Introduction

What/who this book is for

What this book explains/provides

Where to find more SLiM resources

Part I: Installing SLiM for Windows systems

Installing SLiM is trivial on MacOS, however slightly more difficult on Linux based systems

Windows isn’t supported by default, but Windows 10 has a Linux subsystem feature that allows us to run a reasonably quick virtual machine with a much lower footprint than a VMware Mac system. Also considerably more legal.

The SLiM manual provides instructions for how to install SLiM for MacOS and several Linux distros, we will focus mainly on setting it up the Windows Subsystem for Linux (WSL)

Open Powershell as Admin, enable WSL then install

Update to WSL 2 (?)

Installing dependencies/GUI desktop for Ubuntu

Building SLiM for Ubuntu – SLiM Manual instructions

Go through the SLiM Online Workshops! Very helpful and almost necessary

Link directly to glossary, a table to navigate between SLiM manual

Troubleshooting: SLiMgui build problems – black screen

Using cmake, or install script

Try qmake instead

QOpenGLShaderProgram: could not create shader program

Vertex shader for simpleShaderProg (MainVertexShader & PositionOnlyVertexShader) failed to compile

direct rendering: No (LIBGL\_ALWAYS\_INDIRECT set)

export LIBGL\_ALWAYS\_INDIRECT=0

gnome-session

gedit ~/.bashrc change export LIBGL\_ALWAYS\_INDIRECT=1 to export LIBGL\_ALWAYS\_INDIRECT=0

Part II: Polygenic adaptation in SLiM

Brief description of how SLiM models nucleotides etc. – exploit this to model at a QTL level, making sure to adjust recombination rates etc. to a per locus rather than per base pair rate

More detail in SLiM manual

Brief description of quantitative models for polygenic traits – drift/selection/mutation balance etc.

Template for running this: configurable population size, genome length, number of chromosomes, mutation rate, number of QTLs, effect sizes, recombination rates, deleterious mutation rate, heritability, and selection strength – also seed and model index (more on that later)

Description of each parameter

Output: Template includes outputs for allelic effects, frequencies and distributions if as well as phenotypic means, additive variance, heritability, heterozygosity, dN/dS, MK tests, pi, Tajima’s D and theta etc. Stored as single files, with these statistics being calculated at configurable intervals, and each replication also being there and identifiable by the seed number and/or model index.

Part III: Running SLiM in parallel on your home computer

SLiM runs on one core at a time, meaning that you can have multiple instances of SLiM running (for replication, or for different treatments) at the same time. Most computers now have 4 cores, so we’ll assume that and use some of R’s functionality to run SLiM from the command line in parallel.

Rationalise with how long it would take on one thread

The R packages doParallel, future, and foreach can be used in tandem to setup a multicore setup in R

Template R script for running R in parallel

In this script, we are calling a separate SLiM process to start on all available cores via R. We use foreach loops to loop through a list of variables that we want to give to SLiM – these include things like the number of QTLs, effect sizes, etc. Importantly, we also have seed and model index values. The model index is a quick identifier for which combination of variables you are using – it isn’t really necessary in simple cases with only a few variables, but if you are doing more complex stuff with lots of parameters and Latin hypercube sampling (more on that later), it will be very useful

Boxes that go into more detail – so a logical flow of ideas for evreryone and then more detail in boxes for people looking for more detail

Seeds are used to get pseudorandom numbers from computers. Computers are really bad at generating random numbers because they operate on binary conditions – true or false (1 or 0) is at the root of every decision a computer makes. In fact, true randomness can only be achieved through quantum phenomena like the decay of radioactive elements, or the way a laser reflects or is absorbed by a surface. Cool! But we don’t really need ‘true’ randomness very often, we just need to be sure that a ’randomly’ chosen set of numbers is not influenced by human interactions and therefore biased. Computers are good at dealing with this. Most computers have a built in random number generator which uses the computer’s time and current state as an input ‘seed’, which then feeds an algorithm that produces an output pseudorandom number. R has inbuilt functionality just like this, which enables as to produce a list of pseudorandom numbers, which we can then input into SLiM to seed it’s random number generator. SLiM uses RNG to determine where and when mutations should occur, what parents should produce offspring (to an extent, depending on fitness also) etc. By knowing which seed we are feeding SLiM, we can be sure that our experiments are reproducible, and by knowing that the numbers themselves are pseudorandom and unbiased (we can check by making sure the set of numbers follows a uniform distribution) we can be sure our results are not biased by the input we gave.

