# 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 King

# What is RevBayes?

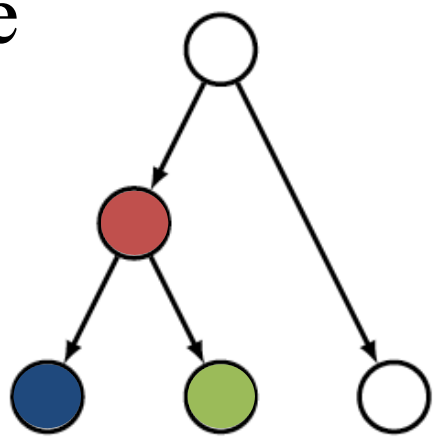
- Software for Bayesian statistical analyses
- Strong focus on phylogenetic models
- Strong focus on MCMC algorithms (Metropolis-Hastings, MCMCMC)
- C++ core for efficiency
- Interpreted R-like language for interactivity
- Built with probabilistic graphical models in mind

# Useful pointers

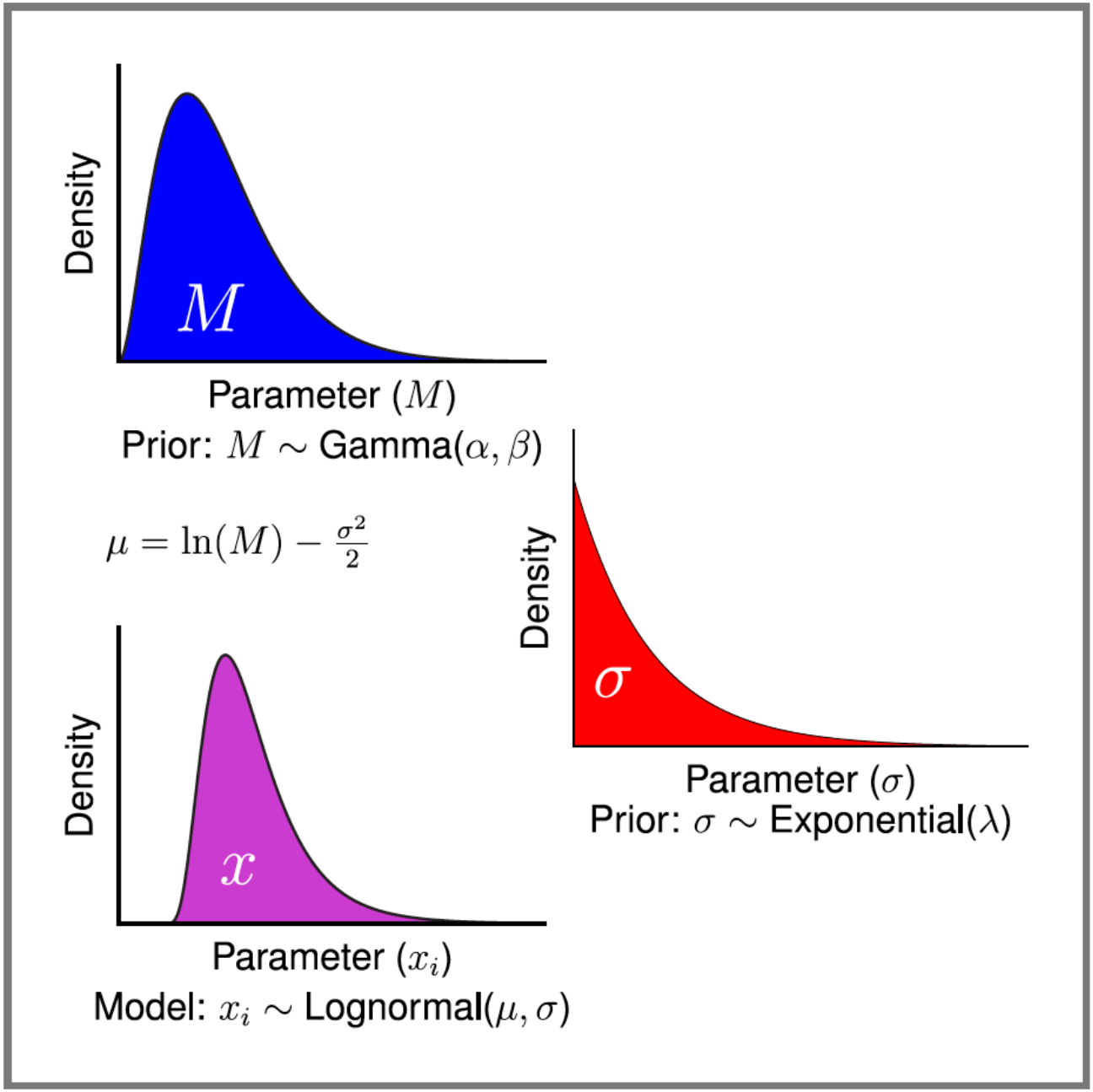
- https://revbayes.github.io/
- http://revbayes.github.io/tutorials/
- https://github.com/revbayes/revbayes
- https://groups.google.com/forum/#!forum/revbayes-users

# Graphical models in RevBayes

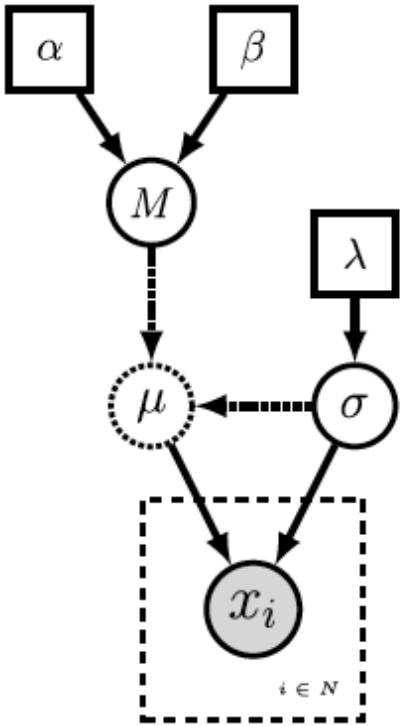
- Graphical models provide a simple way to represent probabilistic models
- They are also a powerful way to identify conditionally independent variables:
  - In RevBayes, objects are programmed in such a way that algorithms naturally benefit from conditional indepence

