

RevBayes: Introduction and Demonstration of the basic elements

**SEBASTIAN HÖHNA, JOHN HUELSENBECK AND
FREDRIK RONQUIST**

**DEPARTMENT OF MATHEMATICS, STOCKHOLM UNIVERSITY
DEPARTMENT OF INTEGRATIVE BIOLOGY, UC BERKELEY
DEPARTMENT OF BIODIVERSITY INFORMATICS, SWEDISH MUSEUM OF
NATURAL HISTORY**

Availability



- Open Source (Sourceforge):
 - <http://sourceforge.net/projects/revbayes/>
 - or <http://www.revbayes.net>
- Download:
 - <http://sourceforge.net/projects/revbayes/files/>

Basic Operations & Commands



- RevBayes:
 - is an interactive environment
 - basic syntax is inspired by 'R'
 - provides standard 'easy-to-use' math-functions

Basic Operations



Simple mathematical operators

$1 + 1$

Addition

$10 - 5$

Subtraction

$5 * 5$

Multiplication

$10 / 2$

Division

2^3

Exponentiation

Basic Functions



Math-Functions

`expf(1)`

exponential function

`ln(1)`

natural logarithm

`sqrt(16)`

square root

`power(2,2)`

exponentiation (=a^b)

Basic Functions

distribution functions

dexp(1,1)

pexp(1,1)

qexp(0.5,1)

rexp(1)
distr

pdf of exponential distr

cdf of exponential distr

quantile of exponential distr

random draw from exponential

dnorm(0,0,1)

pnorm(0.5,0,1)

qnorm(0.95,0,1)

rnorm(0,1)

pdf of normal distr

cdf of normal distr

quantile of normal distr

random draw from normal distr

Function Arguments



- Functions in RevBayes:
 - have arguments with
 - label
 - type
 - return type
- Labeling:
 - `rnorm(mean=0,sd=1)` # fully specified
 - `rnorm(sd=1,mean=0)` # any order
 - `rnorm(m=0,s=1)` # partial label
 - `rnorm(0,1)` # without label, but in order
- Type matching:
 - automatic test if type is equal or of derived type
 - e.g. `rnorm(mean=1.0,sd=1)` expects 'mean' of type 'Real' but get 'Real+'
 - automatic type conversion, if implement
 - e.g. `rnorm(mean=1.0,sd=1)` expects 'sd' of type 'Real+' but gets 'Natural'

Variables



- RevBayes contains 3 types of variables:
 - constant variables (values)
 - deterministic variables (associated with function and parameters)
 - stochastic variables (associated with distribution and parameters)
- RevBayes uses call by reference
 - only constant variables hold a permanent copy

Variables

Variable assignment: constant and deterministic

a <- 1 # assign value '1' to const variable

b := expf(a) # assign exponential function with parameter 'a' (by reference) to deterministic variable

c := ln(b) # assign logarithm function with parameter 'b' to deterministic variable

d <- ln(b) # assign value of logarithm function to const variable

e := c == d # assign '==' function to deterministic variable

a <- 2 # reassignment of 'a' to const variable with value '2'

b <- 1 # reassignment of 'b' to const variable with value '1'

Variables



Variable assignment: stochastic

```
lambda <- 1
```

```
# lambda with const value
```

```
x ~ exp(lambda)
```

```
exponential distribution and parameter 'lambda'
```

```
# assignment of stochastic variable 'x' with
```

```
exponential distribution and parameter 'lambda'
```

```
mu <- 0
```

```
# assignment of const variable 'mu'
```

```
sigma <- 1
```

```
# assignment of const variable 'sigma'
```

```
y ~ norm(mu,sigma)
```

```
normal distribution and parameters 'mu' and 'sigma'
```

```
# assignment of stochastic variable 'y' with
```

```
normal distribution and parameters 'mu' and 'sigma'
```

- Basic types:
 - Boolean, Integer, Real, Natural, Real+, String, Probability
 - `a <- 1`
- Vector types:
 - VectorBoolean, VectorInteger, VectorReal, Simplex, ...
 - `a <- v(1,2,3,4)`
 - `b[1] <- 1` # implicit creation of vector
 - `b[2] <- 2` # vector implicitly resizes
 - `c[1] <- b` # implicit creation of matrix (a vector of a vector)

Loops



- For-loop:

- for (*variable in sequence*) *statement(s)*

```
# loops
```

```
sum <- 0
```

```
for (i in 1:100) {
```

```
  sum <- sum + i
```

```
}
```

```
# create const variable with value '0'
```

```
# loop from 1 to 100 (R-style)
```

```
# add the current value of 'i' to the sum
```

```
# Fibonacci series
```

```
fib[1] <- 1
```

```
fib[2] <- 1
```

```
for (j in 3:10) {
```

```
  fib[j] <- fib[j - 1] + fib[j - 2]
```

```
}
```

```
# implicit creation of array/list with 1 element
```

```
# iterate over the sequence 3 to 10
```

- While-loop:

- **while** (*condition*) *statement(s)*

```
a <- 10
```

```
while (a > 0) {
```

```
  a <- a - 1
```

```
  a
```

```
}
```

If-Condition

```
# if-else condition
```

```
a <- 1
```

```
if (a == 1) {
```

```
  res <- "true"
```

```
} else {
```

```
  res <- "false"
```

```
}
```

User defined functions



#User function

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

```
function Natural fac(i) {
```

```
  if (i > 1) {
```

```
    return i * fac(i - 1)
```

```
  } else {
```

```
    return 1
```

```
  }
```

```
}
```

function with specified return type

Command functions



```
lambda <- 1
```

```
x ~ exp(lambda)
```

```
# built-in functions
```

- `structure(x)` # printing the structure and info about the variable 'x'
- `ls(all=true)` # list the content of the workspace
- `clear()` # clear the content of the workspace
- `quit()` # quit RevBayes (you never should do that)

Help System

- Help system provides information about functions:
 - General information
 - Arguments
 - Usage
 - Usage Examples
 - Author
 - References
- E.g.: ?mcmc

Normal Model



```
# Test file for mcmc on normal distribution
a <- -1.0
b <- 1.0
mu ~ unif(a, b)
moves[1] <- mslide(mu,weight=1.0,delta=1.0)

c <- 0
d <- 1.0
e ~ norm(c,d)
moves[2] <- mslide(e,weight=1.0,delta=1.0)
sigma := expf(e)

filemonitors <- monitor(filename="normaltest.out",printgen=1,mu,e,sigma)

for (i in 1:2) {
  x[i] ~ norm(mu, sigma)
  clamp(x[i], 0.5)
}

mymodel <- model(a)

mymcmc <- mcmc(mymodel, moves=moves, monitors=filemonitors)

mymcmc.run(generations=100)
```

Simple(st) Phylogenetic Model



```
# Phylogenetic model test

# Read data from Nexus file
D <- read("data/primates.nex")

# Create random variable drawn from a uniform topology distribution
tau ~ unifTopology( numberTaxa=D.ntaxa(), tipNames=D.names(), isRooted=false, isBinary=true )

# Create the tree plate that is ordered by the topology
tree <- treeplate(tau)

# create the priors for the parameters of the substitution model

# pi ~ dirichlet(4)
pi_prior <- v(1,1,1,1)
pi ~ dirichlet(pi_prior)
# r ~ dirichlet(6)
r_prior <- v(1,1,1,1,1,1)
r ~ dirichlet(r_prior)

# create the rate matrix for the substitution model
Q := gtr(rates=r, freqs=pi)

dummyState <- D[1][1]

lambda <- 1
```

Simple(st) Phylogenetic Model



Now we can specify the model by using the tree functions for generating node or branch
indices. In trees, T.ancestor(i) would simply give the ancestral node index.

```
nodeCount <- tree.nnodes()
charCount <- D.nchar()

for (i in 1:nodeCount) {
  for (j in 1:charCount) {
    node <- tree.node(i)
    if (node.isRoot()) {
      dna_state[i][j] ~ cat(Q.stationaryfreqs(), dummyState)
    } else {
      if (node.isTip()) {
        parent <- node.ancestor()
        parentIndex <- tree.index(parent)
        bl[i] ~ exp(lambda)
        dna_state[i][j] ~ ctmc(Q, bl[i], dna_state[parentIndex][j])
      } else {
        parent <- node.ancestor()
        parentIndex <- tree.index(parent)
        bl[i] ~ exp(lambda)
        dna_state[i][j] ~ ctmc(Q, bl[i], dna_state[parentIndex][j])
      }
    }
  }
}
```