Instructions for using ARTIMMUS

Dr. Mark Read ~ http://markread.info
http://www.ycil.org.uk/
14th May 2013

About this document, and this simulation

This document serves as instructions on how to install and use ARTIMMUS (artificial murine multiple sclerosis simulation). This extends beyond simply Eclipse, but running it on a cluster (the Sun Grid Engine, though there may be similarities with other clusters), and running the data analysis scripts on simulation data.

I have tried to make these instructions comprehensive, and I hope they cover just about everything (at least at a high level). But I will reiterate the standard research-software-disclaimer. This is a complicated piece of software, developed to meet research needs. Sadly we are not funded to maintain it, or to spend time bug-testing every possible use-case. It will require some (perhaps quite a bit) of effort on your part to get the cluster and analysis script sections working, there's quite a bit to get familiar with. I have done my best to comment code, and use informative (though long) naming conventions for everything in the hope it eases the learning-curve for others. In my humble experience, setting up experiments on the cluster or running the analysis scripts usually does not work first time, you will probably need to dig a little deeper into what is actually going on. This is normal (for research in general, not just this software), it happens to me *a lot*. Please don't let it frustrate you. You are very welcome to contact me with specific questions, I will do my best to answer, if I am able. The best way to reach me is via the contact details on my website which I keep up to date.

The vast majority of this software has been written by myself, Mark Read. Richard Alun Williams and Richard Greaves have also developed parts of the software in the course of their respective Master's degrees.

The biology underpinning this simulation is explained in the "supporting-information" PDF, supplied in "Treg_2D/papers-resources".

I hope this software is of use to you, and I wish you the very best of luck!

System requirements

ARTIMMUS has been developed in Eclipse, using java. It relies on the MASON simulation framework: *http://cs.gmu.edu/~eclab/projects/mason/*
However, MASON is packaged with ARTIMMUS as a jar file, so there is no need to download it separately. You do not need a detailed understanding of how MASON works to use this simulation, use the existing code as a template if you intend to develop the code yourself.

I have developed ARTIMMUS using MacOS, and using eclipse it should (in theory) be platform-independent. ARTIMMUS can be run in a GUI, and this can be launched from eclipse.

For more serious large-scale experimentation and data analysis, I have used the following:
- Access to a computational cluster (mine ran the Sun Grid Engine) on which to run experiments.
- A unix operating system (I have used Ubuntu for this), though MacOS should suffice.
  Really, this is to make use of bash. Other terminals should work too, though I haven't tested them.
- Ruby, a scripting language. This is used in the data analysis scripts.
- Matlab, with access to the Statistics toolbox. Also used for data analysis.

Opening ARTIMMUS as an Eclipse project

First, you will need eclipse. The following has been tested with the Helios eclipse release, which I believe includes java. ARTIMMUS is intended to run on java 5 or later. I'm afraid I cannot answer here questions pertaining to getting java working on eclipse, there's lots of online help for that. This assumes eclipse and java are correctly installed and working together.

The ARTIMMUS simulation is contained within the "Treg_2D" directory.

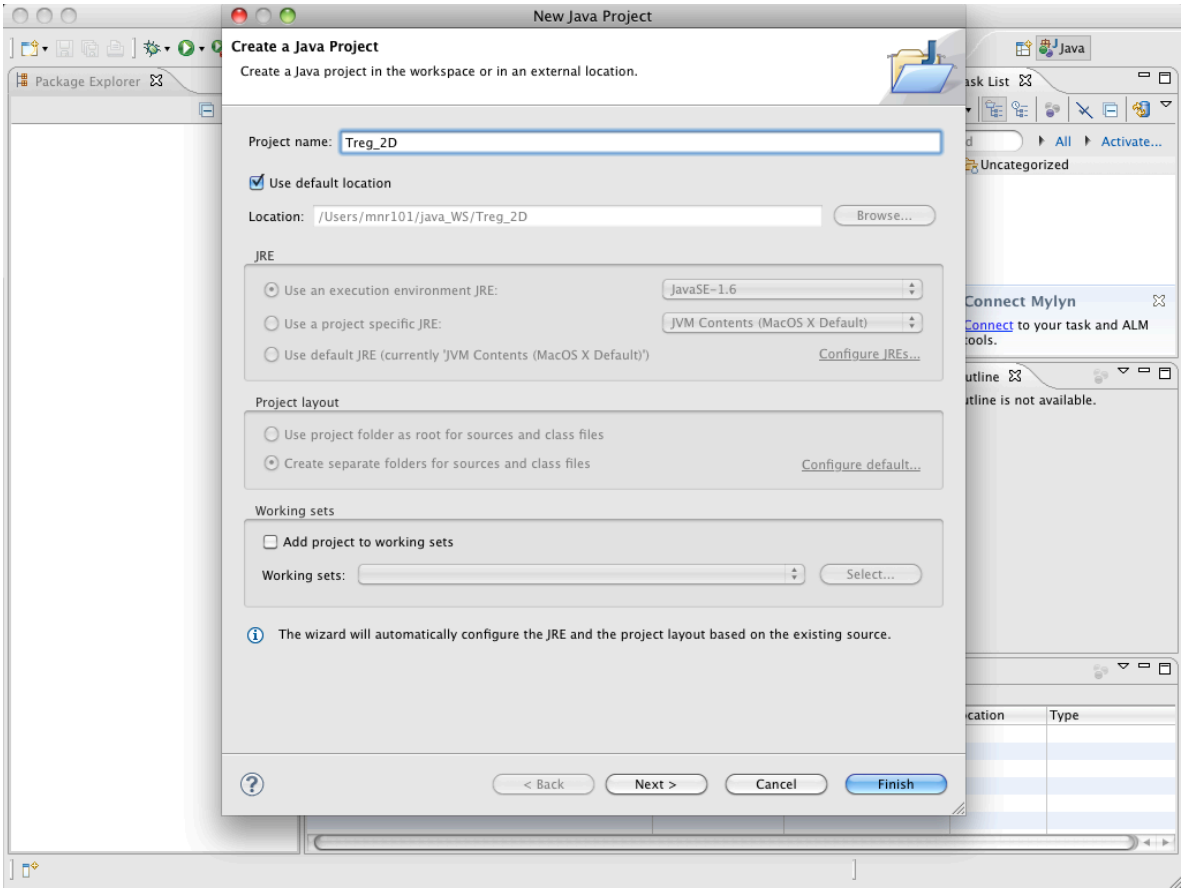
******* PLEASE NOTE THAT Treg_2D SHOULD NOT BE RENAMED *******

There are data analysis scripts that locate one another based on locating a directory of this name somewhere in the current absolute directory path. If Treg_2D is renamed, a lot of scripts will need to be edited. Otherwise, it does not matter where Treg_2D is located on your filesystem.

