_L Na_o;
global Na_i K_o K_i I_s t_on t_dur;
global Er F R T;
y_out = zeros(5,1);

V = y(1);
m = y(2);
h = y(3);
n = y(4);
```

```
p = y(5);

alpha_m = 0.36*(V-22)/(1-exp((22-V)/3));
beta_m = 0.4*(13-V)/(1-exp((V-13)/20));
alpha_h = 0.1*(-10-V)/(1-exp((V+10)/6));
beta_h = 4.5/(1+exp((45-V)/10));
alpha_n = 0.02*(V-35)/(1-exp((35-V)/10));
beta_n = 0.05*(10-V)/(1-exp((V-10)/10));
alpha_p = 0.006*(V-40)/(1-exp((40-V)/10));
beta_p = 0.09*(-25-V)/(1-exp((V+25)/20));

E = V+Er;

i_Na = 1000*m^2*h*P_Na*(E*F^2/(R*T))*...
    (Na_o - Na_i*exp(E*F/(R*T)))...
    /(1-exp(E*F/(R*T)));
i_K = 1000*n^2*P_K*(E*F^2/(R*T))*...
    (K_o - K_i*exp(E*F/(R*T)))...
    /(1-exp(E*F/(R*T)));
i_P = 1000*p^2*P_P*(E*F^2/(R*T))*...
    (Na_o - Na_i*exp(E*F/(R*T)))...
    /(1-exp(E*F/(R*T)));
i_L = g_L*(V-V_L);

if ((t >= t_on)&&(t<t_on+t_dur))
    i_s = 1000*I_s;
else
    i_s = 0;
end;

y_out(1) = -(i_Na+i_K+i_P+i_L-i_s)/Cm;
y_out(2) = alpha_m*(1-m)-beta_m*m;
y_out(3) = alpha_h*(1-h)-beta_h*h;
y_out(4) = alpha_n*(1-n)-beta_n*n;
y_out(5) = alpha_p*(1-p)-beta_p*p;
```

to produce the plot



2.9 (a) For the isometric contraction case, total muscle length L is held at L_0. The total tension T is given by

$$\begin{aligned} T &= T_p + T_s \\ &= \beta\left(\mathrm{e}^{\alpha(L-L_0)} - 1\right) + \beta\left(\mathrm{e}^{\alpha L_s} - 1\right) \\ &= \beta\left(\mathrm{e}^{\alpha L_s} - 1\right) \qquad \text{(since } L = L_0\text{)} \\ &= \beta\left(\mathrm{e}^{\alpha(L_0-L_c)} - 1\right) \end{aligned}$$

Taking derivatives of both sides and using the chain rule:

$$\begin{aligned} \frac{\mathrm{d}T}{\mathrm{d}t} &= \frac{\mathrm{d}}{\mathrm{d}L_c}\left[\mathrm{e}^{\alpha(L_0-L_c)} - 1\right]\frac{\mathrm{d}L_c}{\mathrm{d}t} \\ &= -\alpha\mathrm{e}^{\alpha(L_0-L_c)}\frac{\mathrm{d}L_c}{\mathrm{d}t} \\ &= -\alpha\mathrm{e}^{\alpha(L_0-L_c)}\left(\frac{a\left[T_s - S_0 f(t)\right]}{T_s + \gamma S_0}\right) \end{aligned}$$

Hence, the pair of ODEs describing this isometric contraction is

$$\begin{aligned} \frac{\mathrm{d}T}{\mathrm{d}t} &= -\alpha\mathrm{e}^{\alpha(L_0-L_c)}\left(\frac{a\left[T_s - S_0 f(t)\right]}{T_s + \gamma S_0}\right) \\ \frac{\mathrm{d}L_c}{\mathrm{d}t} &= \frac{a\left[T_s - S_0 f(t)\right]}{T_s + \gamma S_0} \end{aligned}$$

These are implemented in following Matlab code (isometric_solve.m and isometric_prime.m):

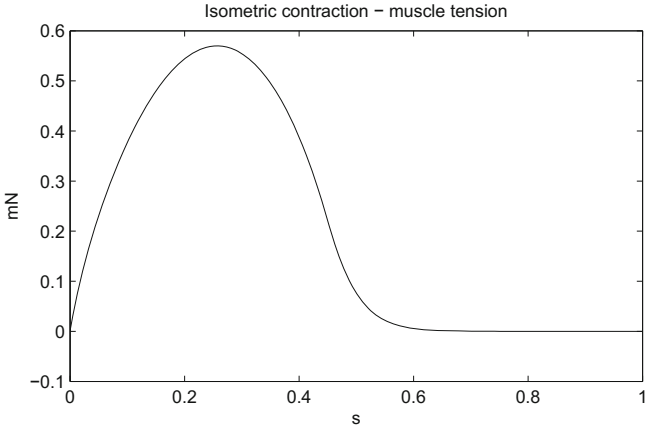
isometric_solve.m

```
global L0 alpha beta a S0 gamma t_0 t_ip;

L0 = 10;         % mm
alpha = 15;      % mm
beta = 5;        % mN
a = 0.66;        % 1/mm
S0 = 4;          % mN
gamma = 0.45;
t_0 = 0.05;      % s
t_ip = 0.2;      % s

Y_init = [0, 10];
options = odeset('MaxStep', 0.01);

[time, Y_out] = ode15s('isometric_prime', [0 1], Y_init, options);
plot(time, Y_out(:,1), 'k'), ...
    xlabel('s'), ylabel('mN'),
    title('Isometric contraction - muscle tension');
```

isometric_prime.m

```
function y_prime = isometric_prime(t,y)
global L0 alpha beta a S0 gamma t_0 t_ip;
y_prime = zeros(2,1);

if (t<2*t_ip+t_0)
    f = sin((pi/2)*(t+t_0)/(t_ip+t_0));
else
    f = 0;
end

T = y(1);
Lc = y(2);
Ls = L0-Lc;
Ts = beta*(exp(alpha*Ls)-1);

y_prime(1) = -alpha*exp(L0-Lc)*...
    a*(Ts-S0*f)/(Ts+gamma*S0);
y_prime(2) = a*(Ts-S0*f)/(Ts+gamma*S0);
```

to produce the plot



(b) For the isotonic contraction case, total muscle tension T is held at 0. This total tension T may be written as

$$\begin{aligned} T &= T_p + T_s \\ &= \beta\left(\mathrm{e}^{\alpha(L-L_0)} - 1\right) + \beta\left(\mathrm{e}^{\alpha L_s} - 1\right) \end{aligned}$$

Taking derivatives of both sides and using the chain rule:

$$\begin{aligned} \frac{\mathrm{d}T}{\mathrm{d}t} &= \frac{\mathrm{d}}{\mathrm{d}L}\left[\mathrm{e}^{\alpha(L-L_0)} - 1\right]\frac{\mathrm{d}L}{\mathrm{d}t} + \frac{\mathrm{d}}{\mathrm{d}L_s}\left[\mathrm{e}^{\alpha L_s} - 1\right]\frac{\mathrm{d}L_s}{\mathrm{d}t} \\ &= \alpha\mathrm{e}^{\alpha(L-L_0)}\frac{\mathrm{d}L}{\mathrm{d}t} + \alpha\mathrm{e}^{\alpha L_s}\frac{\mathrm{d}L_s}{\mathrm{d}t} \\ &= \alpha\mathrm{e}^{\alpha(L-L_0)}\frac{\mathrm{d}L}{\mathrm{d}t} + \alpha\mathrm{e}^{\alpha(L-L_c)}\frac{\mathrm{d}(L-L_c)}{\mathrm{d}t} \end{aligned}$$

and since this contraction is isotonic, $\frac{\mathrm{d}T}{\mathrm{d}t} = 0$. Hence,

$$\begin{aligned} 0 &= \alpha\mathrm{e}^{\alpha(L-L_0)}\frac{\mathrm{d}L}{\mathrm{d}t} + \alpha\mathrm{e}^{\alpha(L-L_c)}\frac{\mathrm{d}(L-L_c)}{\mathrm{d}t} \\ \alpha\mathrm{e}^{\alpha(L-L_c)}\frac{\mathrm{d}L_c}{\mathrm{d}t} &= \alpha\mathrm{e}^{\alpha(L-L_0)}\frac{\mathrm{d}L}{\mathrm{d}t} + \alpha\mathrm{e}^{\alpha(L-L_c)}\frac{\mathrm{d}L}{\mathrm{d}t} \\ \alpha\mathrm{e}^{\alpha(L-L_c)}\frac{\mathrm{d}L_c}{\mathrm{d}t} &= \left(\alpha\mathrm{e}^{\alpha(L-L_0)} + \alpha\mathrm{e}^{\alpha(L-L_c)}\right)\frac{\mathrm{d}L}{\mathrm{d}t} \\ \therefore \quad \frac{\mathrm{d}L}{\mathrm{d}t} &= \left(\frac{\mathrm{e}^{\alpha(L-L_c)}}{\mathrm{e}^{\alpha(L-L_0)} + \mathrm{e}^{\alpha(L-L_c)}}\right)\frac{\mathrm{d}L_c}{\mathrm{d}t} \\ &= \left(\frac{\mathrm{e}^{\alpha L_c}}{\mathrm{e}^{\alpha L_0} + \mathrm{e}^{\alpha L_c}}\right)\frac{\mathrm{d}L_c}{\mathrm{d}t} \end{aligned}$$

$$= \left(\frac{e^{\alpha L_c}}{e^{\alpha L_0} + e^{\alpha L_c}} \right) \frac{a\,[T_s - S_0 f(t)]}{T_s + \gamma S_0}$$

Hence, the pair of ODEs describing the isotonic contraction is

$$\frac{dL}{dt} = \left(\frac{e^{\alpha L_c}}{e^{\alpha L_0} + e^{\alpha L_c}} \right) \frac{a\,[T_s - S_0 f(t)]}{T_s + \gamma S_0}$$
$$\frac{dL_c}{dt} = \frac{a\,[T_s - S_0 f(t)]}{T_s + \gamma S_0}$$

These are implemented in following Matlab code (isotonic_solve.m and isotonic_prime.m):

isotonic_solve.m

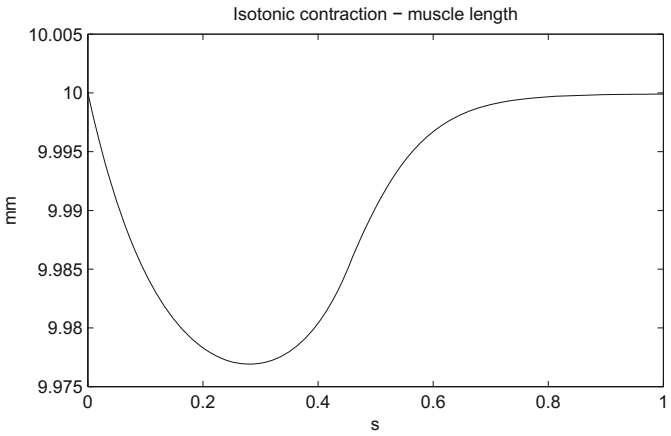
```
global L0 alpha beta a S0 gamma t_0 t_ip;

L0 = 10;         % mm
alpha = 15;      % mm
beta = 5;        % mN
a = 0.66;        % 1/mm
S0 = 4;          % mN
gamma = 0.45;
t_0 = 0.05;      % s
t_ip = 0.2;      % s

Y_init = [10, 10];
options = odeset('MaxStep', 0.01);

[time, Y_out] = ode15s('isotonic_prime', [0 1], Y_init, options);
plot(time, Y_out(:,1), 'k'), ...
    xlabel('s'), ylabel('mm'),
    title('Isotonic contraction - muscle length');
```

isotonic_prime.m

```
function y_prime = isotonic_prime(t,y)
global L0 alpha beta a S0 gamma t_0 t_ip;
y_prime = zeros(2,1);

if (t<2*t_ip+t_0)
    f = sin((pi/2)*(t+t_0)/(t_ip+t_0));
else
    f = 0;
end

L = y(1);
```

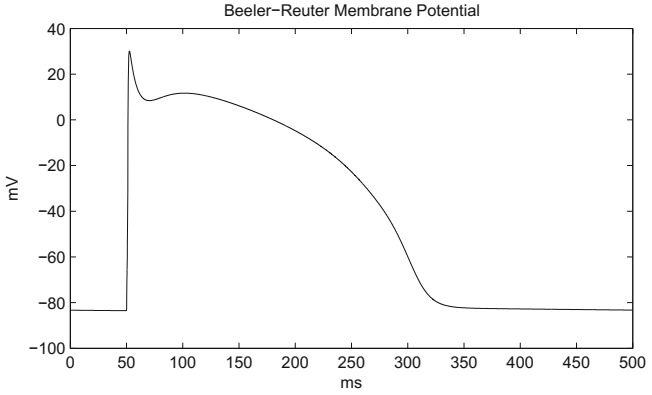
```
Lc = y(2);
Ls = L-Lc;
Ts = beta*(exp(alpha*Ls)-1);

y_prime(1) = (exp(alpha*Lc)/(exp(alpha*L0)+exp(alpha*Lc)))*...
    a*(Ts-S0*f)/(Ts+gamma*S0);
y_prime(2) = a*(Ts-S0*f)/(Ts+gamma*S0);
```

to produce the plot



Problems of Chap. 3

3.1 (a)



This plot was produced by the following Matlab code (BR_solve.m and BR_prime.m). Note that the maximum step for `ode15s` was conservatively set to 0.1 ms to ensure the applied stimulus at $t = 50$ ms was not bypassed:

BR_solve.m

```
% solves the Beeler-Reuter (1977) model
global Cm r_Ca Ca_SR k_up A_K1 A_x1 g_Na;
global g_NaC V_Na g_s A_s t_on t_dur;

Cm = 1;          % uC/cm^2
r_Ca = 1e-7;    % M*cm^2/nC
Ca_SR = 1e-7;    % M
k_up = 0.07;     % 1/ms
A_K1 = 0.35;     % uA/cm^2
A_x1 = 0.8;      % uA/cm^2
V_Na = 50;       % mV
g_Na = 4;        % mS/cm^2
g_NaC = 0.003;   % mS/cm^2
g_s = 0.09;     % mS/cm^2
A_s = 40;        % uA/cm^2
t_on = 50;       % ms
t_dur = 1;       % ms

Y_init = [-83.3, 1.87e-7, 0.1644, 0.01, 0.9814, 0.9673, 0.0033, 0.9884];
options = odeset('MaxStep', 0.1);
[time, Y_out] = ode15s('BR_prime', [0 500], Y_init, options);
plot(time, Y_out(:,1), 'k'), xlabel('ms'), ylabel('mV'), ...
    title('Beeler-Reuter Membrane Potential');
```

BR_prime.m

```
function y_prime = BR_prime(t,y)
global Cm r_Ca Ca_SR k_up A_K1 A_x1 g_Na;
global g_NaC V_Na g_s A_s t_on t_dur;
y_prime = zeros(8,1);

V = y(1);
Ca = y(2);
x1 = y(3);
m = y(4);
h = y(5);
j = y(6);
d = y(7);
f = y(8);
alpha_x1 = 0.0005*exp(0.083*(V+50))/(exp(0.057*(V+50))+1);
beta_x1 = 0.0013*exp(-0.06*(V+20))/(exp(-0.04*(V+20))+1);
alpha_m = -(V+47)/(exp(-0.1*(V+47))-1);
beta_m = 40*exp(-0.056*(V+72));
alpha_h = 0.126*exp(-0.25*(V+77));
beta_h = 1.7/(exp(-0.082*(V+22.5))+1);
alpha_j = 0.055*exp(-0.25*(V+78))/(exp(-0.2*(V+78))+1);
beta_j = 0.3/(exp(-0.1*(V+32))+1);
alpha_d = 0.095*exp(-0.01*(V-5))/(exp(-0.072*(V-5))+1);
beta_d = 0.07*exp(-0.017*(V+44))/(exp(0.05*(V+44))+1);
alpha_f = 0.012*exp(-0.008*(V+28))/(exp(0.15*(V+28))+1);
beta_f = 0.0065*exp(-0.02*(V+30))/(exp(-0.2*(V+30))+1);

V_Ca = -82.3-13.0287*log(Ca);
```

```
i_K1 = A_K1*(4*(exp(0.04*(V+85))-1)/(exp(0.08*(V+53))+exp(0.04*(V+53))) + ...
    0.2*(V+23)/(1-exp(-0.04*(V+23))));
i_x1 = A_x1*x1*(exp(0.04*(V+77))-1)/exp(0.04*(V+35));
i_Na = (g_Na*m^3*h*j + g_NaC)*(V-V_Na);
i_s = g_s*d*f*(V-V_Ca);

if ((t >= t_on)&&(t<t_on+t_dur))
    i_stim = A_s;
else
    i_stim = 0;
end;

y_prime(1) = -(i_K1+i_x1+i_Na+i_s-i_stim)/Cm;
y_prime(2) = -r_Ca*i_s+k_up*(Ca_SR-Ca);
y_prime(3) = alpha_x1*(1-x1)-beta_x1*x1;
y_prime(4) = alpha_m*(1-m)-beta_m*m;
y_prime(5) = alpha_h*(1-h)-beta_h*h;
y_prime(6) = alpha_j*(1-j)-beta_j*j;
y_prime(7) = alpha_d*(1-d)-beta_d*d;
y_prime(8) = alpha_f*(1-f)-beta_f*f;
```

(b) To solve the Beeler–Reuter model using the forward-Euler method, the following code was implemented (BR_forward_Euler_solve.m and BR_forward_Euler.m). Note that BR_forward_Euler.m makes use of the BR_prime.m function defined above:

BR_forward_Euler_solve.m

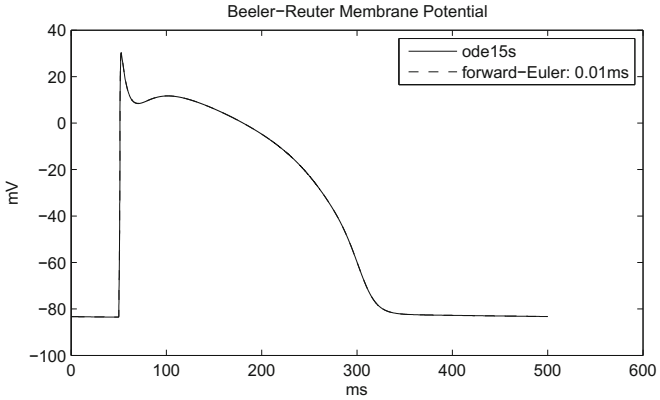
```
% solves the Beeler-Reuter (1977) model with
% the forward-Euler and ode15s algorithms
global Cm r_Ca Ca_SR k_up A_K1 A_x1 g_Na;
global g_NaC V_Na g_s A_s t_on t_dur;
global Y_init;

Cm = 1;          % uC/cm^2
r_Ca = 1e-7;    % M*cm^2/nC
Ca_SR = 1e-7;    % M
k_up = 0.07;     % 1/ms
A_K1 = 0.35;     % uA/cm^2
A_x1 = 0.8;      % uA/cm^2
V_Na = 50;       % mV
g_Na = 4;        % mS/cm^2
g_NaC = 0.003;   % mS/cm^2
g_s = 0.09;     % mS/cm^2
A_s = 40;        % uA/cm^2
t_on = 50;       % ms
t_dur = 1;       % ms

Y_init = [-83.3, 1.87e-7, 0.1644, 0.01, 0.9814, 0.9673, 0.0033, 0.9884];

% solve model using forward-Euler with step-size 0.01 ms
[time_fE, Y_out_fE] = BR_forward_Euler(0.01);

% solve using ode15s
```

```
options = odeset('MaxStep', 0.1);
[time_15s, Y_out_15s] = ode15s('BR_prime', [0 500], Y_init, options);

% plot results
plot(time_15s, Y_out_15s(:,1), 'k', time_fE, Y_out_fE(:,1), 'k--'), ...
    xlabel('ms'), ylabel('mV'), ...
    title('Beeler-Reuter Membrane Potential'), ...
    legend('ode15s', 'forward-Euler: 0.01ms');
```

BR_forward_Euler.m

```
function [time, Y_out] = BR_forward_Euler(h)
% solves the Beeler-Reuter (1977) model
% from t = 0 to 500ms using the forward-Euler
% method with fixed step-size h (in ms)

global Y_init;
S = round(500/h)+1;       % output time length
time = zeros(S,1);
Y_out = zeros(S,8);

t = 0;
Y = Y_init;
Y_out(1,:) = Y_init;

% main loop
for ii = 1:S
    Y = Y + h*BR_prime(t,Y)';
    t = t+h;
    time(ii+1) = t;
    Y_out(ii+1,:) = Y;
end;
```

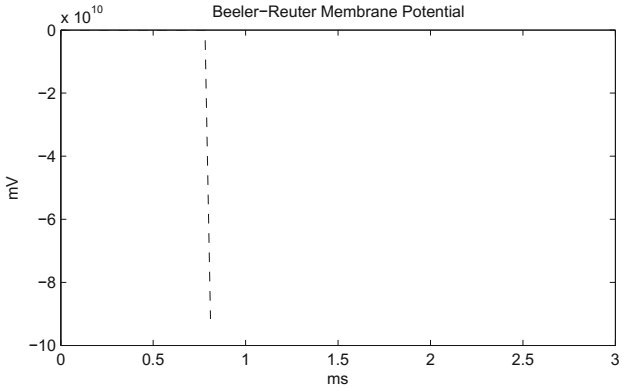
producing the following plot:



(c) The following code was used to plot the forward-Euler solution for V_m over the first few milliseconds using a step size of 0.03 ms:

```
% solves the Beeler-Reuter (1977) model using
% forward-Euler with step-size 0.03 ms
global Cm r_Ca Ca_SR k_up A_K1 A_x1 g_Na;
global g_NaC V_Na g_s A_s t_on t_dur;
global Y_init;
Cm = 1;          % uC/cm^2
r_Ca = 1e-7;    % M*cm^2/nC
Ca_SR = 1e-7;    % M
k_up = 0.07;     % 1/ms
A_K1 = 0.35;     % uA/cm^2
A_x1 = 0.8;      % uA/cm^2
V_Na = 50;       % mV
g_Na = 4;        % mS/cm^2
g_NaC = 0.003;   % mS/cm^2
g_s = 0.09;     % mS/cm^2
A_s = 40;        % uA/cm^2
t_on = 50;       % ms
t_dur = 1;       % ms
Y_init = [-83.3, 1.87e-7, 0.1644, 0.01, 0.9814, 0.9673, 0.0033, 0.9884];

[time_fE, Y_out_fE] = BR_forward_Euler(0.03);

% plot first 100 points only
plot(time_fE(1:100), Y_out_fE(1:100,1), 'k--'), ...
    xlabel('ms'), ylabel('mV'), ...
    title('Beeler-Reuter Membrane Potential');
```

producing the result:



where it is apparent the method has become unstable.

(d) To solve the Beeler–Reuter model using the backward-Euler method, we rewrite Eq. 3.11 in the form:

$$\mathbf{y}_n - \mathbf{y}_{n-1} - h\mathbf{f}(t_n, \mathbf{y}_n) = \mathbf{0}$$

and use Newton's method (Eq. 3.12) to solve for $\mathbf{y}_n$ at each step. This technique has been utilized in the below Matlab code (BR_backward_Euler_solve.m and BR_backward_Euler.m):

BR_backward_Euler_solve.m

```
% solves the Beeler-Reuter (1977) model with
% the backward-Euler and ode15s algorithms
global Cm r_Ca Ca_SR k_up A_K1 A_x1 g_Na;
global g_NaC V_Na g_s A_s t_on t_dur;
global Y_init;

Cm = 1;          % uC/cm^2
r_Ca = 1e-7;     % M*cm^2/nC
Ca_SR = 1e-7;    % M
k_up = 0.07;     % 1/ms
A_K1 = 0.35;     % uA/cm^2
A_x1 = 0.8;      % uA/cm^2
V_Na = 50;       % mV
g_Na = 4;        % mS/cm^2
g_NaC = 0.003;   % mS/cm^2
g_s = 0.09;      % mS/cm^2
A_s = 40;        % uA/cm^2
t_on = 50;       % ms
t_dur = 1;       % ms
```

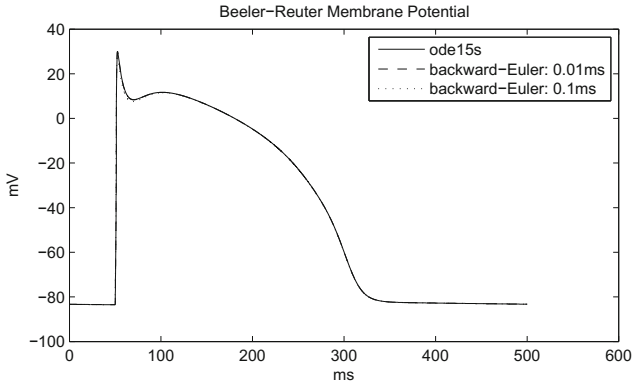
```
Y_init = [-83.3, 1.87e-7, 0.1644, 0.01, 0.9814, 0.9673, ...
    0.0033, 0.9884];

% solve model using backward-Euler with step-sizes of
% 0.01 ms and 0.1 ms
h1 = 0.01;
[time_bE_h1, Y_out_bE_h1] = BR_backward_Euler(h1);
h2 = 0.1;
[time_bE_h2, Y_out_bE_h2] = BR_backward_Euler(h2);

% solve using ode15s
options = odeset('MaxStep', 0.1);
[time_15s, Y_out_15s] = ode15s('BR_prime', [0 500], ...
    Y_init, options);

% plot results
plot(time_15s, Y_out_15s(:,1), 'k', ...
    time_bE_h1, Y_out_bE_h1(:,1), 'k--', ...
    time_bE_h2, Y_out_bE_h2(:,1), 'k:'), ...
    xlabel('ms'), ylabel('mV'), ...
    title('Beeler-Reuter Membrane Potential'), ...
    legend('ode15s', 'backward-Euler: 0.01ms', ...
    'backward-Euler: 0.1ms');
```

BR_backward_Euler.m

```
function [time, Y_out] = BR_backward_Euler(t_step)
% solves the Beeler-Reuter (1977) model
% from t = 0 to 500ms using the backward-Euler
% method with fixed step-size t_step (in ms)

global Y_init Y_prev h_step;

S = round(500/t_step)+1;       % output time length
time = zeros(S,1);
Y_out = zeros(S,8);
t = 0;
Y_prev = Y_init;
h_step = t_step;
Y_out(1,:) = Y_init;

% main loop
for ii = 1:S
    t = t+h_step;
    % use Newton's method to determine Y at the next step
    Y_iter = Y_prev;
    bE_res_0 = Y_iter' - Y_prev' - h_step*BR_prime(t,Y_iter);
    while max(abs(bE_res_0)) > 1e-6
        % determine Jacobian
```

```
        Jac = zeros(8,8);
        for jj = 1:8
            Y_test = Y_iter;
            step_j = 1e-6*max(abs(Y_iter(jj)),1e-3);
            Y_test(jj) = Y_iter(jj) + step_j;
            bE_res = Y_test' - Y_prev' - h_step*BR_prime(t,Y_test);
            Jac(:,jj) = (bE_res-bE_res_0)/step_j;
        end;
        Y_iter = Y_iter - (Jac\bE_res_0)';
        bE_res_0 = Y_iter' - Y_prev' - h_step*BR_prime(t,Y_iter);
    end;
    time(ii+1) = t;
    Y_out(ii+1,:) = Y_iter;
    Y_prev = Y_iter;
end;
```

which produces the following plot:



Note that unlike the forward-Euler method which was unstable for the step-size of 0.03 ms, the backward-Euler algorithm is stable at the even larger step-size of 0.1 ms. However, too large a step may lead to convergence issues with Newton's method.

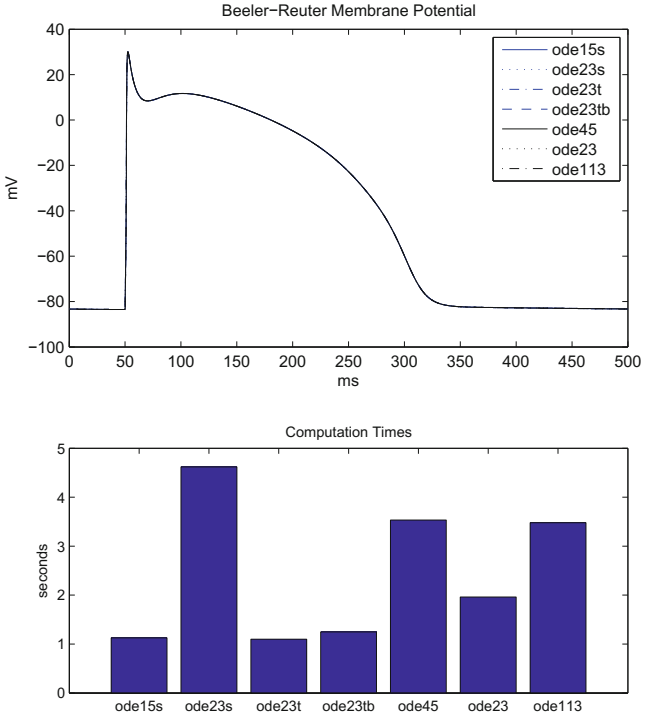
(e) The following Matlab code was used to solve the Beeler–Reuter model for a range of Matlab in-built ODE solvers, determining the computational time taken for each solver (using Matlab's `tic` and `toc` timing commands):

```
% solves the Beeler-Reuter (1977) model with
% several in-built Matlab ODE algorithms
global Cm r_Ca Ca_SR k_up A_K1 A_x1 g_Na;
global g_NaC V_Na g_s A_s t_on t_dur;