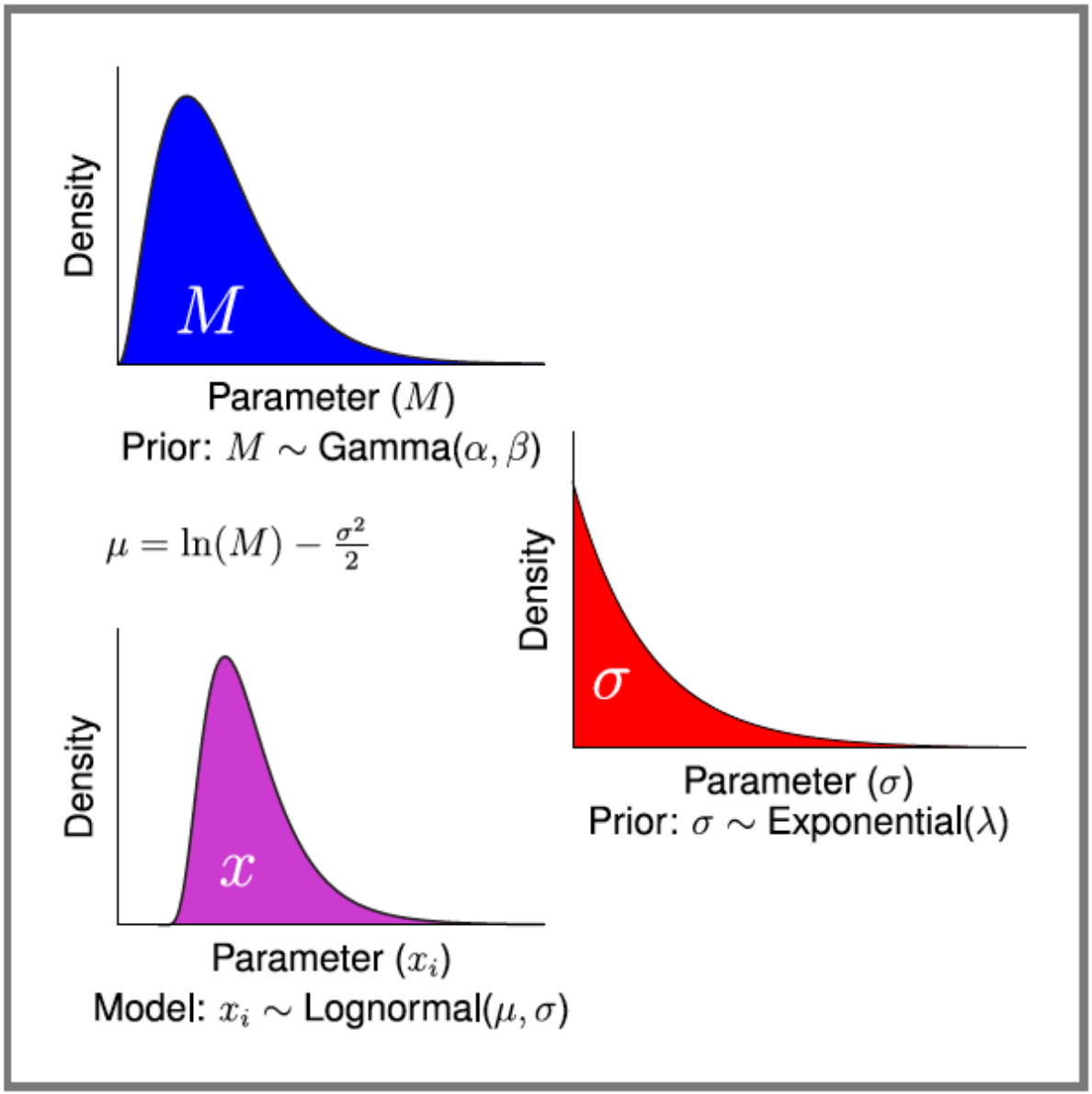


# Distributions and functions

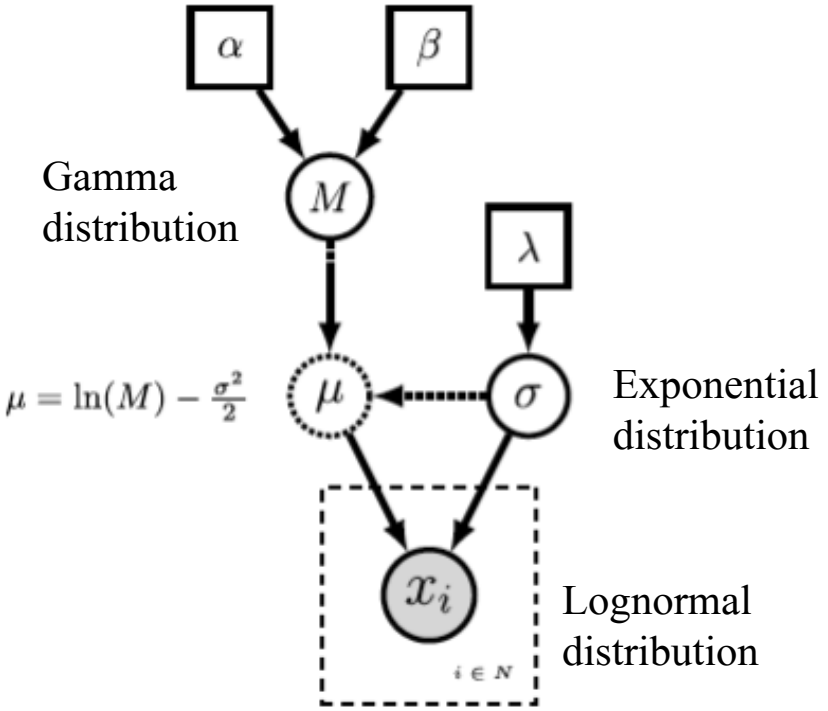


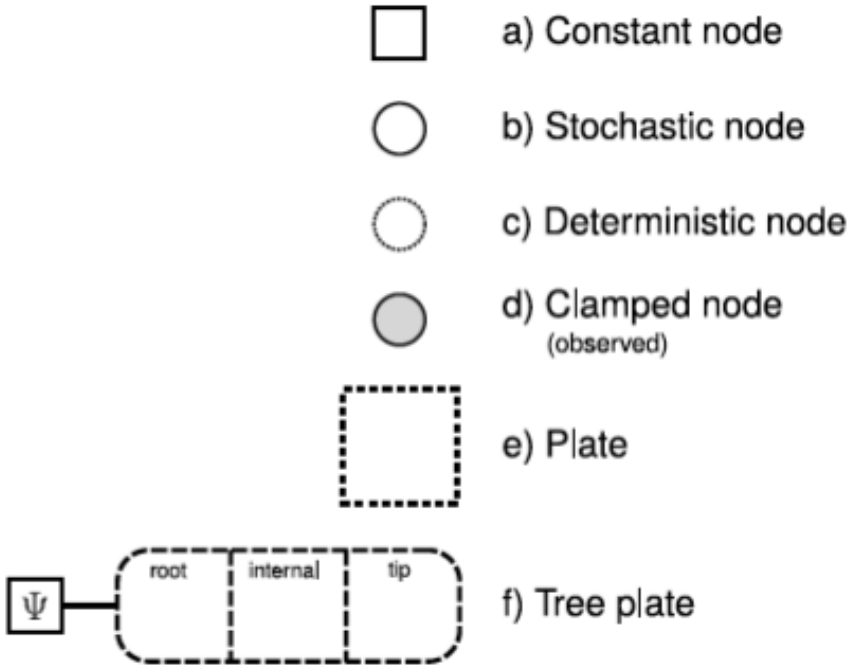
# A probabilistic model is made of functions and distributions



