# Revticulate: An R framework for interaction with RevBayes

Caleb P. Charpentier and April M. Wright[1]

[1]Department of Biological Sciences, Southeastern Louisiana University, Hammond, LA, USA

October 2, 2021

## Running Head: Revticulate

## Keywords

phylogeny, Bayesian phylogenetics, R packages, RevBayes

## 1 Abstract

1: Phylogenetic methods are increasingly complex. Researchers need to make many choices about how to model different aspects of the data appropriately. It is increasingly common to deploy hierarchical Bayesian models in which different data types may be described by different processes. This necessitates tools to help users understand model assumptions more clearly.

2: We describe the package `Revticulate`, which provides an R-based interface to the software RevBayes. RevBayes is a Bayesian phylogenetics program that implements an R-like computing language, but does not interface with R itself. Revticulate was designed to allow communication between an R session, and all of its associated capabilities, such as plotting and simulation, and a RevBayes session.

3: Revticulate can be used to copy objects from RevBayes into R. We provide several usage examples demonstrating how objects, such as such as random variables drawn from probability distributions and phylogenetic trees, can be generated in RevBayes. We then show how these objects can be used with R's phylogenetic ecosystem to plot a phylogenetic tree, or with base R functions to simulate the behavior of a particular probability.

4: Revticulate is a broadly useful software. Revticulate can be used alongside popular document preparation packages, such as Knitr and pkgdown to generate attractive reports, tutorials, and websites. This means that researchers who are looking to communicate their work in RevBayes can do that very easily using Revticulate, enabling rapid generation of reproducible research outputs.

## 2 Introduction

Estimating phylogenetic trees has emerged as one of the predominant challenges in comparative biology. Phylogenetic trees provide researchers with the historical context in which traits and organisms evolved. There is abundant evidence that trying to understand trait evolution without a phylogenetic tree is deeply misleading (Felsenstein, 1985; Uyeda et al., 2018). Phylogenetic trees are often estimated from molecular data (nucleotide sequences, amino acids). However, inclusion of paleontological data are crucially important in comparative analyses (Rabosky, 2010; Slater et al., 2012), and many studies of biogeography are conducted in a phylogenetic context. As such, morphological data, biogeographical information, and stratigraphic data are being used in a wider variety of studies, and across more disciplines. A researcher conducting a modern phylogenetic study may be using multiple data types, described by different mathematical models, and involving layers of statistical assumptions.

Many phylogenetic methods are now quite complex, and many phylogenetic models are hierarchical, and in which submodels may have complex dependencies on one another. For example, the Fossilized Birth-Death model (Stadler, 2010; Heath et al., 2014) and models of deep-time biogeography (Landis et al., 2018) involve hierarchical Bayesian models with multiple subcomponents describing any molecular and morphological data used to estimate the tree, stratigraphy, and geographic locality data. These complex, hierarchical models rely on researchers being able to understand not only the biology of their clade of interest, but also to have deep understandings of statistical concepts in order to be able to set reasonable priors on important model parameters. In developing this type of intuition, it is often is important to be able to explore data visually, to use programmatic statistical tools (such as R (RStudio Team, 2015)) to plot and examine distributions, and to simulate data to understand the behavior of models.

