Getting started with the SPLITT library

Venelin Mitov

SPLITT is a one-header C++ library.

```
knitr::include_graphics("figures/UmlDiagram8.pdf")
```

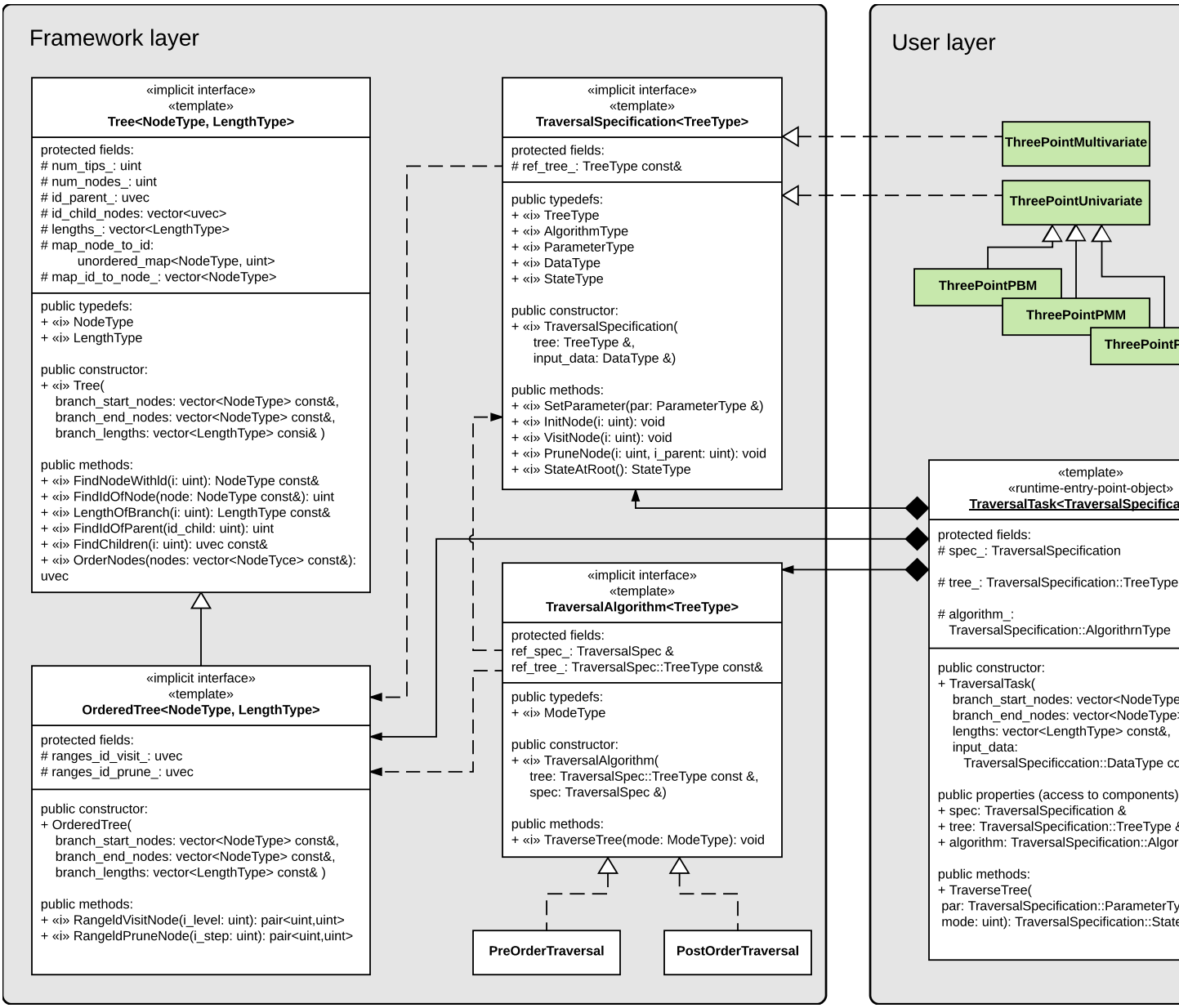
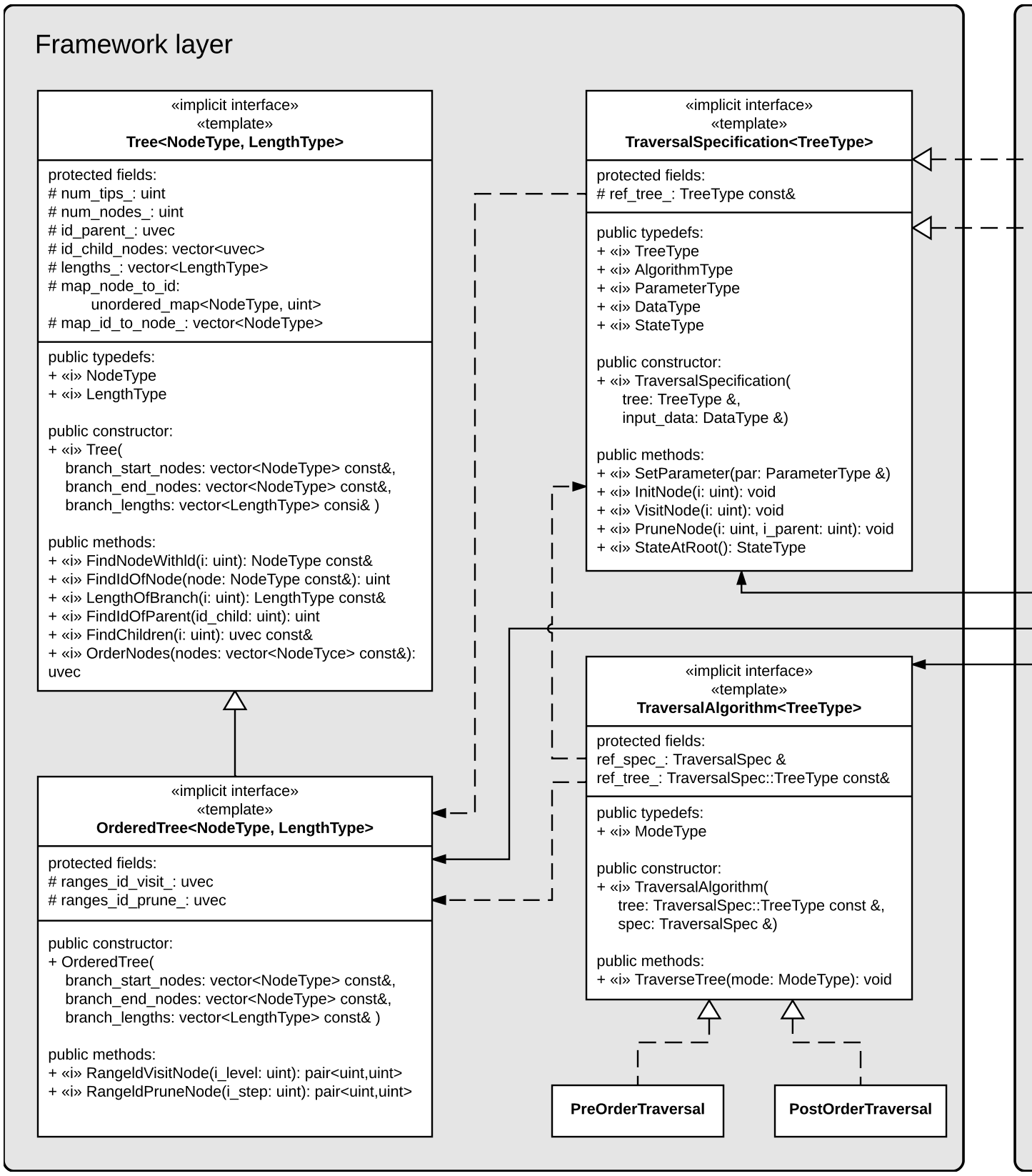