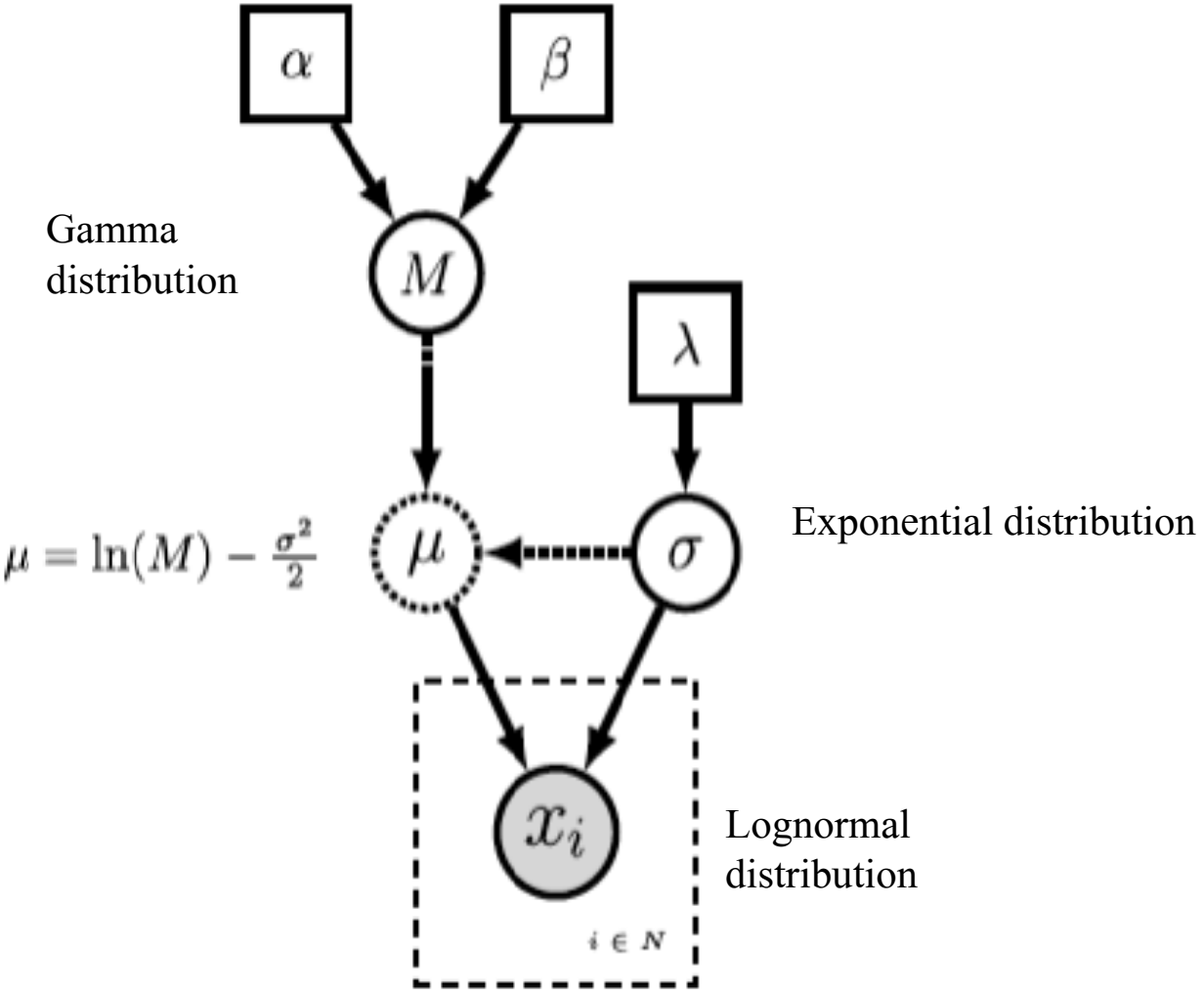



# Graphical model conventions

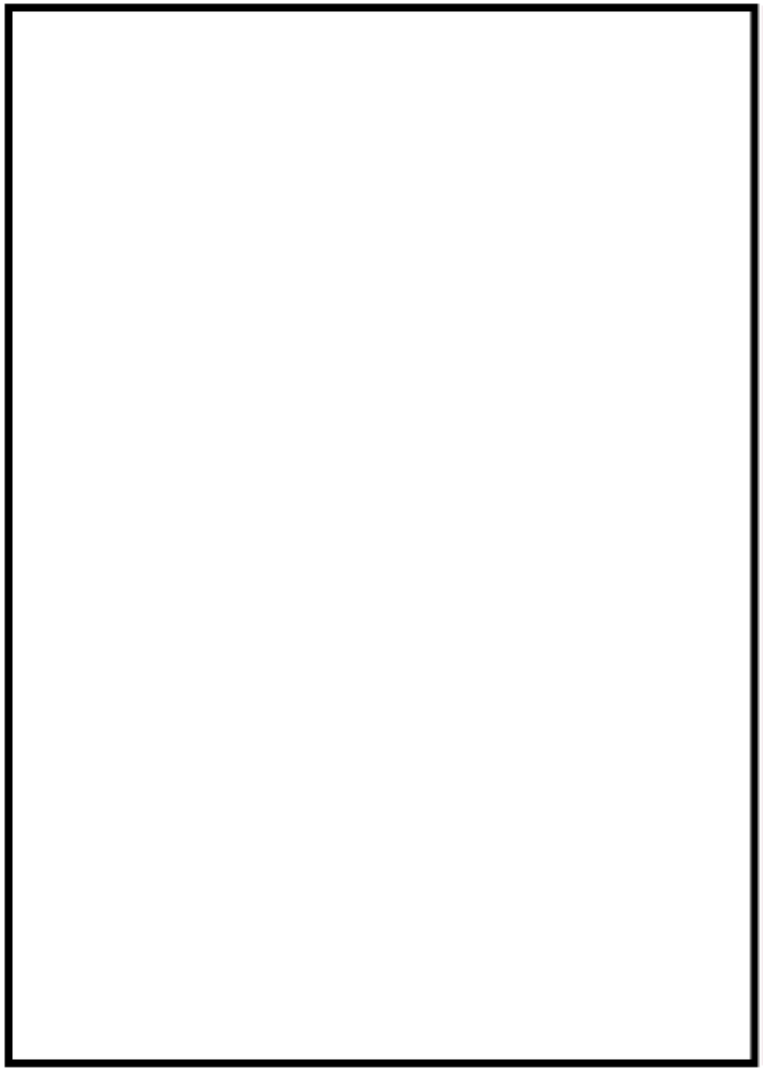


# Using the Rev language to build a model

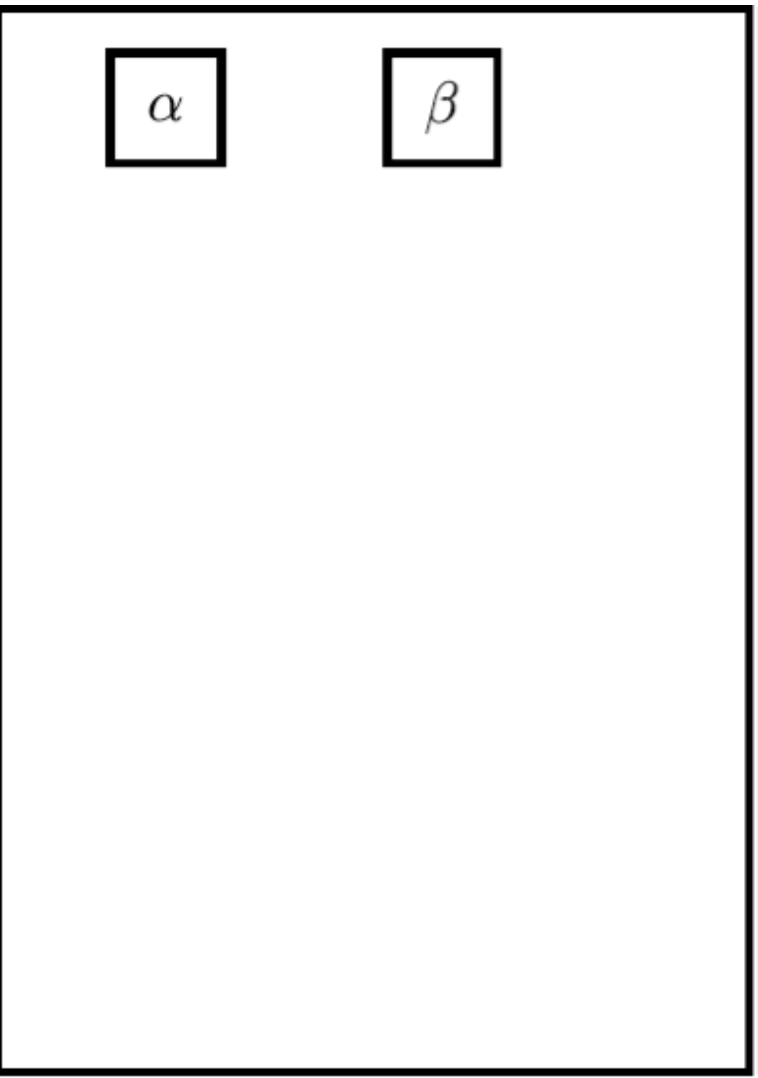


# Using the Rev language to build a model



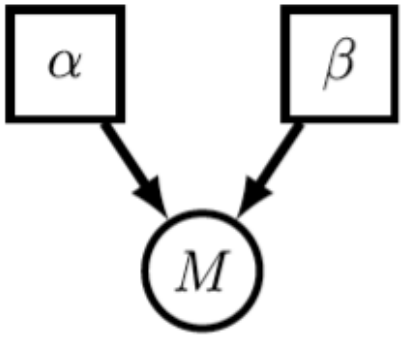
```
observations <- [<your data go here>]
```

