Received: 18 March 2022 | Revised: 20 November 2023 | Accepted: 30 November 2023

DOI: 10.1111/1755-0998.13916

MOLECULAR ECOLOGY RESOURCES WILEY

RESOURCE ARTICLE

slimr: An R package for tailor-made integrations of data in population genomic simulations over space and time

Russell Dinnage[1,2] | Stephen D. Sarre[2] | Richard P. Duncan[2] | Christopher R. Dickman[3] | Scott V. Edwards[4,5] | Aaron C. Greenville[3] | Glenda M. Wardle[3] | Bernd Gruber[2]

[1]Institute of Environment, Department of Biological Sciences, Florida International University, Miami, Florida, USA

[2]Centre for Conservation Ecology and Genomics, Institute for Applied Ecology, University of Canberra, Canberra, Australian Capital Territory, Australia

[3]Desert Ecology Research Group, School of Life and Environmental Sciences, University of Sydney, Camperdown, New South Wales, Australia

[4]Department of Organismic and Evolutionary Biology, Harvard University, Cambridge, Massachusetts, USA

[5]Museum of Comparative Zoology, Harvard University, Cambridge, Massachusetts, USA

Correspondence
Russell Dinnage, Institute of Environment, Department of Biological Sciences, Florida International University, Miami, FL, USA.
Email: rdinnage@fiu.edu

Funding information
Australian Research Council, Grant/Award Number: DP180103844

Handling Editor: Frederic Austerlitz

Abstract

Software for realistically simulating complex population genomic processes is revolutionizing our understanding of evolutionary processes, and providing novel opportunities for integrating empirical data with simulations. However, the integration between standalone simulation software and R is currently not well developed. Here, we present slimr, an R package designed to create a seamless link between standalone software SLiM >3.0, one of the most powerful population genomic simulation frameworks, and the R development environment, with its powerful data manipulation and analysis tools. We show how slimr facilitates smooth integration between genetic data, ecological data and simulation in a single environment. The package enables pipelines that begin with data reading, cleaning and manipulation, proceed to constructing empirically based parameters and initial conditions for simulations, then to running numerical simulations and finally to retrieving simulation results in a format suitable for comparisons with empirical data – aided by advanced analysis and visualization tools provided by R. We demonstrate the use of slimr with an example from our own work on the landscape population genomics of desert mammals, highlighting the advantage of having a single integrated tool for both data analysis and simulation. slimr makes the powerful simulation ability of SLiM directly accessible to R users, allowing integrated simulation projects that incorporate empirical data without the need to switch between software environments. This should provide more opportunities for evolutionary biologists and ecologists to use realistic simulations to better understand the interplay between ecological and evolutionary processes.

KEYWORDS
application, ecology, evolution, evolutionary ecology, landscape genomics, population genomics, simulation, software

1 | INTRODUCTION

Mathematical modelling and simulation, particularly those that simultaneously model genomic, populations, and environmental processes, are cornerstones of population genetic practice. They provide the link between evolutionary and ecological processes through temporal and spatial patterns observed in population genomic data (Haller & Messer, 2019) and enable testable predictions and insights into the interaction between organisms and their environment (Hoban, 2014). In particular, simulation modelling provides the opportunity to forecast the response of species and communities to environmental change (Manel & Holderegger, 2013), infer

dispersal ecology (Storfer et al., 2018), quantify past population dynamics (Patton et al., 2019) and detect outlier loci as evidence of selection or other genetic phenomena (Hoban, 2014). As such, they have application to an increasing variety of ecological and evolutionary problems that will continue to grow as genomic datasets increase in number, quantity, coverage and sophistication. Increasingly too, these genomic datasets will be linked directly to reference genomes enabling the estimation of region-specific chromosomal recombination and mutation rates adding a further layer to the analysis of selection in natural populations (Hoban, 2014).

At a fundamental level, empirical datasets demand analytical toolkits that can accommodate the potentially high complexity of the processes involved in their creation. Recent developments in sophisticated simulation software have the potential to provide mechanistic insight into increasingly complex evolutionary scenarios (Carvajal-Rodríguez, 2010; Haller & Messer, 2019; Hoban, 2014; Kelleher et al., 2016; Messer, 2013; Strand, 2002; Yuan et al., 2012). However, utilizing flexible simulations requires exploration of a large parameter space, which often generates large amounts of data that need sophisticated computational tools to unpack, interrogate and synthesize. Likewise, using simulations to model empirical data is an emerging field ('simulation-based inference') because it allows researchers to fit data to complex mechanistic models where it is difficult or impossible to derive an analytical likelihood for the outcome data under the model (Beaumont et al., 2002; Brehmer et al., 2020; Cranmer et al., 2020; Marjoram et al., 2003; Sisson, 2018; Torada et al., 2019; Wang et al., 2021). Johri et al. (2022) recently argued that to reasonably derive insight into the complex processes involved in population genomics, a 'baseline' model, containing all the minimal evolutionary processes we expect to influence genomes, should be constructed in order to account for 'background' or non-target processes, and reduce the chances of discovering spurious results. Notably, this baseline model, involving genetic drift, demographic history, geographic structure, mutation, recombination, gene conversion and background selection, is already far more complex than most non-simulation-based analytical approaches can handle. One way to deal with this inherent complexity is to increase the routine usage of complex simulations in empirical biology, both for conceptual development of empirical questions and for making empirical inferences. Towards this goal: to increase accessibility and facilitate more rapid and seamless interrogation and synthesis between empirical data and population genomics simulation, we present slimr (https://rdinnager.github.io/slimr/).

slimr is an R package designed to link the very large and widely used ecosystem of analysis and visualization tools in the R statistical language to the SLiM scripting language (Haller & Messer, 2019), a popular, powerful and flexible population genomics simulation tool. The package creates a smooth fusion between the computational power and flexible model specification of SLiM with the advanced statistical analysis, visualization and metaprogramming tools of R. R is not the only powerful framework for data analysis – Python and Julia are also excellent alternatives – but it is a very popular framework amongst biologists, and is particularly strong for metaprogramming – meaning programming code that manipulates programming code – which powers slimr's features and is due to R's non-standard evaluation (NSE) features (Wickham, 2014).

The primary aim of slimr is to facilitate the smooth integration of data processing and analysis with the simulation abilities of SLiM. This allows powerful SLiM-based simulations to be included more easily in complex workflows that might include data as inputs to a simulation, or data generated as outputs of a simulation, facilitating downstream statistical analysis or visualization of simulation results. As a primary example, simulation-based inference methods such as approximate Bayesian computational require running large numbers of simulations with different parameter values, and the comparison of simulation outputs to empirical data. Both of these things are made easier by slimr, which allows users to directly take advantage of simulation-based inference implemented in R itself without having to create potentially complex customized interfaces between R and SLiM.