Cm = 1;          % uC/cm^2
r_Ca = 1e-7;     % M*cm^2/nC
Ca_SR = 1e-7;    % M
```

```
k_up = 0.07;     % 1/ms
A_K1 = 0.35;     % uA/cm^2
A_x1 = 0.8;      % uA/cm^2
V_Na = 50;       % mV
g_Na = 4;        % mS/cm^2
g_NaC = 0.003;   % mS/cm^2
g_s = 0.09;      % mS/cm^2
A_s = 40;        % uA/cm^2
t_on = 50;       % ms
t_dur = 1;       % ms

Y_init = [-83.3, 1.87e-7, 0.1644, 0.01, 0.9814, 0.9673, ...
    0.0033, 0.9884];

% solve using ode15s
options = odeset('MaxStep', 0.1);
tic
[time_15s, Y_out_15s] = ode15s('BR_prime', [0 500], ...
    Y_init, options);
ode15s_time_taken = toc;

% solve using ode23s
tic
[time_23s, Y_out_23s] = ode23s('BR_prime', [0 500], ...
    Y_init, options);
ode23s_time_taken = toc;

% solve using ode23t
tic
[time_23t, Y_out_23t] = ode23t('BR_prime', [0 500], ...
    Y_init, options);
ode23t_time_taken = toc;

% solve using ode23tb
tic
[time_23tb, Y_out_23tb] = ode23tb('BR_prime', [0 500], ...
    Y_init, options);
ode23tb_time_taken = toc;

% solve using ode45
tic
[time_45, Y_out_45] = ode45('BR_prime', [0 500], ...
    Y_init, options);
ode45_time_taken = toc;
```

```
% solve using ode23
tic
[time_23, Y_out_23] = ode23('BR_prime', [0 500], ...
    Y_init, options);
ode23_time_taken = toc;

% solve using ode113
tic
[time_113, Y_out_113] = ode113('BR_prime', [0 500], ...
    Y_init, options);
ode113_time_taken = toc;

time_taken = [ode15s_time_taken, ode23s_time_taken, ...
    ode23t_time_taken, ode23tb_time_taken, ...
    ode45_time_taken, ode23_time_taken, ...
    ode113_time_taken];

% plot results
figure(1), plot(time_15s, Y_out_15s(:,1), 'b', ...
    time_23s, Y_out_23s(:,1), 'b:', ...
    time_23t, Y_out_23t(:,1), 'b-.', ...
    time_23tb, Y_out_23tb(:,1), 'b--', ...
    time_45, Y_out_45(:,1), 'k', ...
    time_23, Y_out_23(:,1), 'k:', ...
    time_113, Y_out_113(:,1), 'k-.'), ...
    xlabel('ms'), ylabel('mV'), ...
    title('Beeler-Reuter Membrane Potential'), ...
    legend('ode15s', 'ode23s', 'ode23t', 'ode23tb',...
    'ode45', 'ode23', 'ode113');

figure(2), bar(time_taken), title('Computation Times'), ...
    set(gca, 'XTickLabel', {'ode15s', 'ode23s', ...
    'ode23t', 'ode23tb', 'ode45', 'ode23', 'ode113'}), ...
    ylabel('seconds');
```

producing the following plots:



From these plots, we can see that all of Matlab's ODE solvers were able to accurately compute the Beeler–Reuter equations, however `ode15s`, `ode23t`, and `ode23tb` exhibited the fastest computational times.[5] Of these, `ode23t` was slightly more efficient than `ode15s`. In most cases however, `ode15s` remains the general-purpose ODE solver of choice for most biological systems, although some other solvers may be slightly more computationally efficient for specific systems.

3.2 (a) The $I_{Na,p} + I_K$ model can be represented by the ODE system

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{f}(t, \mathbf{y})$$

To solve such a system, the third-order Runge–Kutta algorithm with coefficients

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ \frac{3}{4} & 0 & \frac{3}{4} & \\ \hline & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} \end{array}$$

is equivalent to the algorithm

[5] These particular computation times were obtained on a modest 1.3 GHz Intel i5 MacBook Air with 8 GB RAM running OS X 10.8.5.

$$\mathbf{K}_1 = \mathbf{f}(t_{n-1},\ \mathbf{y}_{n-1})$$
$$\mathbf{K}_2 = \mathbf{f}\left(t_{n-1} + \frac{h}{2},\ \mathbf{y}_{n-1} + \frac{h}{2}\mathbf{K}_1\right)$$
$$\mathbf{K}_3 = \mathbf{f}\left(t_{n-1} + \frac{3h}{4},\ \mathbf{y}_{n-1} + \frac{3h}{4}\mathbf{K}_2\right)$$
$$\mathbf{y}_n = \mathbf{y}_{n-1} + \frac{h}{9}\left(2\mathbf{K}_1 + 3\mathbf{K}_2 + 4\mathbf{K}_3\right)$$

which is implemented below in the code INapK_solve.m, INapK_Runge_Kutta.m and INapK_prime.m:

INapK_solve.m

```
% solves the I_Na,p + I_K model
% using a third-order Runge-Kutta algorithm
% and ode15s

global C g_Na E_Na g_K E_K tau_n g_L E_L I;
global Y_init;

C = 1;          % uF/cm^2
g_Na = 20;      % mS/cm^2
E_Na = 60;      % mV
g_K = 10;       % mS/cm^2
E_K = -90;      % mV
tau_n = 1;      % ms
g_L = 8;        % mS/cm^2
E_L = -80;      % mV
I = 40;         % uA/cm^2

Y_init = [-72.9 0.36];
h1 = 0.01;
h2 = 0.1;

% solve model with all steps and methods
[time_h1, Y_out_h1] = INapK_Runge_Kutta(h1);
[time_h2, Y_out_h2] = INapK_Runge_Kutta(h2);
[time_15s, Y_out_15s] = ode15s('INapK_prime', [0 30], Y_init);

% plot solutions
plot(time_15s, Y_out_15s(:,1), 'k', ...
    time_h1, Y_out_h1(:,1), 'k--', ...
    time_h2, Y_out_h2(:,1), 'k:'), ...
    xlabel('ms'), ylabel('mV'), ...
```

```
    title('I_{Na,p}+I_K Model'), ...
    legend('ode15s', 'RK: 0.01 ms', 'RK: 0.1 ms');
```

INapK_Runge_Kutta.m

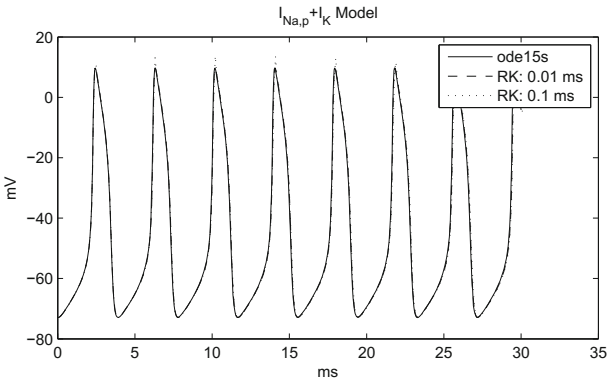
```
function [time, Y_out] = INapK_Runge_Kutta(h)
% solves the I_Na,p + I_K model
% from t = 0 to 30 ms using a
% third-order Runge-Kutta algorithm
% with fixed step-size h (in ms)

global Y_init;

S = round(30/h)+1;       % output time length
time = zeros(S,1);
Y_out = zeros(S,2);

t = 0;
Y = Y_init;
Y_out(1,:) = Y_init;

% main loop
for ii = 1:S
    K1 = INapK_prime(t,Y);
    K2 = INapK_prime(t+h/2,Y+K1'*h/2);
    K3 = INapK_prime(t+3*h/4,Y+K2'*3*h/4);

    Y = Y + (h/9)*(2*K1'+3*K2'+4*K3');
    t = t+h;
    time(ii+1) = t;
    Y_out(ii+1,:) = Y;
end;
```

iNapK_prime.m

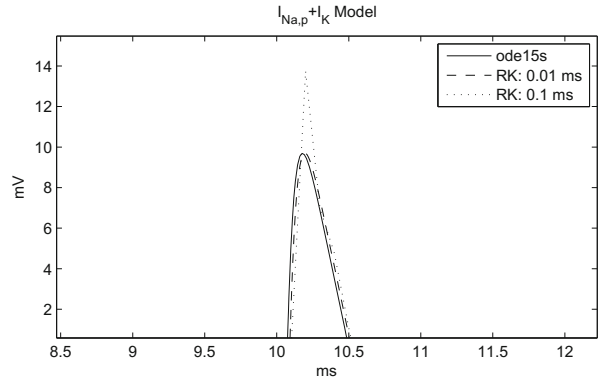
```
function y_prime = INapK_prime(t,y)
% solves the I_Na,p + I_K model
global C g_Na E_Na g_K E_K tau_n g_L E_L I;
y_prime = zeros(2,1);

V = y(1);
n = y(2);
```

```
m_inf = 1/(1+exp(-(V+20)/15));
n_inf = 1/(1+exp(-(V+25)/5));
y_prime(1) = -(1/C)*(g_Na*m_inf*(V-E_Na) + ...
    g_K*n*(V-E_K) + g_L*(V-E_L) - I);
y_prime(2) = (n_inf-n)/tau_n;
```

Executing `INapK_solve` from the Matlab command line produces the following plot of solutions:



The differences between these three methods can be highlighted more clearly when zooming in on the plot near $t = 10$ ms:



(b) To solve the above model using the generalized-α method, we first re-write the implicit algorithm of Eq. 3.26 for use with Newton's method: i.e. in the form $\mathbf{g}(\mathbf{x}) = \mathbf{0}$. For the general first-order ODE system

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{f}(t, \mathbf{y})$$

the generalized-α method can be expressed as:

$$\begin{aligned}\dot{\mathbf{y}}_{n-1} + \alpha_m \left(\dot{\mathbf{y}}_n - \dot{\mathbf{y}}_{n-1}\right) - \mathbf{f}\left(t_{n-1} + h\alpha_f,\ \mathbf{y}_{n-1} + \alpha_f \left(\mathbf{y}_n - \mathbf{y}_{n-1}\right)\right) &= \mathbf{0} \\ \mathbf{y}_n - \mathbf{y}_{n-1} - h\dot{\mathbf{y}}_{n-1} - h\gamma \left(\dot{\mathbf{y}}_n - \dot{\mathbf{y}}_{n-1}\right) &= \mathbf{0}\end{aligned}$$

which is in the required form $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ with $\mathbf{x} = [\dot{\mathbf{y}}_n^T, \mathbf{y}_n^T]$. The method parameters are

$$\alpha_f = \frac{1}{1+\rho_\infty}, \quad \alpha_m = \frac{1}{2}\left(\frac{3-\rho_\infty}{1+\rho_\infty}\right), \quad \gamma = \frac{1}{1+\rho_\infty}$$

where ρ_∞ specifies the required level of high-frequency damping from 0 to 1, with $\rho_\infty = 0$ denoting maximal damping and $\rho_\infty = 1$ no damping. The algorithm is implemented below in the code INapK_generalized_alpha_solve.m and INapK_generalized_alpha.m:

INapK_generalized_alpha_solve.m

```
% solves the I_Na,p + I_K model
% using the generalized-alpha and
% ode15s methods.
global C g_Na E_Na g_K E_K tau_n g_L E_L I;
global Y_init;

C = 1;          % uF/cm^2
g_Na = 20;      % mS/cm^2
E_Na = 60;      % mV
g_K = 10;       % mS/cm^2
E_K = -90;      % mV
tau_n = 1;      % ms
g_L = 8;        % mS/cm^2
E_L = -80;      % mV
I = 40;         % uA/cm^2

Y_init = [-72.9 0.36];

% solve model with ode15s and generalized_alpha method
% using three high-frequency damping factors
[time_15s, Y_out_15s] = ode15s('INapK_prime', [0 30], Y_init);
[time_1, Y_out_1] = INapK_generalized_alpha(0.1,1);
[time_2, Y_out_2] = INapK_generalized_alpha(0.1,0.5);
[time_3, Y_out_3] = INapK_generalized_alpha(0.1,0);

% plot solutions
plot(time_15s, Y_out_15s(:,1), 'k', ...
```

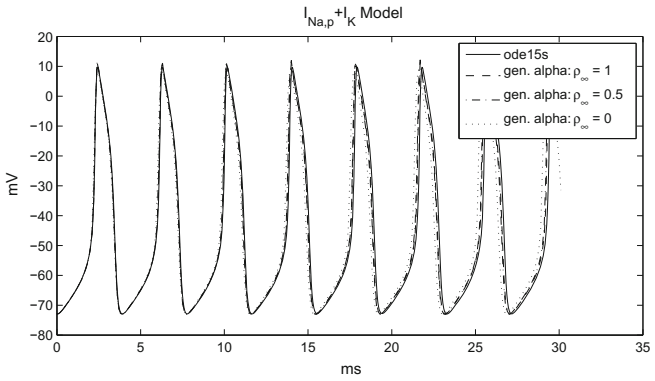
```
    time_1, Y_out_1(:,1), 'k--', ...
    time_2, Y_out_2(:,1), 'k-.', ...
    time_3, Y_out_3(:,1), 'k:'), ...
    xlabel('ms'), ylabel('mV'), ...
    title('I_{Na,p}+I_K Model'), ...
    legend('ode15s', 'gen. alpha: \rho_{\infty} = 1',...
    'gen. alpha: \rho_{\infty} = 0.5',...
    'gen. alpha: \rho_{\infty} = 0');
```

INapK_generalized_alpha.m

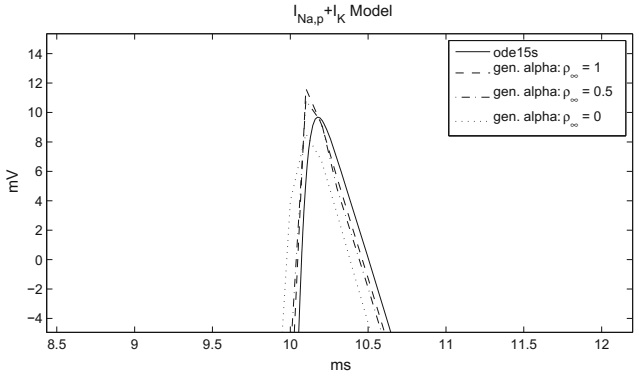
```
function [time, Y_out] = INapK_generalized_alpha(h,rho_inf)
% solves the I_Na,p + I_K model
% from t = 0 to 30 ms using the
% generalized-alpha method
% with fixed step-size h (in ms)
% and high-frequency damping factor rho_inf between 0 and 1

global Y_init;

S = round(30/h)+1;       % output time length
time = zeros(S,1);
Y_out = zeros(S,2);

t = 0;
Y_prev = Y_init;
Y_out(1,:) = Y_init;
Y_prime_prev = INapK_prime(0,Y_init)';

a_f = 1/(1+rho_inf);                % alpha_f
a_m = 0.5*(3-rho_inf)/(1+rho_inf);  % alpha_m
gamma = 1/(1+rho_inf);              % gamma

% main loop
for ii = 1:S
    % Use Newton's method to determine [Y_prime, Y] at the next step
    % First, begin with a constant predictor
    Y_iter = [Y_prime_prev, Y_prev];
    gen_alpha_res_Yprime = (1-a_m)*Y_prime_prev+a_m*Y_iter(1:2)...
        - INapK_prime(t+a_f*h,(1-a_f)*Y_prev+a_f*Y_iter(3:4))';
    gen_alpha_res_Y = Y_iter(3:4)-Y_prev...
        - h*(Y_prime_prev + gamma*(Y_iter(1:2)-Y_prime_prev));
    gen_alpha_res_0 = [gen_alpha_res_Yprime,gen_alpha_res_Y];
    while max(abs(gen_alpha_res_0)) > 1e-6
        % determine Jacobian
        Jac = zeros(4,4);
        for jj = 1:4
            Y_test = Y_iter;
            step_j = 1e-6*max(abs(Y_iter(jj)),1e-3);
            Y_test(jj) = Y_iter(jj) + step_j;
            gen_alpha_res_Yprime = (1-a_m)*Y_prime_prev+a_m*Y_test(1:2)...
                - INapK_prime(t+a_f*h,(1-a_f)*Y_prev+a_f*Y_test(3:4))';
            gen_alpha_res_Y = Y_test(3:4)-Y_prev...
                - h*(Y_prime_prev + gamma*(Y_test(1:2)-Y_prime_prev));
            gen_alpha_res = [gen_alpha_res_Yprime,gen_alpha_res_Y];
```

```
            Jac(:,jj) = (gen_alpha_res-gen_alpha_res_0)/step_j;
        end;
        Y_iter = Y_iter - (Jac\gen_alpha_res_0')';
        gen_alpha_res_Yprime = (1-a_m)*Y_prime_prev+a_m*Y_iter(1:2)...
            - INapK_prime(t+a_f*h,(1-a_f)*Y_prev+a_f*Y_iter(3:4))';
        gen_alpha_res_Y = Y_iter(3:4)-Y_prev...
            - h*(Y_prime_prev + gamma*(Y_iter(1:2)-Y_prime_prev));
        gen_alpha_res_0 = [gen_alpha_res_Yprime,gen_alpha_res_Y];
    end;
    t = t+h;
    time(ii+1) = t;
    Y_out(ii+1,:) = Y_iter(3:4);
    Y_prime_prev = Y_iter(1:2);
    Y_prev = Y_iter(3:4);
end;
```

Executing `INapK_generalized_alpha_solve` from the Matlab command line produces the following plot:



The differences between the three levels of damping can be seen more clearly when zooming in on the plot near $t = 10$ ms:



where it can be seen that membrane potential peak is progressively reduced with increased levels of damping.

3.3 Using Newton's backward difference formula (Eq. 3.42), the interpolating polynomial passing through the $k+1$ step solutions y_{n-1}, y_{n-1},..., y_{n-k-1} is:

$$\begin{aligned}\phi(t) =&\nabla^0 y_{n-1} + \frac{(t-t_{n-1})}{h}\nabla^1 y_{n-1} + \frac{(t-t_{n-1})(t-t_{n-2})}{2!\,h^2}\nabla^2 y_{n-1} + \ldots \\ &+ \frac{(t-t_{n-1})(t-t_{n-2})\ldots(t-t_{n-k})}{k!\,h^k}\nabla^k y_{n-1}\end{aligned}$$

Substituting $t = t_n$, we obtain

$$\begin{aligned}\phi(t_n) &= \nabla^0 y_{n-1} + \frac{(t_n-t_{n-1})}{h}\nabla^1 y_{n-1} + \frac{(t_n-t_{n-1})(t_n-t_{n-2})}{2!\,h^2}\nabla^2 y_{n-1} + \ldots \\ &\quad + \frac{(t_n-t_{n-1})(t_n-t_{n-2})\ldots(t_n-t_{n-k})}{k!\,h^k}\nabla^k y_{n-1} \\ &= \nabla^0 y_{n-1} + \frac{h}{h}\nabla^1 y_{n-1} + \frac{(h)(2h)}{2!\,h^2}\nabla^2 y_{n-1} + \cdots + \frac{(h)(2h)\ldots(kh)}{k!\,h^k}\nabla^k y_{n-1} \\ &= \nabla^0 y_{n-1} + \nabla^1 y_{n-1} + \nabla^2 y_{n-1} + \cdots + \frac{k!\,h^k}{k!\,h^k}\nabla^k y_{n-1} \\ &= \sum_{i=0}^{k} \nabla^i\, y_{n-1} \\ &= y_n^p \qquad \text{as required.}\end{aligned}$$

3.4 The k-step BDF formula is given by Eq. 3.58:

$$\sum_{i=1}^{k} \frac{1}{i}\nabla^i y_n = hf(t_n, y_n)$$

For $k = 7$, we form the backward difference expressions:

$$\begin{aligned}\nabla^1 y_n &= y_n - y_{n-1} \\ \nabla^2 y_n &= y_n - 2y_{n-1} + y_{n-2} \\ \nabla^3 y_n &= y_n - 3y_{n-1} + 3y_{n-2} - y_{n-3} \\ \nabla^4 y_n &= y_n - 4y_{n-1} + 6y_{n-2} - 4y_{n-3} + y_{n-4} \\ \nabla^5 y_n &= y_n - 5y_{n-1} + 10y_{n-2} - 10y_{n-3} + 5y_{n-4} - y_{n-5} \\ \nabla^6 y_n &= y_n - 6y_{n-1} + 15y_{n-2} - 20y_{n-3} + 15y_{n-4} - 6y_{n-5} + y_{n-6} \\ \nabla^7 y_n &= y_n - 7y_{n-1} + 21y_{n-2} - 35y_{n-3} + 35y_{n-4} - 21y_{n-5} + 7y_{n-6} - y_{n-7}\end{aligned}$$

Substituting these into the above k-step formula, and after re-arranging, we obtain the 7-step BDF formula:

$$y_n - \frac{980}{363}y_{n-1} + \frac{490}{121}y_{n-2} - \frac{4900}{1089}y_{n-3} + \frac{1225}{363}y_{n-4} - \frac{196}{121}y_{n-5} + \frac{490}{1089}y_{n-6} - \frac{20}{363}y_{n-7} = \frac{140}{363}hf(t_n, y_n)$$

For the test ODE

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \lambda y$$

with $\lambda < 0$, we take the initial value at $t = 0$ to be y_0. The exact solution is then given by $y_0\mathrm{e}^{\lambda t}$. For a fixed step-size of h, we can write the exact solutions to the first seven steps up to $t = 6h$ as

$$\begin{aligned}
&t = 0 = t_{n-7}, \quad y_{n-7} = y_0\\
&t = h = t_{n-6}, \quad y_{n-6} = y_0\mathrm{e}^{\lambda h}\\
&t = 2h = t_{n-5}, \quad y_{n-5} = y_0\mathrm{e}^{2\lambda h}\\
&t = 3h = t_{n-4}, \quad y_{n-4} = y_0\mathrm{e}^{3\lambda h}\\
&t = 4h = t_{n-3}, \quad y_{n-3} = y_0\mathrm{e}^{4\lambda h}\\
&t = 5h = t_{n-2}, \quad y_{n-2} = y_0\mathrm{e}^{5\lambda h}\\
&t = 6h = t_{n-1}, \quad y_{n-1} = y_0\mathrm{e}^{6\lambda h}
\end{aligned}$$

Substituting these values into the above 7-step BDF formula, and noting that $f(t_n, y_n) = \lambda y_n$, we obtain:

$$\begin{aligned}
y_n - \frac{980}{363}y_0\mathrm{e}^{6\lambda h} + \frac{490}{121}y_0\mathrm{e}^{5\lambda h} - \frac{4900}{1089}y_0\mathrm{e}^{4\lambda h} + \frac{1225}{363}y_0\mathrm{e}^{3\lambda h}\\
- \frac{196}{121}y_0\mathrm{e}^{2\lambda h} + \frac{490}{1089}y_0\mathrm{e}^{\lambda h} - \frac{20}{363}y_0 = \frac{140}{363}h\lambda y_n\\
\therefore \quad y_n\left(\frac{140}{363}h\lambda - 1\right) = y_0\mathrm{e}^{6\lambda h}\left(-\frac{980}{363} + \frac{490}{121}\mathrm{e}^{-\lambda h} - \frac{4900}{1089}\mathrm{e}^{-2\lambda h} + \frac{1225}{363}\mathrm{e}^{-3\lambda h}\right.\\
\left. - \frac{196}{121}\mathrm{e}^{-4\lambda h} + \frac{490}{1089}\mathrm{e}^{-5\lambda h} - \frac{20}{363}\mathrm{e}^{-6\lambda h}\right)\\
y_n(420h\lambda - 1089) = y_{n-1}\left(-2940 + 4410\mathrm{e}^{-\lambda h} - 4900\mathrm{e}^{-2\lambda h} + 3675\mathrm{e}^{-3\lambda h}\right.\\
\left. - 1764\mathrm{e}^{-4\lambda h} + 490\mathrm{e}^{-5\lambda h} - 60\mathrm{e}^{-6\lambda h}\right)
\end{aligned}$$

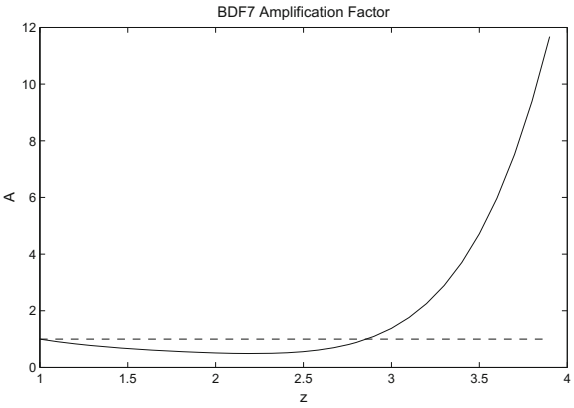
Substituting $z = \mathrm{e}^{-\lambda h}$ and re-arranging, we obtain:

$$\begin{aligned}
y_n &= y_{n-1}\left(\frac{-2940 + 4410z - 4900z^2 + 3675z^3 - 1764z^4 + 490z^5 - 60z^6}{-420\ln z - 1089}\right)\\
&= y_{n-1}\left(\frac{60z^6 - 490z^5 + 1764z^4 - 3675z^3 + 4900z^2 - 4410z + 2940}{420\ln z + 1089}\right)\\
&= y_{n-1}A
\end{aligned}$$

where A is a multiplicative factor that depends on z. For $\lambda < 0$ and h positive, z will be > 1. In the limit as the step-size h becomes infinitely large,

$$\lim_{z\to\infty} A = \frac{60z^6}{420\ln z} = \frac{360z^5}{420/z} = \frac{6}{7}z^6 = \infty$$

where we have used L'Hôpital's rule[6] to evaluate the limit. Hence, the method is unstable for large step-sizes. The plot of A against z is shown below:



where the dashed line represents $A = 1$. Clearly the method is unstable when $A > 1$, corresponding to a value of z near 2.8 on the plot. To numerically determine this value of z, we can use the Matlab `fzero` command, which solves the one-variable equation $g(x) = 0$, to solve the equation $A = 1$. Namely,

$$\frac{60z^6 - 490z^5 + 1764z^4 - 3675z^3 + 4900z^2 - 4410z + 2940}{420\ln z + 1089} = 1$$

which can be re-arranged to

$$60z^6 - 490z^5 + 1764z^4 - 3675z^3 + 4900z^2 - 4410z + 2940 - 420\ln z - 1089 = 0$$

This equation can be solved for z using the Matlab command:

```
z = fzero('BDF_zero', 2.8);
```

where 2.8 is the initial estimate for z, and the function `BDF_zero` is defined as

```
function f_out = BDF_zero(z)
```

[6]Pronounced lopi-tahl, the rule is $\lim_{x\to\infty} \frac{f(x)}{g(x)} = \frac{f'(x)}{g'(x)}$ provided (i) the limit exists, (ii) $g'(x) \neq 0$, and (iii) the corresponding limits of $f(x)$ and $g(x)$ are identical and both equal to 0 or $\pm\infty$.