# Using the Rev language to build a model



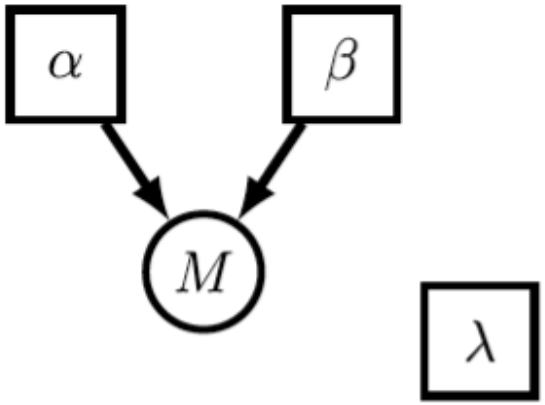

```
observations <- [<your data go here>]

alpha <- 3.0
beta <- 1.0
```

# Using the Rev language to build a model



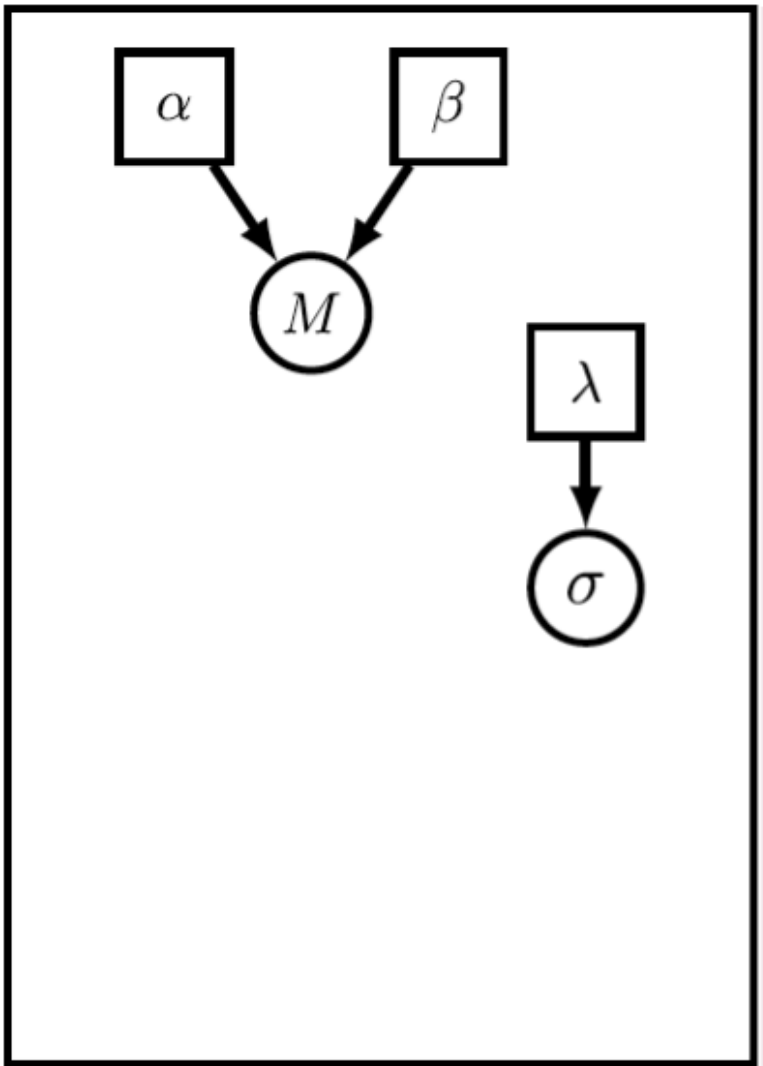

```
observations <- [<your data go here>]

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

# Using the Rev language to build a model



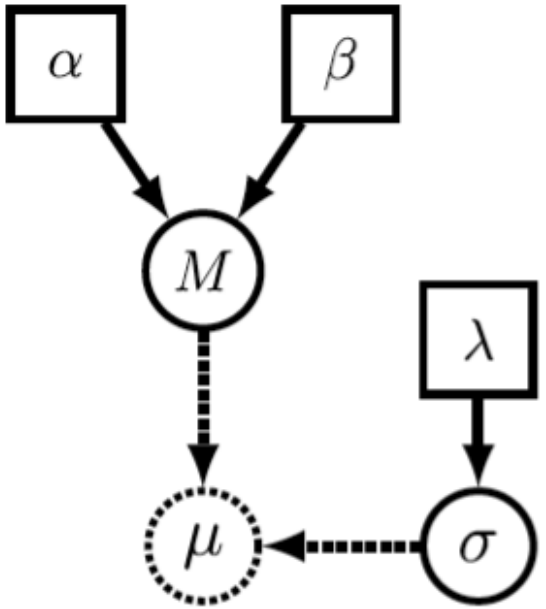

```
observations <- [<your data go here>]

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


lambda <- 1.0
```

# Using the Rev language to build a model



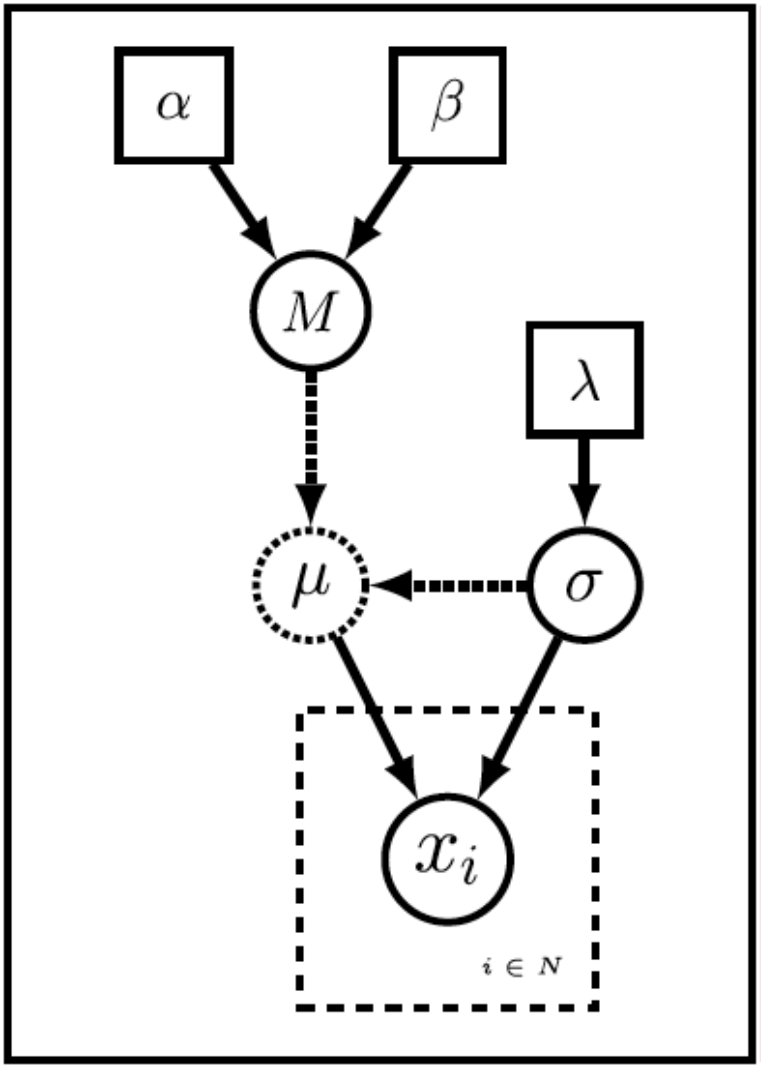

