RevBayes: Bayesian inference in phylogenetics using graphical models and a R-like language

Bastien Boussau
With massive borrowings from
Sebastian Hoehna
Tracy Heath
Michael Landis
Guillaume Kon Kam Kam King
What is RevBayes?

• Software for Bayesian statistical analyses
• Strong focus on phylogenetic models
• Strong focus on MCMC algorithms (Metropolis-Hastings, MCMC)
• C++ core for efficiency
• Interpreted R-like language for interactivity
• Built with probabilistic graphical models in mind
Useful pointers

- https://revbayes.github.io/
- http://revbayes.github.io/tutorials/
- https://github.com/revbayes/revbayes
- https://groups.google.com/forum/#!forum/revbayes-users
Graphical models in RevBayes

• Graphical models provide a simple way to represent probabilistic models

• They are also a powerful way to identify conditionally independent variables:
  – In RevBayes, objects are programmed in such a way that algorithms naturally benefit from conditional independence
Distributions and functions

Parameter ($M$)
Prior: $M \sim \text{Gamma}(\alpha, \beta)$

$\mu = \ln(M) - \frac{\sigma^2}{2}$

Parameter ($\sigma$)
Prior: $\sigma \sim \text{Exponential}(\lambda)$

Model: $x_i \sim \text{Lognormal}(\mu, \sigma)$
A probabilistic model is made of functions and distributions.

Parameter (\(M\))
Prior: \(M \sim \text{Gamma}(\alpha, \beta)\)

\[
\mu = \ln(M) - \frac{\sigma^2}{2}
\]

Parameter (\(\sigma\))
Prior: \(\sigma \sim \text{Exponential}(\lambda)\)

Parameter (\(x_i\))
Model: \(x_i \sim \text{Lognormal}(\mu, \sigma)\)
Graphical model conventions

\[ \mu = \ln(M) - \frac{\sigma^2}{2} \]

**Gamma distribution**

**Exponential distribution**

**Lognormal distribution**

- a) Constant node
- b) Stochastic node
- c) Deterministic node
- d) Clamped node (observed)
- e) Plate
- f) Tree plate
Using the Rev language to build a model

\[ \mu = \ln(M) - \frac{\sigma^2}{2} \]

Gamma distribution

Exponential distribution

Lognormal distribution

\[ i \in N \]
Using the Rev language to build a model

observations <- [your data go here]
Using the Rev language to build a model

\[ \alpha \quad \beta \]

observations <- [<your data go here>]

alpha <- 3.0
beta <- 1.0
Using the Rev language to build a model

\[
\begin{align*}
\alpha & \quad \beta \\
\Rightarrow & \\
M
\end{align*}
\]

\[
\text{observations} \leftarrow [\text{[your data go here]}]
\]

\[
\text{alpha} \leftarrow 3.0 \\
\text{beta} \leftarrow 1.0 \\
M \sim \text{dnGamma}(\text{alpha}, \text{beta})
\]
Using the Rev language to build a model

\[
\begin{align*}
\alpha & \rightarrow M \\
\beta & \rightarrow M \\
M & \rightarrow \lambda
\end{align*}
\]

observations <- [<your data go here>]

alpha <- 3.0
beta <- 1.0
M ~ dnGamma(alpha, beta)

lambda <- 1.0
Using the Rev language to build a model

observations <- [<your data go here>]

alpha <- 3.0
beta <- 1.0
M ~ dnGamma(alpha, beta)

lambda <- 1.0
sigma ~ dnExponential(lambda)
Using the Rev language to build a model

```
observations <- [<your data go here>]

alpha <- 3.0
beta <- 1.0
M ~ dnGamma(alpha, beta)

lambda <- 1.0
sigma ~ dnExponential(lambda)

mu := ln(M) - (power(sigma, 2.0) / 2.0)
```
Using the Rev language to build a model

\[
\begin{align*}
\text{observations} &\leftarrow [\text{<your data go here>}] \\
\text{alpha} &\leftarrow 3.0 \\
\text{beta} &\leftarrow 1.0 \\
M &\sim \text{dnGamma}(\text{alpha, beta}) \\
\lambda &\leftarrow 1.0 \\
\text{sigma} &\sim \text{dnExponential}(\lambda) \\
\mu &:= \ln(M) - \left(\text{power}(\text{sigma}, 2.0) / 2.0\right) \\
N &\leftarrow \text{observations.size()} \\
\text{for} (i \in 1:N) \{ \\
&\quad \text{x}[i] \sim \text{dnLnorm}(\mu, \text{sigma}) \\
\} 
\end{align*}
\]
Using the Rev language to build a model