Framework layer
User layer
«implicit interface»
«template»
Tree<NodeType, LengthType>
protected fields:
num_tips_: uint
num_nodes_: uint
id_parent_: uvec
id_child_nodes: vector<uvec>
lengths_: vector<LengthType>
map_node_to_id:
unordered_map<NodeType, uint>
map_id_to_node_: vector<NodeType>
public typedefs:
+ «i» NodeType
+ «i» LengthType
public constructor:
+ «i» Tree(
branch_start_nodes: vector<NodeType> const&,
branch_end_nodes: vector<NodeType> const&,
branch_lengths: vector<LengthType> consi&)
public methods:
+ «i» FindNodeWithId(i: uint): NodeType const&
+ «i» FindIdOfNode(node: NodeType const&): uint
+ «i» LengthOfBranch(i: uint): LengthType const&
+ «i» FindIdOfParent(id_child: uint): uint
+ «i» FindChildren(i: uint): uvec const&
+ «i» OrderNodes(nodes: vector<NodeTyce> const&):
uvec
«implicit interface»
«template»
TraversalSpecification<TreeType>
protected fields:
ref_tree_: TreeType const&
public typedefs:
+ «i» TreeType
+ «i» AlgorithmType
+ «i» ParameterType
+ «i» DataType
+ «i» StateType
public constructor:
+ «i» TraversalSpecification(
tree: TreeType &,
input_data: DataType &)
public methods:
+ «i» SetParameter(par: ParameterType &)
+ «i» InitNode(i: uint): void
+ «i» VisitNode(i: uint): void
+ «i» PruneNode(i: uint, i_parent: uint): void
+ «i» StateAtRoot(): StateType
«implicit interface»
«template»
TraversalAlgorithm<TreeType>
protected fields:
ref_spec_: TraversalSpec &
ref_tree_: TraversalSpec::TreeType const&
public typedefs:
+ «i» ModeType
public constructor:
+ «i» TraversalAlgorithm(
tree: TraversalSpec::TreeType const &,
spec: TraversalSpec &)
public methods:
+ «i» TraverseTree(mode: ModeType): void
«implicit interface»
«template»
OrderedTree<NodeType, LengthType>
protected fields:
ranges_id_visit_: uvec
ranges_id_prune_: uvec
public constructor:
+ OrderedTree(
branch_start_nodes: vector<NodeType> const&,
branch_end_nodes: vector<NodeType> const&,
branch_lengths: vector<LengthType> const&)
public methods:
+ «i» RangeIdVisitNode(i_level: uint): pair<uint,uint>
+ «i» RangeIdPruneNode(i_step: uint): pair<uint,uint>
PreOrderTraversal
PostOrderTraversal
ThreePointMultivariate
ThreePointUnivariate
ThreePointPBM
ThreePointPMM
«template»
«runtime-entry-point-object»
TraversalTask<TraversalSpecifica
protected fields:
spec_: TraversalSpecification
tree_: TraversalSpecification::TreeType
algorithm_:
TraversalSpecification::AlgorithrnType
public constructor:
+ TraversalTask(
branch_start_nodes: vector<NodeType
branch_end_nodes: vector<NodeType>
lengths: vector<LengthType> const&,
input_data:
TraversalSpecificcation::DataType co
public properties (access to components)
+ spec: TraversalSpecification &
+ tree: TraversalSpecification::TreeType &
+ algorithm: TraversalSpecification::Algor
public methods:
+ TraverseTree(
par: TraversalSpecification::ParameterTy
mode: uint): TraversalSpecification::State