Importantly, slimr is designed to preserve access to the full and complete set of simulation features available in SLiM and to easily accommodate future features of SLiM software which is under very active development. Therefore, slimr does not just wrap around certain features of SLiM, using SLiM as an engine to power a particular class of simulations and make them more easily accessible (that is, not requiring learning SLiM and its complexity). slendr is a recent R package that does take this alternative strategy, allowing the simple specification and running of certain spatial simulations that can make use of some of R's spatial data structures, and greatly increasing the accessibility of this class of simulations (Petr et al., 2022). The trade-off is that slendr has limited flexibility and only implements a subset of SLiM's features. slimr, in contrast, supports all current and future features of SLiM. In fact, because slimr is a general-purpose interface to SLiM, writing packages like slendr, that implement user-friendly interfaces to particular classes of simulations, should be much easier, as slimr can act as an intermediary between R and SLiM, eliminating the need to write large amounts of boilerplate code translating from R to SLiM and back again.

In the following section, we describe the package and its main features from a technical perspective. Lastly, we provide some examples of how to use the package in a simulation workflow from within R, and how this can enable biologists to incorporate advanced simulation into their projects, along with empirical data, for the purposes of experimental design planning, theoretical insight, or to directly model complex mechanistic processes through simulation-based inference.

2 | PACKAGE DESCRIPTION

slimr is an R package that interfaces with SLiM >3.0 software for forward population genetics simulations (see Haller & Messer, 2019; Messer, 2013 for full details on SLiM, as well as the website at https://messerlab.org/slim/). slimr has most recently been updated to work with SLiM version 4.0 and greater, including support for its powerful new 'multispecies' feature (Haller & Messer, 2023), but should also be compatible with any version greater than 3.0 (previous versions

may work but have not been tested). It has already demonstrated its utility in a large-scale simulation projects (Hill et al., 2023).

slimr implements a domain-specific language (DSL) that mimics the syntax of SLiM, allowing you to write and run SLiM scripts and capture resulting simulation output, all within the R environment. Much of the syntax is identical to SLiM, but slimr offers additional R functions that allow users to manipulate SLiM scripts ("slimr verbs") by inserting them directly into any SLiM code block. This enables R users to create SLiM scripts that explore large numbers of different parameters and also automatically produce output from SLiM for powerful downstream analysis within R.

The features of slimr fall into four categories and are as follows:

1. Integrated Development
   a. Autocomplete and Documentation (within R) for SLiM code (Figure 1)
   b. Code highlighting and pretty printing of SLiM code
2. Data Input/Output (Figure 2)
   a. Automatic output generation and extraction from SLiM to R (r_output())
   b. Insert arbitrary R objects into SLiM scripts through inlining (r_inline())
3. Metaprogramming (Figure 2)
   a. Code templating for SLiM scripts (r_template())
   b. Flexible general metaprogramming tools (support for rlang's !! and !!! forcing operators)
4. Interoperability
   a. Full feature preserving conversion from SLiM code to slimr code (as_slimr_script() and as_slimr_code()) and from slimr to SLiM (as_slim_text()).
   b. Run code in SLiM from R

The first set of features is designed to make it easy to develop SLiM scripts in an R development environment such as RStudio, and mostly recapitulates features that SLiM users already have access to in the form of SLiMgui and QtSLiM (https://messerlab.org/slim/). The second and third features are implemented using what we call "slimr verbs", allowing SLiM and R features to be combined in advanced ways. The fourth and final set of features allows users to move smoothly back and forth between SLiM (or SLiMgui) and slimr in R. The integration between R and SLiM provided by slimr compensates knowledgeable users of R for a lack of knowledge of SLiM, helping to lower the barrier to learning and using SLiM. At the same time, it provides powerful metaprogramming abilities to advanced SLiM users, allowing the use of R as a workflow management system for SLiM simulations, which R excels at. For example, slimr allows SLiM simulation to be easily used within R workflow management packages such as targets and its make-like declarative workflows (Landau, 2021), or guildai (Kalinowski, 2023), originally designed to keep track of machine learning runs but also useful for any task that needs to explore a large number of parameters, such as simulations.

In the next section, we describe slimr features in greater detail, showing examples through screenshots and code snippets.

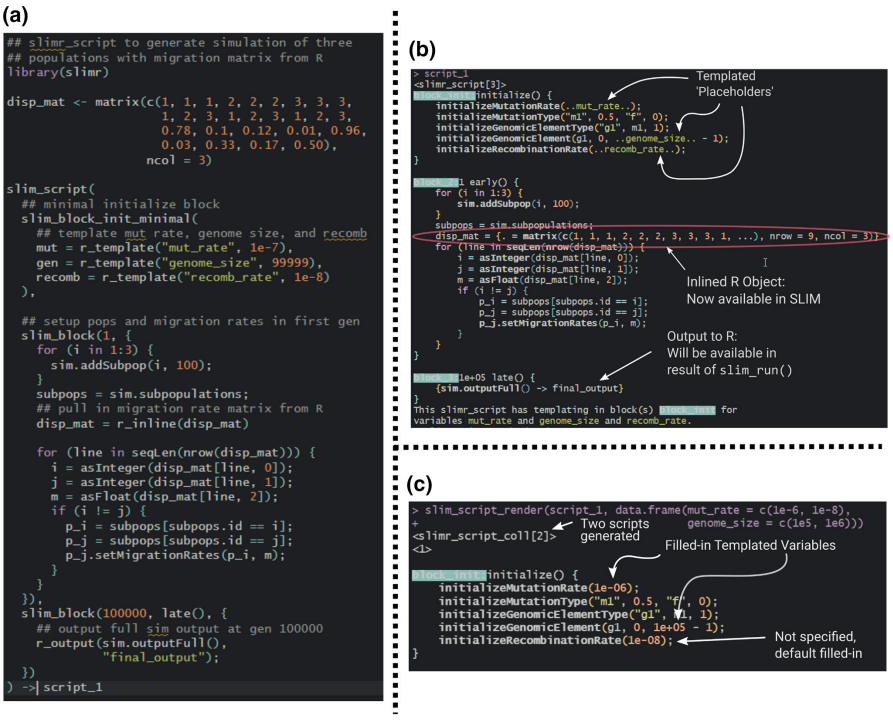

FIGURE 1 Example of a single script using the main slimr verbs (r_template, r_output and r_inline). (a) Code to specify the slimr_script. (b) Pretty printing of the script, showing special slimr syntax. (c) Example of running slim_script_render on the slimr_script object, demonstrating how placeholder variables specified in r_template are replaced with provided values. All code from the above example can be accessed as a package vignette (https://rdinnager.github.io/slimr/articles/simple_example_using_migration_and_fst.html).