```
observations <- [<your data go here>]

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


lambda <- 1.0
sigma ~ dnExponential(lambda)
```

# Using the Rev language to build a model



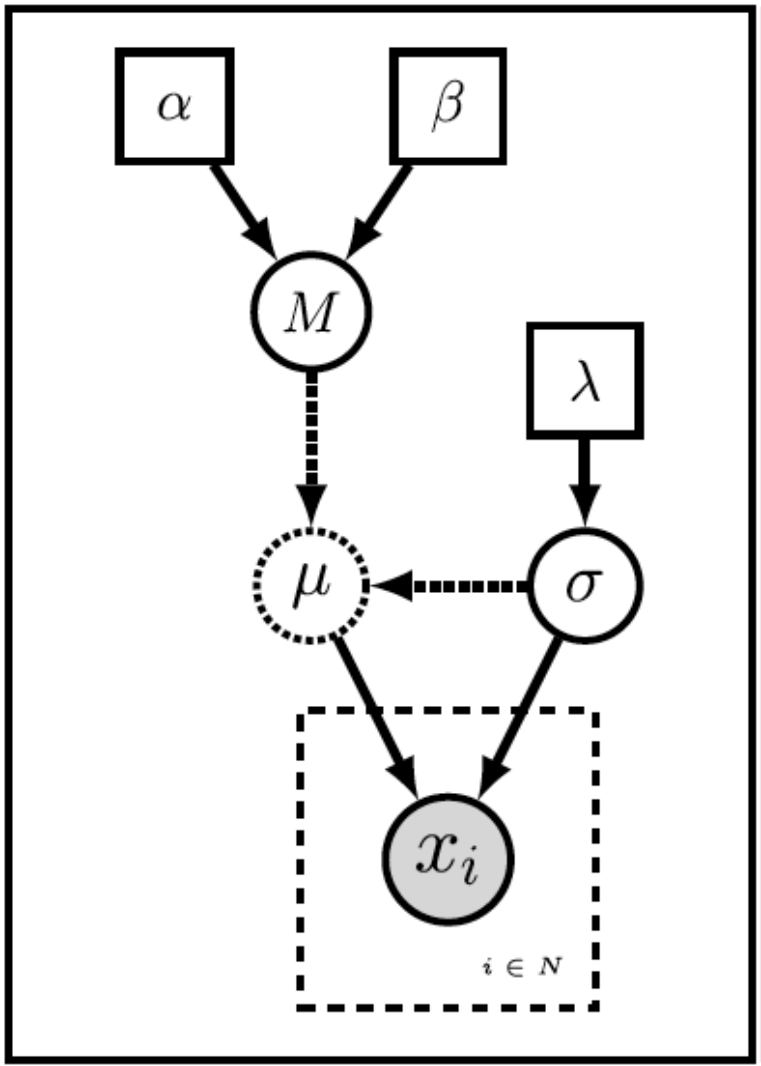

```
observations <- [<your data go here>]

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




```
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)

N <- observations.size()
for( i in 1:N ){
  x[i] ~ dnLnorm(mu, sigma)

}
```