The phylogenetics software RevBayes (Höhna et al., 2014; Höhna et al., 2016) represents an attempt to reconfigure the way phylogenetics software is written. In many software packages developed over the history of phylogenetic estimation, users have been able to select from molecular or morphological evolution models implemented by the developers (examples: RAxML (Stamatakis, 2014), GARLI (Zwickl, 2006), and IQTree (Minh et al., 2020)). In these types of packages, a researcher might be able to choose analytical settings (e.g., how many bootstrap replicates, how to model character change rate heterogeneity, correcting for ascertainment bias), but to implement a new model or method means either collaborating with a developer or interacting with the source code. RevBayes implements a statistical computing language called Rev. This language is broadly similar to the well-known computing language R (RStudio Team, 2015). Rev contains a library of probability distributions, as well as mathematical operations, such as Markov Chain Monte Carlo analysis and associated operators. Further phylogenetic functions, such as tree estimation and comparative phylogenetic methods using trees are available. Using Rev, infinite combinations of models, priors and data can be assembled into custom analysis workflows. A researcher who has a new idea for a model to analyse their data, then, does not have to wait for a developer to implement their method, but is instead empowered to realize their own workflows. Assembling a model from all of its constituent pieces means there are no defaults, enabling a radical transparency in phylogenetic analysis. The researcher must, therefore, become an expert in the properties of their data, and how to use statistical models appropriately to analyse those data.

While this may be greatly empowering, asking researchers and students to learn a new programming language in order to implement their own analyses means asking them to take on a large cognitive load. It also means that researchers are not choosing from a pre-set list of models, but must instead make far more choices about what facets of their data to model. Developing the ability to do this means coming to understand choices in modeling that may be hidden from users of other software packages. In particular for Bayesian analyses, this can mean specifying priors on parameters, which involves knowledge of what different probability distributions look like, and the ability to conceptualize how populations of random variables drawn from them will behave. Researchers might also wish to visualize results or intermediate analysis products using the advanced graphical capabilities and phylogenetic package ecosystem of R. To facilitate the development of deeper statistical expertise, we have developed an interface to R and RStudio for RevBayes. Written entirely in R, Revticulate is intended to provide a set of default functions for translating between Rev code and R objects and visualizations. This manuscript will discuss the technical specifications and use of the RevBayes R interface, Revticulate.

## 3 Materials and Methods

### 3.1 Design of Revticulate

#### 3.1.1 Interaction between R and RevBayes

The Revticulate package is loosely based on the R package Reticulate (Allaire et al., 2018), which allows the use of the Python computing language in an R session. The Revticulate package accesses RevBayes by passing Rev code in a temporary file to the RevBayes executable. On Windows, this executable is named `rb.exe`. On Mac and Linux systems, it is simply called `rb`. Rev code can be run from a command-line interface by passing a Rev file path, containing the Rev script for the analysis the researcher would like to run, as an argument to RevBayes. This causes RevBayes to use its function `source()` to process the Rev code in this file. Any output resulting from the file being processed (e.g., results of calculations, error messages) is then printed to the command line.

Revticulate accepts user input from R or RStudio via the function `callRev()`. `callRev()` is a core function in the package, which formats the user code for use in RevBayes. `callRev()` then writes the code to a file, which is read with RevBayes' `source()` function. The path to these files is supplied as a command line argument to RevBayes via the base R function `system2()`, and the output from RevBayes is returned in character string format to R. A number of built-in data types (vectors, integers, trees) are then formatted for easy and pleasant viewing. The Rev script is then deleted to avoid filling storage on the user's machine.

For Revticulate to function properly, the path to RevBayes must be provided with the function `initRev()`. This function creates environmental variables for locating the RevBayes executable, for locating the temporary files used to manage RevBayes interactions, and for locating the file used to store the Rev code history ('.Revhistory'). `initRev()` takes two arguments: 'searchPath' and 'infoDir'. 'searchPath' is the absolute path to RevBayes's location. It can be the path to the RevBayes executable itself, or the path to a parent directory containing the RevBayes executeable. If multiple versions of RevBayes can be located below the parent directory, the first version `initRev()` locates will be used. 'infoDir' is the path to the directory where the 'RevInfo' folder should be located. The RevInfo folder is used to manage Rev language history and the temporary files used to interact with RevBayes. For example, if a user had a previous RevBayes session they wanted to import and interact with in R, this can be provided as the 'infoDir' argument to the `initRev()` function. If no path is provided, the RevInfo folder will be created in the parent directory of the current R session's working directory. If a 'RevInfo' folder already exists

| Rev Object Type | R Object Type | Example |
|---|---|---|
| Numeric vector | Numeric vector | [1, 2, 3] |
| Character String | Character | “Taxon” |
| Tree | Phylo (Paradis et al., 2004) | ((A, B), (C, D)) |
| Character Matrix | String | *Ctenocystis utahensis*: 0 1 1 ? ? ? |

in the ‘infoDir’ path, the history of the last Revticulate session to refer to this path will be used. Because of this feature, users can save histories from multiple Revticulate sessions, and can access any desired previous history by calling ‘initRev’ with that history as the ‘infoDir’ argument.