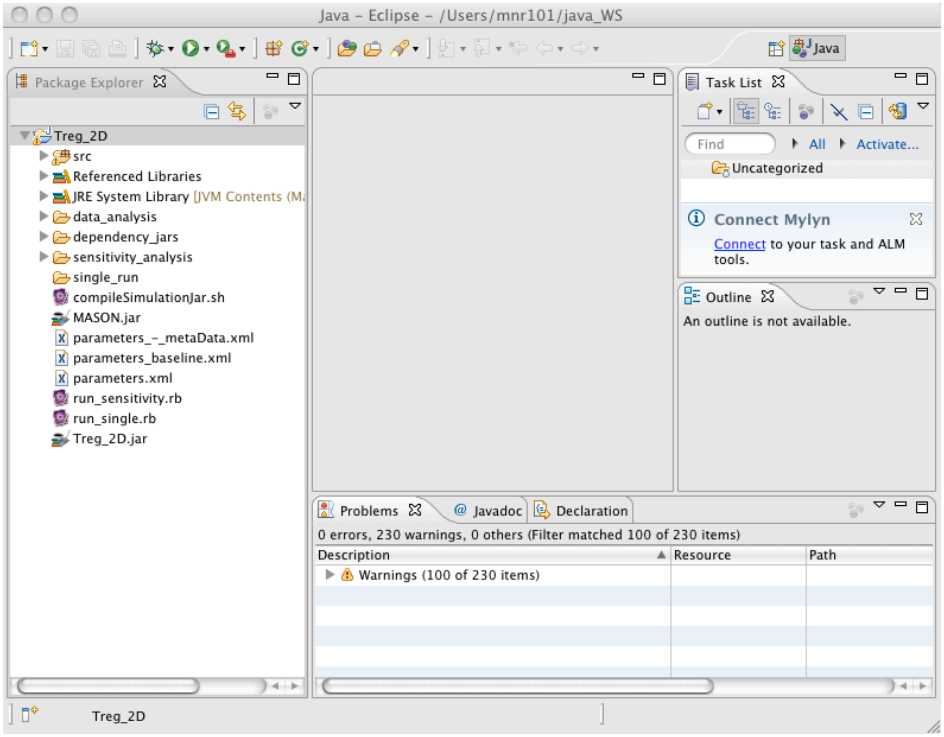
You will need a workspace for eclipse. I have traditionally used "java_workspace" for this, again it does not matter where this directory is located. Copy the Treg_2D directory and all of its contents into your eclipse workspace (referred to hereafter as "java_workspace"). Launch Eclipse, and go "java_workspace". Open the following in Eclipse:

File -> new -> Java Project

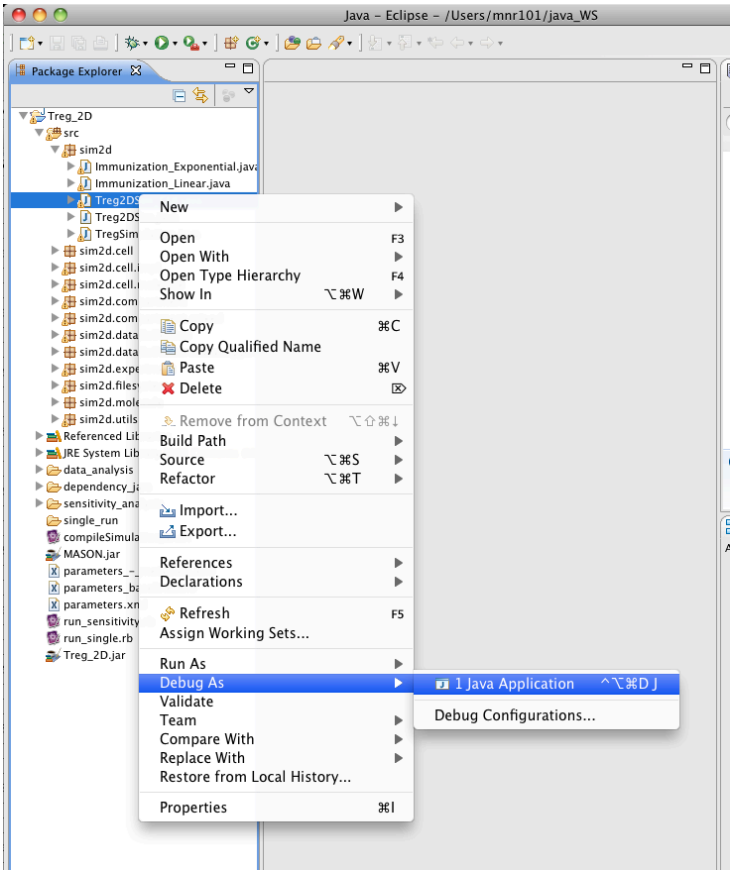
Type "Treg_2D" into the project name box.



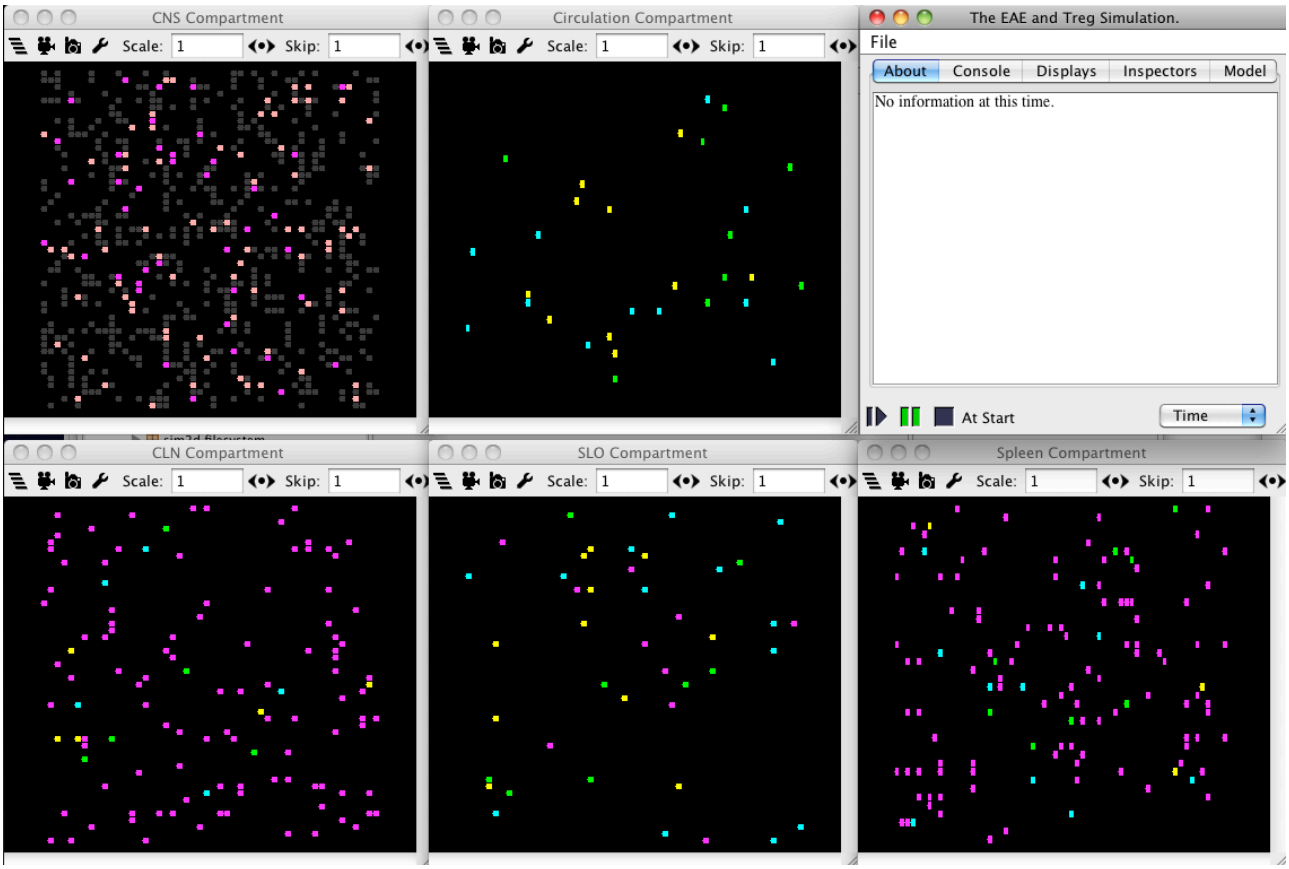
and click Finish. You should be looking at something a bit like this.