2.1 | Integrated development

slimr allows you to write SLiM code from within an R integrated development environment (IDE). slimr is designed to work well with RStudio, but can be used in any R IDE. The syntax used to write SLiM code in slimr is very similar to the native SLiM syntax. As an example, here is a minimal simulation written using slimr.

```
slim_script(
 slim_block(initialize(),
 {
  initializeMutationRate(1e-7);
  initializeMutationType("m1", 0.5, "f", 0.0);
  initializeGenomicElementType("g1", m1, 1.0);
  initializeGenomicElement(g1, 0, 99,999);
  initializeRecombinationRate(1e-8);
 }),
 slim_block(1,
 {
  sim.addSubpop("p1", 500);
 }),
 slim_block(10,000,
 {
  r_output_full();
 })
) ->script_1
```

The above code initializes a simulation with a single species with a single mutation type, setting its mutation and recombination rate, it then adds a single population of the species of 500 individuals and then creates an output of the full simulation state at generation 10,000. The code assigns this code as a slimr script object script_1, which can then be further manipulated, printed prettily and sent to SLiM to run. In this very simple simulation, because we have not included any logic for deterministic evolutionary processes, this simulation would model how the genome would change under genetic drift, for a population of 500 individuals, based on the mutation and recombination rate. This is a simple simulation but SLiM is capable of producing population genomics simulations of great complexity and sophistication, limited primarily by a user's imagination, by simply manipulating the logic of the simulation.

In general, a script is specified in slimr using the slim_script function, within which you create slimr coding blocks, using the slim_block function. The user can create as many slimr code blocks as desired within a slim_script. In the example above, we have added a slimr "verb" (r_output_full), which tells SLiM to output the full state of the simulation and return it to R during the execution of the block. We will discuss slimr verbs in more detail in the next section.

In order to make the development of SLiM simulation within R easier for slimr users, the entire reference documentation for functions in SLiM and Eidos scripting language (on which SLiM is based) is included in slimr (with the original authors' permission). Hence, not only can you look up relevant SLiM functions in their R session (by typing? followed the name of the function, e.g.,? slimr::addRecombinant), but the R IDE can also perform autocompletion.

slimr makes it easy to write SLiM code in R after you learn a few differences between SLiM and slimr (described in Table S1). This means you can learn how to write complex SLiM simulations by reading SLiM documentation and the examples found within it (https://messerlab.org/slim/). In fact, all examples from the SLiM manual, referred to as 'recipes' are available as a character vector within slimr, accessible as the object 'slim_recipes'. In fact, you can directly convert SLiM scripts into slimr code, which we describe in greater detail in the section on interoperability.

2.2 | Slimr verbs

Much of the power of slimr comes from its use of "slimr verbs", which are prefixed with 'r_', and are used for data input and output, and for metaprogramming. What makes slimr verbs unique and powerful is that they are special R functions that can be inserted directly into slimr coding blocks, where they will modify how the SLiM script will be generated and run in SLiM, typically by inserting purpose-built SLiM code in place before SLiM is run. In other words, they are not passed 'as is' to SLiM, but make it easy for R to interact with SLiM. Because of slimr verbs, slimr code appears to be a hybrid between SLiM and R code. slimr verbs allow all set-up and logic required to use SLiM with R to occur inside the slim_block() coding blocks comprising the slimr_script object, thus requiring fewer arguments to be set in preparation for downstream analysis (e.g. slim_run does not require many complex arguments because most of what it needs to know is embedded in the slimr_script object). In our experience, this leads to a very smooth experience using slimr by reducing the frequency of switches between different mental modes. By convention, all slimr verbs have the prefix r_, to denote they are R functions that will be executed from within R, typically to insert something into the eventual SLiM script. They are meant to be used only inside slim_block calls, and will do nothing if called outside this context. All other slimr functions are prefixed with slim_, which generally means they are to be used on slimr_script objects (or to create them), and not inside a slim_block call. Examples of all slimr verbs can be seen in an example script in Figure 1, and are described in Table 1.

The main slimr verbs for input and output are r_inline() and r_output() respectively (Table 1).

2.3 | Metaprogramming

Metaprogramming is programming that generates or manipulates code itself. slimr has facilities for manipulating SLiM programming code and generating scripts. The main slimr verb for doing this is r_template. r_template is designed to help you easily generate many versions of a slimr_script with different parameters, which greatly facilitates parameter exploration, sensitivity analysis and the use of SLiM simulations in simulation-based inference methods such as approximate Bayesian computation (ABC; Beaumont et al., 2002).

Perhaps the simplest way to get data from R into a slimr simulation is by using the forcing operator (!!), which can be used anywhere

in a slim_block() coding block to flag variables that are meant to come from the R environment rather than used as is within the resulting SLiM code. The forcing operator forces R to replace the expression following it with the value to which the expression evaluates, which prevents the expression from being treated as SLiM code. In simple terms, the (!!) is used to flag that the expression following is not a SLiM expression, but a reference to an object in your R environment. slimr supports operators for forcing (!!) and splicing (!!!), which are defined and used extensively in the tidyverse package (and which are imported by slimr). An example of using the (!!) operator for this purpose is provided in the first example in the Examples section.

2.4 | Interoperability

slimr also facilitates interoperability between R and SLiM. The main function for achieving this is slim_run(), which takes a slimr_script object previously created, translates it to SLiM code (including inserting any slimr verb generated content) and then sends it to SLiM to be run, while retrieving any output that may have been specified using the r_output() verb. This simulation output can then be analysed or visualized using any other R functionality.

slimr also provides conversion functions that can convert SLiM scripts to slimr code (functions as_slimr_script() and as_slimr_code()), or convert slimr code to a SLiM script (function as_slim_text()). This makes it easy to move back and forth between SLiM and slimr for users comfortable with both. For example, it is possible to develop a SLiM simulation using SLiMgui, with its convenient interactive tools for developing and visualizing simulations, and then automatically convert this prototype to slimr in order to facilitate running it over large parameter spaces. A new experimental function recently added to slimr, slim_auto(), allows not only automatically converting a SLiM script into slimr code, but automatically adding slimr verbs into the script by inferring how the model should be parameterized based on code analysis (see slimr vignette ref for an example of this).

It is increasingly being seen as vital for biologists to share code used to generate their results in the spirit of open science. A researcher may spend months perfecting a SLiM script that simulates a particular scenario of interest, but this scenario and those similar to it are likely of interest to other researchers as well. slimr allows the sharing of simulations in a very open and easy-to-use way, through the R software ecosystem. It provides tools that can allow researchers, with very little additional code, to make their simulations accept user-defined input and output to common formats used by R users. Simulations can easily be wrapped into an R package, which can then be installed by any R user with a command. Because slimr provides general interfacing functionality from SLiM to R, it allows open development of simulations by developers with much less experience with SLiM coding, and requires far less time.