Figure 1:



Here I show how to use **SPLITT** to estimate the broad-sense heritability, H^2, of a numeric pathogen trait. H^2 is defined as the proportion of the trait variance attributable to genotypic variation in a population of

hosts (Mitov and Stadler 2016; Mitov and Stadler 2018):

Packages used

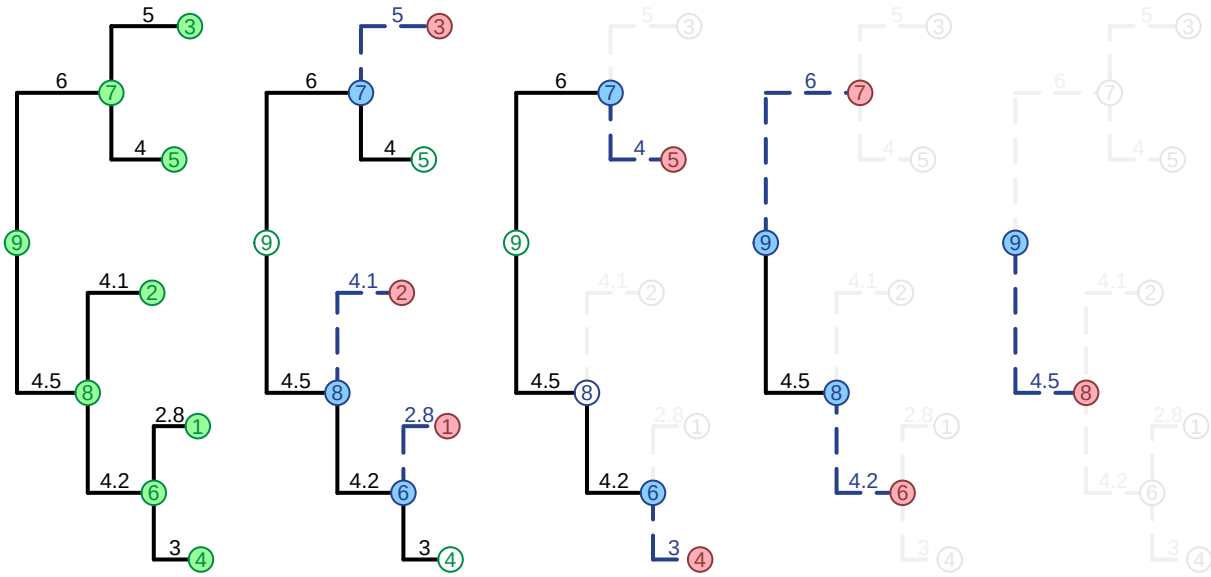
```
newick <- '(((1:3,2:2.8)6:4.2,3:4.1)7:4.5,(4:4,5:5)8:6)0;'
tree <- read.tree(text=newick)

x0 <- 0.1
sigma2 <- 0.25
sigmae2 <- 1

set.seed(1)
g <- rTraitCont(tree, model = "BM", root.value = x0,
                sigma = sqrt(sigma2),
                ancestor = FALSE)

x <- g + rnorm(n = length(tree$tip.label), mean = 0, sd = sqrt(sigmae2))

PMMLogLikCpp(x, tree, x0, sigma2, sigmae2, mode = 1)
```

```
## [1] -8.2
```

```
# output
# InitNode(1): x[i] t[i] a[i] b[i] c[i] : 1.9 2.8 -0.5 1.9 -2.72
# InitNode(2): x[i] t[i] a[i] b[i] c[i] : 0.96 4.1 -0.5 0.96 -1.38
# InitNode(3): x[i] t[i] a[i] b[i] c[i] : -0.74 5 -0.5 -0.74 -1.2
# InitNode(4): x[i] t[i] a[i] b[i] c[i] : 1.79 3 -0.5 1.79 -2.53
# InitNode(5): x[i] t[i] a[i] b[i] c[i] : 0.46 4 -0.5 0.46 -1.03
# InitNode(6): x[i] t[i] a[i] b[i] c[i] : nan 4.2 0 0 0
# InitNode(7): x[i] t[i] a[i] b[i] c[i] : nan 6 0 0 0
# InitNode(8): x[i] t[i] a[i] b[i] c[i] : nan 4.5 0 0 0
# InitNode(9): x[i] t[i] a[i] b[i] c[i] : nan 1.4822e-323 0 0 0
# VisitNode(1): x[i] t[i] a[i] b[i] c[i] : 1.9 2.8 -0.29 1.12 -2.25
# PruneNode(1, 6): x[i] t[i] a[j] b[j] c[j] : 1.9 2.8 -0.29 1.12 -2.25
# VisitNode(2): x[i] t[i] a[i] b[i] c[i] : 0.96 4.1 -0.25 0.47 -1.5
# PruneNode(2, 8): x[i] t[i] a[j] b[j] c[j] : 0.96 4.1 -0.25 0.47 -1.5
# VisitNode(3): x[i] t[i] a[i] b[i] c[i] : -0.74 5 -0.22 -0.33 -1.45
# PruneNode(3, 7): x[i] t[i] a[j] b[j] c[j] : -0.74 5 -0.22 -0.33 -1.45
# VisitNode(4): x[i] t[i] a[i] b[i] c[i] : 1.79 3 -0.29 1.03 -2.12
# PruneNode(4, 6): x[i] t[i] a[j] b[j] c[j] : 1.79 3 -0.58 2.14 -4.36
# VisitNode(5): x[i] t[i] a[i] b[i] c[i] : 0.46 4 -0.25 0.23 -1.32
# PruneNode(5, 7): x[i] t[i] a[j] b[j] c[j] : 0.46 4 -0.47 -0.1 -2.77
# VisitNode(6): x[i] t[i] a[i] b[i] c[i] : nan 4.2 -0.26 0.97 -3.68
# PruneNode(6, 8): x[i] t[i] a[j] b[j] c[j] : nan 4.2 -0.51 1.44 -5.18
# VisitNode(7): x[i] t[i] a[i] b[i] c[i] : nan 6 -0.2 -0.04 -3.2
# PruneNode(7, 9): x[i] t[i] a[j] b[j] c[j] : nan 6 -0.2 -0.04 -3.2
# VisitNode(8): x[i] t[i] a[i] b[i] c[i] : nan 4.5 -0.24 0.67 -5.01
# PruneNode(8, 9): x[i] t[i] a[j] b[j] c[j] : nan 4.5 -0.43 0.63 -8.22
```

```
par(mfrow=c(1,5))
par(mar=c(0,0,0,0))
par(oma=c(0, 0,0, 0))
```

```
PlotParallelTraversal(tree)
```




```
## C++ object <0x10a930ef0> of class 'SPLITT__RCPP__OrderedTreeStringNodes' <0x101f3a7f0>
```

```
#newick <- '(((1:4,2:4.5)11:4.3,3:4)10:1.4,(4:10.25,((5:4,6:3.8)14:4.2,7:4.1)13:4.5,(8:6,9:6)15:6)12:2.
newick <- '(((1:4,2:4.5)11:4.3,3:4)10:3.8,(((5:3,6:2.8)14:4.2,7:4.1)13:4.5,(8:5,9:5)15:6)12:4.2)0;'
tree <- read.tree(text=newick)

par(mfrow=c(1,5))
par(mar=c(0,0,0,0))

plotParallelPruningOrder(tree, TRUE)

fig.a <- recordPlot()

newick <- "(((((1:3.2,2:3.2)10:3.2,3:3.2)11:3.2,4:3.2)12:3.2,5:3.2)13:3.2,6:3.2)14;"
tree <- read.tree(text=newick)

par(mfrow=c(1,6))
par(mar=c(0,0,0,0))
plotParallelPruningOrder(tree, TRUE)

fig.b <- recordPlot()

cowplot::plot_grid(fig.a, fig.b, nrow=2, labels = c("a", "b"))
```