```
f_out = 60*z^6 - 490*z^5 + 1764*z^4 ...
    - 3675*z^3  +4900*z^2 - 4410*z + 2940 ...
    - 420*log(z) - 1089;
```

Executing the above `fzero` command yields a value of $z \approx 2.8596$. For stability, we therefore require

$$\begin{aligned} z = \mathrm{e}^{-\lambda h} &< 2.8596 \\ -\lambda h &< \ln(2.8596) \approx 1.0507 \end{aligned}$$

Hence for absolute stability, we require $h < \frac{1.0507}{|\lambda|}$.

Problems of Chap. 4

4.1 (a) $\nabla f = \begin{pmatrix} 2 \\ -1 \\ 3 \end{pmatrix}$ (b) $\nabla g = \begin{pmatrix} 2x \\ 2y \\ 2z \end{pmatrix}$ (c) $\nabla h = \begin{pmatrix} 1/y \\ -x/y^2 \\ 0 \end{pmatrix}$

4.2 (a) $\nabla \cdot \mathbf{u} = 3$, $\nabla \times \mathbf{u} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ (b) $\nabla \cdot \mathbf{v} = 0$, $\nabla \times \mathbf{v} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$
(c) $\nabla \cdot \mathbf{w} = 4$, $\nabla \times \mathbf{w} = \begin{pmatrix} -3 \\ -2 \\ 2 \end{pmatrix}$

4.3 Writing $\mathbf{u} = (u_x, u_y, u_z)^T$ and $\mathbf{v} = (v_x, v_y, v_z)^T$, we have:
(a)

$$\begin{aligned} \nabla \cdot (\nabla \times \mathbf{u}) &= \nabla \cdot \begin{pmatrix} \frac{\partial u_y}{\partial z} - \frac{\partial u_z}{\partial y} \\ \frac{\partial u_z}{\partial x} - \frac{\partial u_x}{\partial z} \\ \frac{\partial u_x}{\partial y} - \frac{\partial u_y}{\partial x} \end{pmatrix} \\ &= \frac{\partial}{\partial x}\left(\frac{\partial u_y}{\partial z} - \frac{\partial u_z}{\partial y}\right) + \frac{\partial}{\partial y}\left(\frac{\partial u_z}{\partial x} - \frac{\partial u_x}{\partial z}\right) + \frac{\partial}{\partial z}\left(\frac{\partial u_x}{\partial y} - \frac{\partial u_y}{\partial x}\right) \\ &= \frac{\partial^2 u_y}{\partial x \partial z} - \frac{\partial^2 u_z}{\partial x \partial y} + \frac{\partial^2 u_z}{\partial x \partial y} - \frac{\partial^2 u_x}{\partial y \partial z} + \frac{\partial^2 u_x}{\partial y \partial z} - \frac{\partial^2 u_y}{\partial x \partial z} \\ &= 0 \end{aligned}$$

(b)

$$\begin{aligned} \nabla \cdot (\mathbf{u} \times \mathbf{v}) &= \nabla \cdot \begin{pmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{pmatrix} \\ &= \frac{\partial}{\partial x}\Big(u_y v_z - u_z v_y\Big) + \frac{\partial}{\partial y}\Big(u_z v_x - u_x v_z\Big) + \frac{\partial}{\partial z}\Big(u_x v_y - u_y v_x\Big) \end{aligned}$$

$$
\begin{aligned}
&= v_z\frac{\partial u_y}{\partial x} + u_y\frac{\partial v_z}{\partial x} - v_y\frac{\partial u_z}{\partial x} - u_z\frac{\partial v_y}{\partial x} + v_x\frac{\partial u_z}{\partial y} + u_z\frac{\partial v_x}{\partial y} - v_z\frac{\partial u_x}{\partial y} - u_x\frac{\partial v_z}{\partial y} \\
&\qquad + v_y\frac{\partial u_x}{\partial z} + u_x\frac{\partial v_y}{\partial z} - v_x\frac{\partial u_y}{\partial z} - u_y\frac{\partial v_x}{\partial z} \\
&= v_x\left(\frac{\partial u_z}{\partial y} - \frac{\partial u_y}{\partial z}\right) + v_y\left(-\frac{\partial u_z}{\partial x} + \frac{\partial u_x}{\partial z}\right) + v_z\left(\frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}\right) \\
&\qquad + u_x\left(-\frac{\partial v_z}{\partial y} + \frac{\partial v_y}{\partial z}\right) + u_y\left(\frac{\partial v_z}{\partial x} - \frac{\partial v_x}{\partial z}\right) + u_z\left(-\frac{\partial v_y}{\partial x} + \frac{\partial v_x}{\partial y}\right) \\
&= v_x\left(\frac{\partial u_z}{\partial y} - \frac{\partial u_y}{\partial z}\right) + v_y\left(\frac{\partial u_x}{\partial z} - \frac{\partial u_z}{\partial x}\right) + v_z\left(\frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}\right) \\
&\qquad + u_x\left(\frac{\partial v_y}{\partial z} - \frac{\partial v_z}{\partial y}\right) + u_y\left(\frac{\partial v_z}{\partial x} - \frac{\partial v_x}{\partial z}\right) + u_z\left(\frac{\partial v_x}{\partial y} - \frac{\partial v_y}{\partial x}\right) \\
&= \mathbf{v}\cdot(\nabla\times\mathbf{u}) + \mathbf{u}\cdot(-(\nabla\times\mathbf{v})) \\
&= \mathbf{v}\cdot(\nabla\times\mathbf{u}) - \mathbf{u}\cdot(\nabla\times\mathbf{v})
\end{aligned}
$$

(c) Starting from the right-hand side, we have:

$$
\begin{aligned}
\nabla(\nabla\cdot\mathbf{u}) &= \nabla\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z}\right) \\
&= \begin{pmatrix} \frac{\partial}{\partial x}\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z}\right) \\ \frac{\partial}{\partial y}\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z}\right) \\ \frac{\partial}{\partial z}\left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z}\right) \end{pmatrix} \\
&= \begin{pmatrix} \frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_y}{\partial x\partial y} + \frac{\partial^2 u_z}{\partial x\partial z} \\ \frac{\partial^2 u_x}{\partial x\partial y} + \frac{\partial^2 u_y}{\partial y^2} + \frac{\partial^2 u_z}{\partial y\partial z} \\ \frac{\partial^2 u_x}{\partial x\partial z} + \frac{\partial^2 u_y}{\partial y\partial z} + \frac{\partial^2 u_z}{\partial z^2} \end{pmatrix} \\
\nabla^2\mathbf{u} &= \frac{\partial^2\mathbf{u}}{\partial x^2} + \frac{\partial^2\mathbf{u}}{\partial y^2} + \frac{\partial^2\mathbf{u}}{\partial z^2} \\
&= \begin{pmatrix} \frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2} + \frac{\partial^2 u_x}{\partial z^2} \\ \frac{\partial^2 u_y}{\partial x^2} + \frac{\partial^2 u_y}{\partial y^2} + \frac{\partial^2 u_y}{\partial z^2} \\ \frac{\partial^2 u_z}{\partial x^2} + \frac{\partial^2 u_z}{\partial y^2} + \frac{\partial^2 u_z}{\partial z^2} \end{pmatrix} \\
\therefore\quad \nabla(\nabla\cdot\mathbf{u}) - \nabla^2\mathbf{u} &= \begin{pmatrix} \frac{\partial^2 u_y}{\partial x\partial y} + \frac{\partial^2 u_z}{\partial x\partial z} - \frac{\partial^2 u_x}{\partial y^2} - \frac{\partial^2 u_x}{\partial z^2} \\ \frac{\partial^2 u_x}{\partial x\partial y} + \frac{\partial^2 u_z}{\partial y\partial z} - \frac{\partial^2 u_y}{\partial x^2} - \frac{\partial^2 u_y}{\partial z^2} \\ \frac{\partial^2 u_x}{\partial x\partial z} + \frac{\partial^2 u_y}{\partial y\partial z} - \frac{\partial^2 u_z}{\partial x^2} - \frac{\partial^2 u_z}{\partial y^2} \end{pmatrix}
\end{aligned}
$$

On the left-hand side, we have:

$$\nabla \times (\nabla \times \mathbf{u}) = \nabla \times \begin{pmatrix} \frac{\partial u_y}{\partial z} - \frac{\partial u_z}{\partial y} \\ \frac{\partial u_z}{\partial x} - \frac{\partial u_x}{\partial z} \\ \frac{\partial u_x}{\partial y} - \frac{\partial u_y}{\partial x} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial}{\partial z}\left(\frac{\partial u_z}{\partial x} - \frac{\partial u_x}{\partial z}\right) - \frac{\partial}{\partial y}\left(\frac{\partial u_x}{\partial y} - \frac{\partial u_y}{\partial x}\right) \\ \frac{\partial}{\partial x}\left(\frac{\partial u_x}{\partial y} - \frac{\partial u_y}{\partial x}\right) - \frac{\partial}{\partial z}\left(\frac{\partial u_y}{\partial z} - \frac{\partial u_z}{\partial y}\right) \\ \frac{\partial}{\partial y}\left(\frac{\partial u_y}{\partial z} - \frac{\partial u_z}{\partial y}\right) - \frac{\partial}{\partial x}\left(\frac{\partial u_z}{\partial x} - \frac{\partial u_x}{\partial z}\right) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial^2 u_z}{\partial x \partial z} - \frac{\partial^2 u_x}{\partial z^2} - \frac{\partial^2 u_x}{\partial y^2} + \frac{\partial^2 u_y}{\partial x \partial y} \\ \frac{\partial^2 u_x}{\partial x \partial y} - \frac{\partial^2 u_y}{\partial x^2} - \frac{\partial^2 u_y}{\partial z^2} + \frac{\partial^2 u_z}{\partial y \partial z} \\ \frac{\partial^2 u_y}{\partial y \partial z} - \frac{\partial^2 u_z}{\partial y^2} - \frac{\partial^2 u_z}{\partial x^2} + \frac{\partial^2 u_x}{\partial x \partial z} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial^2 u_y}{\partial x \partial y} + \frac{\partial^2 u_z}{\partial x \partial z} - \frac{\partial^2 u_x}{\partial y^2} - \frac{\partial^2 u_x}{\partial z^2} \\ \frac{\partial^2 u_x}{\partial x \partial y} + \frac{\partial^2 u_z}{\partial y \partial z} - \frac{\partial^2 u_y}{\partial x^2} - \frac{\partial^2 u_y}{\partial z^2} \\ \frac{\partial^2 u_x}{\partial x \partial z} + \frac{\partial^2 u_y}{\partial y \partial z} - \frac{\partial^2 u_z}{\partial x^2} - \frac{\partial^2 u_z}{\partial y^2} \end{pmatrix}$$

which is equal to the right-hand side. Hence, we have shown:

$$\nabla \times (\nabla \times \mathbf{u}) = \nabla(\nabla \cdot \mathbf{u}) - \nabla^2 \mathbf{u}$$

4.4 From the divergence theorem (Eq. 4.5), we have

$$\int_V (\nabla . \mathbf{F}) \mathrm{d}V = \int_S \mathbf{F} \cdot \mathrm{d}\mathbf{S}$$

Choosing $\mathbf{F} = \begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix}$, we obtain $\nabla \cdot \mathbf{F} = 1$ and $\mathbf{F} \cdot \mathrm{d}\mathbf{S} = \mathbf{F} \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \mathrm{d}S = x n_x \mathrm{d}S$. Hence, substituting these into the above divergence theorem, we obtain

$$\int_V \mathrm{d}V = V = \int_S x n_x \mathrm{d}S$$

Similarly, we can choose $\mathbf{F} = \begin{pmatrix} 0 \\ y \\ 0 \end{pmatrix}$, to obtain $\nabla \cdot \mathbf{F} = 1$ and $\mathbf{F} \cdot \mathrm{d}\mathbf{S} = y n_y \mathrm{d}S$. Therefore,

$$\int_V \mathrm{d}V = V = \int_S y n_y \mathrm{d}S$$

Finally, choosing $\mathbf{F} = \begin{pmatrix} 0 \\ 0 \\ z \end{pmatrix}$, we have $\nabla \cdot \mathbf{F} = 1$ and $\mathbf{F} \cdot \mathrm{d}\mathbf{S} = z n_z \mathrm{d}S$. Hence,

$$\int_V \mathrm{d}V = V = \int_S z n_z \mathrm{d}S$$

Thus, for an arbitrary closed surface, we have shown its volume is given by

$$V = \int_S x n_x \mathrm{d}S = \int_S y n_y \mathrm{d}S = \int_S z n_z \mathrm{d}S$$

as required.

4.5 (a) Assume we have a spherical shell region V centred on the microsphere, having inner radius r and outer radius $r + \Delta r$. The total amount of drug C within the spherical shell is the volume integral of concentration c, namely:

$$C = \int_V c \, \mathrm{d}V = \int_r^{r+\Delta r} 4\pi\rho^2 c \, \mathrm{d}\rho = 4\pi r^2 c \Delta r \quad (\text{if } \Delta r \text{ is sufficiently small})$$

Since C is a conserved quantity, its time rate of change within the spherical shell must equal the rate of amount of drug entering plus the rate at which it is produced (or lost) within the shell. We can express this statement as a differential equation:

$$\frac{\partial C}{\partial t} = -\int_S \boldsymbol{\Gamma} \cdot \mathrm{d}\mathbf{S} + \int_V f \, \mathrm{d}V$$

where $\boldsymbol{\Gamma}$ is the outward flux (i.e. amount of drug diffusing out of the boundaries S per unit time per unit area), and f is the rate of drug production per unit volume, which equals $-k_{up}c$. To find $\boldsymbol{\Gamma}$, we use Fick's Law of diffusion:

$$\boldsymbol{\Gamma} = -D\nabla c$$

and since the system is spherically symmetric, the concentration gradient will be along the radial direction only. Hence,

$$\boldsymbol{\Gamma} = -D\frac{\partial c}{\partial r}\mathbf{n}_r$$

where $\mathbf{n}_r$ is the unit vector along the local radial axis. For our spherical shell,

$$
\begin{aligned}
-\int_S \boldsymbol{\Gamma}\cdot \mathrm{d}\mathbf{S} &= \int_S D\frac{\partial c}{\partial r}\mathbf{n}_r\cdot \mathrm{d}\mathbf{S}\\
&= \left[D\left(4\pi\rho^2\right)\frac{\partial c}{\partial \rho}\right]_{\rho=r+\Delta r} - \left[D\left(4\pi\rho^2\right)\frac{\partial c}{\partial \rho}\right]_{\rho=r}\\
&= 4\pi\frac{\left(\left[D\rho^2\frac{\partial c}{\partial \rho}\right]_{\rho=r+\Delta r} - \left[D\rho^2\frac{\partial c}{\partial \rho}\right]_{\rho=r}\right)}{\Delta r}\Delta r\\
&= 4\pi\left(\frac{\partial}{\partial r}\left[Dr^2\frac{\partial c}{\partial r}\right]\right)\Delta r \quad \text{(for sufficiently small } \Delta r\text{)}
\end{aligned}
$$

The rate of drug production within the shell is given by

$$
\begin{aligned}
\int_V f\,\mathrm{d}V &= \int_V -k_{up}c\,\mathrm{d}V\\
&= \int_r^{r+\Delta r} -\left(4\pi\rho^2 k_{up}c\right)\,\mathrm{d}\rho\\
&= -4\pi r^2 k_{up}c\Delta r \quad \text{(if } \Delta r \text{ is sufficiently small)}
\end{aligned}
$$

Substituting all these expressions into our earlier conservation law, we have:

$$
\begin{aligned}
\frac{\partial C}{\partial t} &= -\int_S \boldsymbol{\Gamma}\cdot \mathrm{d}\mathbf{S} + \int_V f\,\mathrm{d}V\\
4\pi r^2\Delta r\frac{\partial c}{\partial r} &= 4\pi\Delta r\frac{\partial}{\partial r}\left[Dr^2\frac{\partial c}{\partial r}\right] - 4\pi r^2 k_{up}c\Delta r
\end{aligned}
$$

Dividing all terms by $4\pi r^2\Delta r$, we obtain the required PDE:

$$
\frac{\partial c}{\partial t} = \frac{1}{r^2}\frac{\partial}{\partial r}\left[Dr^2\frac{\partial c}{\partial r}\right] - k_{up}c
$$

(b) Substituting $c = u/r$ into the above PDE, we obtain

$$
\begin{aligned}
\frac{1}{r}\frac{\partial u}{\partial t} &= \frac{1}{r^2}\frac{\partial}{\partial r}\left[Dr^2\left(\frac{1}{r}\frac{\partial u}{\partial r} - \frac{u}{r^2}\right)\right] - k_{up}\left(\frac{u}{r}\right)\\
&= \frac{1}{r^2}\frac{\partial}{\partial r}\left[Dr\frac{\partial u}{\partial r} - Du\right] - k_{up}\left(\frac{u}{r}\right)\\
&= \frac{1}{r^2}\left[D\frac{\partial u}{\partial r} + Dr\frac{\partial^2 u}{\partial r^2} - D\frac{\partial u}{\partial r}\right] - k_{up}\left(\frac{u}{r}\right)\\
&= \frac{1}{r^2}\left[Dr\frac{\partial^2 u}{\partial r^2}\right] - k_{up}\left(\frac{u}{r}\right)\\
&= \left(\frac{D}{r}\right)\frac{\partial^2 u}{\partial r^2} - k_{up}\left(\frac{u}{r}\right)
\end{aligned}
$$

Multiplying all terms by r, leads to the following simplified PDE in u:

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial r^2} - k_{up}u$$

To find the steady-state concentration, we set $\partial u/\partial t = 0$. The PDE is then transformed into the following ODE:

$$D\frac{\mathrm{d}^2 u}{\mathrm{d}r^2} - k_{up}u = 0$$

which has the characteristic equation

$$Dm^2 - k_{up} = 0$$

with roots $m = \pm\sqrt{k_{up}/D}$. Hence, its general solution is

$$u = C_1 \mathrm{e}^{r\sqrt{\frac{k_{up}}{D}}} + C_2 \mathrm{e}^{-r\sqrt{\frac{k_{up}}{D}}}$$

where C_1, C_2 are constants of integration. Since u cannot increase without bound as $r \to \infty$, we must have $C_1 = 0$. Furthermore, since the concentration c at $r = R$ (i.e. on the surface of the microsphere) equals c_0, u must therefore equal to $c_0 R$ at $r = R$. Hence,

$$\begin{aligned} u(R) = c_0 R &= C_2 \mathrm{e}^{-R\sqrt{\frac{k_{up}}{D}}} \\ \therefore \quad C_2 &= c_0 R \mathrm{e}^{R\sqrt{\frac{k_{up}}{D}}} \end{aligned}$$

Hence,

$$\begin{aligned} u &= c_0 R \mathrm{e}^{R\sqrt{\frac{k_{up}}{D}}} \mathrm{e}^{-r\sqrt{\frac{k_{up}}{D}}} \\ &= c_0 R \mathrm{e}^{-(r-R)\sqrt{\frac{k_{up}}{D}}} \end{aligned}$$

and since $c = u/r$, the steady-state concentration of c is given by:

$$c_\infty(r) = c_0 \left(\frac{R}{r}\right) \mathrm{e}^{-(r-R)\sqrt{\frac{k_{up}}{D}}} \qquad (r > R)$$

Including the region inside the microsphere, the complete solution for the steady-state concentration is

$$c_\infty(r) = \begin{cases} c_0 \left(\frac{R}{r}\right) \mathrm{e}^{-(r-R)\sqrt{\frac{k_{up}}{D}}} & r > R \\ c_0 & r \leq R \end{cases}$$

4.6 Let $c^n_{i,j}$ denote the amount of coins held by person (i, j) at time frame n. Each of the three rules for distribution yield an expression for the amount of coins held by person (i, j) at time frame $n + 1$, as follows:

(a) Denote the fixed fraction of coins given to each neighbour as α. Then,

$$\begin{aligned}
c^{n+1}_{i,j} &= c^n_{i,j} - \overbrace{4\alpha c^n_{i,j}}^{\text{giving to 4 neighbours}} + \overbrace{\alpha c^n_{i-1,j} + \alpha c^n_{i+1,j} + \alpha c^n_{i,j-1} + \alpha c^n_{i,j+1}}^{\text{receiving from 4 neighbours}} \\
&= c^n_{i,j} + \alpha c^n_{i-1,j} - 2\alpha c^n_{i,j} + \alpha c^n_{i+1,j} + \alpha c^n_{i,j-1} - 2\alpha c^n_{i,j} + \alpha c^n_{i,j+1} \\
c^{n+1}_{i,j} - c^n_{i,j} &= \alpha\left(c^n_{i-1,j} - 2c^n_{i,j} + c^n_{i+1,j} + c^n_{i,j-1} - 2c^n_{i,j} + c^n_{i,j+1}\right) \\
\Delta t\frac{c^{n+1}_{i,j} - c^n_{i,j}}{\Delta t} &= h^2\alpha\left(\frac{c^n_{i-1,j} - 2c^n_{i,j} + c^n_{i+1,j}}{h^2} + \frac{c^n_{i,j-1} - 2c^n_{i,j} + c^n_{i,j+1}}{h^2}\right) \\
\frac{c^{n+1}_{i,j} - c^n_{i,j}}{\Delta t} &= \frac{\alpha}{\mu}\left(\frac{c^n_{i-1,j} - 2c^n_{i,j} + c^n_{i+1,j}}{h^2} + \frac{c^n_{i,j-1} - 2c^n_{i,j} + c^n_{i,j+1}}{h^2}\right)
\end{aligned}$$

where $\mu = \Delta t/h^2$. The terms in this expression represent finite-difference approximations to first-order derivatives in time and second-order derivatives space. In the limit as $h, \Delta t \to 0$, keeping μ fixed, the expression reduces to the familiar diffusion PDE:

$$\frac{\partial c}{\partial t} = D\left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2}\right)$$

where the diffusion coefficient $D = \alpha/\mu$.

(b) Denote the fixed fraction of coins given to each neighbour by α, the income received per unit time by β. Then,

$$\begin{aligned}
c^{n+1}_{i,j} &= c^n_{i,j} - \overbrace{4\alpha c^n_{i,j}}^{\text{giving to 4 neighbours}} + \overbrace{\alpha c^n_{i-1,j} + \alpha c^n_{i+1,j} + \alpha c^n_{i,j-1} + \alpha c^n_{i,j+1}}^{\text{receiving from 4 neighbours}} + \overbrace{\beta\Delta t}^{\text{income}} \\
c^{n+1}_{i,j} - c^n_{i,j} &= \alpha c^n_{i-1,j} - 2\alpha c^n_{i,j} + \alpha c^n_{i+1,j} + \alpha c^n_{i,j-1} - 2\alpha c^n_{i,j} + \alpha c^n_{i,j+1} + \beta\Delta t \\
\Delta t\frac{c^{n+1}_{i,j} - c^n_{i,j}}{\Delta t} &= h^2\alpha\left(\frac{c^n_{i-1,j} - 2c^n_{i,j} + c^n_{i+1,j}}{h^2} + \frac{c^n_{i,j-1} - 2c^n_{i,j} + c^n_{i,j+1}}{h^2}\right) + \beta\Delta t \\
\frac{c^{n+1}_{i,j} - c^n_{i,j}}{\Delta t} &= \frac{\alpha}{\mu}\left(\frac{c^n_{i-1,j} - 2c^n_{i,j} + c^n_{i+1,j}}{h^2} + \frac{c^n_{i,j-1} - 2c^n_{i,j} + c^n_{i,j+1}}{h^2}\right) + \beta
\end{aligned}$$

where $\mu = \Delta t/h^2$. In the limit as $h, \Delta t \to 0$, keeping μ fixed, the expression reduces to the PDE:

$$\frac{\partial c}{\partial t} = D\left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2}\right) + \beta$$

where $D = \alpha/\mu$.

(c) Denote the fraction of coins received from each neighbour by α, and the proportion given to charity per unit time by β. Then,

$$c_{i,j}^{n+1} = c_{i,j}^n - \overbrace{4\alpha c_{i,j}^n}^{\text{giving to 4 neighbours}} + \overbrace{\alpha c_{i-1,j}^n + \alpha c_{i+1,j}^n + \alpha c_{i,j-1}^n + \alpha c_{i,j+1}^n}^{\text{receiving from 4 neighbours}} - \overbrace{\beta \Delta t c_{i,j}^n}^{\text{giving to charity}}$$

$$c_{i,j}^{n+1} - c_{i,j}^n = \alpha c_{i-1,j}^n - 2\alpha c_{i,j}^n + \alpha c_{i+1,j}^n + \alpha c_{i,j-1}^n - 2\alpha c_{i,j}^n + \alpha c_{i,j+1}^n - \beta \Delta t c_{i,j}^n$$

$$\Delta t \frac{c_{i,j}^{n+1} - c_{i,j}^n}{\Delta t} = h^2 \alpha \left(\frac{c_{i-1,j}^n - 2c_{i,j}^n + c_{i+1,j}^n}{h^2} + \frac{c_{i,j-1}^n - 2c_{i,j}^n + c_{i,j+1}^n}{h^2} \right) - \beta \Delta t c_{i,j}^n$$

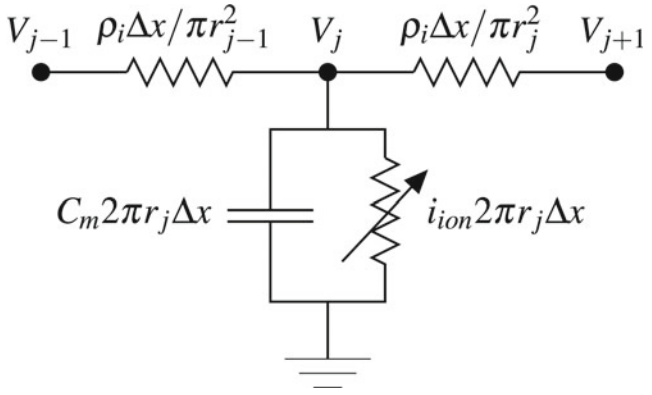
$$\frac{c_{i,j}^{n+1} - c_{i,j}^n}{\Delta t} = \frac{\alpha}{\mu} \left(\frac{c_{i-1,j}^n - 2c_{i,j}^n + c_{i+1,j}^n}{h^2} + \frac{c_{i,j-1}^n - 2c_{i,j}^n + c_{i,j+1}^n}{h^2} \right) - \beta c_{i,j}^n$$

where $\mu = \Delta t / h^2$. In the limit as $h, \Delta t \to 0$, keeping μ fixed, the expression reduces to the PDE:

$$\frac{\partial c}{\partial t} = D \left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2} \right) - \beta c$$

where $D = \alpha / \mu$.

4.7 Discretizing the nerve fibre into elements of length Δx, we obtain the following electrical equivalent circuit for one node, where r_j denotes the radius at node j:



From Kirchhoff's current law, the current flowing into node j must equal the current leaving it. Namely:

$$\overbrace{\frac{V_{j+1} - V_j}{\rho_i \Delta x / \pi r_j^2} + \frac{V_{j-1} - V_j}{\rho_i \Delta x / \pi r_{j-1}^2}}^{\text{current entering}} = \overbrace{C_m 2\pi r_j \Delta x \frac{\mathrm{d}V_j}{\mathrm{d}t} + i_{ion} 2\pi r_j \Delta x}^{\text{current leaving}}$$

$$\frac{\pi}{\rho_i} \left[r_j^2 \frac{V_{j+1} - V_j}{\Delta x} + r_{j-1}^2 \frac{V_{j-1} - V_j}{\Delta x} \right] = 2\pi r_j \Delta x \left(C_m \frac{\mathrm{d}V_j}{\mathrm{d}t} + i_{ion} \right)$$

$$\frac{1}{\Delta x} \left[\frac{r_j^2}{2\rho_i} \frac{V_{j+1} - V_j}{\Delta x} - \frac{r_{j-1}^2}{2\rho_i} \frac{V_j - V_{j-1}}{\Delta x} \right] = r_j \left(C_m \frac{\mathrm{d}V_j}{\mathrm{d}t} + i_{ion} \right)$$

In the limit as $\Delta x \to 0$, the above finite difference expressions approximate to the following PDE:

$$\frac{\partial}{\partial x}\left[\frac{r^2}{2\rho_i}\frac{\partial V}{\partial x}\right] = r\left(C_m\frac{\partial V}{\partial t} + i_{ion}\right)$$

or to be compatible with COMSOL's general PDE form:

$$rC_m\frac{\partial V}{\partial t} + \frac{\partial}{\partial x}\left[-\frac{r^2}{2\rho_i}\frac{\partial V}{\partial x}\right] = -ri_{ion}$$

4.8 To derive the underlying PDE for the muscle displacement, we discretize the muscle strand into N sub-units, each of length $\Delta x = L/N$. We can therefore characterise $N+1$ nodes, each having displacement u_i $(i = 0 \cdots N)$. Denoting the change in length of the i th sub-unit by l_i, and its corresponding l_1, l_2 values by l_{1i}, l_{2i} respectively. Then the following relationships hold:

$$\begin{aligned}
&l_i = l_{1i} + l_{2i} = u_i - u_{i-1} \\
&F_i = \left(\tfrac{k_1}{\Delta x}\right) l_{1i} && \text{for the series element} \\
&F_i = \left(\tfrac{k_2}{\Delta x}\right) l_{2i} + \left(\tfrac{b}{\Delta x}\right)\tfrac{\mathrm{d}l_{2i}}{\mathrm{d}t} && \text{for the parallel spring/dashpot}
\end{aligned}$$

where F_i is the passive force generated by the ith sub-unit, which acts on the point masses to oppose the changes in length. F_i is the same for both the series spring and the parallel spring/dashpot combination in the ith sub-unit, since these elements are in series. Re-arranging the last two of the above relationships, we obtain:

$$\begin{aligned}
l_{1i} &= \left(\frac{F_i}{k_1}\right)\Delta x \\
l_{2i} &= \left(\frac{F_i}{k_2}\right)\Delta x - \left(\frac{b}{k_2}\right)\frac{\mathrm{d}l_{2i}}{\mathrm{d}t}
\end{aligned}$$

Adding these together yields:

$$\begin{aligned}
l_{1i} + l_{2i} &= \left(\frac{1}{k_1} + \frac{1}{k_2}\right)F_i\Delta x - \left(\frac{b}{k_2}\right)\frac{\mathrm{d}l_{2i}}{\mathrm{d}t} \\
l_i &= \left(\frac{1}{k_1} + \frac{1}{k_2}\right)F_i\Delta x - \left(\frac{b}{k_2}\right)\frac{\mathrm{d}l_{2i}}{\mathrm{d}t} \\
&= \left(\frac{1}{k_1} + \frac{1}{k_2}\right)F_i\Delta x - \left(\frac{b}{k_2}\right)\left[\frac{\mathrm{d}l_i}{\mathrm{d}t} - \frac{\mathrm{d}l_{1i}}{\mathrm{d}t}\right]
\end{aligned}$$

and using the series element relationship $F_i = (k_1/\Delta x)l_{1i}$, we obtain $\mathrm{d}F_i/\mathrm{d}t = (k_1/\Delta x)\mathrm{d}l_{1i}/\mathrm{d}t$ and hence $\mathrm{d}l_{1i}/\mathrm{d}t = (\Delta x/k_1)\mathrm{d}F_i/\mathrm{d}t$. Substituting the latter into the above expression, we obtain:

$$l_i = \left(\frac{1}{k_1} + \frac{1}{k_2}\right) F_i \Delta x - \left(\frac{b}{k_2}\right)\left[\frac{\mathrm{d}l_i}{\mathrm{d}t} - \frac{\Delta x}{k_1}\frac{\mathrm{d}F}{\mathrm{d}t}\right]$$

$$l_i + \left(\frac{b}{k_2}\right)\frac{\mathrm{d}l_i}{\mathrm{d}t} = \left(\frac{1}{k_1} + \frac{1}{k_2}\right) F_i \Delta x + \left(\frac{b}{k_1 k_2}\right)\frac{\mathrm{d}F}{\mathrm{d}t}\Delta x$$

Multiplying throughout by $k_1 k_1/\Delta x$ and re-arranging terms, yields:

$$\left(\frac{1}{\Delta x}\right)\left[bk_1\frac{\mathrm{d}l_i}{\mathrm{d}t} + k_1 k_2 l_i\right] = b\frac{\mathrm{d}F_i}{\mathrm{d}t} + (k_1 + k_2)F_i$$

Finally, substituting $l_i = u_i - u_{i-1}$, we obtain:

$$\left(\frac{1}{\Delta x}\right)\left[bk_1\left(\frac{\mathrm{d}u_i}{\mathrm{d}t} - \frac{\mathrm{d}u_{i-1}}{\mathrm{d}t}\right) + k_1 k_2(u_i - u_{i-1})\right] = b\frac{\mathrm{d}F_i}{\mathrm{d}t} + (k_1 + k_2)F_i \quad \text{(B.1)}$$

A similar relation holds also for the $(i + 1)$th sub-unit:

$$\left(\frac{1}{\Delta x}\right)\left[bk_1\left(\frac{\mathrm{d}u_{i+1}}{\mathrm{d}t} - \frac{\mathrm{d}u_i}{\mathrm{d}t}\right) + k_1 k_2(u_{i+1} - u_i)\right] = b\frac{\mathrm{d}F_{i+1}}{\mathrm{d}t} + (k_1 + k_2)F_{i+1} \quad \text{(B.2)}$$

The total force acting on point mass i is given by

$$F_{i+1} - F_i = (M\Delta x)\frac{\mathrm{d}^2 u_i}{\mathrm{d}t^2}$$

Therefore, subtracting Eq. B.1 from Eq. B.2, we obtain:

$$\begin{aligned}\left(\frac{1}{\Delta x}\right)\Bigg[bk_1\left(\frac{\mathrm{d}u_{i+1}}{\mathrm{d}t} - \frac{\mathrm{d}u_i}{\mathrm{d}t} + \frac{\mathrm{d}u_{i-1}}{\mathrm{d}t}\right) &+ k_1 k_2(u_{i+1} - 2u_i + u_{i-1})\Bigg] \\ &= b\left(\frac{\mathrm{d}F_{i+1}}{\mathrm{d}t} - \frac{\mathrm{d}F_i}{\mathrm{d}t}\right) + (k_1 + k_2)\left[F_{i+1} - F_i\right] \\ &= bM\Delta x\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\mathrm{d}^2 u_i}{\mathrm{d}t^2}\right] + (k_1 + k_2)M\Delta x\frac{\mathrm{d}^2 u_i}{\mathrm{d}t^2}\end{aligned}$$

Dividing both sides by Δx:

$$\begin{aligned}bk_1\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta^2 x}\right] + k_1 k_2\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta^2 x}\right) \\ = bM\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\mathrm{d}^2 u_i}{\mathrm{d}t^2}\right] + (k_1 + k_2)M\frac{\mathrm{d}^2 u_i}{\mathrm{d}t^2}\end{aligned}$$

In the limit as $\Delta x \to 0$, this expression reduces to the PDE:

$$bk_1\frac{\partial}{\partial t}\left[\frac{\partial^2 u}{\partial x^2}\right] + k_1 k_2\frac{\partial^2 u}{\partial x^2} = bM\frac{\partial^3 u}{\partial t^3} + (k_1 + k_2)M\frac{\partial^2 u}{\partial t^2}$$

where x is the spatial coordinate along the length of the muscle. Re-arranging terms, the above PDE can also be re-written as:

$$b\frac{\partial}{\partial t}\left[k_1\frac{\partial^2 u}{\partial x^2} - M\frac{\partial^2 u}{\partial t^2}\right] + k_2\left(k_1\frac{\partial^2 u}{\partial x^2} - M\frac{\partial^2 u}{\partial t^2}\right) = k_1 M\frac{\partial^2 u}{\partial t^2}$$

4.9 (a) To solve the atrial tissue PDE system using the method of lines, we first discretize the 2D domain into a square grid of size $N \times N$. There will therefore be a total of N^2 individual nodes, with $2N^2$ variables to be solved for (i.e. V_m and u at each node). The following PDE must be discretized:

$$\begin{aligned}\beta C_m\left(\frac{\partial V_m}{\partial t}\right) &= \nabla\cdot(\sigma\nabla V_m) - \beta i_{ion} + i_{stim}\\ &= \sigma\left(\frac{\partial^2 V_m}{\partial x^2} + \frac{\partial^2 V_m}{\partial y^2}\right) - \beta i_{ion} + i_{stim} \quad \text{(since } \sigma \text{ is constant)}\\ \therefore \quad \frac{\partial V_m}{\partial t} &= \frac{\sigma}{\beta C_m}\left(\frac{\partial^2 V_m}{\partial x^2} + \frac{\partial^2 V_m}{\partial y^2}\right) - \frac{i_{ion}}{C_m} + \frac{i_{stim}}{\beta C_m}\end{aligned}$$

Denoting V_m and u for the (i, j)th node as $V_{i,j}$ and $u_{i,j}$ respectively, this PDE can be discretized as:

$$\begin{aligned}\frac{\mathrm{d}V_{i,j}}{\mathrm{d}t} &= \frac{\sigma}{\beta C_m}\left[\frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{h^2} + \frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{h^2}\right] - \frac{i_{ion,i,j}}{C_m} + \frac{i_{stim,i,j}}{\beta C_m}\\ &= \frac{\sigma}{\beta C_m}\left[\frac{V_{i,j+1} + V_{i,j-1} + V_{i+1,j} + V_{i-1,j} - 4V_{i,j}}{h^2}\right] - \frac{i_{ion,i,j}}{C_m} + \frac{i_{stim,i,j}}{\beta C_m}\end{aligned}$$

where $i_{ion,i,j}$ and $i_{stim,i,j}$ are i_ion and i_stim evaluated at node (i, j). To setup this system in Matlab, its in-built ode solvers require the independent variables to be in a single-column array format. Denoting this array by $\mathbf{Y}$, we can assign the first N^2 elements to the V_m variables and the final N^2 elements for the u, according the following "map":

$$\begin{aligned}V_{i,j} &= Y_{(i-1)N+j}\\ u_{i,j} &= Y_{N^2+(i-1)N+j}\end{aligned}$$

with $i, j = 1 \cdots N$. To implement the zero-flux boundary conditions on the external boundaries, we set the V_m value of points exterior to the boundary to its value at the adjacent boundary point. In Matlab, this can be achieved by padding the 2D V_m-array with extra rows and columns whose V_m values are equal to the adjacent boundary.

The Matlab code below[7] solves this PDE system using a spatial discretization of 51×51 nodes, plotting the solution for V_m at times 0.3, 0.35, 0.4, and 0.45 s, where it can be seen that a self-perpetuating reentrant spiral wave of activation is initiated by the stimulus protocol delivered.

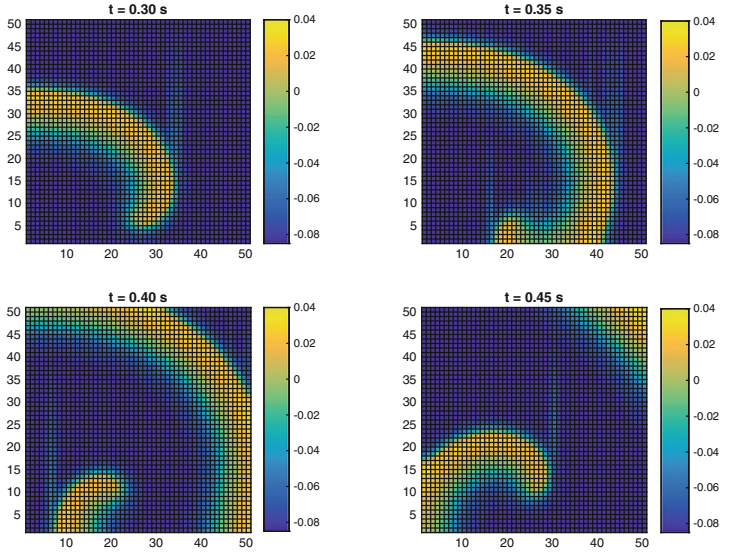
atrial_prime.m:

```
function Y_prime = atrial_prime(t,Y)
global beta sigma Cm a b c1 c2 A B d e N h
V = reshape(Y(1:N^2),N,N)';      % membrane potentials
U = reshape(Y(N^2+1:2*N^2),N,N)';  % u values
Y_prime = zeros(2*N^2,1);
% Next,"pad" the V array to implement zero-flux b.c.'s
VV = [V(1,1), V(1,1:N), V(1,N); ...
    V(1:N,1), V, V(1:N,N); ...
    V(N,1), V(N,1:N), V(N,N)];
% calculate derivatives
for i = 1:N
    for j=1:N
        % obtain Vm value of current node
        % and its four neighbours
        Vm = VV(i+1,j+1);    % padded indices
        Vm_left = VV(i+1,j);
        Vm_right = VV(i+1,j+2);
        Vm_below = VV(i,j+1);
        Vm_above = VV(i+2,j+1);
        % obtain remaining nodal variables
        u = U(i,j);
        x = (j-1)*h;         % x value
        y = (i-1)*h;         % y value
        i_ion = c1*(Vm-a)*(Vm-A)*(Vm-B)+c2*u*(Vm-B);
        if (t >= 0.01)&&(t<=0.011)
            if (y < 0.01)
                i_stim = 50;
            else
                i_stim = 0;
            end;
        elseif (t >= 0.15)&&(t<=0.151)
            if (x < 0.01)
                i_stim = 50;
            else
                i_stim = 0;
            end;
        else
            i_stim = 0;
        end;
        % determine derivative of Vm
        Y_prime((i-1)*N+j) = ...
            sigma/(beta*Cm)*(Vm_left+Vm_right+Vm_below+Vm_above-4*Vm)/(h^2)
...  - i_ion/Cm + i_stim/(beta*Cm);
        % determine derivative of u
        Y_prime(N^2+(i-1)*N+j) = e*(Vm-d*u-b);
    end;
end;
```

[7] This code took just over 12 min to solve using Matlab (R2014b) on a MacBook Air (2013) with 8 GB RAM.

atrial_solve_MOL.m

```
% solves atrial tissue electrical model
% using the method of lines
global beta sigma Cm a b c1 c2 A B d e N h
beta = 100;          % 1/m
sigma = 0.001;       % S/m
Cm = 0.01;           % F/m^2
a = -0.0668;         % V
b = -0.085;          % V
c1 = 530;            % S/(V*m)^2
c2 = 4;              % S/m^2
A = 0.055;           % V
B = -0.085;          % V
d = 0.14;            % V
e = 285.7;           % (1/(V*s)
N = 51;
h = 0.1/(N-1);
Y_init = zeros(2*N^2,1);
Y_init(1:N^2) = -0.085;
options = odeset('MaxStep',0.001);
[t_out, Y_out] = ode15s('atrial_prime', 0:0.001:0.5, Y_init, options);
% plot result
V_array_1 = Y_out(300,1:N^2);
V_out_1 = reshape(V_array_1,N,N);
V_array_2 = Y_out(350,1:N^2);
V_out_2 = reshape(V_array_2,N,N);
V_array_3 = Y_out(400,1:N^2);
V_out_3 = reshape(V_array_3,N,N);
V_array_4 = Y_out(450,1:N^2);
V_out_4 = reshape(V_array_4,N,N);
subplot(2,2,1), pcolor(V_out_1), colorbar, axis('square'), ...
    caxis([-0.085,0.04]), title('t = 0.30 s');
subplot(2,2,2), pcolor(V_out_2), colorbar, axis('square'), ...
    caxis([-0.085,0.04]), title('t = 0.35 s');
subplot(2,2,3), pcolor(V_out_3), colorbar, axis('square'), ...
    caxis([-0.085,0.04]), title('t = 0.40 s');
subplot(2,2,4), pcolor(V_out_4), colorbar, axis('square'), ...
    caxis([-0.085,0.04]), title('t = 0.45 s');
```



(b) To solve the above PDE system using an explicit finite difference scheme, we denote V_m and u for the (i, j)th node and nth time step by $V^n_{i,j}$ and $u^n_{i,j}$ respectively. The PDE can therefore be discretized as:

$$\frac{V^{n+1}_{i,j} - V^n_{i,j}}{\Delta t} = \frac{\sigma}{\beta C_m}\left[\frac{V^n_{i,j+1} + V^n_{i,j-1} + V^n_{i+1,j} + V^n_{i-1,j} - 4V^n_{i,j}}{h^2}\right] - \frac{i^n_{ion,i,j}}{C_m} + \frac{i^n_{stim,i,j}}{\beta C_m}$$

where h and Δt are the spacial resolution and time step respectively, and $i^n_{ion,i,j}$, $i^n_{stim,i,j}$ are i_{ion} and i_{stim} evaluated at node (i, j) and time step n. Re-arranging the above, we have:

$$V^{n+1}_{i,j} = V^n_{i,j} + \frac{\sigma \Delta t}{\beta C_m h^2}\left[V^n_{i,j+1} + V^n_{i,j-1} + V^n_{i+1,j} + V^n_{i-1,j} - 4V^n_{i,j}\right] - \frac{i^n_{ion,i,j}}{C_m} + \frac{i^n_{stim,i,j}}{\beta C_m}$$

Similarly for the u variables, we have

$$\frac{U^{n+1}_{i,j} - U^n_{i,j}}{\Delta t} = e\left(V^n_{i,j} - dU^n_{i,j} - b\right)$$
$$\therefore \quad U^{n+1}_{i,j} = U^n_{i,j} + e\left(V^n_{i,j} - dU^n_{i,j} - b\right)\Delta t$$

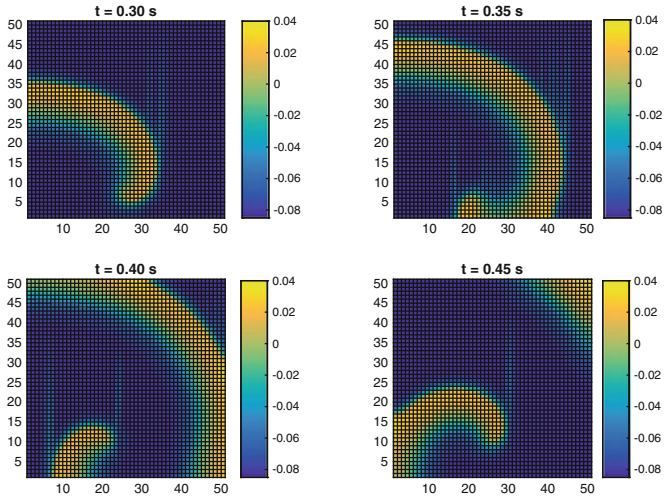
The Matlab code below implements the above explicit FD scheme for V_m and u, plotting the solution for V_m at $t = 0.3, 0.35, 0.4$, and 0.45 s. As in part a), a 'padded' V_m-array was used to implement the zero-flux boundary conditions. The code takes

advantage of Matlab's array processing abilities, and is much more rapid to solve for than the method of lines approach above.[8]

```
% solves atrial tissue electrical model
% using an explicit FD method
beta = 100;          % 1/m
sigma = 0.001;       % S/m
Cm = 0.01;           % F/m^2
a = -0.0668;         % V
b = -0.085;          % V
c1 = 530;            % S/(V*m)^2
c2 = 4;              % S/m^2
A = 0.055;           % V
B = -0.085;          % V
d = 0.14;            % V
e = 285.7;           % (1/(V*s)
N = 51;
h = 0.1/(N-1);       % m
Dt = 1e-5;           % s
% initialise Vm and u
V = ones(N)*-0.085;
U = zeros(N);
% Next, "pad" the V array to implement zero-flux b.c.'s
VV = [V(1,1), V(1,1:N), V(1,N); ...
    V(1:N,1), V, V(1:N,N); ...
    V(N,1), V(N,1:N), V(N,N)];
% initialise spatial coordinates
x = zeros(N);
y = zeros(N);
for j=2:N
    x(:,j) = x(:,j-1)+h;
end; for i=2:N
    y(i,:) = y(i-1,:)+h;
end;
% explicit FD time stepping loop
for t = 0:Dt:0.5
    i_ion = c1*(V-a).*(V-A).*(V-B)+c2*U.*(V-B);
    if (t >= 0.01)&&(t<=0.011)
        i_stim = 50*(y<0.01);
    elseif (t >= 0.15)&&(t<=0.151)
        i_stim = 50*(y<0.01);
    else
        i_stim = zeros(N);
    end;
    V = V + sigma*Dt/(beta*Cm*h^2)*(VV(2:N+1,1:N)+VV(2:N+1,3:N+2) + ...
        VV(1:N,2:N+1)+VV(3:N+2,2:N+1)-4*VV(2:N+1,2:N+1) ...
            - i_ion/Cm + i_stim/(beta*Cm));
    U = U + e*(VV(2:N+1,2:N+1)-d*U-b)*Dt;
    if (t == 0.3)
        V_array_1 = V;
    elseif (t == 0.35)
        V_array_2 = V;
```

[8] This FD scheme took just over 4 s to solve for and plot using Matlab (R2014b) on a MacBook Air (2013) with 8 GB RAM.

```
    elseif (t == 0.4)
        V_array_3 = V;
    elseif (t == 0.45)
        V_array_4 = V;
    end;
end;
% plot result
subplot(2,2,1), pcolor(V_out_1), colorbar, axis('square'), ...
    caxis([-0.085,0.04]), title('t = 0.30 s');
subplot(2,2,2), pcolor(V_out_2), colorbar, axis('square'), ...
    caxis([-0.085,0.04]), title('t = 0.35 s');
subplot(2,2,3), pcolor(V_out_3), colorbar, axis('square'), ...
    caxis([-0.085,0.04]), title('t = 0.40 s');
subplot(2,2,4), pcolor(V_out_4), colorbar, axis('square'), ...
    caxis([-0.085,0.04]), title('t = 0.45 s');
```



4.10 (a) We can write the voltage distribution $V(x, y)$ in the Petri dish in terms of polar coordinates $V(r, \theta)$, to take advantage of the circular geometry of the problem, where $r = \sqrt{x^2 + y^2}$ and $\theta = \cos^{-1}(x/r)$. The relevant PDE is:

$$\nabla \cdot (\sigma \nabla V) = 0$$

and since the conductivity σ is constant throughout the domain, we can divide through by σ to obtain:

$$\nabla \cdot (\nabla V) = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0$$

To express this PDE in terms of r and θ polar coordinates, we first note that

$$\begin{aligned}
\frac{\partial r}{\partial x} &= \frac{\partial}{\partial x}\left[\sqrt{x^2+y^2}\right] \\
&= \frac{x}{\sqrt{x^2+y^2}} \\
&= \frac{x}{r} \\
\frac{\partial \theta}{\partial x} &= \frac{\partial}{\partial x}\left[\cos^{-1}\left(\frac{x}{r}\right)\right] \\
&= \frac{-1}{\sqrt{1-\frac{x^2}{r^2}}}\frac{\partial}{\partial x}\left[\frac{x}{r}\right] \\
&= \frac{-1}{\sqrt{\frac{r^2}{r^2}-\frac{x^2}{r^2}}}\left[\frac{1}{r}-\frac{x}{r^2}\frac{\partial r}{\partial x}\right] \\
&= \frac{-1}{\sqrt{\frac{y^2}{r^2}}}\left[\frac{1}{r}-\left(\frac{x}{r^2}\right)\left(\frac{x}{r}\right)\right] \\
&= -\frac{r}{y}\left[\frac{1}{r}-\frac{x^2}{r^3}\right] \\
&= -\frac{1}{y}\left[1-\frac{x^2}{r^2}\right] \\
&= -\frac{1}{y}\left[\frac{r^2}{r^2}-\frac{x^2}{r^2}\right] \\
&= -\frac{1}{y}\left[\frac{y^2}{r^2}\right] \\
&= -\frac{y}{r^2}
\end{aligned}$$

Using a similar evaluations, we find that

$$\frac{\partial r}{\partial y} = \frac{y}{r} \qquad \frac{\partial \theta}{\partial y} = \frac{x}{r^2}$$

Using the above expressions, we can now evaluate the first term of the PDE, $\partial^2 V/\partial x^2$. We begin with:

$$\begin{aligned}
\frac{\partial V}{\partial x} &= \frac{\partial V}{\partial r}\frac{\partial r}{\partial x}+\frac{\partial V}{\partial \theta}\frac{\partial \theta}{\partial x} && \text{(using the chain rule)} \\
&= \frac{x}{r}\frac{\partial V}{\partial r}-\frac{y}{r^2}\frac{\partial V}{\partial \theta} && \text{(using the previous expressions)}
\end{aligned}$$

And taking one more derivative, we have:

$$\frac{\partial^2 V}{\partial x^2} = \frac{\partial}{\partial x}\left[\frac{x}{r}\frac{\partial V}{\partial r}\right] - \frac{\partial}{\partial x}\left[\frac{y}{r^2}\frac{\partial V}{\partial \theta}\right]$$

Evaluating each of the terms on the right-hand side separately, we have:

$$\begin{aligned}
\frac{\partial}{\partial x}\left[\frac{x}{r}\frac{\partial V}{\partial r}\right] &= \frac{\partial}{\partial r}\left[\frac{x}{r}\frac{\partial V}{\partial r}\right]\frac{\partial r}{\partial x} + \frac{\partial}{\partial \theta}\left[\frac{x}{r}\frac{\partial V}{\partial r}\right]\frac{\partial \theta}{\partial x} \\
&= \frac{x}{r}\left\{\left[-\frac{x}{r^2} + \frac{1}{r}\frac{\partial x}{\partial r}\right]\frac{\partial V}{\partial r} + \frac{x}{r}\frac{\partial^2 V}{\partial r^2}\right\} - \frac{y}{r^2}\left\{\left[\frac{1}{r}\frac{\partial x}{\partial \theta}\right]\frac{\partial V}{\partial r} + \frac{x}{r}\frac{\partial^2 V}{\partial r \partial \theta}\right\} \\
&= \frac{x}{r}\left\{\left[-\frac{x}{r^2} + \frac{1}{r}\cos\theta\right]\frac{\partial V}{\partial r} + \frac{x}{r}\frac{\partial^2 V}{\partial r^2}\right\} - \frac{y}{r^2}\left\{\left[-\frac{1}{r}r\sin\theta\right]\frac{\partial V}{\partial r} + \frac{x}{r}\frac{\partial^2 V}{\partial r \partial \theta}\right\} \\
&= \frac{x}{r}\left\{\left[-\frac{x}{r^2} + \frac{1}{r}\frac{x}{r}\right]\frac{\partial V}{\partial r} + \frac{x}{r}\frac{\partial^2 V}{\partial r^2}\right\} - \frac{y}{r^2}\left\{\left[-\frac{1}{r}r\frac{y}{r}\right]\frac{\partial V}{\partial r} + \frac{x}{r}\frac{\partial^2 V}{\partial r \partial \theta}\right\} \\
&= \frac{y^2}{r^3}\frac{\partial V}{\partial r} + \frac{x^2}{r^2}\frac{\partial^2 V}{\partial r^2} - \frac{xy}{r^3}\frac{\partial^2 V}{\partial r \partial \theta} \\
\frac{\partial}{\partial x}\left[\frac{y}{r^2}\frac{\partial V}{\partial \theta}\right] &= \frac{\partial}{\partial r}\left[\frac{y}{r^2}\frac{\partial V}{\partial \theta}\right]\frac{\partial r}{\partial x} + \frac{\partial}{\partial \theta}\left[\frac{y}{r^2}\frac{\partial V}{\partial \theta}\right]\frac{\partial \theta}{\partial x} \\
&= \frac{x}{r}\left\{\left[\frac{1}{r^2}\frac{\partial y}{\partial r} - \frac{2y}{r^3}\right]\frac{\partial V}{\partial \theta} + \frac{y}{r^2}\frac{\partial^2 V}{\partial r \partial \theta}\right\} - \frac{y}{r^2}\left\{\left[\frac{1}{r^2}\frac{\partial y}{\partial \theta}\right]\frac{\partial V}{\partial \theta} + \frac{y}{r^2}\frac{\partial^2 V}{\partial \theta^2}\right\} \\
&= \frac{x}{r}\left\{\left[\frac{1}{r^2}\sin\theta - \frac{2y}{r^3}\right]\frac{\partial V}{\partial \theta} + \frac{y}{r^2}\frac{\partial^2 V}{\partial r \partial \theta}\right\} - \frac{y}{r^2}\left\{\left[\frac{1}{r^2}r\cos\theta\right]\frac{\partial V}{\partial \theta} + \frac{y}{r^2}\frac{\partial^2 V}{\partial \theta^2}\right\} \\
&= \frac{x}{r}\left\{\left[\frac{1}{r^2}\frac{y}{r} - \frac{2y}{r^3}\right]\frac{\partial V}{\partial \theta} + \frac{y}{r^2}\frac{\partial^2 V}{\partial r \partial \theta}\right\} - \frac{y}{r^2}\left\{\left[\frac{1}{r^2}r\frac{x}{r}\right]\frac{\partial V}{\partial \theta} + \frac{y}{r^2}\frac{\partial^2 V}{\partial \theta^2}\right\} \\
&= \frac{x}{r}\left\{-\frac{y}{r^3}\frac{\partial V}{\partial \theta} + \frac{y}{r^2}\frac{\partial^2 V}{\partial r \partial \theta}\right\} - \frac{y}{r^2}\left\{\frac{x}{r^2}\frac{\partial V}{\partial \theta} + \frac{y}{r^2}\frac{\partial^2 V}{\partial \theta^2}\right\} \\
&= -\frac{xy}{r^4}\frac{\partial V}{\partial \theta} + \frac{xy}{r^3}\frac{\partial^2 V}{\partial r \partial \theta} - \frac{xy}{r^4}\frac{\partial V}{\partial \theta} - \frac{y^2}{r^4}\frac{\partial^2 V}{\partial \theta^2} \\
&= -\frac{2xy}{r^4}\frac{\partial V}{\partial \theta} + \frac{xy}{r^3}\frac{\partial^2 V}{\partial r \partial \theta} - \frac{y^2}{r^4}\frac{\partial^2 V}{\partial \theta^2}
\end{aligned}$$

Combining both terms together, we obtain

$$\frac{\partial^2 V}{\partial x^2} = \frac{y^2}{r^3}\frac{\partial V}{\partial r} + \frac{2xy}{r^4}\frac{\partial V}{\partial \theta} - \frac{2xy}{r^3}\frac{\partial^2 V}{\partial r \partial \theta} + \frac{x^2}{r^2}\frac{\partial^2 V}{\partial r^2} + \frac{y^2}{r^4}\frac{\partial^2 V}{\partial \theta^2}$$

After a similar lengthy derivation, we also obtain for $\partial^2 V/\partial y^2$:

$$\frac{\partial^2 V}{\partial y^2} = \frac{x^2}{r^3}\frac{\partial V}{\partial r} - \frac{2xy}{r^4}\frac{\partial V}{\partial \theta} + \frac{2xy}{r^3}\frac{\partial^2 V}{\partial r \partial \theta} + \frac{y^2}{r^2}\frac{\partial^2 V}{\partial r^2} + \frac{x^2}{r^4}\frac{\partial^2 V}{\partial \theta^2}$$

Adding both of the above second-order partial derivatives, we obtain:

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = \frac{y^2 + x^2}{r^3}\frac{\partial V}{\partial r} + \frac{x^2 + y^2}{r^2}\frac{\partial^2 V}{\partial r^2} + \frac{y^2 + x^2}{r^4}\frac{\partial^2 V}{\partial \theta^2}$$
$$= \frac{1}{r}\frac{\partial V}{\partial r} + \frac{\partial^2 V}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2 V}{\partial \theta^2}$$

Substituting this into the PDE, we obtain

$$\frac{1}{r}\frac{\partial V}{\partial r} + \frac{\partial^2 V}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2 V}{\partial \theta^2} = 0$$

(b) We can discretize the above PDE using a regular grid over (r, θ) coordinate space. Furthermore, due to symmetry of the problem about the x-axis, we will utilise only the half-domain $\theta \in [0, \pi]$, $r \in [0, R]$, such that for the (i, j)th node, we have:

$$\begin{aligned} r_i &= i\Delta r \qquad 0 = 1 \cdots N \\ \theta_j &= j\Delta\theta \qquad 0 = 1 \cdots M \end{aligned}$$

where Δr, $\Delta\theta$ are the spatial resolutions in r and θ respectively, with the total number of grid points in r and θ given by $N + 1$, $M + 1$ respectively, such that

$$\Delta r = \frac{R}{N} \qquad \text{and} \qquad \Delta\theta = \frac{\pi}{M}$$

Denoting the value of V at the (i, j)th node by $V_{i,j}$, the above PDE can be discretized using finite difference approximations of the derivatives as follows:

$$\frac{1}{r_i}\left(\frac{V_{i+1,j} - V_{i,j}}{\Delta r}\right) + \left(\frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{\Delta^2 r}\right) + \frac{1}{r_i^2}\left(\frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{\Delta^2\theta}\right) = 0$$

This approximation, however, cannot be used at the centre of the domain, since evaluation at $r_i = 0$ (i.e. $i = 0$) will lead to a singular value in the expression.[9] However, for $i \geq 1$ we can continue to expand the above as follows:

$$\frac{1}{i\Delta r}\left(\frac{V_{i+1,j} - V_{i,j}}{\Delta r}\right) + \left(\frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{\Delta^2 r}\right) + \frac{1}{i^2\Delta^2 r}\left(\frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{\Delta^2\theta}\right) = 0$$

Multiplying all terms by $i^2\Delta^2 r\Delta^2\theta$, we obtain:

$$i\Delta^2\theta\left(V_{i+1,j} - V_{i,j}\right) + i^2\Delta^2\theta\left(V_{i+1,j} - 2V_{i,j} + V_{i-1,j}\right) + \left(V_{i,j+1} - 2V_{i,j} + V_{i,j-1}\right) = 0$$

or

$$\left(i^2\Delta^2\theta + i\Delta^2\theta\right)V_{i+1,j} + \left(i^2\Delta^2\theta\right)V_{i-1,j} + V_{i,j+1} + V_{i,j-1} - \left(2i^2\Delta^2\theta + i\Delta^2\theta + 2\right)V_{i,j} = 0$$

[9] This property is a consequence of the choice of polar coordinates, raher than any actual discontinuity in the solution for V at the centre of the domain.

At the centre of the Petri dish, where $i = 0$, we can assign V to equal the mean value of all surrounding nodes, namely

$$V_{0,j} = \frac{1}{M+1} \sum_{j=0}^{M} V_{1,j} \qquad j = 0 \cdots M$$

or more simply,

$$(M+1)V_{0,j} - \sum_{j=0}^{M} V_{1,j} = 0 \qquad j = 0 \cdots M$$

Furthermore, along the walls of the dish (i.e. at $i = N$), the following boundary conditions are enforced:

$$V_{N,j} = \begin{cases} V_0 & j\Delta\theta \leq \frac{\theta_0}{2} \quad \text{(active electrode)} \\ 0 & j\Delta\theta \geq \pi - \frac{\theta_0}{2} \quad \text{(ground electrode)} \\ V_{N-1,j} & \frac{\theta_0}{2} < j\Delta\theta < \pi - \frac{\theta_0}{2} \quad \text{(zero} - \text{flux boundaries)} \end{cases}$$

Finally, due to the symmetry of the problem, there is a zero-flux boundary condition for V on the lower edge of the half-domain, corresponding the x-axis, i.e. when $j = 0$ and M. Hence,

$$V_{i,0} = V_{i,1} \qquad \text{and} \qquad V_{i,M} = V_{i,M-1}$$

To solve for the individual $V_{i,j}$ using all the above relationships, we must set up a matrix system of the form

$$\mathbf{Ay} = \mathbf{b}$$

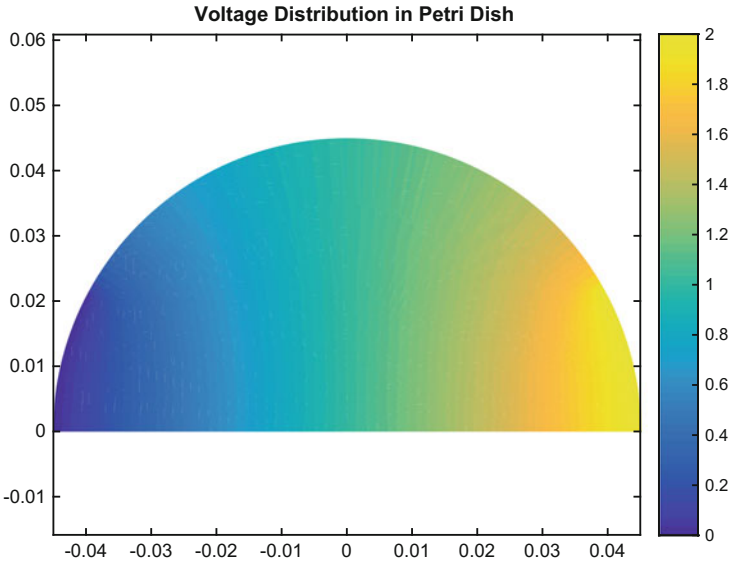
where $\mathbf{y}$ is a column vector of length $(N+1)(M+1)$ containing the unknown $V_{i,j}$, $\mathbf{A}$ is an $(N+1)(M+1) \times (N+1)(M+1)$ matrix, and $\mathbf{b}$ is also a column vector of length $(N+1)(M+1)$. To map the $V_{i,j}$ to the elements of $\mathbf{y}$, we can use:

$$y_{1+j+i(M+1)} = V_{i,j}$$

The Matlab code below implements the above equations for $N, M = 100$, solves the resulting matrix system, and plots the voltage distribution in the symmetric half-domain:

```
% solves for the Petri dish voltage distribution using
% finite differences on a polar coordinate grid
R = 0.045;                                  % m
theta_0 = pi/3;                             % radians
V_0 = 2;                                    % V
N = 100;
M = 100;
```

```
D_r = R/N;
D_theta = pi/M;
B = zeros((N+1)*(M+1),1);
A = sparse((N+1)*(M+1),(N+1)*(M+1));
for i = 0:N
    for j= 0:M
        index = 1+j+i*(M+1);
        if (i == 0)     % i.e. at centre of dish
            A(index,index) = M+1;
            A(index,1+(0:M)+M+1) = -1;
        elseif (i == N) % i.e. on circular boundary
            theta = j*D_theta;
            if (theta <= theta_0/2)
                % active electrode
                A(index,index) = 1;
                B(index) = V_0;
            elseif (theta >= pi-theta_0/2)
                % ground electrode
                A(index,index) = 1;
            else
                % zero-flux boundaries
                A(index,index) = 1;
                A(index,1+j+(N-1)*(M+1)) = -1;
            end;
        elseif (j == 0) % i.e. on symmetric boundary
            A(index,index) = 1;
            A(index,1+1+i*(M+1)) = -1;
        elseif (j == M) % i.e. also on symmetric boundary
            A(index,index) = 1;
            A(index,1+M-1+i*(M+1)) = -1;
        else            % everywhere else
            A(index,index) = -(2*i^2*D_theta^2+i*D_theta^2+2);
            A(index,1+j+(i+1)*(M+1)) = i^2*D_theta^2+i*D_theta^2;
            A(index,1+j+(i-1)*(M+1)) = i^2*D_theta^2;
            A(index,1+j+1+i*(M+1)) = 1;
            A(index,1+j-1+i*(M+1)) = 1;
        end;
    end;
end;
% solve matrix system
Y = A\B; VV = reshape(Y,N+1,M+1);
% plot solution
[RR, WW] = meshgrid(0:D_r:R, 0:D_theta:pi);
XX = RR.*cos(WW);
YY = RR.*sin(WW);
pcolor(XX,YY,VV), colorbar, axis('equal'),...
    shading('interp'),
    title('Voltage Distribution in Petri Dish');
```



(c) The electric field at all points within the dish is given by

$$\mathbf{E} = -\sigma \nabla V = -\sigma \begin{pmatrix} \frac{\partial V}{\partial x} \\ \frac{\partial V}{\partial y} \end{pmatrix}$$

In particular, the electric field magnitude E is given by

$$E = \sigma \sqrt{\left(\frac{\partial V}{\partial x}\right)^2 + \left(\frac{\partial V}{\partial y}\right)^2}$$

From part (a), these derivatives can be expressed in polar coordinates as

$$\begin{aligned} \frac{\partial V}{\partial x} &= \frac{x}{r}\frac{\partial V}{\partial r} - \frac{y}{r^2}\frac{\partial V}{\partial \theta} = \cos\theta \frac{\partial V}{\partial r} - \frac{\sin\theta}{r}\frac{\partial V}{\partial \theta} \\ \frac{\partial V}{\partial y} &= \frac{y}{r}\frac{\partial V}{\partial r} + \frac{x}{r^2}\frac{\partial V}{\partial \theta} = \sin\theta \frac{\partial V}{\partial r} + \frac{\cos\theta}{r}\frac{\partial V}{\partial \theta} \end{aligned}$$

Therefore,

$$\begin{aligned} E &= \sigma \sqrt{\left(\cos\theta \frac{\partial V}{\partial r} - \frac{\sin\theta}{r}\frac{\partial V}{\partial \theta}\right)^2 + \left(\sin\theta \frac{\partial V}{\partial r} + \frac{\cos\theta}{r}\frac{\partial V}{\partial \theta}\right)^2} \\ &= \sigma \sqrt{\begin{array}{l} \cos^2\theta \left(\frac{\partial V}{\partial r}\right)^2 - \frac{2\sin\theta\cos\theta}{r}\left(\frac{\partial V}{\partial r}\right)\left(\frac{\partial V}{\partial \theta}\right) + \frac{\sin^2\theta}{r^2}\left(\frac{\partial V}{\partial \theta}\right)^2 \\ + \sin^2\theta \left(\frac{\partial V}{\partial r}\right)^2 - \frac{2\sin\theta\cos\theta}{r}\left(\frac{\partial V}{\partial r}\right)\left(\frac{\partial V}{\partial \theta}\right) + \frac{\cos^2\theta}{r^2}\left(\frac{\partial V}{\partial \theta}\right)^2 \end{array}} \end{aligned}$$

$$= \sigma \sqrt{\left(\frac{\partial V}{\partial r}\right)^2 - \frac{2\sin 2\theta}{r}\left(\frac{\partial V}{\partial r}\right)\left(\frac{\partial V}{\partial \theta}\right) + \frac{1}{r^2}\left(\frac{\partial V}{\partial \theta}\right)^2}$$

Due to symmetry of the voltage distribution about the x-axis, the gradient of the voltage at the centre of the dish (and hence the electric field) must be parallel to the x-axis itself. Hence, at the centre of the dish, the electric field is given by:

$$\mathbf{E}|_{r=0} = -\sigma \left.\begin{pmatrix} \frac{\partial V}{\partial x} \\ 0 \end{pmatrix}\right|_{x=0,\ y=0}$$

and its magnitude, E_c, is equal to

$$E_c = \sigma \left|\frac{\partial V}{\partial r}\right|_{r=0,\ \theta=0}$$

since the x-axis is aligned with the $\theta = 0$ axis at $r = 0$. Hence, the electric field magnitude in the dish relative to the centre is given by

$$\frac{E}{E_c} = \frac{\sqrt{\left(\frac{\partial V}{\partial r}\right)^2 - \frac{2\sin 2\theta}{r}\left(\frac{\partial V}{\partial r}\right)\left(\frac{\partial V}{\partial \theta}\right) + \frac{1}{r^2}\left(\frac{\partial V}{\partial \theta}\right)^2}}{\left|\frac{\partial V}{\partial r}\right|_{r=0,\ \theta=0}}$$

Defining a scalar function $\Gamma(r, \theta)$ such that

$$\Gamma(r, \theta) = \begin{cases} 1 \ 0.9 \leq \frac{E}{E_c} \leq 1.1 \\ 0 \text{ otherwise} \end{cases}$$

then the area of the dish for which the electric field magnitude is $\pm 10\%$ of E_c is giben by

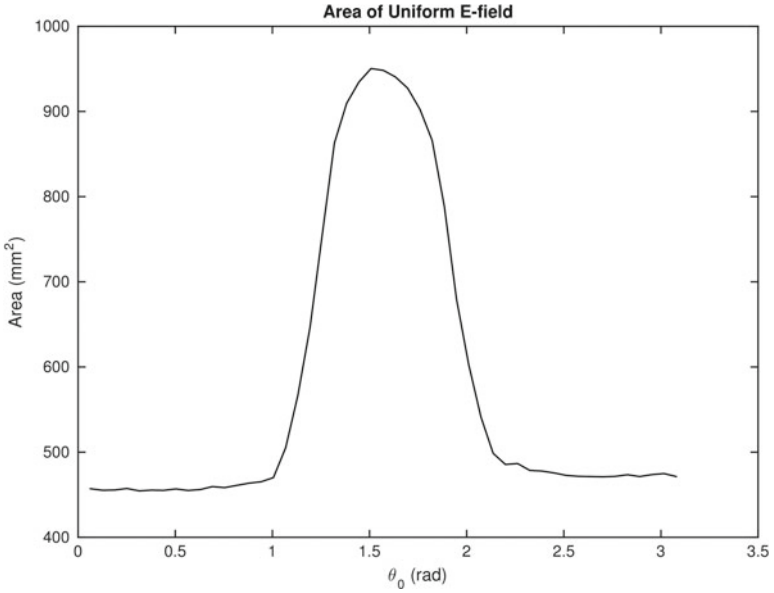
$$\text{Area} = \int_0^{\pi} \int_0^{R} \Gamma(r, \theta) r \, \mathrm{d}r \, \mathrm{d}\theta$$

The Matlab code below approximates this integral by summing the integrand over all elements of the 2D voltage polar-grid array VV, plotting the area against the electrode angle θ_0:

```
% Determines the area for which electric field
% magnitude is with +/- of its value at the
% the centre of the Petri dish, for a range of
% electrode angles.

R = 0.045;                              % m
V_0 = 2;                                % V
N = 100;
M = 100;
D_r = R/N;
```

```
D_theta = pi/M;
Area_array = [];
for theta_0 = pi/50:pi/50:49*pi/50
    B = zeros((N+1)*(M+1),1);
    A = sparse((N+1)*(M+1),(N+1)*(M+1));
    for i = 0:N
        for j= 0:M
            index = 1+j+i*(M+1);
            if (i == 0)     % i.e. at centre of dish
                A(index,index) = M+1;
                A(index,1+(0:M)+M+1) = -1;
            elseif (i == N) % i.e. on circular boundary
                theta = j*D_theta;
                if (theta <= theta_0/2)
                    % active electrode
                    A(index,index) = 1;
                    B(index) = V_0;
                elseif (theta >= pi-theta_0/2)
                    % ground electrode
                    A(index,index) = 1;
                else
                    % zero-flux boundaries
                    A(index,index) = 1;
                    A(index,1+j+(N-1)*(M+1)) = -1;
                end;
            elseif (j == 0) % i.e. on symmetric boundary
                A(index,index) = 1;
                A(index,1+1+i*(M+1)) = -1;
            elseif (j == M) % i.e. also on symmetric boundary
                A(index,index) = 1;
                A(index,1+M-1+i*(M+1)) = -1;
            else            % everywhere else
                A(index,index) = -(2*i^2*D_theta^2+i*D_theta^2+2);
                A(index,1+j+(i+1)*(M+1)) = i^2*D_theta^2+i*D_theta^2;
                A(index,1+j+(i-1)*(M+1)) = i^2*D_theta^2;
                A(index,1+j+1+i*(M+1)) = 1;
                A(index,1+j-1+i*(M+1)) = 1;
            end;
        end;
    end;
    % solve matrix system
    Y = A\B;
    VV = reshape(Y,N+1,M+1)';
    % Determine E-field magnitude at centre of disk using
    % finite difference approximation (normalised to sigma = 1)
    Ec = abs(VV(2,1)-VV(1,1))/D_r;
    % Determine Gamma coefficient
    E_ratio = ones(N+1,M+1);
    Gamma = zeros(N+1,M+1);
    for i = 2:N+1
        r = (i-1)*D_r;
        for j = 1:M+1
            theta = (j-1)*D_theta;
            dVdr = (VV(i,j)-VV(i-1,j))/D_r;
            if (j==1)
                dVdtheta = 0;
            else
                dVdtheta = (VV(i,j)- VV(i,j-1))/D_theta;
            end;
```