3 | EXAMPLES

Here, we demonstrate the use of slimr on a short and simple example, and one more extensive example.

3.1 | Simulating nucleotide evolution

The following script simulates a population of 100 individuals that randomly splits into two equally sized subpopulations with a

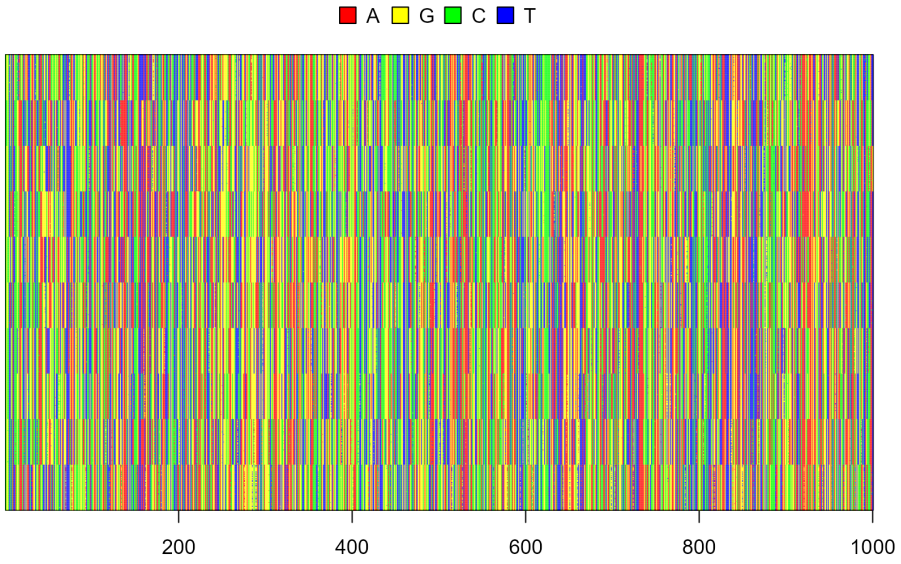

FIGURE 2 Simulated sequences for each individual. The plot shows each position in the genome along the x axis and each individual along the y axis (where individuals in the same subpopulation are adjacent to each other). Grid cells are coloured by their nucleotide identity based on the legend. Subpopulation clustering is visually apparent by the presence of patterns of strong horizontal bands of similar colour that are organized in 10 distinctive patterns.

probability split_prob in each generation, after which the subpopulations are reproductively isolated from each other.

```
## set some parameters.
seed <- 1205;
split_prob <- 0.001;
max_subpops <- 10;

## specify simulation
split_isolate_sim<- slim_script(
 slim_block(initialize(), {.
  setSeed(!!seed);
  ## tell SLiM to simulate nucleotides
  initializeSLiMOptions(nucleotideBased=T);
  initializeAncestralNucleotides(randomNucleotides(1000));
  initializeMutationTypeNuc("m1", 0.5, "f", 0.0);
  initializeGenomicElementType("g1", m1, 1.0, mmJukesCantor(1e-5));
  initializeGenomicElement(g1, 0, 1000-1);
  initializeRecombinationRate(1e-8);
 }),
 slim_block(1, {
  defineGlobal("curr_subpop", 1);
  sim.addSubpop(curr_subpop, 100);
 }),
slim_block(1, 10,000, late(), {
  if(rbinom(1, 1,!! split_prob) ==1) {
```

TABLE 1 The main functions provided by slimr. More details for many of these functions can be found in the Supplementary material, the slimr website (https://rdinnager.github.io/slimr/), and the documentation and vignettes within the slimr package itself. Note that slimr verbs often have variants with different suffixes to deal with certain common situations, such as r_output_nucleotide() to automatically create outputs in the form of nucleotide data. All variants can be found in the package's documentation.

Function	Description
Simulation creation	
slim_script()	Used to create a new slimr_script object. It takes a list of slimr_block objects created by slim_block()
slim_block()	Used to create slimr_block objects, which contain the programming logic for generating a SLiM simulation. The syntax is similar to how code blocks are created in SLiM
Data Input/Output	
r_inline()[a]	Can be used inside a call to slim_block(), allowing the insertion of arbitrary R objects defined in the R environment into the generated SLiM script, so that data created or processed in R can be passed to SLiM for use in a simulation
r_output()[a]	Can be used inside a call to slim_block(), allowing the insertion of SLiM code that generates output from the simulation in a format that can be automatically read within R
Metaprogramming	
r_template()[a]	Can be used inside a call to slim_block(), allowing insertion of dynamically specified parameter values. In combination with slim_script_render() this allows creating large lists of slimr_script objects with different parameter values, which can automatically be run in parallel with slim_run(). Very useful for sensitivity analysis or approximate Bayesian computation (ABC) or other simulation-based inference techniques
slim_script_render()	Takes a slimr_script object and a list or data.frame of parameter values to be inserted where r_template() calls have been made. Can also be used to dynamically insert different objects where r_inline() calls have been made
slim_auto()	Experimental function that automatically determines suitable places to insert r_inline() and r_template() calls based on code analysis (by detecting variables that are inputs to the computational graph implied by the code in a slimr_script object, and by comparing to objects available in the R environment). In the future, it will also support automatic detection of where r_output() calls should be placed as well
Interoperability	
slim_run()	Takes a slimr_script object created by slim_script(), or a list of such objects, translates them into SLiM code while inserting everything specified by embedded slimr verbs, runs it in SLiM while retrieving the simulation output as specified by r_output(), potentially in parallel. SLiM must be installed and linked to slimr before slim_run() will work
as_slim_text()	Takes a slimr_script object created by slim_script and converts it into a SLiM script, as text. This can be directly opened in SLiMgui and run interactively or passed to the SLiM command line program to run the simulation completely outside R if desired
as_slimr_script()	Takes a SLiM script as a character vector and converts it into a slimr_script object containing the equivalent simulation logic
as_slimr_code()	Same as as_slimr_script() except that it converts the SLiM script into the slimr code that would create the equivalent slimr_script object. Therefore, the output of the function is text. This can be copied and pasted into an R console or IDE, and then edited within R. This is very useful if you want to use an existing SLiM script as a starting point for a modified simulation

[a]Designates a slimr 'verb', which can only be used inside slim_block() calls.

```
    ## split a subpop
    subpop_choose=sample(sim.subpopulations, 1);
    curr_subpop=curr_subpop +1;
    sim.addSubpopSplit(subpopID=curr_subpop,
     size=100,
     sourceSubpop=subpop_choose);
    ## if too many subpops, remove one randomly
    if(size(sim.subpopulations)>!! max_subpops) {
     subpop_del=sample(sim.subpopulations, 1);
     subpop_del.setSubpopulationSize(0);
   }
  }
   ## output nucleotide data with slimr verb.
   r_output_nucleotides(subpops=TRUE,do_every=100);
  }),
  slim_block(10,000, late(), {
   sim.simulationFinished();
})
)
results <- slim_run(split_isolate_sim)
```

This simulation simulates genomic evolution with an explicit nucleotide sequence substitution model (Jukes–Cantor model). By default, SLiM only simulates and keeps track of 'mutations' in a more abstract sense (these could be thought of as generating new alleles at a gene, or SNPs, or however the researcher wants to interpret them). However, explicit nucleotide evolution models can be enabled in SLiM using initializeSLiMOptions(nucleotideBased=T) within the initialization block. We then define a particular model of nucleotide substitution using initializeGenomicElementType(), which sets up the model for a particular mutation type, of which we have only one in this simulation. However, multiple mutation types are possible in SLiM, which theoretically could each have a different model of nucleotide substitution.

