



Introduction to RevBayes

Bastien Boussau

With massive borrowings from

Sebastian Hoehna

Tracy Heath

Michael Landis

What is RevBayes?

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

Useful pointers

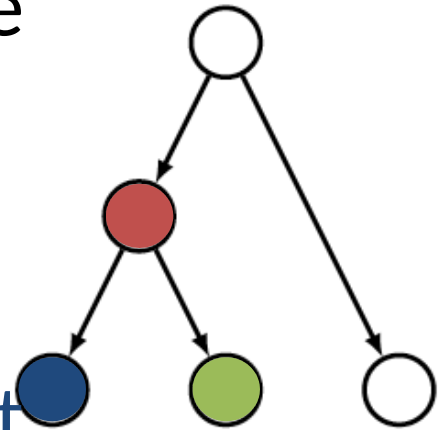
- <http://revbayes.github.io/intro.html>
- <http://revbayes.github.io/tutorials.html>
- <http://revbayes.github.io/installation.html>
- <https://github.com/revbayes>
- <https://groups.google.com/forum/#!forum/revbayes-users>

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



The Rev language

- R-like
- Type inference
- Object-oriented
- Completions
- Case-sensitive
- Math functions:

```
exp(1)
ln(1)
sqrt(16)
power(2,2)
```

- Distributions:

```
dexp(x=1,lambda=1)      # exponential distribution density function
qexp(0.5,1)             # exponential distribution quantile function
rexp(n=10,1)            # random draws from an exponential distribution
dnorm(-2.0,0.0,1.0)     # normal distribution density function
rnorm(n=10,0,1)         # random draws from a normal distribution
```

The Rev language: useful functions

- Structure of a variable :

```
str(a)                                # printing the structure information of 'a'
  _variable      = a
  _RevType       = Natural
  _RevTypeSpec   = [ Natural, Integer, RevObject ]
  _value         = 1
  _dagType       = Constant DAG node
  _children      = [ ]
  .methods       = void function ()
```

- Type of a variable :

```
type(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)

Let's explore the variable types

```
a<-1
```

```
b<-2
```

```
x~dnNormal(mean=a,sd=b)
```

```
print(x)
```

```
y:=x*x
```

```
z<-x*x
```

```
x.redraw();print("x: "+x);print("y: "+y);print("z: "+z)
```

```
x.redraw();print("x: "+x);print("y: "+y);print("z: "+z)
```

```
y.redraw()
```

```
z.redraw()
```


Let's explore the variable types

```
a<-1          Constant node
b<-2          Constant node
x~dnNormal(mean=a,sd=b)  Stochastic node
print(x)
y:=x*x        Deterministic node
z<-x*x        Constant node
x.redraw();print("x: "+x);print("y: "+y);print("z: "+z)
x.redraw();print("x: "+x);print("y: "+y);print("z: "+z)
y.redraw()
z.redraw()
```

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)`
 - ☒ ☐
 - More fun with stochastic variables:

```
x          # print value of stochastic node 'x'
x.probability() # print the probability if 'x'
x.lnProbability() # print the log-probability if 'x'
str(x)      # printing all the information of 'x'
```

The Rev language: more details (1)

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

- User-defined functions:

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

- User-defined functions can be recursive:

```
function Integer sum(Integer j) {  
  if (j > 1) {  
    return j + sum(j-1)  
  } else {  
    return 1  
  }  
}  
  
c <- sum(100)  
c
```

Anatomy of a phylogenetic MCMC analysis

How do we set up inference with MCMC for a simple phylogenetic model in RevBayes ?

Anatomy of a phylogenetic MCMC analysis

```
#####  
# Data #  
#####  
# We read the sequence alignment:  
data = readDiscreteCharacterData("...")  
n_branches = 2 * data.n taxa() - 3
```

Anatomy of a phylogenetic MCMC analysis

```
#####  
# Data #  
#####  
# We read the sequence alignment:  
data = readDiscreteCharacterData("...")  
n_branches = 2 * data.ntaxa() - 3  
  
#####  
# Model of sequence evolution #  
#####  
# Uniform prior on topologies  
topology ~ dnUniformTopology(...)  
  
# Exponential priors on branch lengths  
for (i in 1:n_branches) {  
  bls[i] ~ dnExponential(10)  
}  
# Putting branch lengths and topology together  
psi := treeAssembly(topology, bls)  
# We define a JC rate matrix:  
Q <- fnJC(4)  
# The sequences are drawn from a CTMC running  
along the tree  
seq ~ dnPhyloCTMC( tree=psi, Q=Q, type="DNA" )  
# We condition the CTMC on the sequence  
alignment.  
seq.clamp( data )  
# We declare the model as one big object that  
we are going to use in the MCMC:  
my_model = model(psi)
```

Anatomy of a phylogenetic MCMC analysis