## 3.2 Copying of Objects between R and RevBayes

A core functionality of the `Revticulate` package is its ability to copy objects from `Revbayes` into the current R session in an appropriate R format. ‘Copy’ is used instead of ‘pass’ here, as the totality of an object’s contents are not always passed into the R environment. For example, additional attributes of Rev objects that are not directly translatable to R objects may not be copied. `Revticulate` works by interacting with RevBayes via the command line, and because of this behavior it only retrieves information from output printed to the console.

The most general way for users to submit code to RevBayes is with the `doRev()` function. `doRev()` accesses RevBayes with `callRev()`, while also keeping track of user input from previous `doRev()` calls. This feature allows RevBayes history to be tracked and Rev variables to be referenced in further `doRev()` calls, simulating a continuous RevBayes session. By default, `doRev()` returns RevBayes output to R in character format. For example, if a user creates a Rev vector with `doRev("v(1, 2, 3)")`, the string `"[ 1, 2, 3 ]"` will be returned. However, the `coerceRev()` function allows many Rev object types to be coerced into R equivalents. T his function works by parsing functional elements from RevBayes output strings to determine what R objects or primitive types have similar behaviors and functionalities. For example, the string `"[ 1, 2, 3 ]"` would be coerced to into a numeric vector. The equivalent number of opening and closing brackets allows `coerceRev()` to recognize the string as a vector, and because all of the elements in the vector are numbers, they will be coerced into R numeric objects with the base R function `as.numeric()`. Therefore, to call the function as such: `coerceRev("[ 1, 2, 3 ]")` would cause the Rev vector to be translated to an R vector. The same behavior can be produced using the `coerce` argument to `doRev`: `doRev("v(1, 2, 3)", coerce = TRUE)`. A variety of other objects, including phylogenetic trees, can be identified and coerced into R equivalents through a series of conditional statements in `coerceRev()` . If Revticulate cannot recognize an R equivalent for the RevBayes output, the original output string will be returned in an unformatted text string.

## 3.3 Interfaces to Revticulate

### 3.3.1 Using RevBayes interactively

In addition to `doRev()`, other useful tools are available for user interaction with RevBayes. One such tool is the function `repRev()`. `repRev()` creates an interactive console session that simulates the RevBayes command line. After this function is called, the prompt `rb>>>` will appear in the user's console. While this prompt is visible, all user code will be interpreted as Rev code, with the exception of some helpful Revticulate functions used to manage the '.Revhistory' file. Attempting to use R language code while this session is active may cause RevBayes to return error messages. By default, output generated during the `repRev()` session is returned in string format, but may be coerced into R code with `coerceRev()` via the `repRev()` argument `coerce`. To quit an interactive `repRev()` session, type `quit()`, `q()`, or hit the escape key.