“src” contains the source files of the simulation. Check everything works, try running the simulation within eclipse as follows. Locate the java source file named “Treg2DSim_GUI.java” under “src/sim2d/”, and run it as a java application, by right clicking on it as follows:

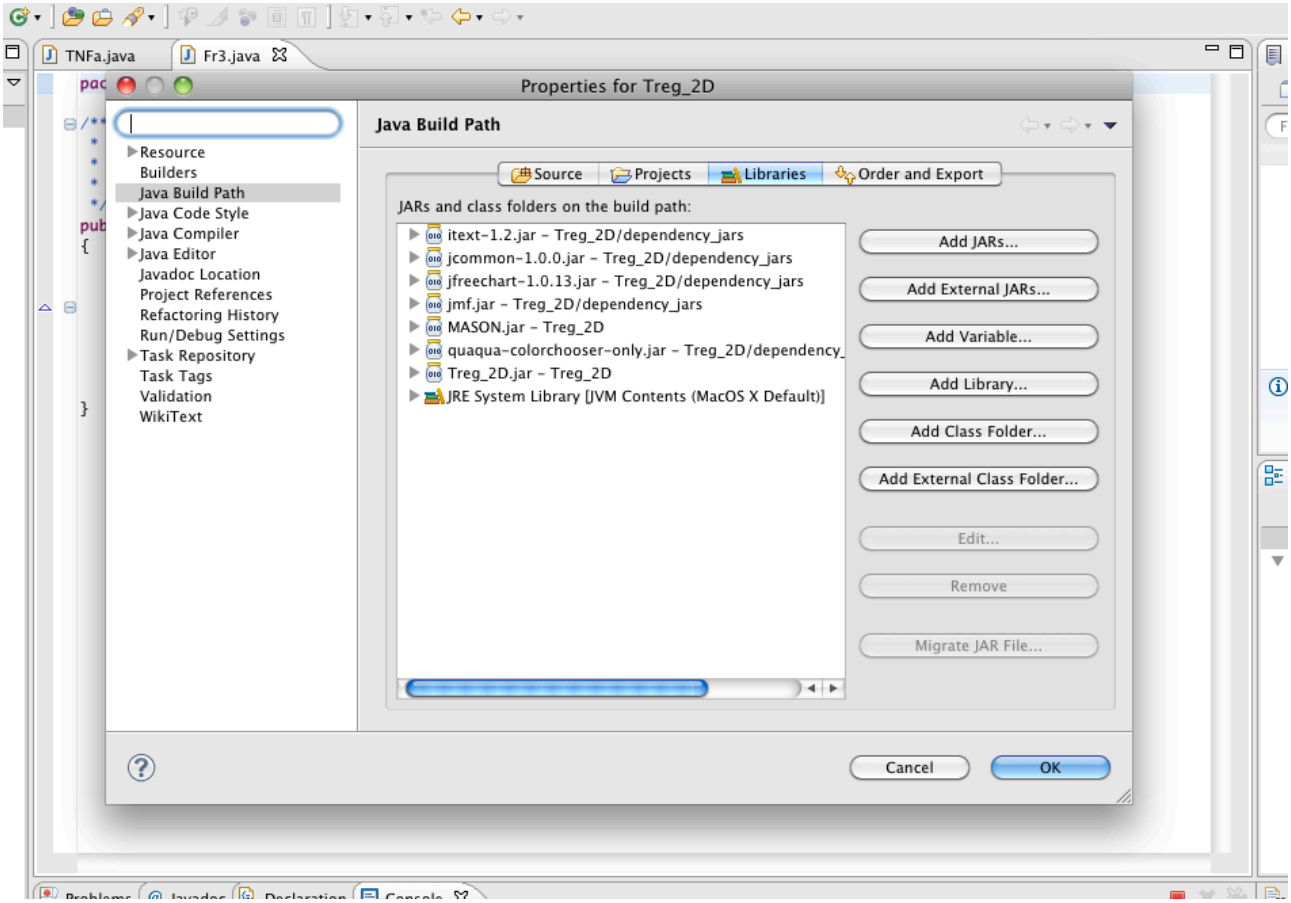


Six windows should appear, one of them is labeled “The EAE and Treg Simulation”, it has a pause button. Click it, and the windows should fill with the following:



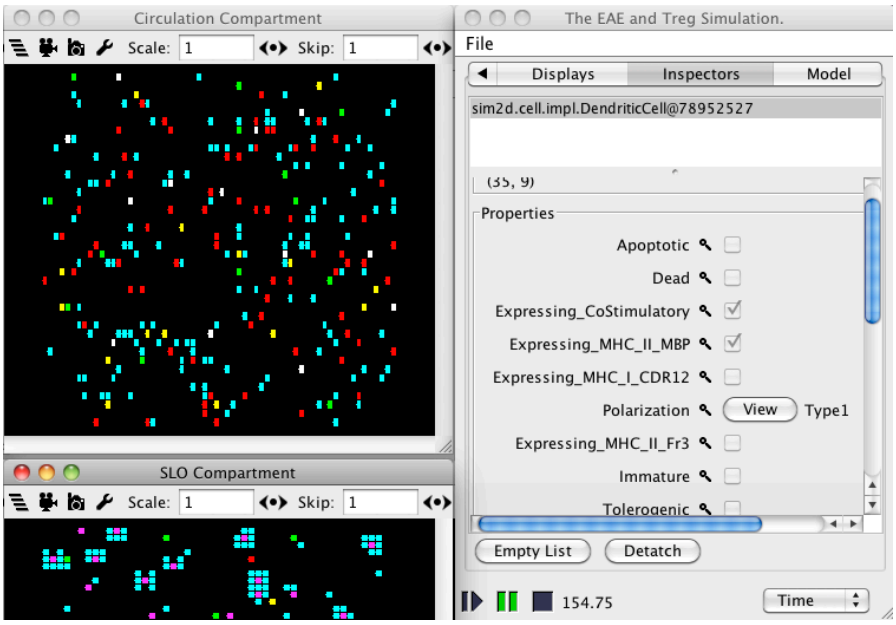
If you click pause again, the simulation starts. Cells (the dots in the windows) will move downwards. For convenience downwards is always assumed to be the direction of blood flow though a compartment.

If the above does not happen, and you instead get error messages, the problem is nearly always related to the java build path. The directory “Treg_2D/dependency_jars” contains jar files that should be added to the Treg_2D java project. This is done by right-clicking the Treg_2D project in the package explorer (traditionally on the left, as in all the above screenshots) and selecting “Properties”. In the window that pops up, select “Java Build Path”, and ensure the following are all present and correct (you may not need quaqua, it seems to be a MacOS specific jar). If any/all of them are missing, add them by clicking “Add External JARs”, and navigate to “dependency_jars” to select them.



What all of this is, and where to find out more information about the biology behind ARTIMMUS

Assuming you have succeeded in launching the simulation GUI, you can double click a cell in a compartment to inspect its state, as follows:

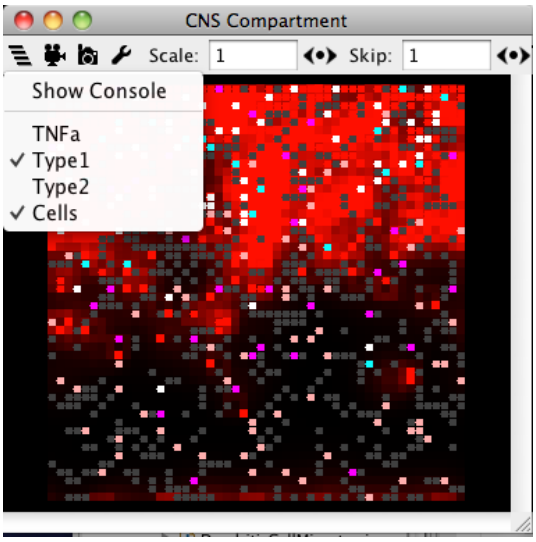


This is very useful for understanding what cells in the simulation are doing. Cells are color coded as follows:
Grey – neurons (often referred to as CNS cells)
Purple – dendritic cells
Peach – microglia (often referred to as CNS macrophages)

Teal – naïve CD4Th cells
Red – encephalitogenic MBP-specific CD4Th1 cells
White – MBP-specific CD4Th2 cells
Yellow – CD4Treg cells
Green – CD8Treg cells

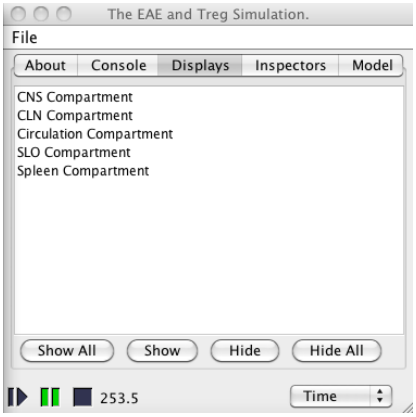
The 5 spatial compartments represented in ARTIMMUS are the CNS, central nervous system; the CLN, the cervical lymph node that drains it; the SLO (secondary lymphoid organ), a generic lymph node draining the region of the mouse where immunization for EAE is subcutaneously administered; the spleen; and the circulatory compartment that links all compartments together (with the exception of cells that leave the CNS and go directly into the CLN). The circulatory compartment can be thought of as “the rest of the body”.

It is also possible to show concentrations of cytokines in the GUI. ARTIMMUS represents TNF-a, and has two generic cytokines that abstractly capture the properties of type 1 and type 2 cytokines. To show these, click the top left button on any of the compartments (the one that looks like 4 horizontal lines not quite aligned), and check whichever views you would like to see:



There is a great deal that can be said about how the biology of this particular murine model of EAE is abstracted into the simulation, and interested readers are directed to the accompanying “supporting-information.pdf” document, or Mark Read’s thesis, which explains much of this. Both are included with this distribution of ARTIMMUS, under “Treg_2D/papers-resources”. Section 2.2 explains the biology of EAE as represented in ARTIMMUS. Chapter 4 provides a detailed explanation of cellular dynamics captured in ARTIMMUS. Chapter 5 provides some implementation details, a very useful class inheritance diagram that outlines code organization, and details of how immunization for EAE is captured in the simulation. Chapter 5 also explains how the simulation has been parameterized against experimental data. Section 6.2 outlines how simulation executions are graded on the 0-5 disease severity scale. The rest of chapter 6 and chapter 7 demonstrate the sorts of experimentation that can be performed using ARTIMMUS.

As the encephalitogenic and regulatory immune responses are mounted and large numbers of cells are created in the simulation, it slows down (there is more computation to do when there are more cells). It can be sped up by hiding the display windows, the buttons for which are located here:



Scoping out behaviors

When clicking “stop” on the GUI, a variety of graphs will be displayed. These depict various aspects of simulation behavior. They are useful for getting a feel of the simulation, or scoping out potential experiments. However, a larger number of much more accurate (compiled from median data) graphs can be obtained by using the analysis scripts included in the “data_analysis” directory, described below. This requires many executions of simulation behavior be sampled, and as such access to a cluster is useful. Furthermore, assigning autoimmunity severity scores to simulation executions can only be done post-experimentation (ie, not here).

ARTIMMUS parameters file

Parameters to ARTIMMUS are provided through XML files. XML can be easily interpreted and written by a variety of scripting languages, and offers a good way of organizing the ~60 parameters. “Treg_2D/Parameters.xml” contains a list of the parameters, and their default values. Values in this file can be changed, and effects observed when using the GUI. The baseline values resulting from calibration of the simulation (see supporting-information.pdf document) are stored in the “Treg_2D/parameters_baseline.xml” file – it is easy to lose or forget the default values when making adjustments to parameters.xml, so this baseline file is useful to retrieving them. **It is highly recommended that you do not change values in this file.** It is the master copy, and can be copied when setting up experiments that are some alteration of the calibrated parameter values. For full explanation of the parameters found in this file please refer to “supporting-information.pdf” or Mark Read’s thesis.