# 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)


N <- observations.size()
for( i in 1:N ){
  x[i] ~ dnLnorm(mu, sigma)
  x[i].clamp(observations[i])
}
```

# 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)

  - 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: final 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: final details (2)

- User-defined functions:

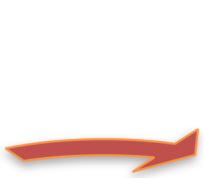
```
function RealPos square ( Real x ) { x * x }
```

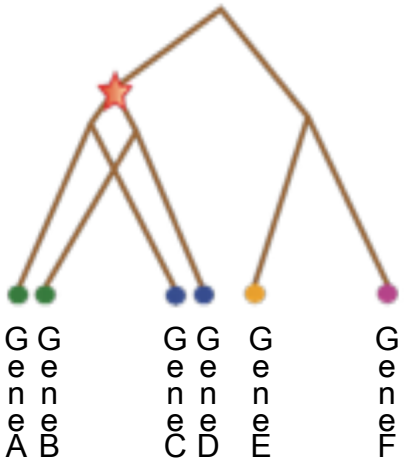
- User-defined functions can be recursive:

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

# Bayesian phylogenetic inference

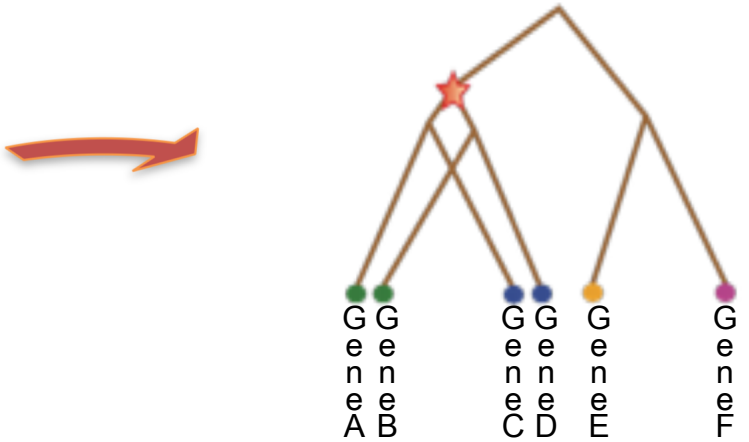
*Homo sapiens*; GeneA: ACTGGTGATGACATAAC…
*Homo sapiens*; GeneB: ACTGTTGATGACATGAC…
*Mus musculus*; GeneC: ACTGATGATGACAAGAC…
*Mus musculus*; GeneD: ACTGGTGA--CCATGAC…
*Bison bison*; GeneE: ACTGGTGATGACACGAC…
*Canis lupus*; GeneF: ACT--TCATGAAACGAC…





# Bayesian phylogenetic inference

*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…



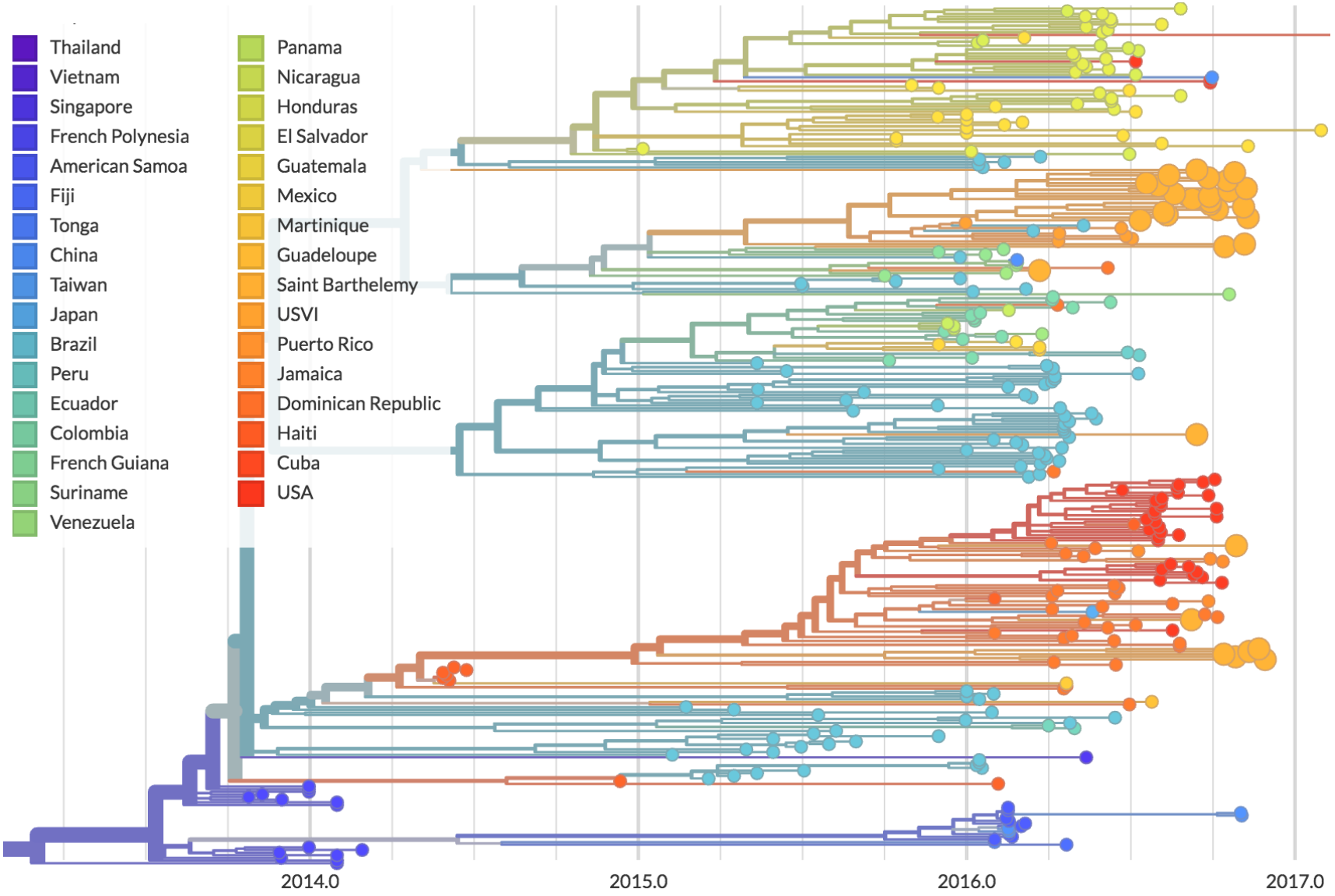

Example of models:

- 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
- …

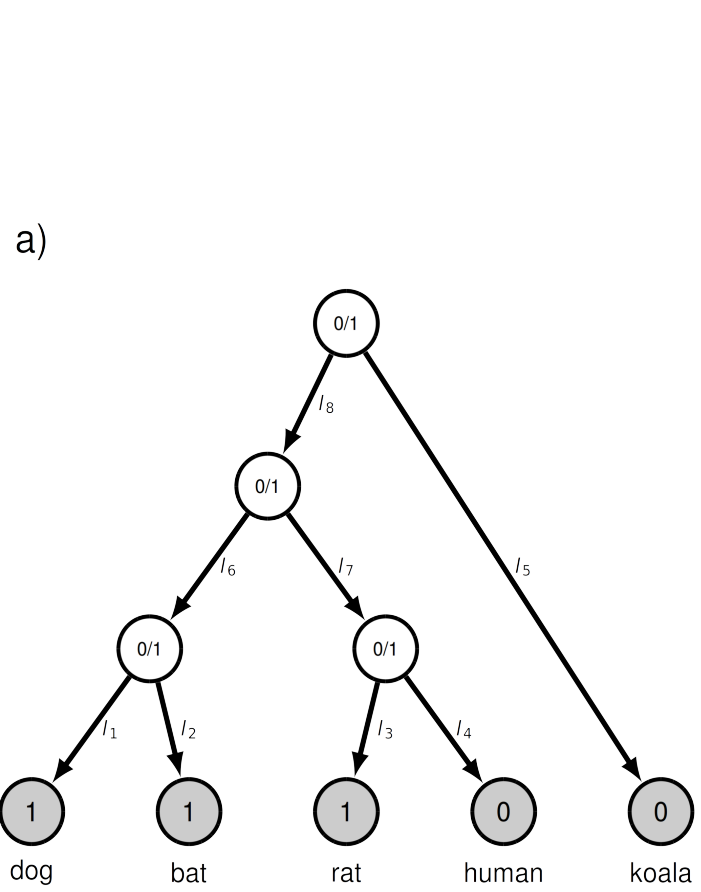
# Bayesian phylogenetic inference

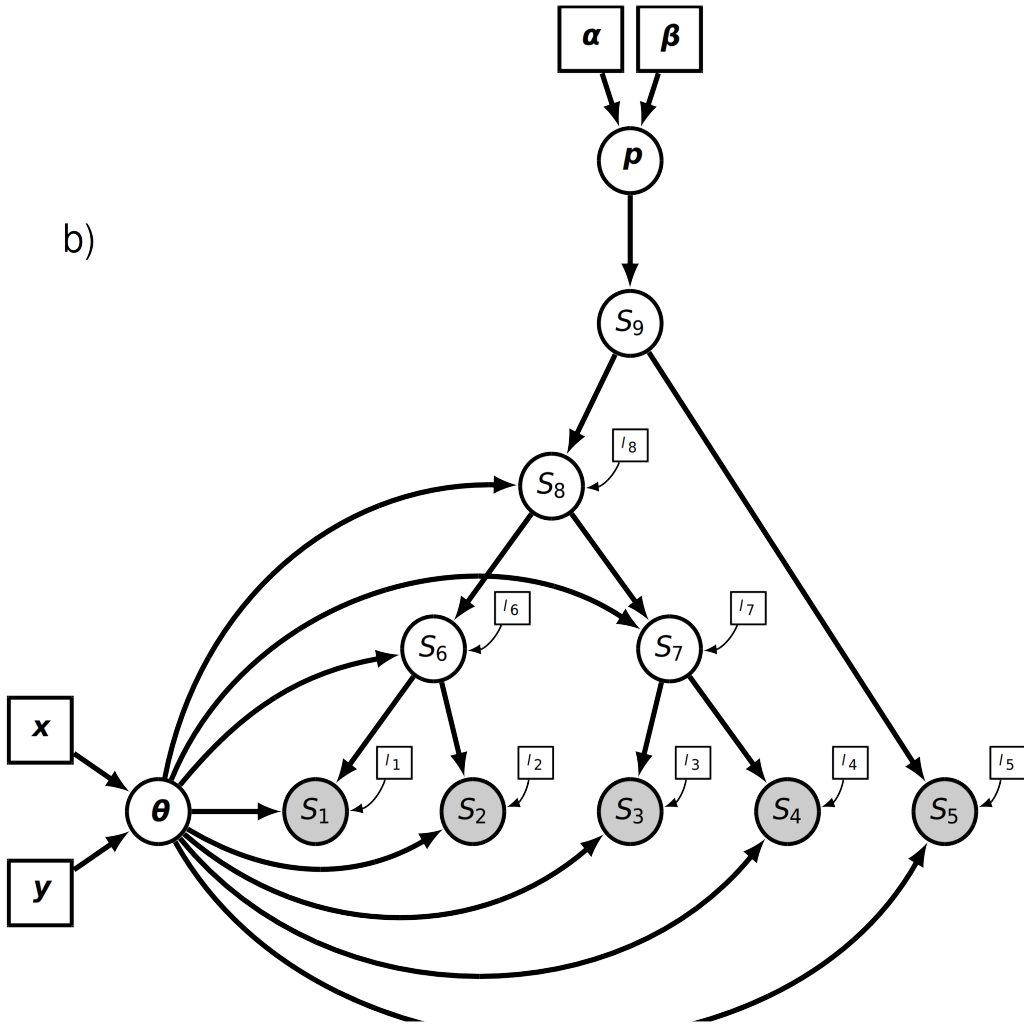
- We may be interested in the topology of the phylogenetic tree
- We may be interested in parameters associated to the branches or the nodes of the phylogenetic tree
- We may be interested in averaging out the uncertainty surrounding the phylogenetic tree to learn about traits at the leaves
- ...

# 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
dnFBDP
dnFBDRMatrix
dnFBDRP
dnFossilizedBirthDeathRange
dnFossilizedBirthDeathRangeMatrix
dnGamma
dnGeom
dnGeometric
dnGossetT
dnHBDP
dnHalfCauchy
dnHalfNormal
dnHeterochronousCoalescent
dnHeterochronousCoalescentSkyline
dnInvWishart
dnInverseGamma
dnInverseWishart
dnLKJ
dnLKJPartial
dnLaplace
dnLnorm
dnLogExponential
dnLognormal
dnLoguniform
dnLorentz
dnMixture
dnMultiSpeciesCoalescent
dnMultiSpeciesCoalescentInverseGamn
dnMultiSpeciesCoalescentUniformPr
dnMultinomial
dnMultivariateNormal ...

# Moves in RevBayes

Move_DiscreteEventCategoryRandomWalk
Move_ElementScale
Move_ElementSlide
Move_ElementSwapSimplex
Move_EllipticalSliceSamplingLognormalIID
Move_EllipticalSliceSamplingSimple
Move_EmpiricalTree
Move_EventTimeBeta
Move_EventTimeSlide
Move_FNPR
Move_GMRFHyperpriorGibbs
Move_GammaScale
Move_GibbsDrawCharacterHistory
Move_GibbsMixtureAllocation__Integer
Move_GibbsMixtureAllocation__Natural
Move_GibbsMixtureAllocation__Probability
Move_GibbsMixtureAllocation__RateGenerator
Move_GibbsMixtureAllocation__Real
Move_GibbsMixtureAllocation__RealPos
Move_GibbsMixtureAllocation__Simplex
Move_GibbsPruneAndRegraft
Move_GraphShiftEdge
Move_HSRFHyperpriorsGibbs
Move_HSRFUnevenGridHyperpriorsGibbs
Move_IndependentTopology
Move_LayeredScaleProposal
Move_LevyJump
Move_LevyJumpSum
Move_MatrixElementScale
Move_MatrixElementSlide
Move_MatrixRealSymmetricSlideMove
Move_MixtureAllocation__Integer
Move_MixtureAllocation__Natural
Move_MixtureAllocation__Probability
Move_MixtureAllocation__RateGenerator
Move_MixtureAllocation__Real
Move_MixtureAllocation__RealPos
Move_MixtureAllocation__Simplex
Move_MixtureAllocation__Tree
Move_MultipleElementScale
Move_NNI
Move_NarrowExchange
Move_NodeTimeSlideBeta
Move_NodeTimeSlidePathTruncate
Move_NodeTimeSlideUniform
Move_NodeTimeSlideUniformAge
Move_RandomGeometricWalk
Move_RandomIntegerWalk
Move_RateAgeBetaShift
Move_ReversibleJumpSwitch__Int
Move_ReversibleJumpSwitch__Na
Move_ReversibleJumpSwitch__Na
Move_ReversibleJumpSwitch__Pro
Move_ReversibleJumpSwitch__Re
Move_ReversibleJumpSwitch__Re
Move_ReversibleJumpSwitch__Sin
Move_ReversibleJumpSwitch__Tre
Move_RootTimeScaleBactrian
Move_RootTimeSlideUniform
Move_SPR
Move_Scale
Move_ScaleBactrian
Move_ScaleBactrianCauchy ...

# Example: toxoplasmosis in boars (from Guillaume Kon Kam King)

- We model toxoplasmosis in boars as follows:

$$i(a) = 1 - \exp\left((A - a) \times \alpha\right)$$

Infected ~ B(Total number, i)

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

$$\log(\alpha) \sim \mathcal{U}(-4, -1)$$

| Age | Infected | Total number |
|---|---|---|
| 4.8 | 13 | 131 |
| 6.0 | 17 | 93 |
| 7.7 | 24 | 82 |
| 9.7 | 32 | 108 |
| 11.5 | 24 | 93 |
| 13.8 | 13 | 60 |
| 17.6 | 26 | 88 |
| 20.6 | 30 | 102 |
| 25.7 | 38 | 82 |
| 51.9 | 30 | 93 |

# 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!*

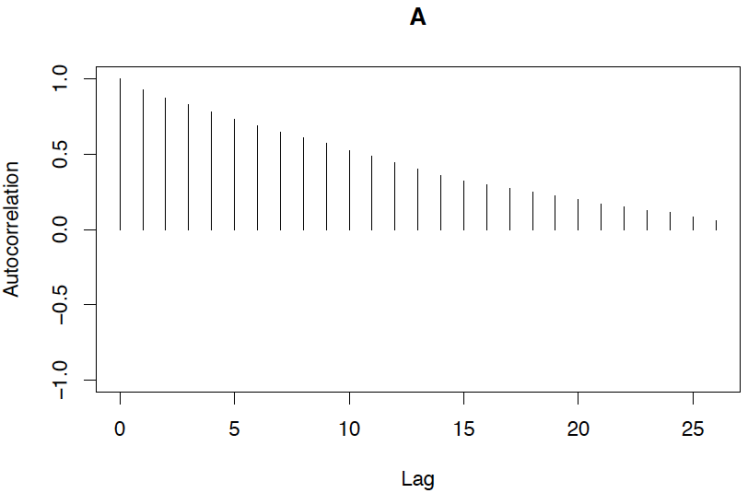
# Preparing for inference

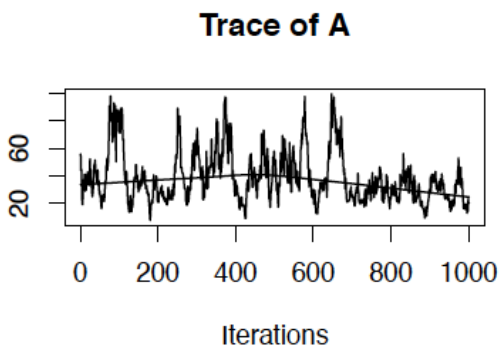
```
# Get a hang on the model (any node will do)
mymodel = model(A)