```
            E_ratio(i,j) = sqrt(dVdr^2-2*sin(2*theta)/r*dVdr*dVdtheta + ...
                dVdtheta^2/r^2)/Ec;
            if ((E_ratio(i,j)>=0.9)&&(E_ratio(i,j)<=1.1))
                Gamma(i,j) = 1;
            end;
        end;
    end;
    % determine area of uniform E-field
    Area = 0;
    for i = 1:N+1
        r = i*D_r;
        for j = 1:M+1
            Area = Area + Gamma(i,j)*r*D_r*D_theta;
        end;
    end;
    Area_array = [Area_array, Area];
end;
% plot solution
plot(pi/50:pi/50:49*pi/50, Area_array*1e6, 'k'), xlabel('\theta_0 (rad)'),
 ...     ylabel('Area (mm^2)'), title('Area of Uniform E-field');
```



From the plot produced by this code (shown above), a maximum uniform electric field area of approximately 950 mm^2 is attained for an electrode angle of around $24\pi/50$ radians, corresponding to $\theta_0 \approx 86°$.

Problems of Chap. 5

5.1 To solve the PDE, we can make use of the system matrices of Eqs. 5.23 and 5.24, determined at the local element level, namely:

$$K_{e,ij} = \int_{\Omega} (\kappa \nabla \varphi_i) \cdot \left(\nabla \varphi_j\right) \, \mathrm{d}V$$
$$f_{e,j} = \int_{\Omega} f \varphi_j \, \mathrm{d}V$$

where $i, j = 1, 2$, $\kappa = 1$, and $f = 2$. Using 1D Lagrange shape functions, we have $\varphi_1 = (1 - \xi)$ and $\varphi_2 = \xi$. The (1,1) component of the element stiffness matrix can be determined from

$$K_{e,11} = h \int_0^1 \left(\frac{1}{h}\right) \frac{\partial \varphi_1(\xi)}{\partial \xi} \left(\frac{1}{h}\right) \frac{\partial \varphi_1(\xi)}{\partial \xi} \, \mathrm{d}\xi$$

where h denotes the element size, and the various h factors convert the shape function derivatives and integral from the local element to the spatial frame. Hence,

$$\begin{aligned} K_{e,11} &= h \int_0^1 \left(\frac{1}{h}\right) (-1) \left(\frac{1}{h}\right) (-1) \, \mathrm{d}\xi \\ &= \frac{1}{h} \end{aligned}$$

Similarly for the other components,

$$\begin{aligned} K_{e,12} &= h \int_0^1 \left(\frac{1}{h}\right) \frac{\partial \varphi_1(\xi)}{\partial \xi} \left(\frac{1}{h}\right) \frac{\partial \varphi_2(\xi)}{\partial \xi} \, \mathrm{d}\xi \\ &= h \int_0^1 \left(\frac{1}{h}\right) (-1) \left(\frac{1}{h}\right) (1) \, \mathrm{d}\xi \\ &= -\frac{1}{h} \\ K_{e,21} &= h \int_0^1 \left(\frac{1}{h}\right) \frac{\partial \varphi_2(\xi)}{\partial \xi} \left(\frac{1}{h}\right) \frac{\partial \varphi_1(\xi)}{\partial \xi} \, \mathrm{d}\xi \\ &= h \int_0^1 \left(\frac{1}{h}\right) (1) \left(\frac{1}{h}\right) (-1) \, \mathrm{d}\xi \\ &= -\frac{1}{h} \\ K_{e,22} &= h \int_0^1 \left(\frac{1}{h}\right) \frac{\partial \varphi_2(\xi)}{\partial \xi} \left(\frac{1}{h}\right) \frac{\partial \varphi_2(\xi)}{\partial \xi} \, \mathrm{d}\xi \\ &= h \int_0^1 \left(\frac{1}{h}\right) (1) \left(\frac{1}{h}\right) (1) \, \mathrm{d}\xi \\ &= \frac{1}{h} \end{aligned}$$

For the element load vector calculations, we have

$$
\begin{aligned}
f_{e,1} &= h \int_0^1 2\varphi_1(\xi)\,\mathrm{d}\xi \\
&= h \int_0^1 2\,(1-\xi)\,\mathrm{d}\xi \\
&= h\left[2\xi - \xi^2\right]_0^1 \\
&= h \\
f_{e,2} &= h \int_0^1 2\varphi_2(\xi)\,\mathrm{d}\xi \\
&= h \int_0^1 2\xi\,\mathrm{d}\xi \\
&= h\left[\xi^2\right]_0^1 \\
&= h
\end{aligned}
$$

Hence, the element stiffness matrix and load vector are

$$
\mathbf{K}_e = \frac{1}{h}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad \mathbf{f}_e = h\begin{bmatrix} 1 \\ 1 \end{bmatrix}
$$

Assembling the individual matrices into the global system matrices, noting that $h = 0.25$ and that matrix components are added wherever the element matrices overlap, we obtain:

$$
\mathbf{K} = 4\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \qquad \mathbf{f} = 0.25\begin{bmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 1 \end{bmatrix}
$$

The matrix system may then be written as:

$$
\begin{bmatrix} 4 & -4 & 0 & 0 & 0 \\ -4 & 8 & -4 & 0 & 0 \\ 0 & -4 & 8 & -4 & 0 \\ 0 & 0 & -4 & 8 & -4 \\ 0 & 0 & 0 & -4 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.25 \end{bmatrix}
$$

To enforce the Dirichlet boundary conditions $u_1 = 1$ and $u_5 = -1$, we add two rows to the load vector and two additional rows and columns to the stiffness matrix, corresponding to two dummy Lagrange multiplier variables λ_1 and λ_2, to obtain the full matrix system as follows:

$$\begin{bmatrix} 4 & -4 & 0 & 0 & 0 & 1 & 0 \\ -4 & 8 & -4 & 0 & 0 & 0 & 0 \\ 0 & -4 & 8 & -4 & 0 & 0 & 0 \\ 0 & 0 & -4 & 8 & -4 & 0 & 0 \\ 0 & 0 & 0 & -4 & 4 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.25 \\ 1 \\ -1 \end{bmatrix}$$

where we have preserved symmetry in the square matrix by appending symmetric columns to its end. Inverting this matrix system in Matlab, we obtain (correct to 4 decimal places) $u_1 = 1.0000$, $u_2 = 0.6875$, $u_3 = 0.2500$, $u_4 = -0.3125$, $u_5 = -1.0000$, $\lambda_1 = -1$, $\lambda_2 = 3$.

To obtain the exact solution to the PDE, we rewrite it as

$$-\frac{\partial^2 u}{\partial x^2} = 2$$

Integrating twice, we obtain:

$$u = -x^2 + c_1 x + c_2$$

where c_1, c_2 are constants of integration. Their value can be determined from the boundary conditions, namely

$$\begin{aligned} u(0) &= 1 = c_2 \\ u(1) &= -1 = -1^2 + c_1 + c_2 \end{aligned} \tag{.3}$$

yielding $c_1 = -1$, $c_2 = 1$. Hence the exact PDE solution is

$$u = -x^2 - x + 1$$

Evaluating this solution at the node positions $x = 0$, 0.25, 0.5, 0.75 and 1, we obtain: $u(0) = 1$, $u(0.25) = 0.6875$, $u(0.5) = 0.25$, $u(0.75) = -0.3125$, and u(1) = -1. These values correspond to the nodal values u_1, u_2, u_3, u_4, and u_5 obtained by solving the matrix system earlier. Therefore, the FEM solution agrees with the exact solution.

5.2 To solve this PDE using FEM, we begin by first formulating the strong PDE form of the problem into its equivalent weak form. Rewriting the PDE as

$$\begin{aligned} -\frac{\partial^2 u}{\partial x^2} &= x \\ u(0) &= 1 \\ \frac{\partial u}{\partial x}(1) &= -1 \end{aligned}$$

we first multiply by our test function u_{test}, where we impose $u_{test}(0) = 0$, namely, at the Dirichlet boundary.[10] Integrating across the domain, we obtain:

$$\int_0^1 -u_{test} \frac{\partial^2 u}{\partial x^2} \, \mathrm{d}x = \int_0^1 x u_{test} \, \mathrm{d}x$$

Integrating the left-hand side by parts, yields:

$$\begin{aligned} \left[-u_{test} \frac{\partial u}{\partial x} \right]_0^1 - \int_0^1 -\frac{\partial u_{test}}{\partial x} \frac{\partial u}{\partial x} \, \mathrm{d}x &= \int_0^1 x u_{test} \, \mathrm{d}x \\ -u_{test}(1) \frac{\partial u}{\partial x}(1) + u_{test}(0) \frac{\partial u}{\partial x}(0) - \int_0^1 -\frac{\partial u_{test}}{\partial x} \frac{\partial u}{\partial x} \, \mathrm{d}x &= \int_0^1 x u_{test} \, \mathrm{d}x \\ u_{test}(1) + \int_0^1 \frac{\partial u_{test}}{\partial x} \frac{\partial u}{\partial x} \, \mathrm{d}x &= \int_0^1 x u_{test} \, \mathrm{d}x \end{aligned}$$

where we have used $\frac{\partial u}{\partial x}(1) = -1$ and $u_{test}(0) = 0$. Utilising our 1D Lagrange basis functions $\varphi_i(x)$, we employ Galerkin's method to substitute $u_{test} = \varphi_j$ and $u = \sum_i u_i \varphi_i$, to obtain:

$$\begin{aligned} \varphi_j(1) + \sum_{i=1}^{N} \int_0^1 \frac{\partial \varphi_j}{\partial x} \frac{\partial \varphi_i}{\partial x} \, \mathrm{d}x &= \int_0^1 x \varphi_j \, \mathrm{d}x \\ \sum_{i=1}^{N} \int_0^1 \frac{\partial \varphi_j}{\partial x} \frac{\partial \varphi_i}{\partial x} \, \mathrm{d}x &= -\varphi_j(1) + \int_0^1 x \varphi_j \, \mathrm{d}x \end{aligned}$$

For our 1D Lagrange functions, $\varphi_j(1)$ will equal 1 only for $j = N$, where N is the number of basis functions (and global nodes). For all other values of j, it will be 0. The above is equivalent to the following matrix system:

$$\begin{aligned} \mathbf{Ku} &= \mathbf{f} \\ K_{ij} &= \int_0^1 \frac{\partial \varphi_j}{\partial x} \frac{\partial \varphi_i}{\partial x} \, \mathrm{d}x \\ f_i &= -\delta_i^N + \int_0^1 x \phi_i \, \mathrm{d}x \end{aligned}$$

where δ_i^N is the *Kronecker delta*, defined by

$$\delta_i^j = \begin{cases} 1 \; i = j \\ 0 \; i \neq j \end{cases}$$

[10] Later, the actual Dirichlet condition $x(0) = 1$ will be enforced through the method of Lagrange multipliers, but for now, this constraint on u_{test} will allow us to conveniently obtain the PDE weak form.

To evaluate these components, it is convenient to work with local element versions of these matrices, namely:

$$\begin{aligned} K_{e,ij} &= h \int_0^1 \left(\frac{1}{h}\right) \frac{\partial \varphi_j}{\partial \xi} \left(\frac{1}{h}\right) \frac{\partial \varphi_i}{\partial \xi} \, \mathrm{d}\xi \\ &= \frac{1}{h} \int_0^1 \frac{\partial \varphi_j}{\partial \xi} \frac{\partial \varphi_i}{\partial \xi} \, \mathrm{d}\xi \\ f_{e,i} &= -\delta_x(1) + h \int_0^1 x \phi_i \, \mathrm{d}\xi \end{aligned}$$

where $i, j = 1, 2$, h is the element size, and $\delta_x(1) = 1$ if $x = 1$, 0 otherwise. Using the 1D Lagrange shape functions, $\varphi_1 = (1 - \xi)$ and $\varphi_2 = \xi$, the four components of the element stiffness matrix can be determined using:

$$\begin{aligned} K_{e,11} &= \frac{1}{h} \int_0^1 \frac{\partial \varphi_1(\xi)}{\partial \xi} \frac{\partial \varphi_1(\xi)}{\partial \xi} \, \mathrm{d}\xi \\ &= \frac{1}{h} \int_0^1 (-1)(-1) \, \mathrm{d}\xi \\ &= \frac{1}{h} \\ K_{e,12} &= \frac{1}{h} \int_0^1 \frac{\partial \varphi_1(\xi)}{\partial \xi} \frac{\partial \varphi_2(\xi)}{\partial \xi} \, \mathrm{d}\xi \\ &= \frac{1}{h} \int_0^1 (-1)(1) \, \mathrm{d}\xi \\ &= -\frac{1}{h} \\ K_{e,21} &= \frac{1}{h} \int_0^1 \frac{\partial \varphi_2(\xi)}{\partial \xi} \frac{\partial \varphi_1(\xi)}{\partial \xi} \, \mathrm{d}\xi \\ &= \frac{1}{h} \int_0^1 (1)(-1) \, \mathrm{d}\xi \\ &= -\frac{1}{h} \\ K_{e,22} &= \frac{1}{h} \int_0^1 \frac{\partial \varphi_2(\xi)}{\partial \xi} \frac{\partial \varphi_2(\xi)}{\partial \xi} \, \mathrm{d}\xi \\ &= \frac{1}{h} \int_0^1 (1)(1) \, \mathrm{d}\xi \\ &= \frac{1}{h} \end{aligned}$$

For the element load vector, we must evaluate the integral of $x\varphi_j$ with respect to the local element coordinate ξ. To do this, we first express x in local element coordinates.

Since the element is isoparametric, the value of x within the element is simply the weighted sum of its shape functions, namely:

$$\begin{aligned} x &= x_1\varphi_1(\xi) + x_2\varphi_2(\xi) \\ &= x_1(1-\xi) + x_2\xi \end{aligned}$$

where x_1, x_2 are the values of x at nodes 1 and 2 of the element. The components of the load vector are therefore

$$\begin{aligned} f_{e,1} &= -\delta_x(1) + h\int_0^1 x\varphi_1(\xi)\,\mathrm{d}\xi \\ &= -\delta_x(1) + h\int_0^1 [x_1(1-\xi) + x_2\xi]\,(1-\xi)\,\mathrm{d}\xi \\ &= -\delta_x(1) + h\int_0^1 \left[x_1(1-\xi)^2 + x_2\xi\,(1-\xi)\right]\mathrm{d}\xi \\ &= -\delta_x(1) + h\left[-\frac{x_1(1-\xi)^3}{3} + x_2\left(\frac{\xi^2}{2} - \frac{\xi^3}{3}\right)\right]_0^1 \\ &= -\delta_x(1) + h\left[\frac{1}{3}x_1 + \frac{1}{6}x_2\right] \\ f_{e,2} &= -\delta_x(1) + h\int_0^1 x\varphi_2(\xi)\,\mathrm{d}\xi \\ &= -\delta_x(1) + h\int_0^1 [x_1(1-\xi) + x_2\xi]\,\xi\,\mathrm{d}\xi \\ &= -\delta_x(1) + h\int_0^1 \left[x_1(1-\xi)\xi + x_2\xi^2\right]\mathrm{d}\xi \\ &= -\delta_x(1) + h\left[x_1\left(\frac{\xi^2}{2} - \frac{\xi^3}{3}\right) + x_2\frac{\xi^3}{3}\right]_0^1 \\ &= -\delta_x(1) + h\left[\frac{1}{6}x_1 + \frac{1}{3}x_2\right] \end{aligned}$$

Using $h = 0.25$, we can write the stiffness matrix of each element as

$$\mathbf{K}_e = \frac{1}{h}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 4 & -4 \\ -4 & 4 \end{bmatrix}$$

Furthermore, the element load vectors $\mathbf{f}_e$ are given by

$$\begin{aligned} \mathbf{f}_e &= -\delta_x(1) + \frac{h}{6}\begin{bmatrix} 2x_1 + x_2 \\ x_1 + 2x_2 \end{bmatrix} \\ &= -\delta_x(1) + \frac{1}{24}\begin{bmatrix} 2x_1 + x_2 \\ x_1 + 2x_2 \end{bmatrix} \end{aligned}$$

These element load vectors will be different for each element, and are given as follows:

$$\text{Element 1}: x_1 = 0,\ x_2 = 0.25 \quad \mathbf{f}_e = \tfrac{1}{24}\begin{bmatrix} 0.25 \\ 0.5 \end{bmatrix}$$

$$\text{Element 2}: x_1 = 0.25,\ x_2 = 0.5 \quad \mathbf{f}_e = \tfrac{1}{24}\begin{bmatrix} 1 \\ 1.25 \end{bmatrix}$$

$$\text{Element 3}: x_1 = 0.5,\ x_2 = 0.75 \quad \mathbf{f}_e = \tfrac{1}{24}\begin{bmatrix} 1.75 \\ 2 \end{bmatrix}$$

$$\text{Element 4}: x_1 = 0.75,\ x_2 = 1 \quad \mathbf{f}_e = \begin{bmatrix} 0 \\ -1 \end{bmatrix} + \tfrac{1}{24}\begin{bmatrix} 2.5 \\ 2.75 \end{bmatrix}$$

Assembling the individual element matrices into the global system matrices, noting that matrix components are added wherever the element matrices overlap, we obtain:

$$\mathbf{K} = \begin{bmatrix} 4 & -4 & 0 & 0 & 0 \\ -4 & 8 & -4 & 0 & 0 \\ 0 & -4 & 8 & -4 & 0 \\ 0 & 0 & -4 & 8 & -4 \\ 0 & 0 & 0 & -4 & 4 \end{bmatrix} \qquad \mathbf{f} = \frac{1}{24}\begin{bmatrix} 0.25 \\ 1.5 \\ 3 \\ 4.5 \\ -21.25 \end{bmatrix}$$

The matrix system may then be written as:

$$\begin{bmatrix} 4 & -4 & 0 & 0 & 0 \\ -4 & 8 & -4 & 0 & 0 \\ 0 & -4 & 8 & -4 & 0 \\ 0 & 0 & -4 & 8 & -4 \\ 0 & 0 & 0 & -4 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0.0104 \\ 0.0625 \\ 0.1250 \\ 0.1875 \\ -0.8854 \end{bmatrix}$$

To enforce the Dirichlet boundary condition at $x = 0$, namely $u_1 = 1$, we add a row to the load vector and one additional row and column to the stiffness matrix, corresponding to a dummy Lagrange multiplier variable λ_1, to obtain the full matrix system as follows:

$$\begin{bmatrix} 4 & -4 & 0 & 0 & 0 & 1 \\ -4 & 8 & -4 & 0 & 0 & 0 \\ 0 & -4 & 8 & -4 & 0 & 0 \\ 0 & 0 & -4 & 8 & -4 & 0 \\ 0 & 0 & 0 & -4 & 4 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ \lambda_1 \end{bmatrix} = \begin{bmatrix} 0.0104 \\ 0.0625 \\ 0.1250 \\ 0.1875 \\ -0.8854 \\ 1 \end{bmatrix}$$

Inverting this matrix system in Matlab, we obtain (correct to 4 decimal places) $u_1 = 1.0000$, $u_2 = 0.8724$, $u_3 = 0.7292$, $u_4 = 0.5547$, $u_5 = 0.3333$, $\lambda_1 = -0.5000$.

To obtain the exact solution to the PDE, we have

$$-\frac{\partial^2 u}{\partial x^2} = x$$

Integrating twice, we obtain:

$$\frac{\partial u}{\partial x} = -\frac{x^2}{2} + c_1$$
$$u = -\frac{x^3}{6} + c_1 x + c_2$$

where c_1, c_2 are constants of integration, whose value can be determined from the boundary conditions, namely

$$u(0) = 1 = c_2$$
$$\frac{\partial u}{\partial x}(1) = -1 = -\frac{1}{2} + c_1 \tag{.4}$$

yielding $c_1 = -\frac{1}{2}$, $c_2 = 1$. Hence the exact PDE solution is

$$u = -\frac{x^3}{6} - \frac{x}{2} + 1$$

Evaluating this solution at the node positions $x = 0$, 0.25, 0.5, 0.75 and 1, we obtain: $u(0) = 1$, $u(0.25) = 0.8724$, $u(0.5) = 0.7292$, $u(0.75) = 0.5547$, and u(1) = 0.3333. These values correspond to the nodal values u_1, u_2, u_3, u_4, and u_5 obtained by solving the matrix system earlier. Therefore, the FEM solution agrees with the exact solution.

5.3 For the cubic Lagrange 1D element, there are four nodes located at $\xi = 0$, $\xi = 1/3, \xi = 2/3$ and $\xi = 1$. The four cubic Lagrange functions can be determined as follows:
For $\varphi_1(\xi)$, we require $\varphi_1(0) = 1$, $\varphi_1(1/3) = 0$, $\varphi_1(2/3) = 0$ and $\varphi_1(1) = 0$. The cubic polynomial must therefore satisfy the following form:

$$\varphi_1(\xi) = c_1 \left(\xi - \tfrac{1}{3}\right)\left(\xi - \tfrac{2}{3}\right)(\xi - 1)$$

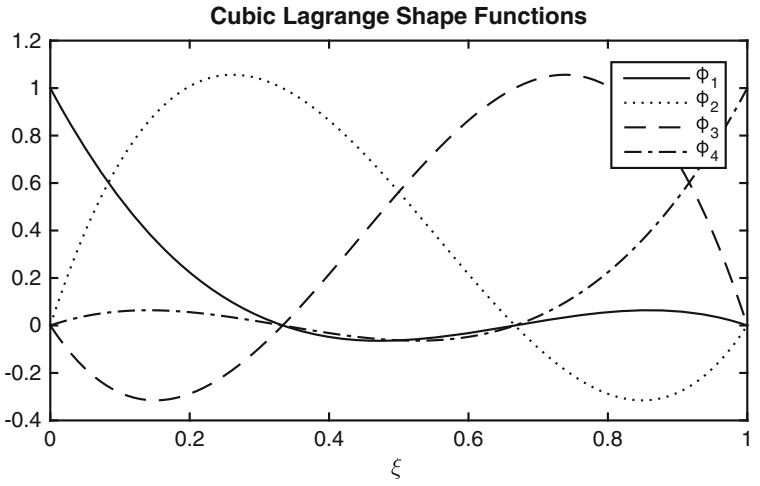
where c_1 is a constant. Its value can be determined from the requirement that $\varphi_1(0) = 1$. Hence, $c_1 = -9/2$, and we obtain

$$\varphi_1(\xi) = \tfrac{9}{2}\left(\xi - \tfrac{1}{3}\right)\left(\xi - \tfrac{2}{3}\right)(1 - \xi)$$

A similar analysis can be applied to the other three shape functions, requiring these to equal 1 at one node and 0 at all others. This yields the remaining shape functions as follows:

$$\varphi_2(\xi) = \tfrac{27}{2}\xi\left(\xi - \tfrac{2}{3}\right)(\xi - 1)$$
$$\varphi_3(\xi) = \tfrac{27}{2}\xi\left(\xi - \tfrac{1}{3}\right)(1 - \xi)$$
$$\varphi_4(\xi) = \tfrac{9}{2}\xi\left(\xi - \tfrac{1}{3}\right)\left(\xi - \tfrac{2}{3}\right)$$

A plot of these shape functions is shown below:



5.4 This diffusion PDE is similar to Example 5.1, in which we computed the global damping, stiffness and load matrices/vectors using Eqs. 5.5–5.7

$$D_{ij} = \int_0^1 \varphi_i(x)\varphi_j(x)\,\mathrm{d}x \qquad \text{(damping matrix)}$$
$$K_{ij} = \int_0^1 D\frac{\mathrm{d}\varphi_i(x)}{\mathrm{d}x}\frac{\mathrm{d}\varphi_j(x)}{\mathrm{d}x}\,\mathrm{d}x \qquad \text{(stiffness matrix)}$$
$$f_i = q(t)\varphi_i(1) = q(t)\delta_i^N \qquad \text{(load vector)}$$

where D is the diffusion coefficient ($= 1$ for this problem), δ_i^N is the Kronecker delta, and $q(t) = \left[D\frac{\partial c}{\partial x}\right]_{x=1}$, which represents the specified flux at $x = 1$. As in Example 5.1, since zero-flux boundary conditions have been specified at both ends of the domain, all elements of the load vector will be 0.

To determine the damping and stiffness matrices, is convenient to express these at the local element level, namely:

$$D_{e,ij} = h\int_0^1 \varphi_i(\xi)\varphi_j(\xi)\,\mathrm{d}\xi$$
$$K_{e,ij} = h\int_0^1 \left(\frac{1}{h}\right)\frac{\mathrm{d}\varphi_i(\xi)}{\mathrm{d}\xi}\left(\frac{1}{h}\right)\frac{\mathrm{d}\varphi_j(\xi)}{\mathrm{d}\xi}\,\mathrm{d}\xi$$
$$= \frac{1}{h}\int_0^1 \frac{\mathrm{d}\varphi_i(\xi)}{\mathrm{d}\xi}\frac{\mathrm{d}\varphi_j(\xi)}{\mathrm{d}\xi}\,\mathrm{d}\xi$$

We can then substitute the three quadratic Lagrange shape functions,

$$\begin{aligned}\varphi_1(\xi) &= 2(0.5-\xi)(1-\xi)\\ \varphi_2(\xi) &= 4\xi(1-\xi)\\ \varphi_3(\xi) &= 2\xi(\xi-0.5)\end{aligned}$$

into these expressions to determine the above system matrices. For the damping matrix, we have:

$$\begin{aligned}D_{e,11} &= h\int_0^1 \varphi_1(\xi)\varphi_1(\xi)\,\mathrm{d}\xi\\ &= 4h\int_0^1 (0.5-\xi)^2(1-\xi)^2\,\mathrm{d}\xi\\ &= 4h\int_0^1 (0.25-\xi+\xi^2)(1-2\xi+\xi^2)\,\mathrm{d}\xi\\ &= 4h\int_0^1 \left[0.25-\xi+\xi^2-0.5\xi+2\xi^2-2\xi^3+0.25\xi^2-\xi^3+\xi^4\right]\mathrm{d}\xi\\ &= 4h\int_0^1 \left[0.25-1.5\xi+3.25\xi^2-3\xi^3+\xi^4\right]\mathrm{d}\xi\\ &= 4h\left[0.25\xi-0.75\xi^2+\frac{3.25}{3}\xi^3-0.75\xi^4+0.2\xi^5\right]_0^1\\ &= \frac{2h}{15}\\ D_{e,12} &= h\int_0^1 \varphi_1(\xi)\varphi_2(\xi)\,\mathrm{d}\xi\\ &= 8h\int_0^1 (0.5-\xi)(1-\xi)\xi(1-\xi)\,\mathrm{d}\xi\\ &= 8h\int_0^1 (0.5\xi-\xi^2)(1-2\xi+\xi^2)\,\mathrm{d}\xi\\ &= 8h\int_0^1 \left[0.5\xi-\xi^2-\xi^2+2\xi^3+0.5\xi^3-\xi^4\right]\mathrm{d}\xi\\ &= 8h\int_0^1 \left[0.5\xi-2\xi^2+2.5\xi^3-\xi^4\right]\mathrm{d}\xi\\ &= 8h\left[0.25\xi^2-\frac{2}{3}\xi^3+\frac{2.5}{4}\xi^4-0.2\xi^5\right]_0^1\\ &= \frac{h}{15}\\ D_{e,13} &= h\int_0^1 \varphi_1(\xi)\varphi_3(\xi)\,\mathrm{d}\xi\end{aligned}$$

$$
\begin{aligned}
&= 4h \int_0^1 (0.5 - \xi)(1 - \xi)\xi(\xi - 0.5)\,\mathrm{d}\xi \\
&= 4h \int_0^1 (\xi - 0.5)^2 \xi(\xi - 1)\,\mathrm{d}\xi \\
&= 4h \int_0^1 (\xi^2 - \xi + 0.25)(\xi^2 - \xi)\,\mathrm{d}\xi \\
&= 4h \int_0^1 \left[\xi^4 - \xi^3 + 0.25\xi^2 - \xi^3 + \xi^2 - 0.25\xi\right]\mathrm{d}\xi \\
&= 4h \int_0^1 \left[\xi^4 - 2\xi^3 + 1.25\xi^2 - 0.25\xi\right]\mathrm{d}\xi \\
&= 4h \left[0.2\xi^5 - 0.5\xi^4 + \frac{1.25}{3}\xi^3 - 0.125\xi^2\right]_0^1 \\
&= -\frac{h}{30}
\end{aligned}
$$

$$
\begin{aligned}
D_{e,22} &= h \int_0^1 \varphi_2(\xi)\varphi_2(\xi)\,\mathrm{d}\xi \\
&= 16h \int_0^1 \xi^2 (1 - \xi)^2\,\mathrm{d}\xi \\
&= 16h \int_0^1 \xi^2 (1 - 2\xi + \xi^2)\,\mathrm{d}\xi \\
&= 16h \int_0^1 \left[\xi^2 - 2\xi^3 + \xi^4\right]\mathrm{d}\xi \\
&= 16h \left[\frac{1}{3}\xi^3 - 0.5\xi^4 + 0.2\xi^5\right]_0^1 \\
&= \frac{8h}{15}
\end{aligned}
$$

$$
\begin{aligned}
D_{e,23} &= h \int_0^1 \varphi_2(\xi)\varphi_3(\xi)\,\mathrm{d}\xi \\
&= 8h \int_0^1 \xi(1 - \xi)\xi(\xi - 0.5)\,\mathrm{d}\xi \\
&= 8h \int_0^1 \xi^2 (-0.5 + 1.5\xi - \xi^2)\,\mathrm{d}\xi \\
&= 8h \int_0^1 \left[-0.5\xi^2 + 1.5\xi^3 - \xi^4\right]\mathrm{d}\xi \\
&= 8h \left[-\frac{0.5}{3}\xi^3 + \frac{1.5}{4}\xi^4 - 0.2\xi^5\right]_0^1 \\
&= \frac{h}{15}
\end{aligned}
$$

$$
\begin{aligned}
D_{e,33} &= h\int_0^1 \varphi_3(\xi)\varphi_3(\xi)\,\mathrm{d}\xi \\
&= 4h\int_0^1 \xi^2(\xi-0.5)^2\,\mathrm{d}\xi \\
&= 4h\int_0^1 \xi^2(\xi^2-\xi+0.25)\,\mathrm{d}\xi \\
&= 4h\int_0^1 \left[\xi^4-\xi^3+0.25\xi^2\right]\mathrm{d}\xi \\
&= 4h\left[0.2\xi^5-0.25\xi^4+\frac{0.25}{3}\xi^3\right]_0^1 \\
&= \frac{2h}{15}
\end{aligned}
$$

with remaining terms being symmetric, that is: $D_{e21} = D_{e12}$, $D_{e31} = D_{e13}$, $D_{e32} = D_{e23}$. Hence, the 3×3 element damping matrix is given by:

$$
\mathbf{D}_e = \frac{h}{30}\begin{bmatrix} 4 & 2 & -1 \\ 2 & 16 & 2 \\ -1 & 2 & 4 \end{bmatrix}
$$

For the element stiffness matrix, we must first calculate the derivatives of our shape functions:

$$
\begin{aligned}
\frac{\mathrm{d}\varphi_1(\xi)}{\mathrm{d}\xi} &= \frac{\mathrm{d}}{\mathrm{d}\xi}\Big[2(0.5-\xi)(1-\xi)\Big] \\
&= \frac{\mathrm{d}}{\mathrm{d}\xi}\Big[2\xi^2-3\xi+1\Big] \\
&= 4\xi-3 \\
\frac{\mathrm{d}\varphi_2(\xi)}{\mathrm{d}\xi} &= \frac{\mathrm{d}}{\mathrm{d}\xi}\Big[4\xi(1-\xi)\Big] \\
&= \frac{\mathrm{d}}{\mathrm{d}\xi}\Big[4\xi-4\xi^2\Big] \\
&= 4-8\xi \\
\frac{\mathrm{d}\varphi_3(\xi)}{\mathrm{d}\xi} &= \frac{\mathrm{d}}{\mathrm{d}\xi}\Big[2\xi(\xi-0.5)\Big] \\
&= \frac{\mathrm{d}}{\mathrm{d}\xi}\Big[2\xi^2-\xi\Big] \\
&= 4\xi-1
\end{aligned}
$$

Hence, the components of the element stiffness matrix can be determined as follows:

$$
\begin{aligned}
K_{e,11} &= \frac{1}{h}\int_0^1 \frac{\mathrm{d}\varphi_1(\xi)}{\mathrm{d}\xi}\frac{\mathrm{d}\varphi_1(\xi)}{\mathrm{d}\xi}\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 (4\xi-3)^2\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 \left(16\xi^2-24\xi+9\right)\,\mathrm{d}\xi \\
&= \frac{1}{h}\left[\frac{16}{3}\xi^3-12\xi^2+9\xi\right]_0^1 \\
&= \frac{7}{3h} \\
K_{e,12} &= \frac{1}{h}\int_0^1 \frac{\mathrm{d}\varphi_1(\xi)}{\mathrm{d}\xi}\frac{\mathrm{d}\varphi_2(\xi)}{\mathrm{d}\xi}\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 (4\xi-3)(4-8\xi)\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 \left(-12+40\xi-32\xi^2\right)\,\mathrm{d}\xi \\
&= \frac{1}{h}\left[-12\xi+20\xi^2-\frac{32}{3}\xi^3\right]_0^1 \\
&= -\frac{8}{3h} \\
K_{e,13} &= \frac{1}{h}\int_0^1 \frac{\mathrm{d}\varphi_1(\xi)}{\mathrm{d}\xi}\frac{\mathrm{d}\varphi_3(\xi)}{\mathrm{d}\xi}\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 (4\xi-3)(4\xi-1)\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 \left(16\xi^2-16\xi+3\right)\,\mathrm{d}\xi \\
&= \frac{1}{h}\left[\frac{16}{3}\xi^3-8\xi^2+3\xi\right]_0^1 \\
&= \frac{1}{3h} \\
K_{e,22} &= \frac{1}{h}\int_0^1 \frac{\mathrm{d}\varphi_2(\xi)}{\mathrm{d}\xi}\frac{\mathrm{d}\varphi_2(\xi)}{\mathrm{d}\xi}\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 (4-8\xi)^2\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 \left(16-64\xi+64\xi^2\right)\,\mathrm{d}\xi
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{h}\left[16\xi - 32\xi^2 + \frac{64}{3}\xi^3\right]_0^1 \\
&= \frac{16}{3h} \\
K_{e,23} &= \frac{1}{h}\int_0^1 \frac{\mathrm{d}\varphi_2(\xi)}{\mathrm{d}\xi}\frac{\mathrm{d}\varphi_3(\xi)}{\mathrm{d}\xi}\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 (4 - 8\xi)(4\xi - 1)\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 \left(-4 + 24\xi - 32\xi^2\right)\,\mathrm{d}\xi \\
&= \frac{1}{h}\left[-4\xi + 12\xi^2 - \frac{32}{3}\xi^3\right]_0^1 \\
&= -\frac{8}{3h} \\
K_{e,33} &= \frac{1}{h}\int_0^1 \frac{\mathrm{d}\varphi_3(\xi)}{\mathrm{d}\xi}\frac{\mathrm{d}\varphi_3(\xi)}{\mathrm{d}\xi}\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 (4\xi - 1)^2\,\mathrm{d}\xi \\
&= \frac{1}{h}\int_0^1 \left(16\xi^2 - 8\xi + 1\right)\,\mathrm{d}\xi \\
&= \frac{1}{h}\left[\frac{16}{3}\xi^3 - 4\xi^2 + \xi\right]_0^1 \\
&= \frac{7}{3h}
\end{aligned}
$$

with the remaining components being symmetric. Thus, the element stiffness matrix is given by

$$
\mathbf{K}_e = \frac{1}{3h}\begin{bmatrix} 7 & -8 & 1 \\ -8 & 16 & -8 \\ 1 & -8 & 7 \end{bmatrix}
$$

Assembling the four individual element matrices into the global system matrices, using $h = 0.25$ and noting that matrix components are added wherever the element matrices overlap, we obtain the 9×9 global system matrices:

$$\mathbf{D} = \frac{1}{120}\begin{bmatrix} 4 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 16 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & 8 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 16 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & 8 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 16 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 8 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 16 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & 4 \end{bmatrix} \quad \text{(damping matrix)}$$

$$\approx \begin{bmatrix} 3.33 & 1.67 & -0.83 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1.67 & 13.33 & 1.67 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.83 & 1.67 & 6.67 & 1.67 & -0.83 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.67 & 13.33 & 1.67 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.83 & 1.67 & 6.67 & 1.67 & -0.83 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.67 & 13.33 & 1.67 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.83 & 1.67 & 6.67 & 1.67 & -0.83 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.67 & 13.33 & 1.67 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.83 & 1.67 & 3.33 \end{bmatrix} \times 10^{-2}$$

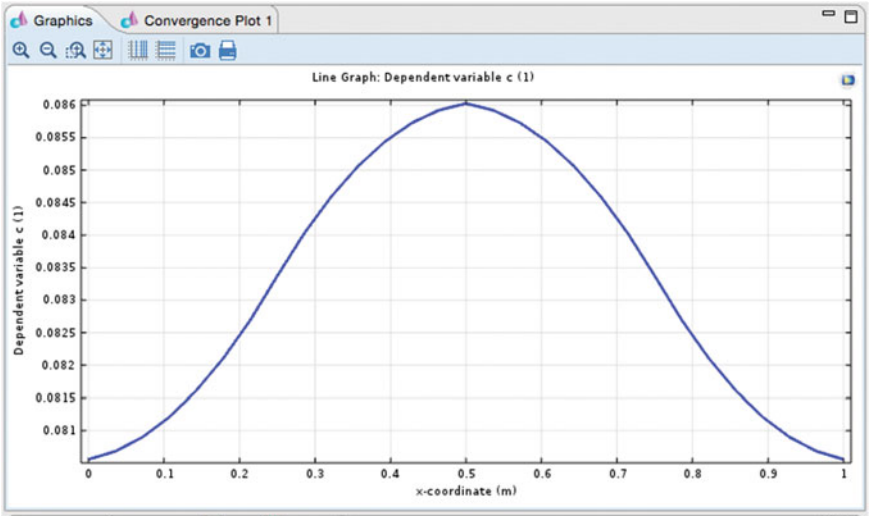
and

$$\mathbf{K} = \frac{4}{3}\begin{bmatrix} 7 & -8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -8 & 16 & -8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -8 & 14 & -8 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -8 & 16 & -8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -8 & 14 & -8 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -8 & 16 & -8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -8 & 14 & -8 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -8 & 16 & -8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -8 & 7 \end{bmatrix} \quad \text{(stiffness matrix)}$$

$$\approx \begin{bmatrix} 9.33 & -10.67 & 1.33 & 0 & 0 & 0 & 0 & 0 & 0 \\ -10.67 & 21.33 & -10.67 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1.33 & -10.67 & 18.67 & -10.67 & 1.33 & 0 & 0 & 0 & 0 \\ 0 & 0 & -10.67 & 21.33 & -10.67 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.33 & -10.67 & 18.67 & -10.67 & 1.33 & 0 & 0 \\ 0 & 0 & 0 & 0 & -10.67 & 21.33 & -10.67 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.33 & -10.67 & 18.67 & -10.67 & 1.33 \\ 0 & 0 & 0 & 0 & 0 & 0 & -10.67 & 21.33 & -10.67 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.33 & -10.67 & 9.33 \end{bmatrix}$$

To solve the PDE in COMSOL and inspect the resulting damping and stiffness matrices, we can implement the same steps as Example 5.1, however this time specifying 4 elements under the mesh distribution settings, and 'Quadratic' for the Lagrange element order under the Discretization tab of the General Form PDE node in the model

tree. The solution at $t = 0.1$ is shown plotted below, along with the COMSOL-generated damping matrix.



Messages | Progress | Log | Table

0.033333	0.016667	-0.0083333	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.016667	0.13333	0.016667	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
-0.0083333	0.016667	0.066667	0.016667	-0.0083333	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.016667	0.13333	0.016667	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	-0.0083333	0.016667	0.066667	0.016667	-0.0083333	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.016667	0.13333	0.016667	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	-0.0083333	0.016667	0.066667	0.016667	-0.0083333
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.016667	0.13333	0.016667
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-0.0083333	0.016667	0.033333

The COMSOL-generated stiffness matrix is shown below:

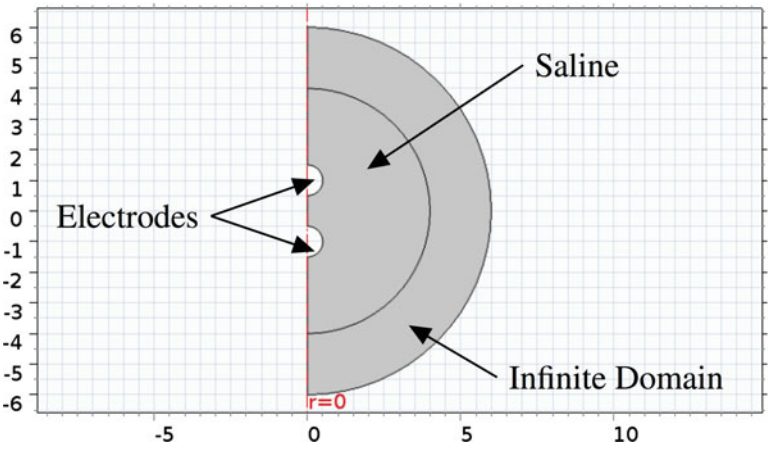
Messages | Progress | Log | Table

9.3333	-10.667	1.3333	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
-10.667	21.333	-10.667	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
1.3333	-10.667	18.667	-10.667	1.3333	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	-10.667	21.333	-10.667	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	1.3333	-10.667	18.667	-10.667	1.3333	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	-10.667	21.333	-10.667	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	1.3333	-10.667	18.667	-10.667	1.3333
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-10.667	21.333	-10.667
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.3333	-10.667	9.3333

Both of these COMSOL matrices agree with those we obtained analytically.

Problems of Chap. 6

6.1 To solve this problem in COMSOL, we can utilise the 2D axisymmetric geometry shown below, where all dimensions are in mm. Note that the electrodes are defined as semi-circular boundaries of the saline domain.



Using the electric currents physics mode of the AC/DC module, we set the conductivity of the saline as $1\,\mathrm{S\,m^{-1}}$, the boundaries one electrode at ground, and the boundaries of the other electrode to a potential of 1 V. The surrounding hemispherical domain is assigned an infinite element with zero-flux boundary. Defining an integration coupling operator along the active electrode boundaries, we can determine the total current i flowing into the saline domain as

$$i = \int_{A_e} 2\pi r J_n \, \mathrm{d}s$$

where A_e is the active electrode boundary, s is the arc-length along the boundary, and J_n is the normal component of the inward current density, given by the COMSOL variable `ec.nJ`. The resistance R between the electrodes can then be determined using

$$R = \frac{1}{i}$$

Setting the mesh distribution along the boundaries of each electrode to be 100 elements, COMSOL yields a value of $R = 237.23\,\Omega$.

6.2 Using Eq. 6.16 for the 2D case, we can determine the conductivity tensor in the tissue slab using

$$\boldsymbol{\sigma} = \sigma_1 \mathbf{n}_1 \mathbf{n}_1^{\mathrm{T}} + \sigma_2 \mathbf{n}_2 \mathbf{n}_2^{\mathrm{T}}$$

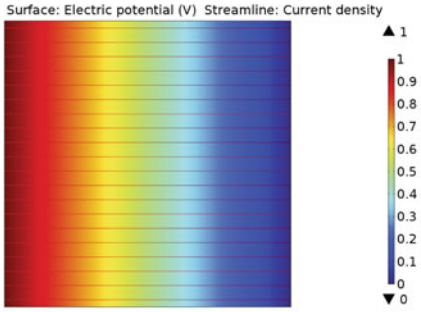
where σ_1, σ_2 are the conductivities in the fibre and transverse-fibre directions, and $\mathbf{n}_1$, $\mathbf{n}_2$ are the corresponding orthogonal unit vectors in the fibre and transverse-fibre directions. For a fibre angle of θ, these directions are given by

$$\mathbf{n}_1 = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} \qquad \mathbf{n}_2 = \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix}$$

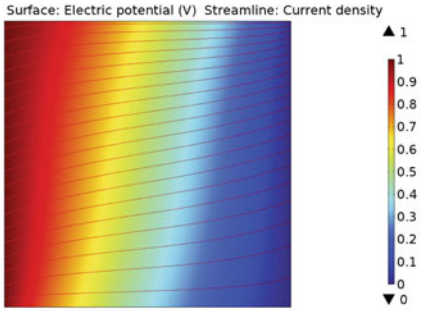
with resulting conductivity tensor

$$\boldsymbol{\sigma} = \sigma_1 \begin{pmatrix} \cos^2\theta & \cos\theta\sin\theta \\ \cos\theta\sin\theta & \sin^2\theta \end{pmatrix} + \sigma_2 \begin{pmatrix} \sin^2\theta & -\sin\theta\cos\theta \\ -\sin\theta\cos\theta & \cos^2\theta \end{pmatrix}$$
$$= \begin{pmatrix} \sigma_1\cos^2\theta + \sigma_2\sin^2\theta & (\sigma_1 - \sigma_2)\sin\theta\cos\theta \\ (\sigma_1 - \sigma_2)\sin\theta\cos\theta & \sigma_1\sin^2\theta + \sigma_2\cos^2\theta \end{pmatrix}$$

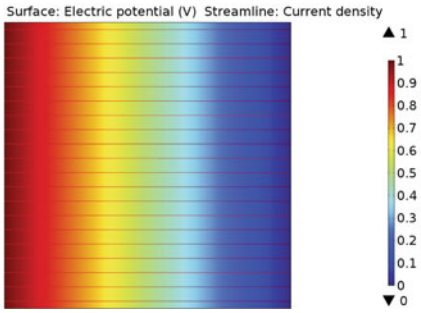
Using $\sigma_1 = 0.2\,\mathrm{mS\,cm^{-1}}$, $\sigma_2 = 0.1\,\mathrm{mS\,cm^{-1}}$, along with values of $\theta = 0°$, $45°$, $90°$, we obtain the following COMSOL plots of voltage distributions and current streamlines (using the 2D streamline plot type):



$\theta = 0°$



$\theta = 45°$



$\theta = 90°$

6.3 This problem is similar to that of Sect. 6.1.5, with the exception that the electric potential on the electrode disc is replaced with a normal current density boundary

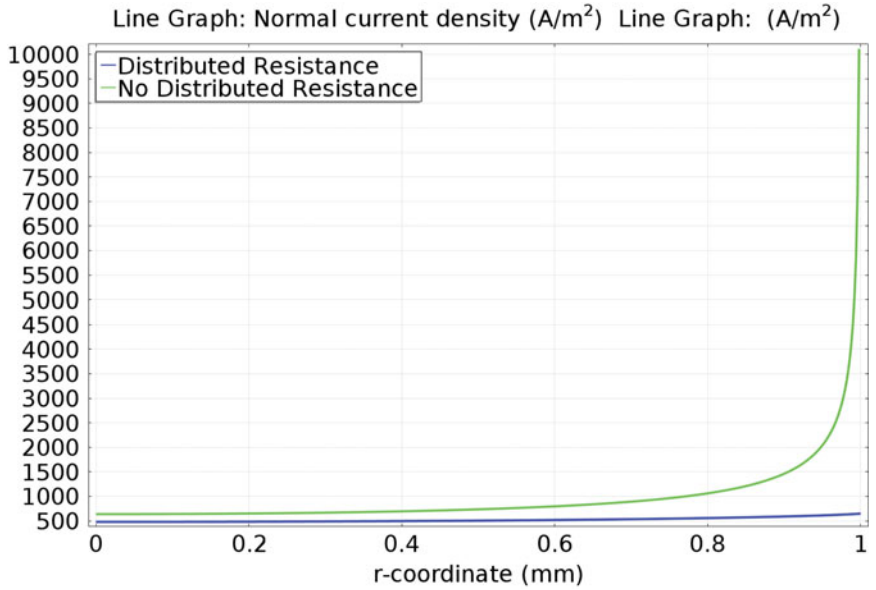
condition, with inward normal current density J_n given by:

$$J_n = \frac{V_s - V}{R}$$

where V_s is the supply voltage (1 V), V is the potential in the saline medium adjacent to the electrode, and R is the distributed resistance ($0.001\,\Omega\,\mathrm{m}^2$). The COMSOL-generated plot of current density as a function of radial position along the disc electrode is shown below, where the theoretical plot with no distributed resistance has been generated using

$$J_n = \frac{2\sigma V_s}{\pi\sqrt{R_e^2 - r^2}}$$

where r is the radial position along the disc, σ is the conductivity of the saline medium ($1\,\mathrm{S\,m^{-1}}$), V_s is the supply voltage (1 V) and R_e is the electrode radius (1 mm). Note that the effect of the distributed resistance is to smooth out the variations in current density, particularly at the edge of the disc, resulting in a near-constant current across the disc electrode.



Problems of Chap. 7

7.1 At steady-state, $\partial c/\partial t = 0$ and the PDE reduces to the ODE

$$D\frac{\mathrm{d}^2 c}{\mathrm{d}t^2} - k_{up}c = 0$$

which can be solved for using the methods of Chap. 2. The characteristic equation of this ODE is

$$Dm^2 - k_{up} = 0$$
$$\therefore \quad m = \pm\sqrt{\frac{k_{up}}{D}}$$

Hence the solution is of the form

$$c(x) = C_1 \mathrm{e}^{x\sqrt{\frac{k_{up}}{D}}} + C_2 \mathrm{e}^{-x\sqrt{\frac{k_{up}}{D}}}$$

where C_1, C_2 are constants which can be determined from the boundary conditions.

Specifically, when $x = 0$, $c = C_0$. Hence

$$C_0 = C_1 + C_2$$

Also, when $x = d_c$, $c = 0$. Hence

$$0 = C_1 \mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}} + C_2 \mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}}$$

Substituting $C_2 = C_0 - C_1$ into the above, we have

$$\begin{aligned} C_1 \mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}} + (C_0 - C_1)\mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}} &= 0 \\ C_1 \mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}} - C_1 \mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}} &= -C_0 \mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}} \\ C_1 \left[\mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}} - \mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}}\right] &= -C_0 \mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}} \\ \therefore \quad C_1 &= \frac{C_0 \mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}}}{\mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}} - \mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}}} \end{aligned}$$

and using $C_2 = C_0 - C_1$, we also obtain

$$\begin{aligned} C_2 &= C_0 - \frac{C_0 \mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}}}{\mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}} - \mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}}} \\ &= C_0 \left[\frac{\mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}} - \mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}}}{\mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}} - \mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}}}\right] - \frac{C_0 \mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}}}{\mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}} - \mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}}} \\ &= -\frac{C_0 \mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}}}{\mathrm{e}^{-d_c\sqrt{\frac{k_{up}}{D}}} - \mathrm{e}^{d_c\sqrt{\frac{k_{up}}{D}}}} \end{aligned}$$

Substituting these values of C_1, C_2 into the general solution form, we obtain

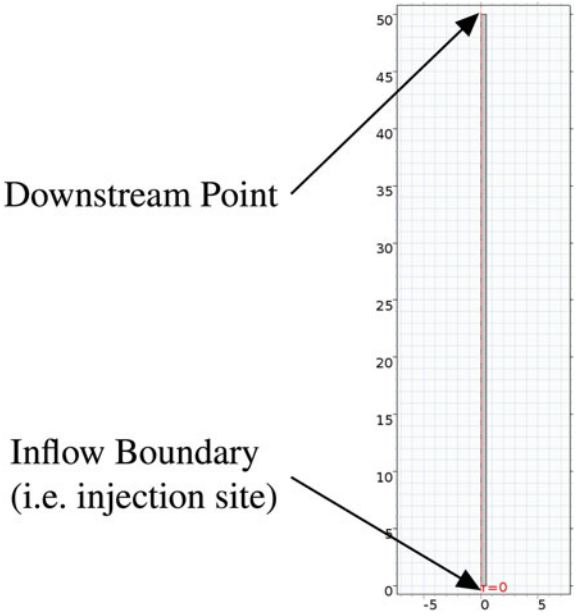
$$c(x) = \overbrace{\left[\frac{C_0 e^{-d_c\sqrt{\frac{k_{up}}{D}}}}{e^{-d_c\sqrt{\frac{k_{up}}{D}}} - e^{d_c\sqrt{\frac{k_{up}}{D}}}}\right]}^{C_1} e^{x\sqrt{\frac{k_{up}}{D}}} + \overbrace{\left[\frac{-C_0 e^{d_c\sqrt{\frac{k_{up}}{D}}}}{e^{-d_c\sqrt{\frac{k_{up}}{D}}} - e^{d_c\sqrt{\frac{k_{up}}{D}}}}\right]}^{C_2} e^{-x\sqrt{\frac{k_{up}}{D}}}$$
$$= \frac{C_0}{e^{-d_c\sqrt{\frac{k_{up}}{D}}} - e^{d_c\sqrt{\frac{k_{up}}{D}}}}\left[e^{(x-d_c)\sqrt{\frac{k_{up}}{D}}} - e^{-(x-d_c)\sqrt{\frac{k_{up}}{D}}}\right]$$

which may also be written as

$$c(x) = C_0\left[\frac{\sinh\left((x-d_c)\sqrt{\frac{k_{up}}{D}}\right)}{\sinh\left(-d_c\sqrt{\frac{k_{up}}{D}}\right)}\right]$$

where sinh is the hyperbolic sine function.

7.2 To simulate this system in COMSOL, we can utilise the 2D axisymmetric geometry shown below (all dimensions are in cm), corresponding to a single rectangular domain with axisymmetric axis coinciding with the left long edge of the rectangle. Note that the lower boundary is the site of indicator injection, and the upper left corner of the rectangular domain is the downstream site of concentration measurement.



We can also use two physics to implement the model: (1) Transport of Diluted Species and (2) Global ODEs and DAEs, and define the following parameters: Q (volume

flow rate), `R` (radius of vessel), and `M0` (total amount of indicator injected). Under the Transport of Diluted Species node, the diffusion coefficient of the indicator species is set to a user-defined value $1 \times 10^{-9}\,\mathrm{m^2\,s^{-1}}$. The components of the velocity field are also specified as a function of the total volumetric flow, Q, to correspond to a parabolic velocity profile. To determine this velocity field, we have from Eq. 7.7:

$$u = \frac{u_{\max}}{R^2}\left(R^2 - r^2\right)$$

where $u_{\max}$ is the maximum velocity at the axis of the vessel, R is its radius, and r is the radial coordinate. The total flow through the vessel is then determined by:

$$\begin{aligned} Q &= \int_0^R 2\pi r u \,\mathrm{d}r \\ &= \frac{2\pi u_{\max}}{R^2}\int_0^R r\left(R^2 - r^2\right) \\ &= \frac{2\pi u_{\max}}{R^2}\left[\frac{r^2R^2}{2} - \frac{r^4}{4}\right]_0^R \\ &= \frac{2\pi u_{\max}}{R^2}\left[\frac{R^4}{2} - \frac{R^4}{4}\right] \\ &= \frac{\pi R^2 u_{\max}}{2} \end{aligned}$$

Hence,

$$u_{\max} = \frac{2Q}{\pi R^2}$$

Substituting this expression for u_{max} into the expression for the parabolic velocity profile, we obtain

$$u = \frac{2Q}{\pi R^4}\left(R^2 - r^2\right)$$

This expression for u is entered into the velocity field for the z-component, and a value of 0 entered for the r-component. We also enter a flux boundary condition on the lower boundary (i.e. the site of injection), with flux given by