Where to start looking at the code

You may notice that the ARTIMMUS simulation is quite a big piece of software (at least by many academic standards I’ve seen). It may take you some time to get to grips with it and learn where everything is. This section provides a bit of a roadmap, where to start looking, and where the main essentials are located. Note that the “supporting-information.pdf” document provides a good overview of the code in terms of its inheritance structure. In getting a feel for the code I would highly recommend that readers do not dwell on any particular part for long, at least in the first instance, just get a feeling for what a class does (not how it does it) and what methods are in there. Only later, if you need to at all, should you go and figure exactly what a method actually does in detail.

Sim2d.TregSimulation.java is the main driver of the simulation. There are other classes that contain main methods needed to run the simulation (sim2d.Treg2DSim_GUI.java for the GUI, and classes in sim2d.experiment for running without the GUI), but these quickly hand over to TregSimulation. It sets up the simulation to run within MASON (you do not need to look at any MASON code to understand how ARTIMMUS works), it creates the spatial compartments and links them together, it populates the simulation with cells, it loads in parameters from the parameters file, and it sets up the immunization mechanism to induce EAE. This is where I recommend anyone start looking at code, because this is where it all starts.

For cells, the sim2d.cell package contains a few interfaces, but these don’t do very much. It’s in the abstract classes in sim2d.cell.impl where things start to get interested. The APC_Impl.java and TCell_Impl.java

classes are worth looking at. Microglia (called CNS macrophages in the simulation), DCs and all the various T cells inherit from these classes and provide more specific functionality.

Following that, the sim2d.compartment package contains the code representing spatial compartments. A lot of the functions that compartments perform are generic to all of them, and Compartment_Impl2D.java provides a lot of abstract functions that concrete sub classes make use of. The sim2D.cell.molecule package contains interfaces that guard whether or not particular cells interact with one another. Sim2d.molecule contains classes representing cytokines and some cell-surface molecules. These classes generally use the *instance* design pattern (there is only ever one of them).

The remainder of the packages not yet discussed contain house-keeping and utility functions, such as reading and writing from/to the file system, logging data, and drawing graphs.

Execution on a cluster

Execution within Eclipse, and observation of simulation dynamics using the GUI are a good way of getting a feel for the simulation. However, for more serious experimentation a computational cluster of some sort is needed. This section explains how to run the simulation on a cluster administered using the Sun Grid Engine; there are other cluster systems out there, but this is the only system I have used. There may be a lot of parallels.

Firstly, the simulation must be compiled into a jar file that can be run from the command line. There is a script to this do packaged with ARTIMMUS:

```
mark@host:Treg_2D$ ./compileSimulationJar.sh
```

The easiest way to run ARTIMMUS from the command line is to copy the entire Treg_2D directory to wherever the simulation is to be executed, including the MASON.jar and Treg_2D.jar files. Executing the simulation from the command line requires something like:

```
mark@host:~$ cd ~/java_workspace/Treg_2D
mark@host:Treg_2D$ /usr/bin/java -Xms600M -Xmx600M -classpath Treg_2D.jar:MASON.jar
sim2d.experiment.SingleRun -medians false -seed 1 -runs 1 -startNum 1 -rawData true -time 1300 -timeOut
60 -path splenectomy_immunizationStrengths/control_10.0 -param
splenectomy_immunizationStrengths/paramFiles/parameters_control_-_10.0.xml
```

Some things to note:

- “/usr/bin/java” – the location on java on your machine may differ. Its location may be checked with “mark@host:directory$ whereis java”
- “-Xmx600M” and “-Xmx600M” are virtual machine arguments for java. ARTIMMUS can require more Heap space than java is allocated by the operating system by default. These increase that space, and depending on the exact nature of your experiments, you may need more or less than this. Use google if you don’t know what this stuff.
- “-classpath Treg_2D.jar:MASON.jar” – this tells java where to find the ARTIMMUS simulation, and MASON. For simplicity, the current working directory should be “Treg_2D” and should contain these jars.
- “sim2d.experiment.SingleRun” – the class containing a main method that is to be run by java. This is the entry point into the simulation. This is the best to use in most situations, but there is an alternative, “sim2d.experiment.SensitivityAnalysis”, explained below.
- “-medians false” – there it is possible to have a single java call to SingleRun to run several simulations, and compile a simulation output file containing the median data of those runs. In practice I found that this required a lot of java Heap space, especially for large numbers of executions, and prefer to do this using scripts (described below) after all the simulation data is collected.
- “-seed 1” – the seed for the simulation execution, on a cluster this should be set manually, because if several simulations are run in parallel on the cluster and set off at the same time, they may have the same seeds. From a statistical point of view, these would comprise identical samples, best avoided.
- “-runs 1” – as stated above, a single call to the simulation can run several simulations one after the other (best avoided, in my opinion, but the option is here). This specifies how many runs should be performed.

- "-startNum 1" – when compiling data from multiple simulation executions the output files are appended with a unique ID, this is the ID to start from. In this particular ARTIMMUS call, one simulation execution is performed, and the output file will be appended with the number "1". A counter example might involve running the simulation 15 times, and starting the IDs from 37; in this case "-runs 15 –startNum 37" would produce simulation output files appended with "37, 38, 39…52".
- "-rawData true" – this tells ARTIMMUS to write simulation execution data files to the file system. It is possible to write compile one file containing median data and not write all the individual execution data with "-medians true –rawData false".
- "-time 1300" – the time, in simulated hours, after which the simulation terminates.
- "-timeOut 60" – it is possible to select parameters such that the simulation has trouble executing. For instance, if cells are very long lived and proliferate very quickly, but have very little space in which to reside; the simulation will spend the bulk of its time trying to move cells around in overcrowded spatial compartments. This timeout utility terminates an ARTIMMUS execution is it has been running for more than the specified number of minutes. Its not a perfect system, the time check is not in a separate thread from the simulation, meaning that if the simulation is never able to get to the time-check, then it will not terminate. You should still keep an eye on things when running on a cluster.
- "-path splenectomy_immunizationStrengths/control_10.0" – this is where data is to be written. If running the SingleRun main method, then "single_run" will be prefixed to this path (it is unwise to delete the "Treg_2D/single_run" directory for this reason). Setting the path like this makes for better organization of simulation experimental data. The path here is specific to one of my experiments, you should supply your own.
- "-param splenectomy_immunizationStrengths/paramFiles/parameters_control_-_10.0.xml" – this is where the parameters file that the simulation should use is located. Again, this is only an example, you should supply your own.