\[
\begin{align*}
\text{observations} & \leftarrow [\text{<your data go here>}] \\
\alpha & \leftarrow 3.0 \\
\beta & \leftarrow 1.0 \\
M & \sim \text{dnGamma}(\alpha, \beta) \\
\lambda & \leftarrow 1.0 \\
\sigma & \sim \text{dnExp}(\lambda) \\
\mu & := \ln(M) - \left(\text{power}(\sigma, 2.0)/2.0\right) \\
N & \leftarrow \text{observations.size()} \\
\text{for}(i \in 1:N)\{ \\
& \quad x[i] \sim \text{dnLnorm}(\mu, \sigma) \\
& \quad x[i].\text{clamp}(\text{observations}[i]) 
\}
\end{align*}
\]
The Rev language

- R-like
- Type inference
- Object-oriented
- Completions
- Case-sensitive
- Math functions:
  - $\exp(1)$
  - $\ln(1)$
  - $\sqrt{16}$
  - $\text{power}(2, 2)$

- Distributions:
  - $\text{dexp}(x=1, \lambda=1)$
  - $\text{qexp}(0.5, 1)$
  - $\text{rexp}(n=10, 1)$
  - $\text{dnorm}(-2.0, 0.0, 1.0)$
  - $\text{rnorm}(n=10, 0, 1)$
The Rev language: useful functions

• Structure of a variable

```r
str(a)  # printing the structure information of 'a'
```

```
_data = a
_Revttype = Natural
_Revtypespec = [ Natural, Integer, RevObject ]
_value = 1
_dagtype = Constant DAG node
_children = [
.methods = void function()
```

• Type of a variable

```r
type(a)  # printing the type of 'a'
```

```
Natural
```

• Help: ?mean

• Working directory: getwd()

• What’s in my environment: ls()

• What commands are available? ls(all=TRUE)

• Sourcing a file: source("file")
Variable declaration in Rev

- 2 main types of variables:
  - Environment variable: name = « MyAnalysis »
  - Model variables:
    - Constant variable: c <- 1
    - Deterministic variable: d := exp(c)
    - Stochastic variable: x ~ dnExponential(c)

- Fun with stochastic variables:

  x
  x.probability()
  x.lnProbability()
  str(x)
The Rev language: final details (1)

• Vectors: `v <- v(1,2,3)` or: `w <- [1,2,3]` or: `z[1] <- 1
  z[3] <- 3`

• Convenience functions:
  ```
  1:10
  rep(10,1)
  seq(1,20,2)
  ```

• Vectors are objects: `v.methods()`

• Control structures:
  – for loops
  – while loops
  ```
  sum <- 0
  for (i in 1:100) {
    sum <- sum + i
  }
  sum
  ```
The Rev language: final details (2)

- User-defined functions:

```rev
function RealPos square ( Real x ) { x * x }
```

- User-defined functions can be recursive:

```rev
function Integer sum(Integer j) {
    if (j > 1) {
        return j + sum(j-1)
    } else {
        return 1
    }
}

c <- sum(100)
c
```
Bayesian phylogenetic inference

Homo sapiens; GeneA: ACTGGTGATGACATAAC...
Homo sapiens; GeneB: ACTGTTGATGACATGAC...
Mus musculus; GeneC: ACTGATGATGACAAGAC...
Mus musculus; GeneD: ACTGGTGA---CCATGAC...
Bison bison; GeneE: ACTGGTGATGACACGAC...
Canis lupus; GeneF: ACT---TCATGAAACGAC...
Bayesian phylogenetic inference

**Example of models:**

- Model of sequence data evolution: Markov model, all sites are independent
- Model of continuous trait: Brownian motion, or Ornstein Uhlenbeck, or Levy process
- Prior for the tree: Birth-death process
- ...
Bayesian phylogenetic inference

- We may be interested in the topology of the phylogenetic tree
- We may be interested in parameters associated to the branches or the nodes of the phylogenetic tree
- We may be interested in averaging out the uncertainty surrounding the phylogenetic tree to learn about traits at the leaves
- ...
Bayesian phylogenetic inference
Bayesian phylogenetic inference
Bayesian phylogenetic inference