# 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[3] = srMinESS(50,myOutputFile,10000)
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)
```

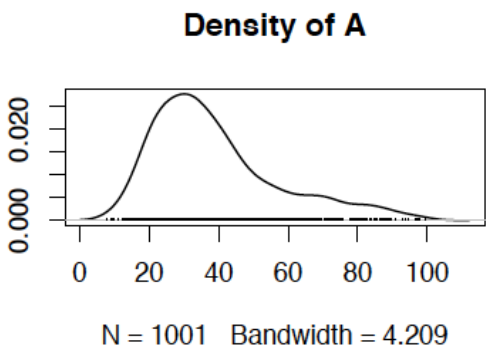
# Performing inference

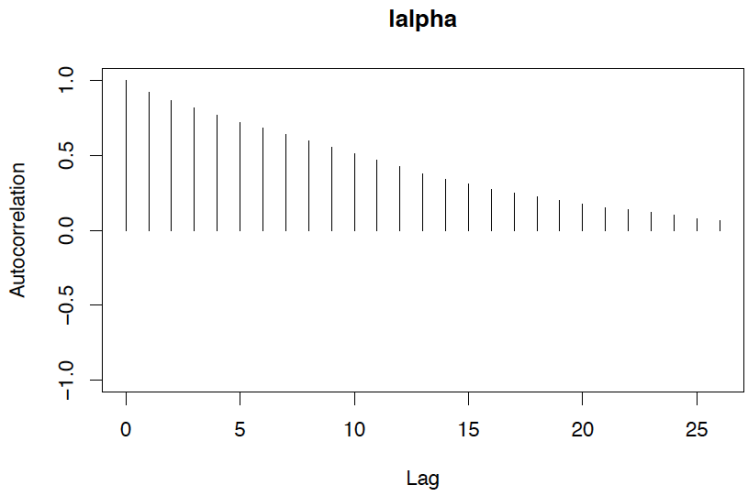
```
# Creating the MCMC object
mymcmc = mcmc(mymodel, monitors, moves,
    moveschedule="random", nruns=2)