The main simulation logic is contained in a slim_block call that looks like slim_block(1, 10,000, late(), {...}). This specifies that the code in {...} should run in all generations from 1 to 10,000 – late() is a callback which tells the simulation to run the code as late as possible in the simulation cycle.

After specifying the simulation as a slimr_script object, we run the simulation using slim_run(), and place the results in an object named results.

This example also demonstrates the forcing operator!!. In the script above, we have highlighted where this is occurring in bold. Where!! occurs slimr will insert the value of the variable immediately following, as defined within the R environment, so it will not be passed as is to SLiM. The script is also available as a vignette in the slimr package (which can be viewed here: https://rdinnager.github.io/slimr/articles/simple_nucleotide_example.html).

Once the simulation runs, we can extract the data, and print it to see what the resulting object looks like. Then, we can plot one of the sequence alignments as an image plot (Figure 2).

```
res_data <- slim_results_to_data(results)
res_data
## # A tibble: 100 x 6
## type expression generation name data
##
##  1 slim_nucleotides slimr_output_nucleotide~100
seqs
##  2 slim_nucleotides slimr_output_nucleotide~200
seqs
```

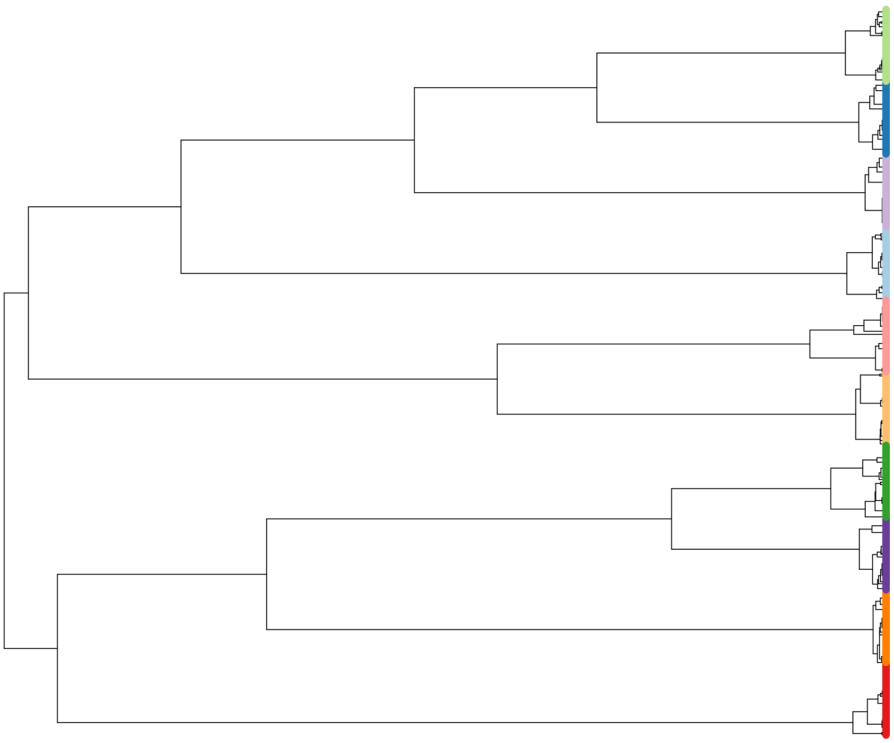

FIGURE 3 UPGMA tree of subpopulations simulated in the nucleotide evolution example in section 3.1, with tip points coloured by subpopulation.

```
## 3 slim_nucleotides slimr_output_nucleotide~300 seqs
## 4 slim_nucleotides slimr_output _nucleotide~400 seqs
## 5 slim_nucleotides slimr_output_nucleotide~500 seqs
## 6 slim_nucleotides slimr_output_nucleotide~600 seqs
## 7 slim_nucleotides slimr_output_nucleotide~700 seqs
## 8 slim_nucleotides slimr_output_nucleotide~800 seqs
## 9 slim_nucleotides slimr_output_nucleotide~900 seqs
## 10 slim_nucleotides slimr_output_nucleotide~1000 seqs
## … with 90 more rows
image(ape::as.DNAbin(res_data$data[[100]]))
```

And then we use some other R packages to quickly build a tree based on the simulated nucleotides to see if it looks like what we would expect from a sequentially splitting population (Figure 3).

```
## convert to ape::DNAbin
al<- ape::as.DNAbin(res_data$data[[100]])
dists <- ape::dist.dna(al)
upgma_tree <- ape::as.phylo(hclust(dists, method="average"))
pal <- paletteer::paletteer_d("RColorBrewer::Paired", 10)
plot(upgma_tree, show.tip.label=FALSE)
ape::tiplabels(pch=19, col=pal[as.numeric(as.factor(res_data$subpops[[100]]))])
```

4 | SCIENTIFIC HYPOTHESIS EXPLORATION EXAMPLE: INVESTIGATING POPULATION GENOMICS OF SMALL MAMMALS IN A PERIODIC ENVIRONMENT

In this section, we provide a brief description of a full example analysis using simulation that is fully described in the accompanying Supplementary Material and a substantial R vignette available here: https://rdinnager.github.io/slimr/articles/Main_manuscript_example_v2.html.

The context for this example is a long-term ecological study in the Simpson Desert in central Australia. Several authors of this paper have studied the population dynamics of small mammals and reptiles in this desert for more than 30 years (Dickman et al., 2014; Greenville et al., 2016, 2017). Recently, we have begun sequencing tissue samples taken from animals captured during the past 15 years, and obtained single nucleotide polymorphism (SNP) data using DArT (Diversity Arrays Technology Pty Ltd) technology. Here, we use SNP data from 167 individuals of a common native rodent species, the sandy inland mouse *Pseudomys hermannsburgensis*, sampled at seven sites over 3 years (2006–2008), and subsequently aggregated to three subpopulations for analysis. The three sample years span periods before and after a major rainfall event at the end of 2006; big rains occur infrequently in the study region (every 8–12 years) (Greenville et al., 2012) but drive major population eruptions.