Simulation output file

An execution of ARTIMMUS using the SingleRun.java class will produce a file containing time-series data of the simulation's execution. These files appear in the path described by "-path" when running from the command line. They are named "simOutputData_XXX.txt" with "XXX" replaced by the run number (see ""-startNum" in the section above). Be aware that each of these files can be quite big, especially when running the simulation for a long period of time. Various aspects of simulation behavior are recorded in the columns, and rows represent each hour of simulated time. The columns record the following (format is column number, short name, explanation (where appropriate)):

1. time_hours, current simulation time, in hours.
2. total_CD4Th, all CD4Th cells currently in the system, regardless of state.

The number of T cells in particular states are recorded. For states, see the domain model in the accompanying "supporting-information.pdf" document. This is done separately for MBP-specific CD4Th cells, CD4Treg, and CD8Tregs.
3. total_CD4ThNaive
4. total_CD4ThPartial
5. total_CD4ThProliferating
6. total_CD4Th1
7. total_CD4Th2
8. total_CD4ThApoptotic
9. total_CD4Treg
10. total_CD4TregNaive
11. total_CD4TregPartial
12. total_CD4TregProliferating
13. total_CD4TregActivated
14. total_CD4TregApoptotic
15. total_CD8Treg
16. total_CD8TregNaive
17. total_CD8TregPartial
18. total_CD8TregProliferating
19. total_CD8TregActivated
20. total_CD8TregApoptotic

Number of APCs and DCs in each compartment, in each state.

21. cns_APC
22. cns_APCImmature
23. cns_APCTolerogenic
24. cns_APCImmunogenic
25. cns_APCApoptotic
26. CLN_DC
27. CLN_DCImmature
28. CLN_DCTolerogenic
29. CLN_DCImmunogenic
30. CLN_DCApoptotic
31. SLO_DC
32. SLO_DCImmature
33. SLO_DCTolerogenic
34. SLO_DCImmunogenic
35. SLO_DCApoptotic

number of CD4Th1 and CD4Th2 in the CNS compartment.
36. CNS_CD4Th1
37. CNS_CD4Th2

The average specificities (binding strength of TCR with cognate MHC:peptide) of each T cell population.
38. CD4Th_Specificity
39. CD4Th1_Specificity
40. CD4Th2_Specificity

41. cumulative_Th1Killed; cumulative count of CD4Th1 apoptosed by CD8Treg to date.
42. CLN_DC_PolarizationType1; number of type-1 polarizing DCs in the CLN compartment
43. CLN_DC_PolarizationType2 ; similar to above, but type 2.
44. cumulative_CNS_DC_PolarizationType1
45. cumulative_CNS_DC_PolarizationType2

Total number of DCs in the spleen, and a break down of their states
46. Spleen_DC
47. Spleen_DCImmature
48. Spleen_DCTolerogenic
49. Spleen_DCImmunogenic
50. Spleen_DCApoptotic

Number of T cells of particular types and states in the spleen. Similar for the CLN
51. Spleen_CD4TregTotal
52. Spleen_CD4TregEffector
53. Spleen_CD8TregTotal
54. Spleen_CD8TregEffector
55. Spleen_CD4Th1
56. Spleen_CD4Th2
57. Spleen_CD4TregProlif
58. Spleen_CD8TregProlif
59. Spleen_Th1Prolif
60. Spleen_Th2Prolif
61. CLN_CD4TregProlif
62. CLN_CD8TregProlif
63. CLN_Th1Prolif
64. CLN_Th2Prolif
65. CLN_CD4TregEffector
66. CLN_CD8TregEffector
67. CLN_Th1Effector
68. CLN_Th2Effector

Cumulative counts of CD4Th1 cells apoptosed by CD8Tregs in each compartment
69. cumulativeCD4Th1KilledCirculatory
70. cumulativeCD4Th1KilledCLN
71. cumulativeCD4Th1KilledCNS
72. cumulativeCD4Th1KilledSLO

73. cumulativeCD4Th1KilledSpleen

Cumulative counts of the number of T cells (of particular types) primed in various compartments over time.
74. CD4ThPrimedCirculatory
75. CD4ThPrimedCLN
76. CD4ThPrimedCNS
77. CD4ThPrimedSLO
78. CD4ThPrimedSpleen
79. CD4TregPrimedCirculatory
80. CD4TregPrimedCLN
81. CD4TregPrimedCNS
82. CD4TregPrimedSLO
83. CD4TregPrimedSpleen
84. CD8TregPrimedCirculatory
85. CD8TregPrimedCLN
86. CD8TregPrimedCNS
87. CD8TregPrimedSLO
88.CD8TregPrimedSpleen
89. NeuronsKilledCumulative; a cumulative count of how many neurons have been apoptosed to date.

Setting up experiments on a cluster, and executing them (on Sun Grid Engine)

This section explains how I organize my file system and data when performing experiments on the cluster. This is not the only way to do things, however the analysis scripts supplied with ARITMMUS were created with this sort of organization in mind – they can be changed, but to get going quickly I suggest you try this first.

The standard, and recommended, way of running ARTIMMUS on a cluster is to call it using java from the command line (as outlined above) using the SingleRun class's main method. Data will be output into the "Treg_2D/single_run" directory, in a sub-directory of your specification. Experiments normally comprise changing simulation parameters, which is easy to handle. It's also possible that an experiment might comprise some mechanistic (code/cellular behaviour rather than parameters) change in the simulation: eg, "what happens if we try a different model for cytokine diffusion". This can involve changing code rather than parameters. One might be tempted to put some code on the cluster, run the experiment, and then switch the code and run the other experiment. Given my personal experience I would recommend against this, its too easy to forget which version of the code you are currently running, and if you're using the cluster to run multiple experiments simultaneously then it can become really tricky. You can put an if-statement in the simulation, dependent on a Boolean parameter, and which selects between the two pieces of code to run. This is much easier to manage. Its not hard to add parameters to the simulation's XML files, and the code to pull parameters out of these XML files are generally contained at the bottom of each class (classes pertaining to particular cells or functions are responsible for managing and retrieving their own parameters).

Experiments are ideally contained within their own directory. To perform a splenectomy experiment I set up a directory structure that looks like this:

1. "Treg_2D/single_run/splenectomy" – the top level directory for the experiment
2. "Treg_2D/single_run/splenectomy/paramFiles" – I put all the parameter files that comprise the experiment in one place, here. Generally I create a new parameter file for each parameter setting. The simulation will automatically copy this into the directory where experimental data is being written to. Its very helpful to keep the parameter files with the data that was generated from them, you can always figure out what the data represents. This is science, if you're ever in doubt about your data you have to do the experiment again; there's a good argument to keep a copy of the code with the data too, though I never went this far. It may sound tedious to create this many parameter files, but its not hard to create a script to do it for you.
3. "Treg_2D/single_run/splenectomy/SGE" – I put all files pertaining to running experiments on the grid in the same directory, SGE stands for sun grid engine.
4. "Treg_2D/single_run/splenectomy/splenectomy_-_0" – this would be the control group, no splenectomy. 0 is synonymous with "false".
5. "Treg_2D/single_run/splenectomy/splenectomy_-_1" – this is the experiment/treatment group, splenectomy performed here. Note that its possible to name these directories something more human-readable like "baseline" "treatment" or "control" and "splenectomy". The reason I don't do this is because there are a lot of automated analysis scripts that like to use numbers that can be easily ordered. Categorical data is more difficult to deal with. There's no reason these scripts can't be adapted to deal with categorical data more easily… but I suggest you look at how complex these

scripts already are first, and then decide if it's a worthwhile investment of your time for the sake of some naming convenience. Of course, you can always rename files before the analysis stage, I have done this often. I should mention that "_-_*value*" features a lot, I use "_-_" to separate terms in the directory names that scripts need to read to contextualize what's going on.