Objects created in a `repRev()` session may also be exported for use in R. While in the `RepRev` environment, RevBayes functions can be carried out on any created obejcts. However, if the desired behavior is to work with these objects in R, they must be exported. For example, if we created a numeric variable in a `repRev()` session like so:

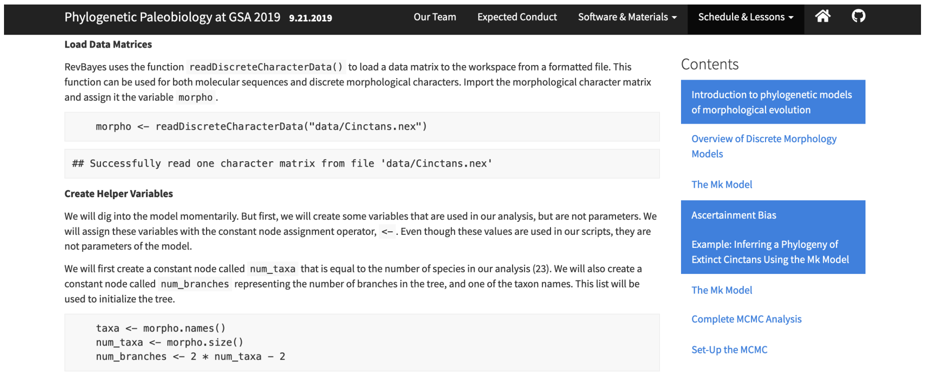
$$a \sim dnLognormal(10, .1)$$

the value of the variable could be viewed in the `repRev()` session by simply echoing it to the screen like so `print(a)`. Once the researcher has exited the `repRev()` session, the object can be exported to R using the `getRevObj` function. `exported_a <- getRevObj("a", coerce = TRUE)` will locate the variable 'a' in the RevHistory and coerce it to an appropriate R object type, in this case a numeric value of either integer (if the number is whole) or double (if a decimal). This object can now be used with any R functions available to that data type.

### 3.3.2 Knitting RevBayes Documents

Revticulate can also be integrated with the R package `knitr` (Xie, 2013). `knitr` allows for dynamic report generation and smooths the process of communicating the results of programmatic analyses. `knitr` works with a file format called RMarkdown, which can contain code, text, and image files. `knitr` can generate documents in a variety of formats, including PDF and HTML files. These documents contain 'chunks' which contain code. This format allows the user to demonstrate code usage in a variety of languages, including R, Python, C++, and many others. To use `knitr` with RevBayes, Revticulate provides the `knitRev` function, which creates a RevBayes engine called 'rb' via `knit_engines$set()`. To establish this functionality, the user must place `initRev()` and then `knitRev()` in the initiation chunk of the RMarkdown document that will be knitted. Because 'knitr' chunks are interpreted in different R sessions, `initRev()` ensures that the same Rev language history is passed between chunks and defined Rev variables can be used across them.

Revticulate can also be used with other R packages that are based on knitr and the RMarkdown format. pkgdown (Wickham and Hesselberth), for example, which can be used to generate static websites for R packages, renders the HTML for the wesbite based on knitr. Blogdown (Xie et al., 2017), which is used for generation of blogs and websites can also render Rev code via RMarkdown. Together, these packages create a powerful interface for generating tutorials and course materials. See Fig. 1 for an example website generated with Revticulate and pkgdown.



**Load Data Matrices**

RevBayes uses the function `readDiscreteCharacterData()` to load a data matrix to the workspace from a formatted file. This function can be used for both molecular sequences and discrete morphological characters. Import the morphological character matrix and assign it the variable `morpho`.

```
    morpho <- readDiscreteCharacterData("data/Cinctans.nex")
```

```
## Successfully read one character matrix from file 'data/Cinctans.nex'
```

**Create Helper Variables**

We will dig into the model momentarily. But first, we will create some variables that are used in our analysis, but are not parameters. We will assign these variables with the constant node assignment operator, `<-`. Even though these values are used in our scripts, they are not parameters of the model.

We will first create a constant node called `num_taxa` that is equal to the number of species in our analysis (23). We will also create a constant node called `num_branches` representing the number of branches in the tree, and one of the taxon names. This list will be used to initialize the tree.

```
    taxa <- morpho.names()
    num_taxa <- morpho.size()
    num_branches <- 2 * num_taxa - 2
```

Contents

Figure 1: An example of a tutorial website built using Revticulate and pkgdown. Content is written using RMarkdown with embedded Rev code, while the HTML for the website is autogenerated via pkgdown.