We used the SNP data to calculate pairwise F_{st} values among the three subpopulations in each year, revealing that pairwise F_{st} values dropped rapidly to nearly zero immediately after the rainfall event from a high recorded just prior to the event when the populations were more genetically differentiated. We interpreted this result to mean that the rainfall event, which caused the sandy inland mouse population to rapidly increase, also allowed animals to move out of spatially scattered refuge patches to which they had been confined during the preceding dry period (Dickman et al., 2011). This

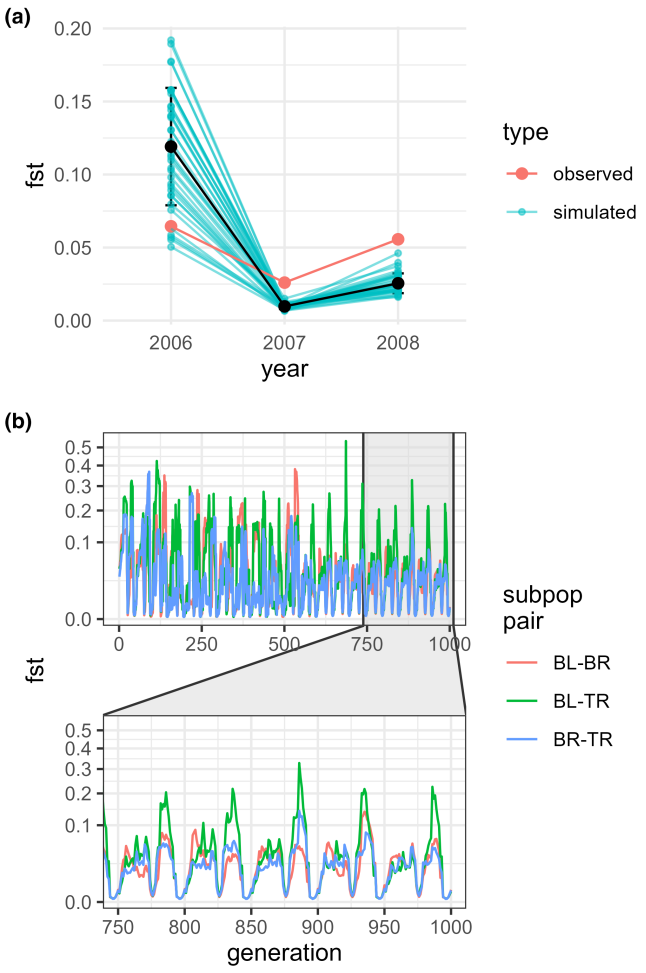

FIGURE 4 (a) Mean F_{st} values from 36 replicate simulations simulated under our hypothesized mechanism to explain Fst fluctuations in small mammal population in the Simpson Desert using hand-chosen parameter values. Blue values and lines represent simulated values and red values and line represent the observed F_{st} values. Details of simulation including code are in the Data S1. (b) Same simulation run over many generations, showing the three subpopulation pairs separately. The subpop pair refers to pairwise combinations of three subpopulations named BL, BR and TR (further explanation can be found in the vignette of this example in the Data S1, or as included in the slimr package).

movement allowed the subpopulations to mix, leading to a decrease in population genetic structure as measured by F_{st}.

In the example, we use simulations to evaluate our interpretation regarding the processes driving changes in F_{st} values. We found that our initial hypothesis, that the rainfall event led to the mixing of previously unconnected populations in refuge patches, provided a good match to the data when we simulated the population and genomic processes (Figure 4). However, we also identified several other processes that could generate similar outcomes, which raises the question as to what data or analyses would be required to distinguish among these competing processes.

To formalize our ideas a little more, we ran an approximate Bayesian computation (ABC) analysis to derive an approximate posterior distribution of model parameters that produced a good fit to our short F_{st} time series (see Data S1: ABC Analysis for code used). We were able to easily move from simulation exploration to a more formal fitting exercise because the simulation was already in R (thanks to slimr), and so only a small amount of code was required to convert the input and output of our simulation to the format required by the easyABC package, which we used for this analysis.

Simulations using parameter values drawn from the approximate posterior of our ABC analysis are shown in Figure 5a, and the pairwise joint posterior distribution of those parameter values is shown in Figure 5b. Overall, the ABC analysis confirms that these three data points do not constrain much of the parameter space of