- A phylogenetic tree is a type of graphical model
- Its structure can change during the MCMC: we need special moves to alter the topology of the tree
- It is also a parameter, and requires a prior distribution
- RevBayes includes many moves dedicated to phylogenetic inference, and many priors dedicated to phylogenetic objects
Distributions in RevBayes

dnBDBP
dnBDP
dnBernoulli
dnBeta
dnBimodalLognormal
dnBimodalNormal
dnBinom
dnBinomial
dnBirthDeath
dnBirthDeathBurstProcess
dnBirthDeathMultiRate
dnCBDSP
dnCDBDP
dnCDCladoBDP
dnCDFBDP
dnCDSSBDP
dnCat
dnCategorical
dnCauchy
dnCauchyPlus
dnChisq
dnCoalescent
dnCoalescentSkyline
dnCompleteBDP
dnCompleteBirthDeath
dnConstrainedNodeAge
dnConstrainedNodeOrder
dnConstrainedTopology
dnCppNormal
dnDPP
dnDecomposedInvWishart
dnDirichlet
dnDiversityDependentYule
dnDuplicationLoss
dnEBDP
dnEmpiricalSample
dnEmpiricalTree
dnEpisodicBirthDeath
dnEvent
dnExp
dnExponential
dnFBBDP
dnFBDRMatrix
dnFBDRP
dnFossilizedBirthDeathRange
dnGamma
dnGeom
dnGeometric
dnGilbertGraph
Moves in RevBayes

Move_DiscreteEventCategoryRandomWalk  Move_GraphShiftEdge  Move_NodeTimeSlideBeta
Move_ElementScale  Move_HSRFHyperpriorsGibbs  Move_NodeTimeSlidePathTruncatedNormal
Move_ElementSlide  Move_HSRFUnevenGridHyperpriorsGibbs  Move_NodeTimeSlideUniform
Move_ElementSwapSimplex  Move_IndependentTopology  Move_NodeTimeSlideUniformAge
Move_EllipticalSliceSamplingLognormalIID  Move_LayeredScaleProposal  Move_RandomGeometricWalk
Move_EllipticalSliceSamplingSimple  Move_LevyJump  Move_RandomIntegerWalk
Move_EmpiricalTree  Move_LevyJumpSum  Move_RateAgeBetaShift
Move_EventTimeBeta  Move_MatrixElementScale  Move_ReversibleJumpSwitch__Integer
Move_EventTimeSlide  Move_MatrixElementSlide  Move_ReversibleJumpSwitch__Natural
Move_FNPR  Move_MatrixRealSymmetricSlideMove  Move_ReversibleJumpSwitch__Probability
Move_GMRFHyperpriorGibbs  Move_MixtureAllocation__Integer  Move_ReversibleJumpSwitch__Real
Move_GammaScale  Move_MixtureAllocation__Natural  Move_ReversibleJumpSwitch__RealPos
Move_GibbsDrawCharacterHistory  Move_MixtureAllocation__Probability  Move_ReversibleJumpSwitch__Simplex
Move_GibbsMixtureAllocation__Integer  Move_MixtureAllocation__RateGenerator  Move_ReversibleJumpSwitch__Tree
Move_GibbsMixtureAllocation__Natural  Move_MixtureAllocation__Real  Move_ReversibleJumpSwitch__Tree
Move_GibbsMixtureAllocation__Probability  Move_MixtureAllocation__RealPos  Move_ReversibleJumpSwitch__Vector
Move_GibbsMixtureAllocation__RateGenerator  Move_MixtureAllocation__Simplex  Move_ReversibleJumpSwitch__Vector
Move_GibbsMixtureAllocation__Real  Move_MixtureAllocation__Tree  Move_ShrinkExpand
Move_GibbsMixtureAllocation__RealPos  Move_MultipleElementScale  Move_SingleElementScale
Move_GibbsMixtureAllocation__Simplex  Move_NarrowExchange  Move_SingleElementSlide
Move_GibbsPruneAndRegraft  Move_TipTimeSlideUniform  Move_SliceScampling
Move_GibbsReparameterization  Move_TreeScale  Move_Speciation"
Example: toxoplasmosis in boars (from Guillaume Kon Kam King)

- We model toxoplasmosis in boars as follows:

\[ i(a) = 1 - \exp((A - a) \times \alpha) \]

<table>
<thead>
<tr>
<th>Age</th>
<th>Infected</th>
<th>Total number</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8</td>
<td>13</td>
<td>131</td>
</tr>
<tr>
<td>6.0</td>
<td>17</td>
<td>93</td>
</tr>
<tr>
<td>7.7</td>
<td>24</td>
<td>82</td>
</tr>
<tr>
<td>9.7</td>
<td>32</td>
<td>108</td>
</tr>
<tr>
<td>11.5</td>
<td>24</td>
<td>93</td>
</tr>
<tr>
<td>13.8</td>
<td>13</td>
<td>60</td>
</tr>
<tr>
<td>17.6</td>
<td>26</td>
<td>88</td>
</tr>
<tr>
<td>20.6</td>
<td>30</td>
<td>102</td>
</tr>
<tr>
<td>25.7</td>
<td>38</td>
<td>82</td>
</tr>
<tr>
<td>51.9</td>
<td>30</td>
<td>93</td>
</tr>
</tbody>
</table>

\[ \text{Infected} \sim \mathcal{B} (\text{Total number}, i) \]

\[ A \sim \mathcal{U}(-100, 0) \]

\[ \log(\alpha) \sim \mathcal{U}(-4, -1) \]
Entering the data and setting up the model