### 3.4 Limitations

Rev and R are still two separate languages, and as such seamless passing of objects and commands between the two is not possible. Thus, translating custom functions from Rev to R will still require the researcher to manually change functions if they would like to use them in both Rev and R. Additionally, since `system2` is used to pass commands from R to Rev, long computations may run into time-out warnings. Thus, it is best practice to save long computations to a plain text file and run them directly in RevBayes.

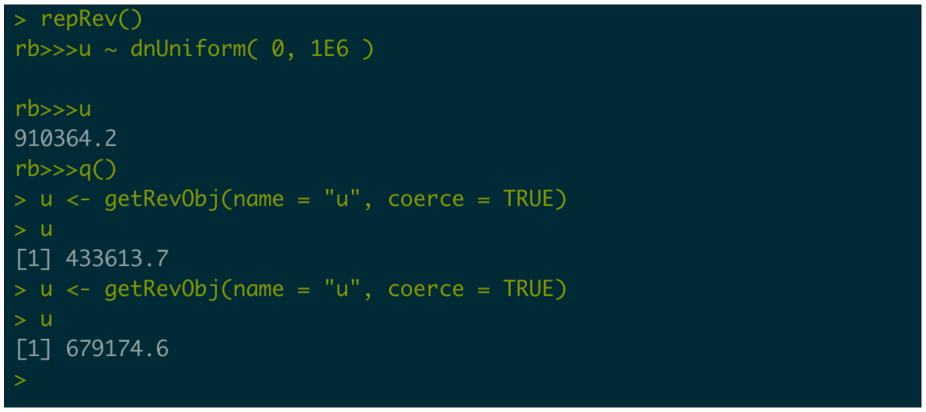
## 4 Usage Example

### 4.1 Installation of Revticulate

Revticulate can be installed in two ways. The first is via CRAN, using the default `install.packages` function in R:

`install.packages("Revticulate")`.

The second is via the remotes package (Hester et al., 2020), a lightweight package enabling installation from GitHub repositories.



```
> repRev()
rb>>>u ~ dnUniform( 0, 1E6 )

rb>>>u
910364.2
rb>>>q()
> u <- getRevObj(name = "u", coerce = TRUE)
> u
[1] 433613.7
> u <- getRevObj(name = "u", coerce = TRUE)
> u
[1] 679174.6
>
```

Figure 2: A `repRev()` session in an RStudio session. In this `repRev()` session, a value, u, is drawn from a Uniform distribution. Then, it is converted to an R object using the function `getRevObj`. The value in this figure is used to simulate the distribution in Fig. 3.

```
remotes::install_github("revbayes/Revticulate")
```

The GitHub repository for Revticulate contains cutting-edge features and may contain bugfixes, but the CRAN is known to be stable for everyday use.

## 4.2 Use of RevBayes in Console

To simulate command line RevBayes usage in R, the function `repRev()` is available. Calling `repRev()` begins a loop that simulates an interactive session with RevBayes in the R console. Because `repRev()` accesses the same `.Revhistory` file as the other Revticulate functions, Rev variables defined prior to the `repRev()` session can be referenced during the session, and variables defined during the session can be referenced after it is closed. For example, a value can be drawn from Uniform distribution in a `repRev()` session with the Rev language line

$$u \sim dnUniform(0, 1E6)$$

This command creates a stochastic variable `u` which returns a random value in a Uniform distribution between 0 and 1. Fig. 1 demonstrates how this looks in an RStudio window.