The following is an example of how a splenectomy experiment might be run on a cluster. In this example "splenectomy_-_0" is replaced with "baseline" and "splenectomy_-_1" is replaced with "splemectomy". These names would be changed to the format outlined above before running analyses on the data. Job submissions to the sun grid engine is through a script of the following format.

mark@host:Treg_2D/single_run/splenectomy/SGE/$ cat baseline.sge

```
#!/bin/sh
#$-S /bin/bash
#$ -o java_workspace/Treg_2D/single_run/splenectomy/SGE/baseline.out
#$ -e java_workspace/Treg_2D/single_run/splenectomy/SGE/baseline.err
#$ -t 1-500
date
RUNNUM=$(($SGE_TASK_ID-1))
SEEDFILE=~/java_workspace/Treg_2D/single_run/splenectomy/SGE/baseline.seeds
SEED=$(cat $SEEDFILE | head -n $SGE_TASK_ID | tail -n 1)
cd ~/java_workspace/Treg_2D
if [ ! -f Treg_2D/single_run/splenectomy/baseline/simOutputData_$RUNNUM.txt ]
then
/usr/bin/java -Xms600M -Xmx600M -classpath Treg_2D.jar:MASON.jar sim2d.experiment.SingleRun -
medians false -seed $SEED -runs 1 -startNum $RUNNUM -rawData true -time 5280 -timeOut 60 -path
splenectomy/baseline -param splenectomy/paramFiles/parameters_baseline.xml
fi
date
```

Some explanation. The first two lines ensure that the correct shell environment is being used (I use bash). The "#$ -o" command dictates where to write standard output from a job that the cluster is running. Likewise, the "-e" line species the path to which standard error is written. This job comprises 500 simulation executions, and rather than create 500 independent jobs, this is an *array job*, meaning that it will itself administer 500 simulation executions as one job. That's what the "#$ -t 1-500" does. Look up sun grid engine array jobs on google for more information. "date" simply prints the date, it does this at the end of the job too (in case you wonder how long it all took). Each of the 500 array jobs has a job ID, a unique identifier. I use this to label the simulation output files, which is what "RUNNUM=$(($SGE_TASK_ID-1))" does. I also start my job numberings from zero. Recall that simulation seeds should be set manually. I create a file containing a unique integer on every line, and use the run number to index into this file and pull out the seed on a particular line. That's what the lines starting SEEDFILE and SEED do. Next the grid engine is told to set the current working directory to the Treg_2D directory. The "if [! -f Treg_2D/single_run/splenectomy/baseline/simOutputData_$RUNNUM.txt]" line is very useful; clusters do a lot of heavy computational work very quickly and are often used by many people simultaneously, they occasionally crash. When they do, its very cumbersome to identify which simulation executions completed successfully and which did not, at least by eye. Far easier to use this if statement and just set the whole experiment off again. Only those experiments for which an output file does not already exist are executed. The long "/usr/bin/java…" line was covered above, but note here the use of $RUNNUM and $SEED to provide run numbers and seeds. What the grid engine does is essentially call this script 500 times (because its an array job) using the job ID number to provide unique run numbers and seeds for the simulation.

I would not recommend writing a lot of these files by hand, I use a ruby script to generate them automatically. I did specify all the parameter files by hand, but its not hard to write a script to do this also.

Another quick tip, in my case I could not run data analysis on the same cluster that runs the simulations. Rsync is a very good tool for moving data between machines, even across university networks.

SensitivityAnalysis class

SingleRun is not the only way to run experimentation on the cluster, there is also the SensitivityAnalysis class (found in sim2d.experiment). Nonetheless, I consider the latter somewhat deprecated and would recommend using SingleRun for everything, and just being clever in how you set up your experiments.

SensitivityAnalysis provided a way to perform a robustness analysis (see Read's MCMDS paper, and Alden's Spartan paper, also shipped with this software) relatively quickly. I won't provide detailed instructions on this here, I suggest interested readers look at the code.

Data analysis

This section covers the scripts provided with ARTIMMUS for analyzing data, typically used for large-scale experiments or compiling graphs from multiple simulation executions. This assumes that you have set up your experiments as described in the above section on using the cluster, and that matlab (with statistics toolbox) and ruby are installed on the machine you are using.

There is an important caveat to inform the reader of. In my experience, different users (even myself on different days) *always* have subtle differences in how they set up and organize their experiments. As such, some of the scripts below will probably fail rather than complete correctly on the first try. I wrote the scripts, and even I have to go into them and figure out what went wrong from time to time. I have tired to comment all scripts as I wrote them, and this should help. Its usually something to do with how data is organized, or the naming of the directories representing parameter adjustments. If it happens to you, please don't worry, it is normal and has happened to *everyone* I know who has used the simulation to do large scale experimentation (including me). I'm afraid this is the nuts and bolts reality of running large-scale computational biology simulations for research purposes, it would be wonderful to have an integrated framework and highly robust scripts, but the reality is that we are all engaged in research and not software suite development (the todo list is long, time and money are short).

This section is written with the above splenectomy as a running example. In the above section it is noted that although human-readable names for directories relating to parameter changes (like "control" or "baseline" and "splenectomy") are useful for humans, they are not so useful for the scripts described here as they are categorical data. I have renamed the directories containing splenectomy data as follows:

```
mark@host:splenectomy$ mv baseline splenectomy_-_0
mark@host:splenectomy$ mv splenectomy splenectomy_-_1
```

Firstly, check that all files are present and accounted for. Copy (or create a symbolic link if you find that better) this file:

```
mark@host:Treg_2D/single_run/splenectomy$ cp ../../data_analysis/checkConsistencyofSingleRunDataFilesTopLevelDriver.rb .
```

You run it from the bash terminal as follows:

```
mark@host:…$ ruby checkConsistencyofSingleRunDataFilesTopLevelDriver.rb -numRuns 500 -timeSamples 1300
```

"-numRuns XXX" specifies the number of simulation executions that *should* be in each directory containing simulation data. "-timeSamples" specifies how many rows there should be in each table, ie, how long the simulation as run for. This checks that the file was not partially written. This script identifies all sub-directories containing simulation output data, go into each one in turn, and runs a matlab script that checks they are all of the correct format and that there are the correct number of them. When it's finished, you should find a text file called consistencyOfDataResult. Hopefully it contains something like "true", if it has anything labelled "false", then there were problems. There is a similar named file in each of the sub-directories, you can find more details in those.