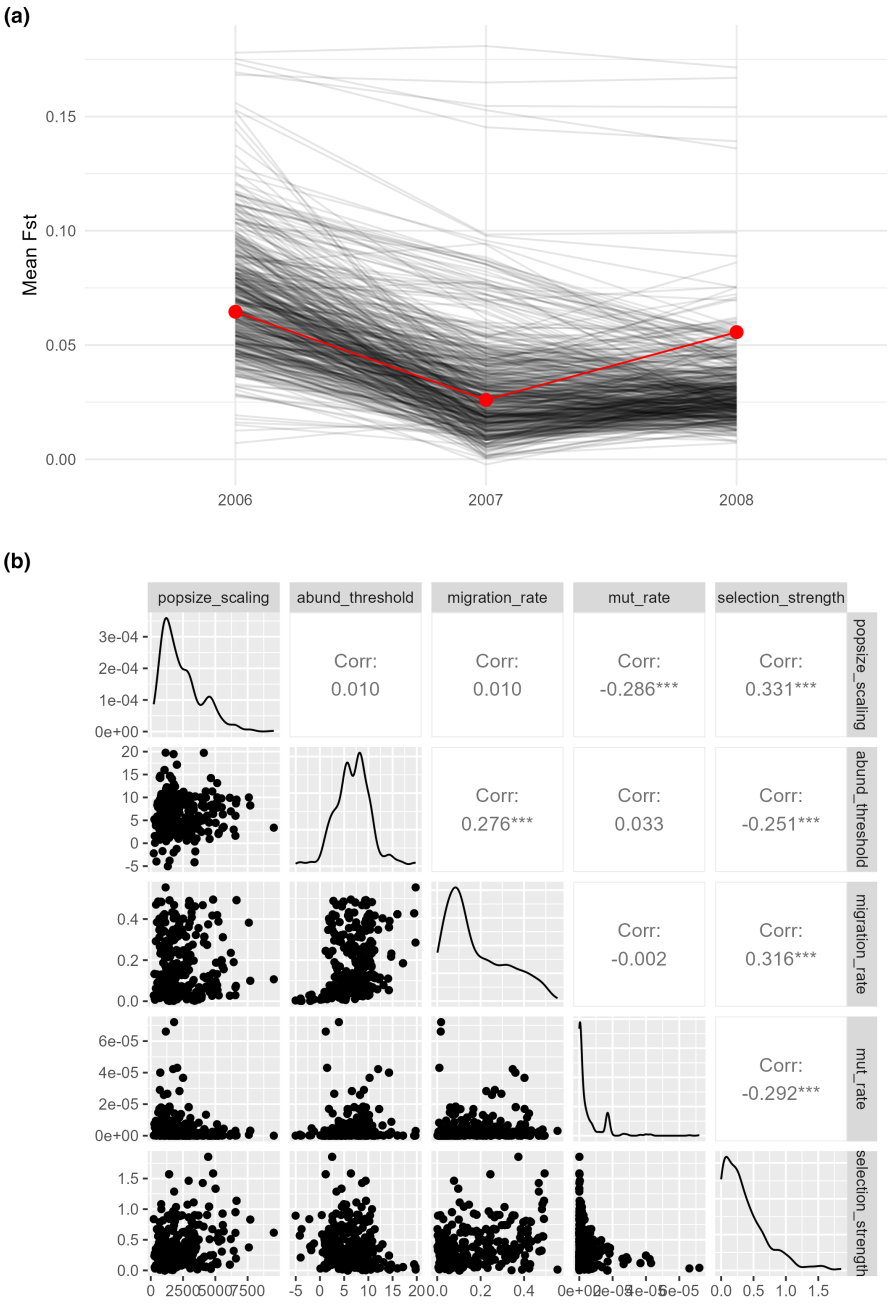

FIGURE 5 (a) Fst values calculated from simulations based on 500 parameter value sets drawn from the approximate posterior distribution of our simulation based on an ABC analysis. Partially transparent black lines represent the simulations. Red points and line represent the observed Fst values from the study described in this section. (b) Samples from the approximate posterior distribution for our simulation model fit with approximate Bayesian computation. Lower left panels show the five parameters of our simulation model estimated, plotted against each other in a pairwise fashion, giving an indication of their joint posterior distribution. Some parameters are highly correlated in the posterior. Upper right panels show the Pearson correlation coefficient, and the panels in the diagonal show the marginal distribution of each parameter estimated using kernel density estimation on the samples.

plausible models, though in some cases joint distribution appears to be somewhat constrained for certain combinations of parameters.

The marginal posterior distribution based on samples from the ABC analysis confirms that our data do not constrain individual parameters much, with a fairly wide distribution for most parameters providing a good fit to our data (Figure 5b, diagonal panels). The only exception was perhaps mutation rate, for which the lower values that we simulated tended to provide a better fit. The parameter of most interest to us was the abundance threshold (abund_threshold in Figure 5b), which specified the population size above which a subpopulation would 'turn on' migration, that is, start exporting individuals to the other subpopulations (in the real system, this population size change is driven by rainfall). In this simulation, an abundance threshold of zero or less would be migration always happening, and one of 20 or over would be migration almost never happening. Some simulations produced well-fitting Fst values for nearly all relevant values of the abundance threshold, with some falloff at either end. However, when we start looking at combinations of multiple parameters we see that the value of the abundance threshold parameter does constrain what values of other parameters will make for a good fit to the data. For example, if the abundance threshold is low, and thus migration is always on, only simulations with very low migration rates and very low mutation rates can provide a good fit to the data (Figure 5b, panels in rows 3 and 4 in column 2). All in all, this suggests that there are two approaches to improving our ability to distinguish how different processes lead to the patterns we see (besides just collecting more data): (1) try adding new summary statistics besides just pairwise Fst, which may capture some other aspect of the data, and (2) use some independent sources of data or information to estimate and constrain the parameter space of our simulations closer to that of the real system. In particular, approach 1 could be tested without having to collect more data by doing more simulations: we could simulate our model, then simulate data collection and calculate our new summary statistic on the simulated data. We can then see if we can recover the parameters of our simulation better than we could before incorporating our new statistic.

The results from these preliminary simulations will, thus, be invaluable in guiding which individuals and time periods we should focus our sequencing on, and what summary statistics to use, to maximize the chances of distinguishing among competing hypotheses that might explain the combined population and genetic patterns in the data. Ultimately, we aim to use this approach to understand how future climate change could alter the population and genetic structure of desert animals, highlighting the value of slimr in a scientific workflow.

AUTHOR CONTRIBUTIONS

RD, BG, SS and RPD developed the concept for the package. SE, CD, GW and AG provided feedback on the package design. RD coded the package and wrote the manuscript draft. CD, GW and AG contributed data for testing of the package, and BG helped test the package as a user. All authors contributed critically to manuscript drafts and gave final approval for publication.

ACKNOWLEDGEMENTS

We thank Benjamin C. Haller and Phillip W. Messer for permission to reproduce the documentation and examples of SLiM in slimr, and for valuable feedback on the package (from B.C.H.). Thanks to Emily Stringer for providing additional test data to help develop the methods used in this manuscript. This work was funded by Australian Research Council Discovery Project grant DP180103844.

DATA AVAILABILITY STATEMENT

The data and code that support the findings of this study are openly available in figshare at http://doi.org/10.6084/m9.figshare.24758082 or are included within the slimr package which can be installed or downloaded at https://github.com/rdinnager/slimr.

ORCID

Russell Dinnage https://orcid.org/0000-0003-0846-2819
Stephen D. Sarre https://orcid.org/0000-0002-7158-2517
Richard P. Duncan https://orcid.org/0000-0003-2295-449X
Christopher R. Dickman https://orcid.org/0000-0002-1067-3730
Scott V. Edwards https://orcid.org/0000-0003-2535-6217
Aaron C. Greenville https://orcid.org/0000-0002-0113-4778
Glenda M. Wardle https://orcid.org/0000-0003-0189-1899
Bernd Gruber https://orcid.org/0000-0003-0078-8179

REFERENCES

Beaumont, M. A., Zhang, W., & Balding, D. J. (2002). Approximate Bayesian computation in population genetics. *Genetics*, *162*(4), 2025–2035. https://doi.org/10.1093/genetics/162.4.2025

Brehmer, J., Louppe, G., Pavez, J., & Cranmer, K. (2020). Mining gold from implicit models to improve likelihood-free inference. *Proceedings of the National Academy of Sciences of the United States of America*, *117*(10), 5242–5249. https://doi.org/10.1073/pnas.1915980117

Carvajal-Rodríguez, A. (2010). Simulation of genes and genomes forward in time. *Current Genomics*, *11*(1), 58–61. https://doi.org/10.2174/138920210790218007

Cranmer, K., Brehmer, J., & Louppe, G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences of the United States of America*, *117*(48), 30055–30062. https://doi.org/10.1073/pnas.1912789117