To use this value with R code, first close the `repRev()` session with `quit()`,`q()`, or the `esc` key. Then run the R code `u <- getRevObj(name = "u", coerce = TRUE)`. This command will assign an R variable `u`, to be drawn from the Uniform(0, 1E6) distribution. As demonstrated in Fig. 1, each time `getRevObj` is called, a new draw is performed, allowing the user to continually pull new values from Rev into an R session. Because RevBayes output is returned to R in character format by default, the argument `coerce = TRUE` is required to transform the value of 'u' into a more usable numeric format. 'u' can then be used to generate a gamma distribution in R, that can be plotted as a histogram like so:

```
> draws <- rgamma(1000, shape = u, rate = u)
> hist(draws, xlab = "Value")
```

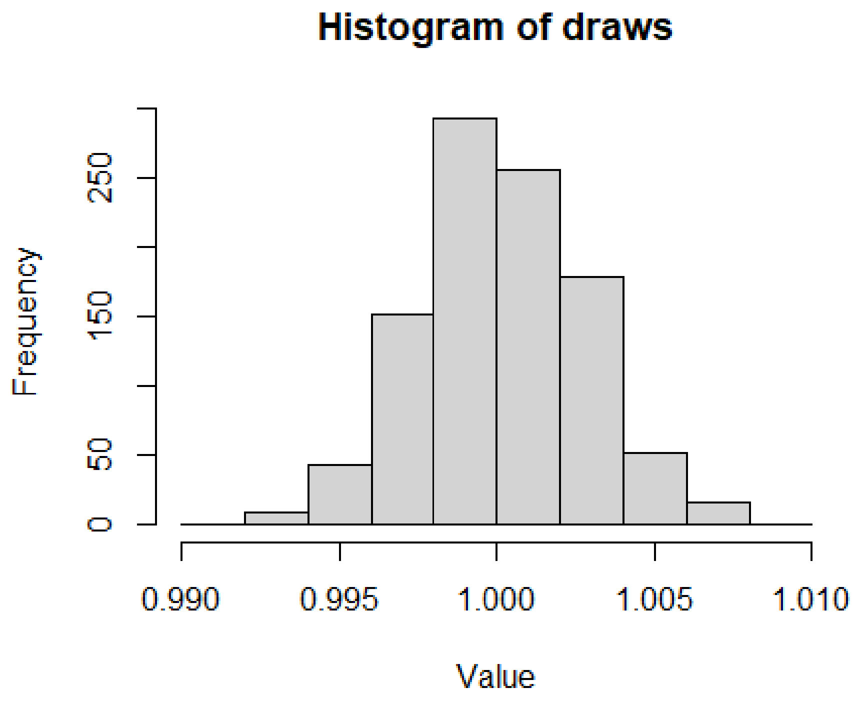


Figure 3: A distribution made by drawing values from the a Gamma distribution with shape and rate u, in which u is a value simulated from a Uniform distribution in RevBayes and passed to R via the `getRevObj` function in Revticulate. Value u was drawn from a distribution in Fig. 1.

To empty the Rev language history, use the function `clearRev()`. This function can be called during or out of a `repRev()` session.

## 4.3 Use of RevBayes in KnitR

In addition to command line simulation, another possible use of the 'Revticulate' package is integration with the package 'knitr'. RevBayes lacks inherent code visualization capabilities, but knitr provides a smooth and convenient format for generating markdown documents. A RevBayes engine for knitr can be created with the function `knitRev()`, and can then be used by changing the language name in the knitr chunk headers to 'rb'. When knitting a document, knitr chunks are individually interpreted in separate R sessions. Because of this behavior, the functions `knitRev()` and `initRev()` should be placed in the knitr setup chunk to ensure each Rev language chunk accesses the same `.Revhistory` file. This practice allows Rev variables defined in one chunk to be referenced in other chunks, a feature not present in many other knitr engines. This inter-chunk accession allows for Rev language chunk output to be accessed in R language chunks via Revticulate functions, allowing for clean and seamless inter-language document creation. In the example below, the variable `myTree` is created using Rev language, and is coerced into a `phylo` object with the function `getRevObj()`. The code `‘‘‘{rb}` indicates to knitr that this code should be interpreted via the Rev language kernel. The code `‘‘‘{r}` indicates to knitr that this code should be interpreted via the standard R language kernel.

```
‘‘‘{rb}
tips <- 2^4
myTree <- simTree(tips)
‘‘‘
```