Apart from base R functionality, the patherit package uses a number of 3rd party R-packages:

- For tree processing: ape v5.1 (Paradis et al. 2016);
- For reporting: data.table v1.11.2 (Dowle and Srinivasan 2016);
- For the POUMM and PP estimates: POUMM v2.1.0 (Mitov and Stadler 2017), boot v1.3.20 (Canty and Ripley 2016);
- For testing: testthat v2.0.0 (Wickham 2016).

References

Canty, Angelo, and Brian Ripley. 2016. *Boot: Bootstrap R (S-Plus) Functions. R-Package.* https://cran.r-project.org/web/packages/boot/index.html.

Dowle, Matt, and Arun Srinivasan. 2016. *Data.table: Extension of 'Data.frame'.* https://CRAN.R-project.org/package=data.table.

Mitov, Venelin, and Tanja Stadler. 2016. "The heritability of pathogen traits - definitions and estimators." *Unpublished Data* https://www.biorxiv.org/content/early/2016/06/12/058503.

———. 2017. *Fast and Robust Inference of Phylogenetic Ornstein-Uhlenbeck Models Using Parallel Likelihood Calculation. BiorXiv.* http://biorxiv.org/content/early/2017/05/30/115089.

———. 2018. "A Practical Guide to Estimating the Heritability of Pathogen Traits." *Molecular Biology and Evolution*, msx328. doi:10.1093/molbev/msx328.

Paradis, Emmanuel, Simon Blomberg, Ben Bolker, Julien Claude, Hoa Sien Cuong, Richard Desper, Gilles Didier, et al. 2016. *Ape: Analyses of Phylogenetics and Evolution.* https://CRAN.R-project.org/package=ape.

Wickham, Hadley. 2016. *Testthat: Unit Testing for R.* https://CRAN.R-project.org/package=testthat.