Dickman, C., Wardle, G., Foulkes, J., & de Preu, N. (2014). *Desert complex environments. Biodiversity and environmental change: Monitoring, challenges and direction* (pp. 379–438). CSIRO Publishing. https://www.publish.csiro.au/book/7009/

Dickman, C. R., Greenville, A. C., Tamayo, B., & Wardle, G. M. (2011). Spatial dynamics of small mammals in central Australian desert habitats: The role of drought refugia. *Journal of Mammalogy*, *92*(6), 1193–1209. https://doi.org/10.1644/10-MAMM-S-329.1

Greenville, A. C., Dickman, C. R., & Wardle, G. M. (2017). 75 years of dryland science: Trends and gaps in arid ecology literature. *Plos One*, *12*(4), e0175014. https://doi.org/10.1371/journal.pone.0175014

Greenville, A. C., Wardle, G. M., & Dickman, C. R. (2012). Extreme climatic events drive mammal irruptions: Regression analysis of 100-year trends in desert rainfall and temperature. *Ecology and Evolution*, *2*(11), 2645–2658. https://doi.org/10.1002/ece3.377

Greenville, A. C., Wardle, G. M., Nguyen, V., & Dickman, C. R. (2016). Population dynamics of desert mammals: Similarities and contrasts within a multispecies assemblage. *Ecosphere*, *7*(5), e01343. https://doi.org/10.1002/ecs2.1343

Haller, B. C., & Messer, P. W. (2019). SLiM 3: Forward genetic simulations beyond the Wright-fisher model. *Molecular Biology and Evolution*, *36*(3), 632–637. https://doi.org/10.1093/molbev/msy228

Haller, B. C., & Messer, P. W. (2023). SLiM 4: Multispecies eco-evolutionary modeling. *The American Naturalist*, *201*(5), E127–E139. https://doi.org/10.1086/723601

Hill, P., Dickman, C. R., Dinnage, R., Duncan, R. P., Edwards, S. V., Greenville, A., Sarre, S. D., Stringer, E. J., Wardle, G. M., & Gruber, B. (2023). Episodic population fragmentation and gene flow reveal a trade-off between heterozygosity and allelic richness. *Molecular Ecology*, *32*(24), 6766–6776. https://doi.org/10.1111/mec.17174

Hoban, S. (2014). An overview of the utility of population simulation software in molecular ecology. *Molecular Ecology*, *23*(10), 2383–2401. https://doi.org/10.1111/mec.12741

Johri, P., Aquadro, C. F., Beaumont, M., Charlesworth, B., Excoffier, L., Eyre-Walker, A., Keightley, P. D., Lynch, M., McVean, G., Payseur, B. A., Pfeifer, S. P., Stephan, W., & Jensen, J. D. (2022). Recommendations for improving statistical inference in population genomics. *PLoS Biology*, *20*(5), e3001669. https://doi.org/10.1371/journal.pbio.3001669

Kalinowski, T. (2023). *guildai: Track Machine Learning Experiments*.

Kelleher, J., Etheridge, A. M., & McVean, G. (2016). Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLoS Computational Biology*, *12*(5), e1004842. https://doi.org/10.1371/journal.pcbi.1004842

Landau, W. (2021). The targets R package: A dynamic make-like function-oriented pipeline toolkit for reproducibility and high-performance computing. *The Journal of Open Source Software*, *6*(57), 2959. https://doi.org/10.21105/joss.02959

Manel, S., & Holderegger, R. (2013). Ten years of landscape genetics. *Trends in Ecology & Evolution*, *28*(10), 614–621. https://doi.org/10.1016/j.tree.2013.05.012

Marjoram, P., Molitor, J., Plagnol, V., & Tavare, S. (2003). Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences of the United States of America*, *100*(26), 15324–15328. https://doi.org/10.1073/pnas.0306899100

Messer, P. W. (2013). SLiM: Simulating evolution with selection and linkage. *Genetics*, *194*(4), 1037–1039. https://doi.org/10.1534/genetics.113.152181

Patton, A. H., Margres, M. J., Stahlke, A. R., Hendricks, S., Lewallen, K., Hamede, R. K., Ruiz-Aravena, M., Ryder, O., McCallum, H. I., Jones, M. E., Hohenlohe, P. A., & Storfer, A. (2019). Contemporary demographic reconstruction methods are robust to genome assembly quality: A case study in tasmanian devils. *Molecular Biology and Evolution*, *36*(12), 2906–2921. https://doi.org/10.1093/molbev/msz191

Petr, M., Haller, B. C., Ralph, P. L., & Racimo, F. (2022). slendr: A framework for spatio-temporal population genomic simulations on geographic landscapes. *BioRxiv*. https://doi.org/10.1101/2022.03.20.485041

Sisson, S. A. (2018). *Handbook of approximate bayesian computation*. CRC Press, [2019]: Chapman and Hall/CRC. https://doi.org/10.1201/9781315117195

Storfer, A., Patton, A., & Fraik, A. K. (2018). Navigating the interface between landscape genetics and landscape genomics. *Frontiers in Genetics*, *9*, 68. https://doi.org/10.3389/fgene.2018.00068

Strand, A. E. (2002). Metasim 1.0: An individual-based environment for simulating population genetics of complex population dynamics. *Molecular Ecology Notes*, *2*(3), 373–376. https://doi.org/10.1046/j.1471-8286.2002.00208.x

Torada, L., Lorenzon, L., Beddis, A., Isildak, U., Pattini, L., Mathieson, S., & Fumagalli, M. (2019). ImaGene: A convolutional neural network to quantify natural selection from genomic data. *BMC Bioinformatics*, *20*(Suppl 9), 337. https://doi.org/10.1186/s12859-019-2927-x

Wang, Z., Wang, J., Kourakos, M., Hoang, N., Lee, H. H., Mathieson, I., & Mathieson, S. (2020). Automatic inference of demographic parameters using generative adversarial networks. *Molecular Ecology Resources*, *21*(8), 2689–2705. https://doi.org/10.1111/1755-0998.13386

Wickham, H. (2014). *Advanced R*. Chapman and Hall/CRC. https://doi.org/10.1201/b17487

Yuan, X., Miller, D. J., Zhang, J., Herrington, D., & Wang, Y. (2012). An overview of population genetic data simulation. *Journal of Computational Biology*, *19*(1), 42–54. https://doi.org/10.1089/cmb.2010.0188

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: Dinnage, R., Sarre, S. D., Duncan, R. P., Dickman, C. R., Edwards, S. V., Greenville, A. C., Wardle, G. M., & Gruber, B. (2024). slimr: An R package for tailor-made integrations of data in population genomic simulations over space and time. *Molecular Ecology Resources*, *24*, e13916. https://doi.org/10.1111/1755-0998.13916