The next thing to do is to compile from all these simulation executions the median data.

```
mark@host:…$ cd ~/java_workspace/Treg_2D/single_run/splenectomy
mark@host:splenectomy$ cp ../../data_analysis/compileMedianDataRunFromSingleRunsTopLevelDriver.rb .
mark@host:splenectomy$ ruby compileMedianDataRunFromSingleRunsTopLevelDriver.rb
```

The above 3 lines ensure you are in the correct directory, copies the analysis script locally, and executes it. This script works much in the same way as the previous script; it identifies all sub-directories that contain simulation data, and in each sub-directory it executes a matlab script to open and read all simulation data files, and write a file (of the same format as the simulation output data file) containing median data. This file is named "multipleDataOutput.txt".

Next, creating graphs of these data. Again you will need to copy across the script, or at least create a symbolic link.

```
mark@host:splenectomy$ cp ../../data_analysis/robustnessAnalysis/compileSimOutputGraphs.rb .
mark@host:splenectomy$ ruby compileSimOutputGraphs.rb –end 50 –error 10
```

Again this script identifies all sub directories containing simulation data and calls a matlab script to compile draw a graph of the simulation data. Actually it does this twice, the first time to find out what the upper limits on graphs for each parameter value, and the second time to re-draw and save the graphs using the same upper limits. This makes comparing results much easier. "-end 50" is used to specify the upper time range (in days) over which to draw graphs, it always starts from day zero. For convoluted reasons not worth going into here[1], it is necessary to get accurate EAE severity scores data (script for that explained below) to have about 4 days worth (~100 hours) of extra simulation time series data than is actually wanted. 50 days is 1200 hours, but for this reason I execute ARTIMMUS for 1300 hours of simulated time. "-error 10" places error bars on some of the graphs, the "10" bit is the period of error bar placement in days. This means error bars are slightly staggered horizontally, and makes them easier to read as they would otherwise overlap. When this script completes there should be a new directory created:
"Treg_2D/single_run/splenectomy/simulationOutputPngs"
that contains all the graphs generated. There are a lot of them. Another piece of advice based on experience: this graphing script runs matlab "headless" (no GUI, just in the terminal). For reasons I have never been able to figure out, if you try doing this over an ssh connection (or, presumably, on a machine with no X display) the resolution of the graphs that matlab draws are terrible. If you do it locally, matlab (even though it runs without a GUI) draws and displays the graphs, and they are of a far better quality.

It is nearly always useful to do some stats on the data you have collected from an experiment, and my first step has always been to perform a robustness analysis on it (see Read's thesis, Read's MCMDS paper, and the Spartan paper. These scripts came before spartan, but I will provide the instructions here anyway). Before you do this, there is presumable a control in your experiment. If there isn't, there probably should be, identify the most appropriate. For example, in the case of the splenectomy it would be "0", since this means "false", no splenectomy performed. You need to create a file containing the baseline/control parameter value:

```
mark@host:splenectomy$ echo 0 > defaultParameterValue.
```

Note that the value entered into the file has to *exactly* match the text at the end of the directory containing that control data. If it was "splenectomy_-_0.000" then I would need to enter "0.000" into defaultParameterValue. Do not put any other text in this file, just that number, character for character. To perform the robustness analysis,

```
$mark@host:splenectomy$ cp ../../data_analysis/robustnessAnalysis/generateResponses.rb .
$mark@host:splenectomy$ ruby generateResponses -end 50
```

In statistical terms, a "response" is something that is measured in a system, which may explain the name. "-end" you should be familiar with by now. This script does a lot, and may compile median data and draw the graphs again (you can delete or comment out that line in the file if you wish… but remember this may cause problems later when you come to other experiments and wonder why the script isn't working properly! Remember that caveat at the start…). When this has completed there should be a new directory "Treg_2D/single_run/splenectomy/robustness_sensitivity_analysis" which contains graphs showing how various aspects of simulation behaviour have changed under different parameter values. There will also be other statistical information in there. There is more analysis to come however…

Next, grade the simulation executions in terms of disease severity. All being well, one script should handle all of it (though it calls a lot of other scripts along the way…)

```
mark@host:splenectomy$ cp ../../data_analysis/EAESeverity/performEAESeverityAnalysis.rb .
mark@host:splenectomy$ ruby performEAESeverityAnalysis.rb –end 50
```

This will assign time-series disease severity scores to each and every simulation execution in every sub-directory containing simulation data. It draws graphs of the average disease scores, of the proportions of simulations that experience particular disease scores over time, proportions of simulations that experience each maximum disease score at any point in time, the durations of remissions and relapses into disease.

---

[1] This is because there is some smoothing of neuronal apoptosis time series data in order to obtain the EAE disease severity score data. The smoothing algorithm used is a sliding window filter, however when the filter nears the end of the time-series data the window gets smaller and this interferes with the smoothed data, hence the extra 100 hours. This is not a problem at the zero-days end of the data, because neuronal apoptosis does not occur at a sufficient rate to cause clinical disease in the first 4 days anyway.

Some of this data is written to files also. I encourage readers to have a hunt around and see what has been produced. A new directory will be created that contains these graphs,
“Treg_2D/single_run/splenectomy/EAESeverityAnalysis”
and additional information will be added into the “robustness_sensitivity_analysis” directory. It may be worth re-running the robustnessAnalysis.m script (previously performed as part of generateResponses):

```
mark@host:splenectomy$ cd robustness_sensitivity_analysis
mark@host:robustness_sensitivity_analysis$ matlab –nosplash –nodesktop
> robustnessAnalysis('')
```

Lastly, should you wish to draw graphs of individual simulation executions (rather than the median data series), scripts to do this are provided in “Treg_2D/data_analysis/robustnessAnalysis” and “Treg2D/data_analysis/EAESeverityAnalysis”.

This should complete the analysis. There is a lot here, and these scripts can seem daunting to anyone who did not write them. I encourage patience, and re-iterate my earlier statement: don’t expect these to work first time (the same goes for the cluster), this is the nuts and bolts of doing this sort of mass-experimentation; its hard, things go wrong. I’m sure its not dissimilar to normal biological experimentation in that regard. Take your time, because making mistakes can be very costly.

If you have any questions, please feel free to contact me. The best way to do this is probably through the contact details on my website (the most up to date):
http://markread.info

I hope these instructions will have been useful to you. Good luck!