```
M0/(pi*R^2*(100 [ms]))*rect1(t [1/s])
```

where `rect1` is a user-defined rectangular function having lower limit of 0.005, upper limit 0.105, and smoothing factor 0.01. This defines a rectangular pulse of duration 0.1 that begins at $t = 0$, taking the smooth onset of the pulse into account. We also specify an outflow boundary condition for the upper boundary. Finally for this physics node, we specify 'Isotropic diffusion' under the Inconsistent stabilization tab (make sure the Stabilization option is checked under the view menu).

Under the Component definitions, we define a point integration operator (intop1) for the upper left-hand corner of the rectangular domain. In the variables sub-node,

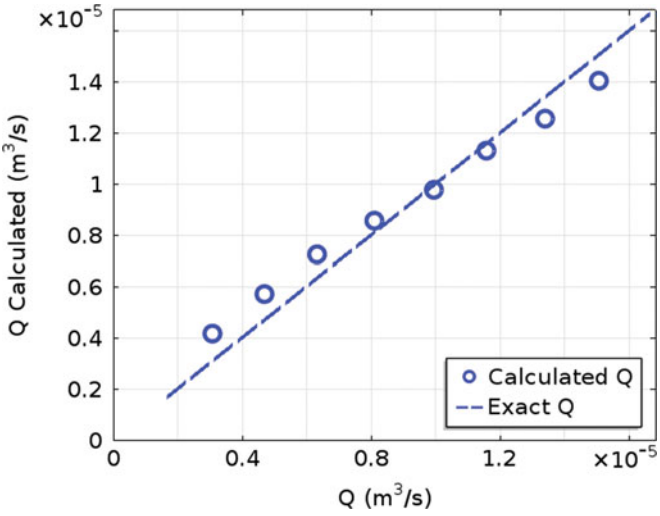
we can then define a global variable `c_d` with expression `intop1(c)` to define the indicator concentration at the downstream site. Under the Global ODEs and DAEs physics node, we can then specify a global ODE variable `c_int` satisfying the equation

`c_intt-c_d` (=0)

which states that the time-derivative of `c_int` equals `c_d`. This is equivalent to the integral

$$\texttt{c\_int} = \int_0^t \texttt{c\_d}\,\mathrm{d}t$$

We can then define a variable `c_id` with expression equal to `M0/c_int`. Finally, we mesh the model geometry using the mapped mesh option consisting of 5×500 quadrilateral elements over the rectangular domain. For the time-dependent solver, specify the times from 0 to 20 s in steps of 0.1 s, using strict time stepping. Performing a parameter sweep on parameter `Q` generates the following result for `Q_id`, evaluated at the final output time of $t = 20$ s.

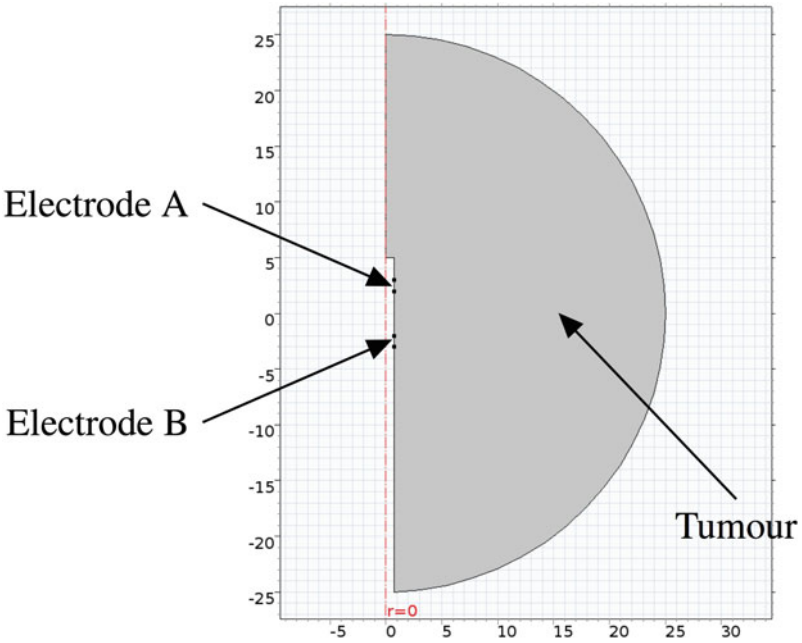


This result verifies that the method of indicator dilution is able to approximate flow rate from the expression $Q = M_0 \left[\int_0^\infty c\,\mathrm{d}t\right]^{-1}$.

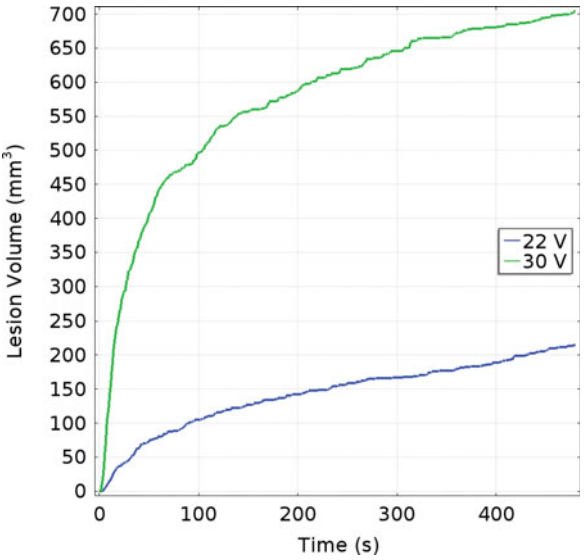
7.3 To simulate this model in COMSOL, we utilise a 2D axisymmetric geometry as shown below (all dimensions are in cm), and then follow a similar implementation to that of Example 7.2.3, using the three physics nodes (1) Electric Currents, (2) Heat Transfer in Solids and (3) the General Form PDE, with the following boundary conditions:

- An electric ground for the boundary of Electrode B

- An electric potential condition for Electrode boundary A (22 and 30 V, implemented as a parametric sweep on a defined applied voltage parameter).
- Electric insulation on all other boundaries (except the axisymmetric axis, which employs a similar axisymmetric condition).
- A temperature boundary condition (37 °C for the outer boundaries of the tumour.
- Thermal insulation on all other boundaries.



For the mesh, we set the general element size as 'Extra fine', and for the time-dependent solver, we use output times from 0 to 480 s in steps of 1 s. Defining variables and integration operators as in Example 7.2.3, we obtain the following result for the lesion volume for both 22 and 30 V probe voltages.



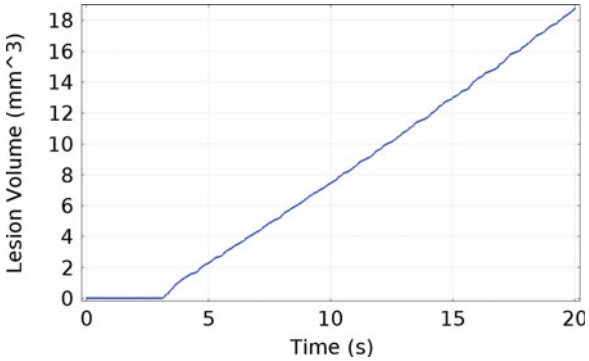
7.4 To determine the negative complex part of the permittivity of heart tissue, we can employ parameter definitions within COMSOL to enter the permittivity parameters of Eq. 7.19 and undertake the necessary complex number calculations, as shown below:

Name	Expression	Value
e_inf	4	4
delta_e1	50	50
tau_1	7.96 [ps]	7.9600E-12 s
alpha_1	0.1	0.1
delta_e2	1200	1200
tau_2	159.15 [ns]	1.5915E-7 s
alpha_2	0.05	0.05
delta_e3	4.5e5	4.5000E5
tau_3	72.34 [us]	7.2340E-5 s
alpha_3	0.22	0.22
delta_e4	2.5e7	2.5000E7
tau_4	4.547 [ms]	0.004547 s
alpha_4	0	0
e1	delta_e1/(1+(j*omega*tau_1)^(1-alpha_1))	49.999 - 0.0035632i
e2	delta_e2/(1+(j*omega*tau_2)^(1-alpha_2))	925.57 - 458.97i
e3	delta_e3/(1+(j*omega*tau_3)^(1-alpha_3))	2284.9 - 6085.7i
e4	delta_e4/(1+(j*omega*tau_4)^(1-alpha_4))	0.12252 - 1750.1i
e_r	e_inf+e1+e2+e3+e4	3264.6 - 8294.8i
epsilon_prime_prime	-epsilon0_const*imag(e_r)	7.3444E-8 F/m

Note that the last column of the above table represents the numerical values calculated by COMSOL. As can be seen from the last row of the table, COMSOL yields a value of $\varepsilon'' = 7.3444 \times 10^{-8}\,\mathrm{F\,m^{-1}}$. This parameter can then be used in the user-defined heat source as follows:

```
sigma*ec.normE^2 +
epsilon_prime_prime*omega*ec.normE^2 +
rho_b*C_b*omega_b*(T_b-T)
```

where the first term denotes the conductive (i.e. Joule) heating component, the second term denotes dielectric heating, and the third term denotes blood perfusion heating. All other model settings are as given in Example 7.2.3. Solving this model yields the following lesion volume against time plot:



Comparing this result with that of Example 7.2.3 (Fig. 7.13), we see that addition of the dielectric heating component at 500 kHz doubles the lesion volume. At this frequency, dielectric heating is therefore significant and should not be neglected when simulating RF atrial ablation.

Problems of Chap. 8

8.1 (a) Using the definitions of scalar dot and vector cross products given by Eqs. 8.1 and 8.2, we have:

$$
\begin{aligned}
(\mathbf{a}\times\mathbf{b})\cdot\mathbf{c} &= \begin{vmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}\cdot\mathbf{c} \\
&= \left\{\mathbf{e}_1\begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} - \mathbf{e}_2\begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} + \mathbf{e}_3\begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix}\right\}\cdot\mathbf{c} \\
&= (\mathbf{e}_1\cdot\mathbf{c})\begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} - (\mathbf{e}_2\cdot\mathbf{c})\begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} + (\mathbf{e}_3\cdot\mathbf{c})\begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \\
&= c_1\begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} - c_2\begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} + c_3\begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \\
&= \begin{vmatrix} c_1 & c_2 & c_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \\
&= \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix}
\end{aligned}
$$

Expanding this determinant, we obtain:

$$\begin{aligned}\begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} &= a_1 \begin{vmatrix} b_2 & b_3 \\ c_2 & c_3 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & b_3 \\ c_1 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix} \\ &= a_1 (b_2 c_3 - b_3 c_2) - a_2 (b_1 c_3 - b_3 c_1) + a_3 (b_1 c_2 - b_2 c_1) \\ &= a_1 b_2 c_3 - a_1 b_3 c_2 - a_2 b_1 c_3 + a_2 b_3 c_1 + a_3 b_1 c_2 - a_3 b_2 c_1 \\ &= \varepsilon_{ijk} a_i b_j c_k\end{aligned}$$

Hence,

$$(\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c} = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = \varepsilon_{ijk} a_i b_j c_k$$

(b) We recall that

$$\sigma_{ij}^{\mathrm{H}} = \tfrac{1}{3}\sigma_{\alpha\alpha}\delta_{ij} \qquad \varepsilon_{ij}^{\mathrm{H}} = \tfrac{1}{3}\varepsilon_{\alpha\alpha}\delta_{ij}$$

and writing these hydrostatic tensors in terms of their components, we have:

$$\boldsymbol{\sigma}^{\mathrm{H}} = \begin{pmatrix} \frac{1}{3}(\sigma_{11}+\sigma_{22}+\sigma_{33}) & 0 & 0 \\ 0 & \frac{1}{3}(\sigma_{11}+\sigma_{22}+\sigma_{33}) & 0 \\ 0 & 0 & \frac{1}{3}(\sigma_{11}+\sigma_{22}+\sigma_{33}) \end{pmatrix}$$

$$\boldsymbol{\varepsilon}^{\mathrm{H}} = \begin{pmatrix} \frac{1}{3}(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) & 0 & 0 \\ 0 & \frac{1}{3}(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) & 0 \\ 0 & 0 & \frac{1}{3}(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) \end{pmatrix}$$

For the deviatoric tensors, we have

$$\boldsymbol{\sigma}^{\mathrm{D}} = \boldsymbol{\sigma} - \boldsymbol{\sigma}^{\mathrm{H}} \qquad \boldsymbol{\varepsilon}^{\mathrm{D}} = \boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^{\mathrm{H}}$$

and therefore

$$\boldsymbol{\sigma}^{\mathrm{D}} = \begin{pmatrix} \frac{1}{3}(2\sigma_{11}-\sigma_{22}-\sigma_{33}) & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \frac{1}{3}(-\sigma_{11}+2\sigma_{22}-\sigma_{33}) & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \frac{1}{3}(-\sigma_{11}-\sigma_{22}+2\sigma_{33}) \end{pmatrix}$$

$$\boldsymbol{\varepsilon}^{\mathrm{D}} = \begin{pmatrix} \frac{1}{3}(2\varepsilon_{11}-\varepsilon_{22}-\varepsilon_{33}) & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{21} & \frac{1}{3}(-\varepsilon_{11}+2\varepsilon_{22}-\varepsilon_{33}) & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & \frac{1}{3}(-\varepsilon_{11}-\varepsilon_{22}+2\varepsilon_{33}) \end{pmatrix}$$

In particular, we note that

$$\begin{aligned}\mathrm{trace}\left(\boldsymbol{\sigma}^{\mathrm{D}}\right) &= \tfrac{1}{3}\{2\sigma_{11}-\sigma_{22}-\sigma_{33}-\sigma_{11}+2\sigma_{22}-\sigma_{33}-\sigma_{11}-\sigma_{22}+2\sigma_{33}\} = 0 \\ \mathrm{trace}\left(\boldsymbol{\varepsilon}^{\mathrm{D}}\right) &= \tfrac{1}{3}\{2\varepsilon_{11}-\varepsilon_{22}-\varepsilon_{33}-\varepsilon_{11}+2\varepsilon_{22}-\varepsilon_{33}-\varepsilon_{11}-\varepsilon_{22}+2\varepsilon_{33}\} = 0\end{aligned}$$

Writing

$$\boldsymbol{\sigma}^{\mathrm{H}} = \begin{pmatrix} \sigma_H & 0 & 0 \\ 0 & \sigma_H & 0 \\ 0 & 0 & \sigma_H \end{pmatrix}, \qquad \boldsymbol{\varepsilon}^{\mathrm{H}} = \begin{pmatrix} \varepsilon_H & 0 & 0 \\ 0 & \varepsilon_H & 0 \\ 0 & 0 & \varepsilon_H \end{pmatrix}$$

with

$$\sigma_H = \tfrac{1}{3}\left(\sigma_{11} + \sigma_{22} + \sigma_{33}\right), \qquad \varepsilon_H = \tfrac{1}{3}\left(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}\right)$$

we have:

$$\sigma_{ij}^{\mathrm{H}}\varepsilon_{ij}^{\mathrm{D}} = \sigma_{11}^{\mathrm{H}}\varepsilon_{11}^{\mathrm{D}} + \sigma_{22}^{\mathrm{H}}\varepsilon_{22}^{\mathrm{D}} + \sigma_{33}^{\mathrm{H}}\varepsilon_{33}^{\mathrm{D}} = \sigma_H \,\mathrm{trace}\left(\boldsymbol{\varepsilon}^{\mathrm{D}}\right) = 0$$

and

$$\sigma_{ij}^{\mathrm{D}}\varepsilon_{ij}^{\mathrm{H}} = \sigma_{11}^{\mathrm{D}}\varepsilon_{11}^{\mathrm{H}} + \sigma_{22}^{\mathrm{D}}\varepsilon_{22}^{\mathrm{H}} + \sigma_{33}^{\mathrm{D}}\varepsilon_{33}^{\mathrm{H}} = \varepsilon_H \,\mathrm{trace}\left(\boldsymbol{\sigma}^{\mathrm{D}}\right) = 0$$

Hence we have verified that $\sigma_{ij}^{\mathrm{H}}\varepsilon_{ij}^{\mathrm{D}} = \sigma_{ij}^{\mathrm{D}}\varepsilon_{ij}^{\mathrm{H}} = 0$.

8.2 (a) For the simple shear deformation given, we have

$$u_1 = \lambda x_2, \qquad u_2 = 0$$

Therefore,

$$\varepsilon_{11} = \frac{\partial u_1}{\partial x_1} = 0$$
$$\varepsilon_{12} = \varepsilon_{21} = \tfrac{1}{2}\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x 1}\right) = \tfrac{1}{2}\left(\lambda + 0\right) = \frac{\lambda}{2}$$
$$\varepsilon_{22} = \frac{\partial u_2}{\partial x_2} = 0$$

and

$$E_{11} = \tfrac{1}{2}\left(\frac{\partial u_1}{\partial x_1} + \frac{\partial u_1}{\partial x_1} + \frac{\partial u_1}{\partial x_1}\frac{\partial u_1}{\partial x_1}\right) = 0$$
$$E_{12} = E_{21} = \tfrac{1}{2}\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2}\frac{\partial u_2}{\partial x_1}\right) = \tfrac{1}{2}\left(\lambda + 0 + \lambda \cdot 0\right) = \frac{\lambda}{2}$$
$$E_{22} = \tfrac{1}{2}\left(\frac{\partial u_2}{\partial x_2} + \frac{\partial u_2}{\partial x_2} + \frac{\partial u_2}{\partial x_2}\frac{\partial u_2}{\partial x_2}\right) = 0$$

In matrix form, these strain tensors are

$$\boldsymbol{\varepsilon} = \begin{pmatrix} 0 & \frac{\lambda}{2} \\ \frac{\lambda}{2} & 0 \end{pmatrix}, \qquad \mathbf{E} = \begin{pmatrix} 0 & \frac{\lambda}{2} \\ \frac{\lambda}{2} & 0 \end{pmatrix}$$

(b) For the uniform inflation deformation, we have

$$u_1 = (R-1)x_1, \qquad u_2 = (R-1)x_2$$

Therefore,

$$\varepsilon_{11} = \frac{\partial u_1}{\partial x_1} = R - 1$$
$$\varepsilon_{12} = \varepsilon_{21} = \tfrac{1}{2}\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x1}\right) = 0$$
$$\varepsilon_{22} = \frac{\partial u_2}{\partial x_2} = R - 1$$

and

$$E_{11} = \tfrac{1}{2}\left(\frac{\partial u_1}{\partial x_1} + \frac{\partial u_1}{\partial x_1} + \frac{\partial u_1}{\partial x_1}\frac{\partial u_1}{\partial x_1}\right) = \tfrac{1}{2}\left(2R - 2 + (R-1)^2\right) = \tfrac{1}{2}\left((R-1)(R+1)\right)$$
$$= \tfrac{1}{2}\left(R^2 - 1\right)$$
$$E_{12} = E_{21} = \tfrac{1}{2}\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2}\frac{\partial u_2}{\partial x_1}\right) = 0$$
$$E_{22} = \tfrac{1}{2}\left(\frac{\partial u_2}{\partial x_2} + \frac{\partial u_2}{\partial x_2} + \frac{\partial u_2}{\partial x_2}\frac{\partial u_2}{\partial x_2}\right) = \tfrac{1}{2}\left(2R - 2 + (R-1)^2\right) = \tfrac{1}{2}\left((R-1)(R+1)\right)$$
$$= \tfrac{1}{2}\left(R^2 - 1\right)$$

In matrix form, these strain tensors are

$$\boldsymbol{\varepsilon} = \begin{pmatrix} R-1 & 0 \\ 0 & R-1 \end{pmatrix}, \qquad \mathbf{E} = \begin{pmatrix} \frac{1}{2}\left(R^2-1\right) & 0 \\ 0 & \frac{1}{2}\left(R^2-1\right) \end{pmatrix}$$

(c) In the case of rotation, a point originally at (x_1, x_2) is rotated to the new point $(\bar{x}_1, \bar{x}_2)$ according to the rotation transformation:

$$\begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Hence the displacements are given by

$$u_1 = x_1\cos\theta - x_2\sin\theta - x_1 = x_2\left(\cos\theta - 1\right) - x_2\sin\theta$$
$$u_2 = x_2\cos\theta + x_1\sin\theta - x_2 = x_1\left(\cos\theta - 1\right) + x_1\sin\theta$$

Therefore,

$$\varepsilon_{11} = \frac{\partial u_1}{\partial x_1} = \cos\theta - 1$$
$$\varepsilon_{12} = \varepsilon_{21} = \tfrac{1}{2}\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x 1}\right) = \tfrac{1}{2}\left(-\sin\theta + \sin\theta\right) = 0$$
$$\varepsilon_{22} = \frac{\partial u_2}{\partial x_2} = \cos\theta - 1$$

and

$$E_{11} = \tfrac{1}{2}\left(\frac{\partial u_1}{\partial x_1} + \frac{\partial u_1}{\partial x_1} + \frac{\partial u_1}{\partial x_1}\frac{\partial u_1}{\partial x_1}\right) = \tfrac{1}{2}\left(2(\cos\theta - 1) + (\cos\theta - 1)^2\right)$$
$$= \tfrac{1}{2}\left((\cos\theta - 1)(\cos\theta + 1)\right) = \tfrac{1}{2}\left(\cos^2\theta - 1\right) = -\tfrac{1}{2}\sin^2\theta$$
$$E_{12} = E_{21} = \tfrac{1}{2}\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2}\frac{\partial u_2}{\partial x_1}\right) = \tfrac{1}{2}\left(-\sin\theta + \sin\theta + -\sin\theta\cdot\sin\theta\right)$$
$$= -\tfrac{1}{2}\sin^2\theta$$
$$E_{22} = \tfrac{1}{2}\left(\frac{\partial u_2}{\partial x_2} + \frac{\partial u_2}{\partial x_2} + \frac{\partial u_2}{\partial x_2}\frac{\partial u_2}{\partial x_2}\right) == \tfrac{1}{2}\left(2(\cos\theta - 1) + (\cos\theta - 1)^2\right)$$
$$= \tfrac{1}{2}\left((\cos\theta - 1)(\cos\theta + 1)\right) = \tfrac{1}{2}\left(\cos^2\theta - 1\right) = -\tfrac{1}{2}\sin^2\theta$$

In matrix form, these strain tensors are therefore given by

$$\boldsymbol{\varepsilon} = \begin{pmatrix} \cos\theta - 1 & 0 \\ 0 & \cos\theta - 1 \end{pmatrix}, \qquad \mathbf{E} = \tfrac{1}{2}\begin{pmatrix} -\sin^2\theta & -\sin^2\theta \\ -\sin^2\theta & -\sin^2\theta \end{pmatrix}$$

8.3 (a) For an incompressible deformation, the volume of the spherical shell must be preserved. That is:

$$\tfrac{4}{3}\pi\left(b^3 - a^3\right) = \tfrac{4}{3}\pi\left(B^3 - A^3\right)$$

Therefore,

$$b^3 = a^3 + B^3 - A^3$$
$$b = \sqrt[3]{a^3 + B^3 - A^3}$$

(b) For a particle in the myocardial wall initially located at a radial position of R, we can use a similar analysis as part a) to show that the volume of the shell between radial distances of A and R must be preserved. Hence, if this particle moves to a new radial position of r, then we must have

$$r = \sqrt[3]{a^3 + R^3 - A^3}$$

which corresponds to a radial displacement $u(R) = r - R$, with u_1, u_2, u_3 components of

$$u_1 = \left(\frac{x_1}{R}\right) u(R) = \frac{x_1}{R}\left(\sqrt[3]{a^3 + R^3 - A^3} - R\right) = x_1\left(\sqrt[3]{1 + \frac{a^3 - A^3}{R^3}} - 1\right)$$
$$u_2 = \left(\frac{x_2}{R}\right) u(R) = \frac{x_2}{R}\left(\sqrt[3]{a^3 + R^3 - A^3} - R\right) = x_2\left(\sqrt[3]{1 + \frac{a^3 - A^3}{R^3}} - 1\right)$$
$$u_3 = \left(\frac{x_3}{R}\right) u(R) = \frac{x_3}{R}\left(\sqrt[3]{a^3 + R^3 - A^3} - R\right) = x_3\left(\sqrt[3]{1 + \frac{a^3 - A^3}{R^3}} - 1\right)$$

where $R = \sqrt{x_1^2 + x_2^2 + x_3^2}$. For convenience, we let $\gamma = \sqrt[3]{1 + \frac{a^3 - A^3}{R^3}} - 1$ and write these displacement components as

$$u_1 = \gamma x_1, \quad u_2 = \gamma x_2, \quad u_3 = \gamma x_3$$

To determine the components of Cauchy strain, we need to determine the partial derivatives of these displacements with respect to x_1, x_2 and x_3. These in turn require the following derivatives:

$$\begin{aligned}
\frac{\mathrm{d}\gamma}{\mathrm{d}R} &= \tfrac{1}{3}\left(1 + \frac{a^3 - A^3}{R^3}\right)^{-\frac{2}{3}}\left[\frac{-3\left(a^3 - A^3\right)}{R^4}\right] \\
&= -\left[\frac{a^3 - A^3}{R^4}\right]\left(1 + \frac{a^3 - A^3}{R^3}\right)^{-\frac{2}{3}} \\
&= -\left[\frac{(\gamma+1)^3 - 1}{R}\right](\gamma+1)^{-2} \\
&= -\frac{1}{R}\left[\gamma + 1 - \frac{1}{(\gamma+1)^2}\right] \\
\frac{\partial R}{\partial x_1} &= \tfrac{1}{2}\left(x_1^2 + x_2^2 + x_3^2\right)^{-\frac{1}{2}} 2x_1 = \frac{x_1}{R} \\
\frac{\partial R}{\partial x_2} &= \tfrac{1}{2}\left(x_1^2 + x_2^2 + x_3^2\right)^{-\frac{1}{2}} 2x_2 = \frac{x_2}{R} \\
\frac{\partial R}{\partial x_3} &= \tfrac{1}{2}\left(x_1^2 + x_2^2 + x_3^2\right)^{-\frac{1}{2}} 2x_3 = \frac{x_3}{R}
\end{aligned}$$

Hence, the derivatives of displacement u_1 with respect to each of x_1, x_2, and x_3 are:

$$\frac{\partial u_1}{\partial x_1} = \gamma + x_1\frac{\partial \gamma}{\partial x_1} = \gamma + x_1\frac{\mathrm{d}\gamma}{\mathrm{d}R}\frac{\partial R}{\partial x_1} = \gamma - \frac{x_1^2}{R^2}\left[\gamma + 1 - \frac{1}{(\gamma+1)^2}\right]$$
$$\frac{\partial u_1}{\partial x_2} = x_1\frac{\partial \gamma}{\partial x_2} = x_1\frac{\mathrm{d}\gamma}{\mathrm{d}R}\frac{\partial R}{\partial x_2} = -\frac{x_1 x_2}{R^2}\left[\gamma + 1 - \frac{1}{(\gamma+1)^2}\right]$$

$$\frac{\partial u_1}{\partial x_3} = x_1 \frac{\partial \gamma}{\partial x_3} = x_1 \frac{\mathrm{d}\gamma}{\mathrm{d}R} \frac{\partial R}{\partial x_3} = -\frac{x_1 x_3}{R^2} \left[\gamma + 1 - \frac{1}{(\gamma+1)^2} \right]$$

Again for convenience, we let $\beta = \gamma + 1 - \frac{1}{(\gamma+1)^2}$. The above derivatives then become:

$$\frac{\partial u_1}{\partial x_1} = \gamma - \frac{\beta x_1^2}{R^2}, \quad \frac{\partial u_1}{\partial x_2} = -\frac{\beta x_1 x_2}{R^2}, \quad \frac{\partial u_1}{\partial x_3} = -\frac{\beta x_1 x_3}{R^2}$$

Similarly, we obtain the remaining derivatives of the other displacement terms as

$$\frac{\partial u_2}{\partial x_1} = -\frac{\beta x_1 x_2}{R^2}, \quad \frac{\partial u_2}{\partial x_2} = \gamma - \frac{\beta x_2^2}{R^2}, \quad \frac{\partial u_2}{\partial x_3} = -\frac{\beta x_2 x_3}{R^2}$$
$$\frac{\partial u_3}{\partial x_1} = -\frac{\beta x_1 x_3}{R^2}, \quad \frac{\partial u_2}{\partial x_3} = -\frac{\beta x_2 x_3}{R^2}, \quad \frac{\partial u_3}{\partial x_3} = \gamma - \frac{\beta x_3^2}{R^2}$$

The Cauchy strain is given by $\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$. Inserting the above derivatives, we obtain its following matrix form:

$$\boldsymbol{\varepsilon} = \begin{pmatrix} \gamma - \frac{\beta x_1^2}{R^2} & -\frac{\beta x_1 x_2}{R^2} & -\frac{\beta x_1 x_3}{R^2} \\ -\frac{\beta x_1 x_2}{R^2} & \gamma - \frac{\beta x_2^2}{R^2} & -\frac{\beta x_2 x_3}{R^2} \\ -\frac{\beta x_1 x_3}{R^2} & -\frac{\beta x_2 x_3}{R^2} & \gamma - \frac{\beta x_3^2}{R^2} \end{pmatrix}$$

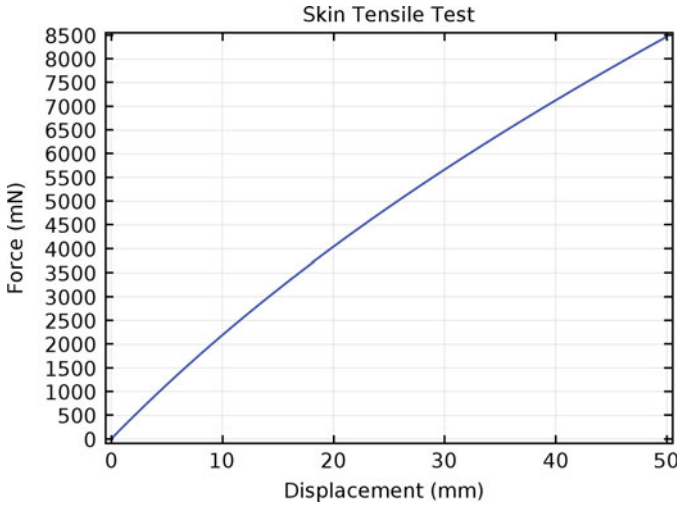
where, as indicated earlier, $\gamma = \sqrt[3]{1 + \frac{a^3 - A^3}{R^3}} - 1$ and $\beta = \gamma + 1 - \frac{1}{(\gamma+1)^2}$.

8.4 (a) Since the skin thickness of the sample is small compared to its other dimensions, a 2D implementation in COMSOL is preferable, using the plane stress condition (i.e. there are no components of stress perpendicular to the plane). Since the model is symmetric, we can implement only half of the geometry, employing a symmetric boundary condition on the lower face. The following are key points regarding this COMSOL implementation

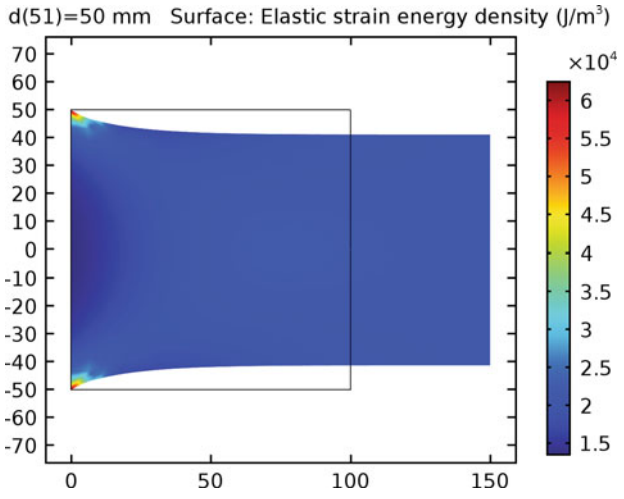
- In the Solid Mechanics settings, specify a thickness of 1 mm and the plane stress 2D approximation mode.
- Specify a hyperelastic material employing the Mooney-Rivlin, two parameters material model. Enter all user-defined material values as given.
- Specify a prescribed displacement boundary condition on the right boundary with only a prescribed x-displacement of `d`, where `d` is a user-defined global model parameter. Specify a symmetry boundary condition for the lower boundary a fixed constraint boundary condition for the left boundary.
- Define an integration boundary operator, `intop1`, acting over the right boundary. Define a model variable `F` representing the applied force, given by the expression `2*intop1(solid.Tax)*(1 [mm])`. In this expression, `Tax` denotes a COMSOL in-built variable for the x-component of traction, `(1 [mm])` denotes

the skin thickness, and the factor 2 takes into account that only half the geometry is implemented.

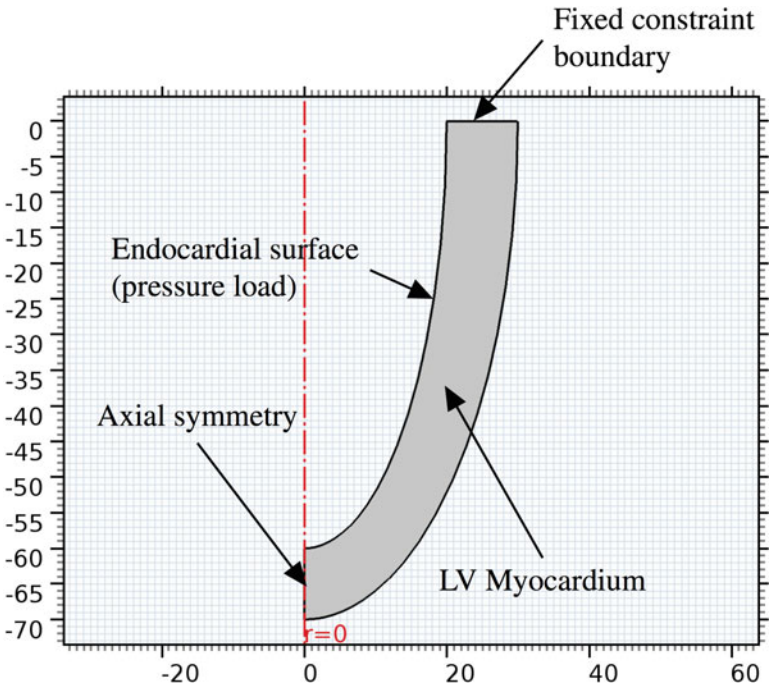
- Perform a parameter sweep of parameter `d` from 0 to 50 mm.
- Setup a 1D plot group (global plot) to plot the resulting applied force vs displacement, as shown below:



(b) To plot the elastic strain energy, first create a 2D mirror dataset so that the entire geometry of the skin sample can be visualised. Setup a new 2D plot group (surface plot) and plot COMSOL's in-built elastic strain energy variable `solid.ws`. Right-click the surface plot-sub-node and specify deformation with a scale factor of 1. The resulting plot of strain energy in the deformed sample is shown below:



8.5 To implement this model in COMSOL, setup the 2D axisymmetric geometry shown below, where all dimensions are shown in mm:



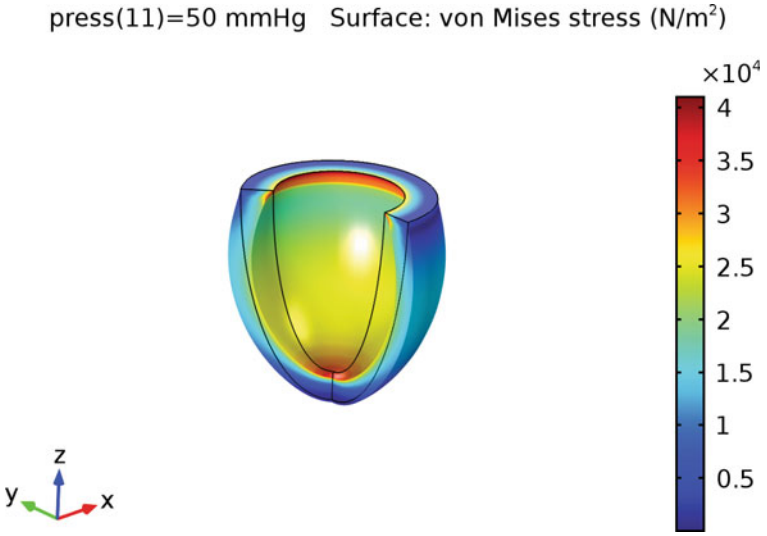
Additional points regarding model implementation are as follows:

- Define a model parameter `press` for the endocardial pressure with a default value of `50[mmHg]`.
- For the model geometry, use two ellipses for the epi and endocardial surfaces, perform a boolean subtraction of one from the other, and subtract a rectangle covering their upper half from both.
- Define an integration boundary operator, `intop1`, acting over the endocardial edge. This operator is then used in a variable expression to calculate the volume of the LV cavity according to `intop1(-pi*nr*r^2)`. Note the negative sign for `nr` is to specify the outward normal of the LV *cavity* as opposed to the LV wall. To understand this expression, we note that the volume of a solid of revolution is given by

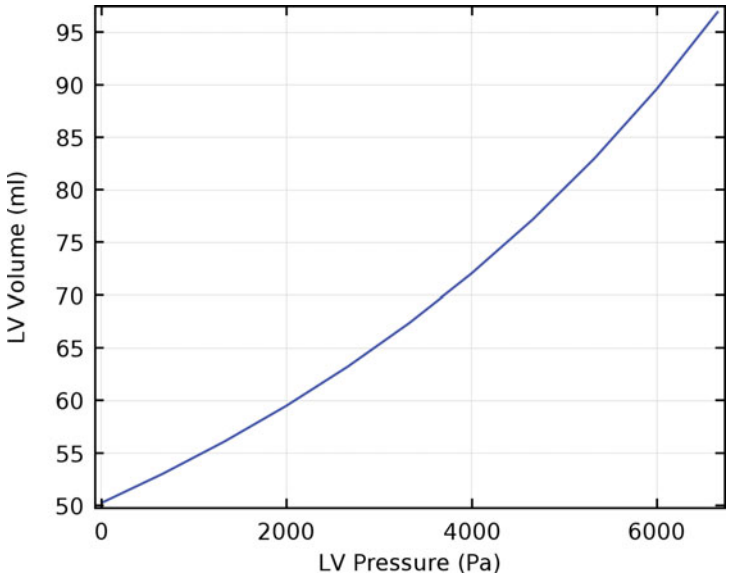
$$\begin{aligned} V &= \int_A 2\pi r \,\mathrm{d}r\,\mathrm{d}z \\ &= \int_A \pi \nabla \cdot \begin{pmatrix} r^2 \\ 0 \end{pmatrix} \mathrm{d}A \quad (\text{where } \mathrm{d}A = \mathrm{d}r\,\mathrm{d}z) \\ &= \int_L \pi n_r r^2 \,\mathrm{d}L \quad (\text{which follows from the divergence theorem}) \end{aligned}$$

where L is the edge boundary of the axisymmetric solid of revolution, and n_r is the r-component of the outward normal along the boundary.

- Select the hyperelastic/Mooney-Rivlin, two parameters material model and enter all material coefficients as described.
- For the base boundary, select the Fixed Constraint boundary condition. For the endocardial edge, select the Boundary Load boundary condition, and specify the load type as 'pressure'. Enter the value as the pressure parameter `press`.
- For the parameter sweep, specify parameter `press` ranging from 0 to 50 mmHg in steps of 5 mmHg.
- For the mesh, select the 'Fine' mesh setting.
- Solving the model produces the following von Mises stress distribution in the deformed (i.e. inflated) state at 50 mmHg, where we have used the default 3D stress plot:



- Specifying a new 1D Plot Group (Global plot), produces the following plot of variable `V` against `press`:



Problems of Chap. 9

9.1 Following the same principles as Sect. 9.1.1, we can represent the fluid motion in the circular tube using a series of sliding tubes. Denoting the radius of one such tube as r, the forces acting on its surface boundaries will be:

1. the force on the upstream end due to the upstream pressure, given by $F_{up} = \pi r^2 P$.
2. the force on the downstream end due to the downstream pressure, given by $F_{down} = 0$, since in this case there is no downstream pressure.
3. the traction acting on the curved surface of the tube, due to the viscous force from the relative velocities of layers sliding past each other. This viscous force equals the viscous stress τ multiplied by the curved surface area, or

$$F_{viscous} = 2\pi r L\tau = 2\pi r L\mu \frac{\partial v}{\partial r}$$

where v is the fluid velocity.

Adding the above three forces together yields the total force on the inner tube, which is the mass of the tube multiplied by its acceleration. This total force is given by

$$F = \int_0^r 2\pi r L\rho \frac{\partial v}{\partial t}\,\mathrm{d}r$$

Hence,

$$\pi r^2 P + \mu \frac{\partial v}{\partial r} 2\pi r L = F = \int_0^r 2\pi r L\rho \frac{\partial v}{\partial t}\,\mathrm{d}r$$

Differentiating both sides with respect to r, we obtain:

$$2\pi r P + \frac{\partial^2 v}{\partial r^2} 2\pi r \mu L + 2\pi \mu L \frac{\partial v}{\partial r} = 2\pi r L \rho \frac{\partial v}{\partial t}$$

Dividing throughout by $2\pi r \mu L$ and re-arranging:

$$\frac{\rho}{\mu}\frac{\partial v}{\partial t} - \frac{\partial^2 v}{\partial r^2} - \frac{1}{r}\frac{\partial v}{\partial r} = \frac{P}{\mu L}$$

or

$$\frac{\rho}{\mu}\frac{\partial v}{\partial t} + \frac{1}{r}\frac{\partial}{\partial r}\left[-r\frac{\partial v}{\partial r}\right] = \frac{P}{\mu L}$$

Multiplying throughout by r, we obtain the resulting PDE:

$$\frac{\rho r}{\mu}\frac{\partial v}{\partial t} + \frac{\partial}{\partial r}\left[-r\frac{\partial v}{\partial r}\right] = \frac{Pr}{\mu L}$$

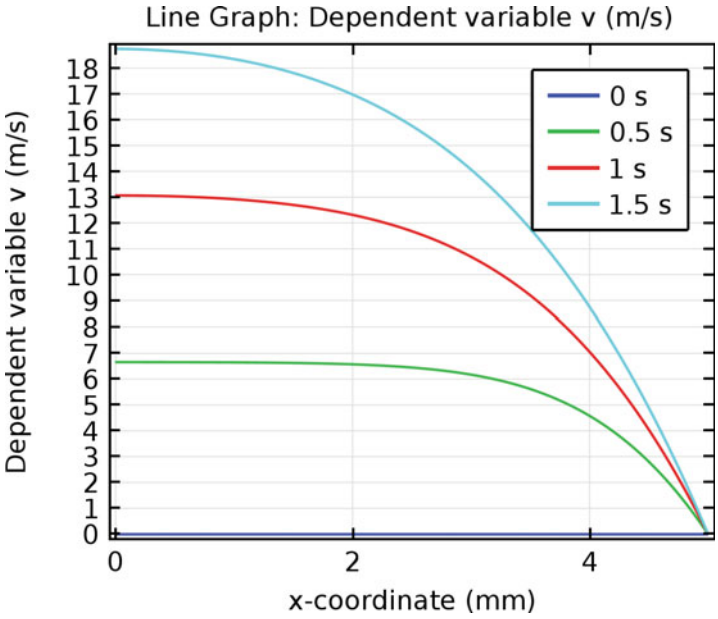
with initial and boundary conditions on $v(r, t)$ given by

$$\begin{aligned} v(r, 0) &= 0 \quad \text{(initial value)} \\ v(D/2, t) &= 0 \quad \text{(no} - \text{slip wall)} \\ \partial v(0, t)/\partial r &= 0 \quad \text{(radial symmetry at } r = 0\text{)} \end{aligned}$$

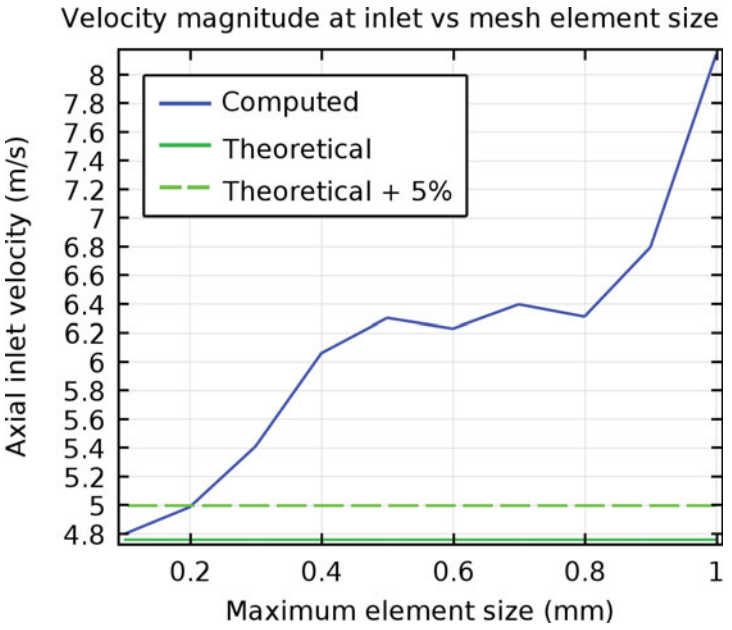
To solve this PDE, we can use COMSOL's mathematics PDE interface (General Form PDE) over a 1D spatial dimension, using the following settings:

$$\begin{aligned} \text{flux}: &\quad \Gamma = -r\partial v/\partial r \\ \text{damping coefficient}: &\quad d_a = \rho r/\mu \\ \text{source term}: &\quad f = Pr/\mu L \end{aligned}$$

and using $r \equiv$ `x` for COMSOL's 1D PDE General form, the above quantities are written as `-x*vx`, `rho*x/mu`, and `P*x/(mu*L)` respectively. Implementing this PDE in COMSOL, using a Dirichlet boundary condition of $v = 0$ at $x = D/2$ and zero-flux boundary condition at $x = 0$, produces the following solution for v at $t = 0,\ 0.5,\ 1,$ and 1.5 s:

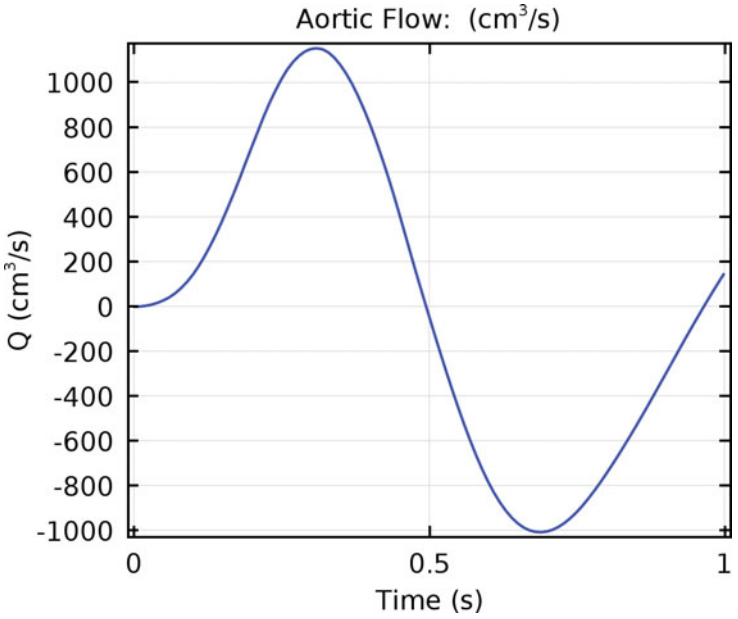


9.2 We can implement the axisymmetric model of Sect. 9.1.1 in COMSOL using an additional parameter `mesh` to denote the maximum element size. Specifying a custom mesh size with maximum element size of `mesh`, and performing a parameter sweep, results in the following plot of axial velocity at the inlet of the tube as a function of element size:



from which it is evident that a maximum element size of 0.2 mm achieves an error of 5% of theoretical axial velocity of $V_{th} = \frac{\Delta P D^2}{16 \mu L} \approx 4.76\,\mathrm{m\,s^{-1}}$.

9.3 We can utilize the COMSOL model of Sect. 9.4.2, this time defining an interpolation function (under Global Definitions | Functions | Interpolation) and entering a table of the pressure values and times specified. In the Units tab of the interpolation function settings, specify the units of the function argument as 'ms' and the function output as 'mmHg'. Under the Interpolation and Extrapolation tab, specify the interpolation type as 'Cubic Spline'. This will create an interpolation function with default name 'int1'. Under the component 1 variable definitions, we can then specify the variable `P_in` as the expression `int1(t)`. Specify the mesh element size as 'Finer', and specify the global ODE variable `P_out` to have the initial value `72 [mmHg]`. All other settings are the same as per the COMSOL model of Sect. 9.4.2. Using these settings, the resulting plot of aortic flow is shown below:


Aortic Flow: (cm³/s)
Q (cm³/s)
1000
800
600
400
200
0
-200
-400
-600
-800
-1000
0
0.5
1
Time (s)

Index

S. Dokos, *Modelling Organs, Tissues, Cells and Devices*,
Lecture Notes in Bioengineering, DOI 10.1007/978-3-642-54801-7