Note that Rev and R cannot be used in the same code chunk. This is due to the structure of Knitr, in which there may only be one language per code chunk.

```
‘‘‘{r}
thisTree <- getRevObj("myTree", coerce = TRUE)
phytools::plot(thisTree)
‘‘‘
```

### 4.3.1 pkgdown, blogdown, and automating tutorial service

A considerable strength of the R environment is the ability to prepare and manage documents using KnitR (preparation of reports), pkgdown (generation of websites for R packages), and blogdown (generation of weblogs without associated packages). This set of tools makes generating and updating tutorials and associated instructional materials rapid and reproducible. Revticulate contains a KnitR kernel for the Rev language, enabling real-time execution of code in RMarkdown documents. This allows instructors and developers to show both syntax and expected outputs in a document. pkgdown websites have an articles menu for the service of vignettes, but this can also be used to manage tutorials for a class or workshop. Likewise, blogdown enables course and workshop content to be served as a continuous list of blog articles rendered from R Markdown format. Either of these packages make the process of providing

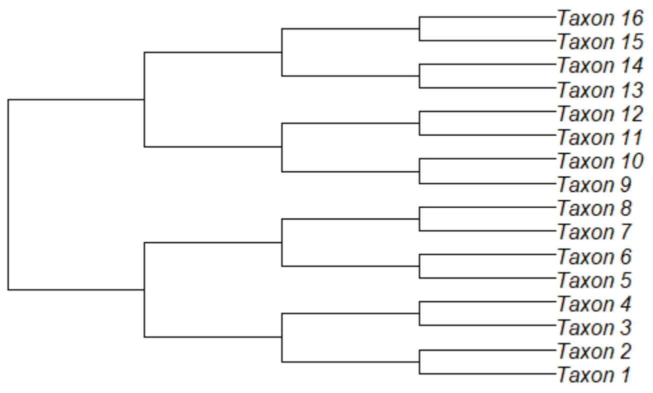


Figure 4: A tree simulated in R, using RevBayes' `simTree` function. This tree is then imported into R using the `getRevObj()` function and plotted via phytools (Revell, 2012). Trees are one of a number of default phylogenetic objects that can be passed between RevBayes and R.

educational content in Rev very simple. Markdown documents form the basis of pkgdown and blogdown websites, enabling developers and instructors to serve websites for free via services such as GitHub. An example of a workshop website generated using Revticulate and pkgdown can be seen at https://dwbapst.github.io/PaleoSoc_phylo_short_course_2019/index.html and in Fig. 1.

## 5 Conclusion

Phylogenetic methods have become ever more complex in recent decades, and along with this boon of techniques has become an expanding pool of technologies to implement them. One of these technologies, RevBayes, provides a very powerful and flexible interface for custom statistical analysis. However, is usage requires the user to learn a new, R-like programming language, a daunting task for many biologists with limited programming backgrounds. Common phylogenetic methods also involve developing advanced statistical intuition by researchers who may have little or no formal training in statistics. To assist researchers in developing this intuition, and to increase the interoperability of RevBayes with other common phylogenetics tools, we have developed the R package Revticulate, which serves as an R-language interface for RevBayes. Written entirely in R to allow for operation of RevBayes in R and RStudio, Revticulate implements a number of flexible functions to help both researchers and educators make use of RevBayes.

## 6 Author Contributions

Revticulate was written mostly by CPC with assistance from AMW. AMW and CPC prepared the manuscript together.

## References

JJ Allaire, Kevin Ushey, Yuan Tang, Dirk Eddelbuettel, Bryan Lewis, and Marcus Geelnard. reticulate: Interface to'python'. *R package version*, 1(8), 2018.

J. Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, 39(4):783–791, 1985. doi: 10.1111/j.1558-5646.1985.tb00420.x.

Tracy A Heath, John P Huelsenbeck, and Tanja Stadler. The fossilized birth-death process for coherent calibration of divergence-time estimates. *Proceedings of the National Academy of Sciences*, 111(29):E2957–E2966, 2014. doi: 10.1073/pnas.1319091111.

Jim Hester, Gábor Csárdi, Hadley Wickham, Winston Chang, Martin Morgan, and Dan Tenenbaum. remotes: R package installation from remote repositories, including 'github'. *R package version*, 2(1), 2020.

Sebastian Höhna, Tracy A. Heath, Bastien Boussau, Michael J. Landis, Fredrik Ronquist, and John P. Huelsenbeck. Probabilistic graphical model representation in phylogenetics. *Systematic Biology*, 63(5):753–771, 2014. doi: 10.1093/sysbio/syu039.

Sebastian Höhna, Michael J. Landis, Tracy A. Heath, Bastien Boussau, Nicolas Lartillot, Brian R. Moore, John P. Huelsenbeck, and Fredrik Ronquist. RevBayes: Bayesian phylogenetic inference using graphical models and an interactive model-specification language. *Systematic Biology*, 65(4):726–736, 2016. doi: 10.1093/sysbio/syw021.

Michael J Landis, William A Freyman, and Bruce G Baldwin. Retracing the hawaiian silversword radiation despite phylogenetic, biogeographic, and paleogeographic uncertainty. *bioRxiv*, page 301887, 2018. doi: 10.1101/301887.

Bui Quang Minh, Heiko A Schmidt, Olga Chernomor, Dominik Schrempf, Michael D Woodhams, Arndt Von Haeseler, and Robert Lanfear. Iq-tree 2: New models and efficient methods for phylogenetic inference in the genomic era. *Molecular biology and evolution*, 37(5):1530–1534, 2020.

Emmanuel Paradis, Julien Claude, and Korbinian Strimmer. Ape: analyses of phylogenetics and evolution in r language. *Bioinformatics*, 20(2):289–290, 2004.

Daniel L Rabosky. Extinction rates should not be estimated from molecular phylogenies. *Evolution: International Journal of Organic Evolution*, 64(6): 1816–1824, 2010.

Liam J Revell. phytools: an r package for phylogenetic comparative biology (and other things). *Methods in ecology and evolution*, 3(2):217–223, 2012.

RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA, 2015. URL http://www.rstudio.com/.

Graham J. Slater, Luke J. Harmon, and Michael E. Alfaro. Integrating fossils with molecular phylogenies improves inference of trait evolution. *Evolution*, 66(12):3931–3944, 2012. doi: 10.1111/j.1558-5646.2012.01723.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1558-5646.2012.01723.x.

T. Stadler. Sampling-through-time in birth-death trees. *Journal of Theoretical Biology*, 267(3):396–404, 2010. doi: 10.1016/j.jtbi.2010.09.010.

Alexandros Stamatakis. Raxml version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

Josef C Uyeda, Rosana Zenil-Ferguson, and Matthew W Pennell. Rethinking phylogenetic comparative methods. *Systematic Biology*, 67(6):1091–1109, 04 2018. ISSN 1063-5157. doi: 10.1093/sysbio/syy031. URL https://doi.org/10.1093/sysbio/syy031.

Hadley Wickham and Jay Hesselberth. Pkgdown: Make static html documentation for a package, 2018. *URL https://CRAN. R-project. org/package= pkgdown. R package version*, 1(0):560.

Yihui Xie. knitr: A general-purpose tool for dynamic report generation in r. *R package version*, 1(1), 2013.

Yihui Xie, Amber Thomas, and Alison Presmanes Hill. *Blogdown: creating websites with R markdown.* Chapman and Hall/CRC, 2017.

Derrick Joel Zwickl. *Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion.* PhD thesis, 2006.