```
#####  
# Data #  
#####  
# We read the sequence alignment:  
data = readDiscreteCharacterData("...")  
n_branches = 2 * data.ntaxa() - 3  
  
#####  
# Model of sequence evolution #  
#####  
# Uniform prior on topologies  
topology ~ dnUniformTopology(...)  
  
# Exponential priors on branch lengths  
for (i in 1:n_branches) {  
  bls[i] ~ dnExponential(10)  
}  
# Putting branch lengths and topology together  
psi := treeAssembly(topology, bls)  
# We define a JC rate matrix:  
Q <- fnJC(4)  
# The sequences are drawn from a CTMC running  
along the tree  
seq ~ dnPhyloCTMC( tree=psi, Q=Q, type="DNA" )  
# We condition the CTMC on the sequence  
alignment.  
seq.clamp( data )  
# We declare the model as one big object that  
we are going to use in the MCMC:  
my_model = model(psi)
```

```
#####  
# Moves #  
#####  
# We create a vector of moves to store them  
all:  
moves = VectorMoves()  
# Move on the topology  
moves.append(mvNNI(topology, weight=10.0))  
# We define moves on the branch lengths, one  
for each branch.  
for (i in 1:n_branches) {  
  moves.append(mvScaleBactrian(bls[i],  
tune=TRUE))  
}
```


Anatomy of a phylogenetic MCMC analysis

```
#####
# Data #
#####
# We read the sequence alignment:
data = readDiscreteCharacterData(...)
n_branches = 2 * data.n taxa() - 3

#####
# Model of sequence evolution #
#####
# Uniform prior on topologies
topology ~ dnUniformTopology(...)

# Exponential priors on branch lengths
for (i in 1:n_branches) {
  bls[i] ~ dnExponential(10)
}
# Putting branch lengths and topology together
psi := treeAssembly(topology, bls)
# We define a JC rate matrix:
Q <- fnJC(4)
# The sequences are drawn from a CTMC running
along the tree
seq ~ dnPhyloCTMC( tree=psi, Q=Q, type="DNA" )
# We condition the CTMC on the sequence
alignment.
seq.clamp( data )
# We declare the model as one big object that
we are going to use in the MCMC:
my_model = model(psi)
```

```
#####
# Moves #
#####
# We create a vector of moves to store them
all:
moves = VectorMoves()
# Move on the topology
moves.append(mvNNI(topology, weight=10.0))
# We define moves on the branch lengths, one
for each branch.
for (i in 1:n_branches) {
  moves.append(mvScaleBactrian(bls[i],
tune=TRUE))
}

#####
# MCMC analysis #
#####
# Now we define monitors to keep track of
what's happening during the MCMC.
# One monitor to store the parameter
distributions into a file:
monitors[1] = mnModel(filename="...",
printgen=10, separator = TAB)
# We create an MCMC object:
analysis = mcmc(my_model, monitors, moves,
...)
# We run the MCMC for 20,000 iterations:
analysis.run(20000)
```

Anatomy of a phylogenetic MCMC analysis

DATA

```
#####  
# Data #  
#####  
# We read the data  
data = read...")  
n_branches = 2 * data.ntaxa() - 3
```

```
#####  
# Model of sequence evolution #  
#####  
# Uniform prior on topologies  
topology ~ dnUniformTopology(...)
```

```
# Exponential priors on branch lengths  
for (i in 1:n_branches) {  
  bls[i] ~ dnExponential(10)  
}  
# Putting branch lengths and topology together  
psi := treeAssembly(topology, bls)  
# We define a JC rate matrix:  
Q <- fnJC(  
# The sequence is running  
along the  
seq ~ dnPhylo("DNA" )  
# We condition the tree on the sequence  
alignment.  
seq.clamp( data )  
# We declare the model as one big object that  
we are going to use in the MCMC:  
my_model = model(psi)
```

MODEL

MOVES

```
#####  
# Moves #  
#####  
# We create them  
all:  
moves = VectorMoves()  
# Move on the topology  
moves.append(mvNNI(topology, weight=10.0))  
# We define moves on the branch lengths, one  
for each branch.  
for (i in 1:n_branches) {  
  moves.append(mvScaleBactrian(bls[i],  
tune=TRUE))  
}
```

```
#####  
# MCMC analysis #  
#####  
# Now we define monitors to keep track of  
what's happening  
# One monitor  
distributions  
monitors[...]  
printgen=10, separator = TAB)  
# We create an MCMC object:  
analysis = mcmc(my_model, monitors, moves,  
...)  
# We run the MCMC for 20,000 iterations:  
analysis.run(20000)
```

MCMC

Advice on organizing an analysis

- create a folder for the analysis
- put data in a folder « data »
- put the scripts in a folder « scripts »
- store the output files in « analyses » or « output »
- to run an analysis :
 `rb scripts/myscript.rev`
- or, from within rb :
 `source("scripts/myscript.rev")`