It’s important to note that SLiM uses two random number generators – a 64bit slower one and a 32 bit faster one – this limits the seed that we give to SLiM to a 32 bit integer, as anything more is at a bit level truncated. This still leaves us with over 4 billion possible numbers (and with every parameter change, the seeds will produce different results), so it isn’t too much of an issue: but it is important to remember that you should only sample your random numbers in a range of 0 to 4294967295 or you could see some weird behaviour.

Part IV: Running SLiM on a HPC

PBS -> SLURM

Now that we’ve got SLiM running on multiple cores, we can use a similar idea to run it on a remote computer with considerably more cores. You can connect to UQ’s HPC via ssh.

First, you’ll need to sign up to the HPC you want to use – information is here: <https://rcc.uq.edu.au/high-performance-computing>

I recommend applying for either Awoonga or Tinaroo, as these provide the most cores and adequate RAM. Tinaroo allows for multiple nodes, which is invaluable for larger projects where you have a lot of simulations to run (i.e. runs >> the number of cores on a single HPC node).

There’s a training session at the end of each month which I would recommend attending to familiarise yourself with the system.

On your WSL system, you can install filezilla to transfer files from your local system to the HPC, such as SLiM and R scripts. The same can be done vice versa to copy your output to your local drive, or alternatively, to an RDM drive.

Job queue system, PBS scripts. <https://amandamiotto.github.io/INTRO_HPC/11-cluster/>

If you want to run a simple SLiM job on a single node, there a several ways to go about it. You’ll have to write a PBS script to give to the HPC to load your job, but you can choose to have this script run SLiM through multiple bash commands, or through R in a similar way to what we did above. Bash has less overhead (and so should be faster), but having access to R while running the scripts is also convenient if you want to manipulate the data via some R package afterwards.

Bash template for single node job

R template for single node job + R script call

It is important to mention that we are combining our output into a single file in the temp directory. This reduces strain on the HPC’s network, and is much faster than having a billion files appear on the HPC for it to contend with. Remember it is almost always better to concatenate/append your output than to create a new file!

Part V: Latin Hypercube and multi-node SLiM HPC jobs

So now let’s explore how to get a multi-node job running on Tinaroo. There are two main ways of running SLiM on Tinaroo, Embedded Nimrod or job arrays.

Job arrays use a PBS command to reserve a certain number of nodes and have each of them run a subset of the total number of replicates/treatments. It’s really no different from having a separate script for each subset except it’s easier to do. Here’s a template:

Embedded Nimrod is a system that is much more efficient. It allows for each individual SLiMulation to be run as each core becomes available on each node, leading to shorter waiting times between a run ending and a new one starting, and a more streamlined job run. The downside is you have to wait for all the nodes you request to be available before it will start running, whereas job arrays will start a part of your job on each node as that node becomes available. Here’s a template:

Embedded Nimrod does have some bugs occasionally. I found that some runs simply didn’t go through the first time, so I had to identify them via an R script (Appendix) and rerun those particular treatments. However, there are easier ways to get around this by including a check in your Nimrod script itself. That’s shown in the above template …

Ordinarily, you shouldn’t need to use these for small experiments. 24 cores is a lot of simultaneous simulations, but if you are looking at a heap of replicates it might be a good timesaver. If you’re looking at a lot of variables, you’ll need them as well as Latin hypercube sampling to make it feasible

You might have noticed in the simple home computer multi-core template that as you add more variables, the number of necessary trials tends to increase exponentially. This is a problem for effective sampling. It becomes impossible to get a good read on what levels of various parameters are doing anything, and which interactions matter. Latin hypercube sampling somewhat alleviates this issue by sampling a parameter space: all of the variables are considered at once. A single Latin hypercube sample contains a value for each of the parameters considered. Together, all of the LHC samples describe each of the parameters independently, minimising correlations between parameters, and maximising how much of the entire range of parameter values you are trying to explain. Of course, there are limits, and for each parameter you add, you need more and more samples to adequately describe them. The DoE.wrapper package in R can be used to generate LHC samples via a number of different methods – maximin seems to work best, although you can check your case by using the cor() function in baseR to determine correlation between variables in your final LHC.

When using a LHC approach, running SLiM can be done slightly differently. Since each treatment is effectively a combination of values for each of your parameters, you can treat your hypercube sample as a ‘model index’, which allows you to easily sort via the parameter combination, and feed individual parameter values to SLiM via the LHC itself. Here’s an example of that in action:

Part VI: Statistics and data wrangling for polygenic SLiM data

rPCA

Eigentensor analysis

Random skewers

Allele frequency/effect size data

RegEx for dealing with the huge sizes of files

Part VII: Other SLiM models and tools

Moving WSL to another drive (in case your C:/ is small/full)

Nucleotide based models (?)

Migration models

HPC file storage