# Setting up the data
ages<-v(4.8, 6, 7.7, 9.7, 11.5, 13.8, 17.6, 20.6, 25.7, 51.9)
infected <-v(13, 17, 24, 32, 24, 13, 26, 30, 38, 30)
total <- v(131, 93, 82, 108, 93, 60, 88, 102, 82, 93)

# Setting up the model
A~dnUniform(0,100)
lalpha~dnUniform(-4, -1)

# The model is replicated across age categories
for (i in 1:ages.size()) {
  intermediate[i] := Probability ( 1-exp( (-A -ages[i]) * (10^lalpha) ) )
  infectedV[i] ~dnBinomial(p=intermediate[i], n=total[i])
  infectedV[i].clamp(infected[i])
}

Need to convert from RealPos to Probability

... so that it works in there!
Preparing for inference

# Get a hang on the model (any node will do)
mymodel = model(A)

# Moves
moveIndex = 0
moves[moveIndex++] = mvSlide(A)
moves[moveIndex++] = mvSlide(lalpha)
moves[moveIndex++] = mvScale(A)

# Some monitors to see how the MCMC is going
myOutputFile = "boars.log"
monitors[1] = mnModel(filename=myOutputFile, printgen=10, separator=" ")
monitors[2] = mnScreen(printgen=10, A, lalpha)

# Automatic stopping rules when convergence has occurred or when too much time has passed
stopping_rules[1] = srMaxIteration(200000)
stopping_rules[2] = srMaxTime(15,"hours")
stopping_rules[4] = srGelmanRubin(1.01,myOutputFile,10000)
stopping_rules[5] = srGeweke(prob=0.001, file=myOutputFile,freq=10000)
stopping_rules[6] = srStationarity(prob=0.01, file=myOutputFile,freq=10000)
Performing inference

# Creating the MCMC object
mymcmc = mcmc(mymodel, monitors, moves,
    moveschedule="random", nruns=2)

# Alternatively we could create a MCMCMC object
# or mymcmc = mcmcmc(mymodel, monitors, moves,
#    moveschedule="random", nchains=4, nruns=1)

# Running the analysis: first some burnin...
mymcmc.burnin(generations=10000,200)

# Then the real thing
mymcmc.run(stopping_rules)
Convergence plots with coda
Changing the moves

moves[moveIndex++] = mvSlice(A)
moves[moveIndex++] = mvSlice(lalpha)
Convergence plots with coda

A

Trace of A

Density of A

Trace of lalpha

Density of lalpha
Combining moves

moveIndex = 0

moves[moveIndex++] = mvSlice(A)
moves[moveIndex++] = mvSlice(lalpha)
moves[moveIndex++] = mvSlide(A)
moves[moveIndex++] = mvSlide(lalpha)
moves[moveIndex++] = mvScale(A)
Convergence plots with coda
Combining moves and using MCMC MCMC

moveIndex = 0
moves[moveIndex++] = mvSlice(A)
moves[moveIndex++] = mvSlice(lalpha)
moves[moveIndex++] = mvSlide(A)
moves[moveIndex++] = mvSlide(lalpha)
moves[moveIndex++] = mvScale(A)
#

mymcmc = mcmcmc(mymodel, monitors, moves, moveschedule="random", nchains=4, nruns=2)
Convergence plots with coda
Comparison with Jags

A

Trace of A

Density of A

Trace of lalpha

Density of lalpha

Autocorrelation

Lag

Iterations

N = 1000  Bandwidth = 4.383

N = 1000  Bandwidth = 0.03682
Comparison with Stan

A

Trace of A

Density of A

Trace of Ialpha

Density of Ialpha
Comparison of the lag for various moves and vs Jags and Stan

Slide moves

Slice+Slide+Scale moves

Jags

Slice moves

Slice+Slide+Scale moves + MC³

Stan
Things I did not talk about

- RevBayes can compute marginal likelihoods for model comparison (stepping stone sampling, path sampling)
- RevBayes can handle mixture models
- RevBayes can handle infinite mixture models (Dirichlet process)
- RevBayes can be run on a cluster through MPI, with parallelisation by the data