# 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)
```

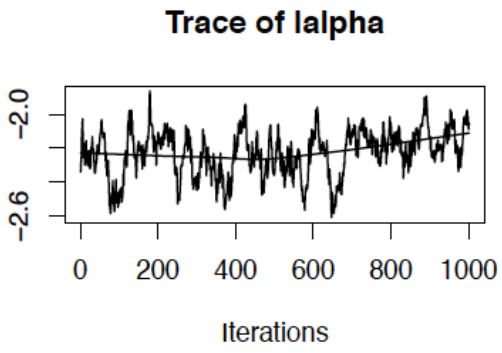
# Convergence plots with coda

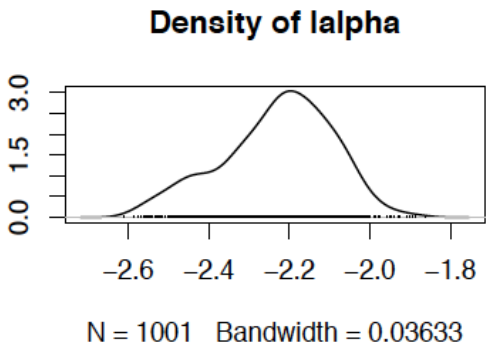


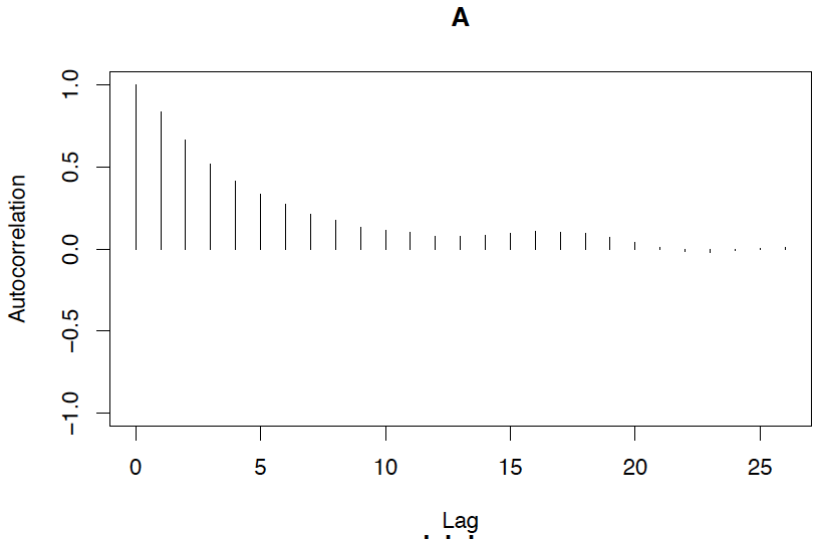



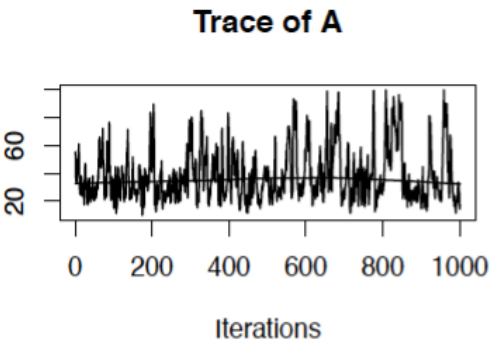
# Changing the moves

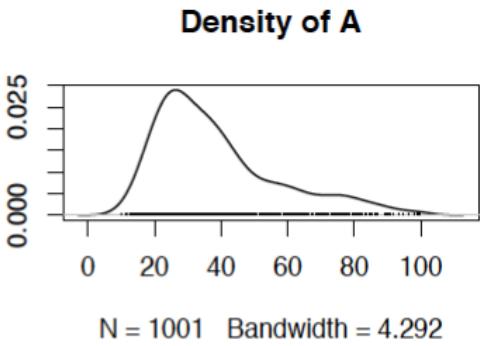
moves[moveIndex++] = mvSlice(A)

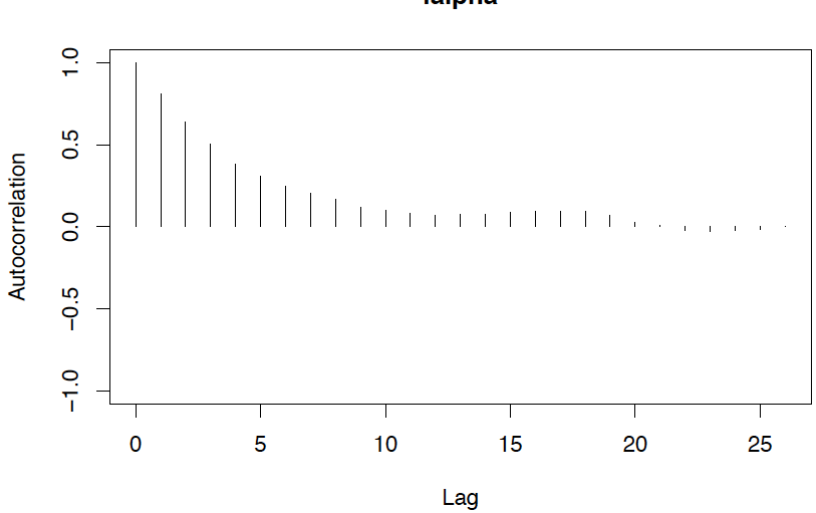
moves[moveIndex++] = mvSlice(lalpha)

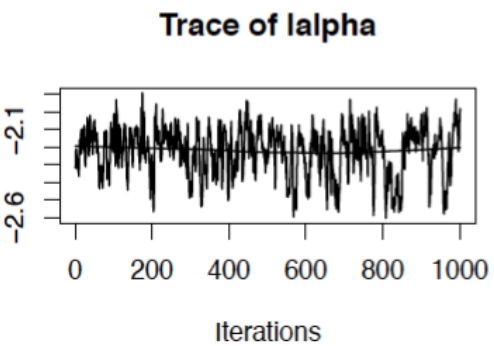
# Convergence plots with coda

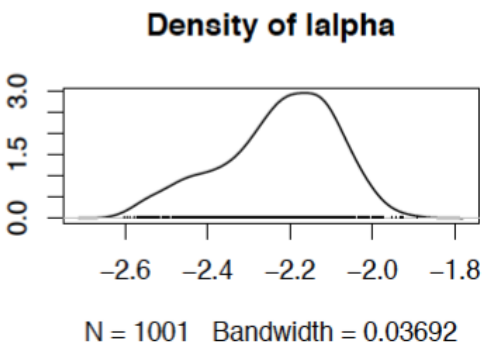


# Combining moves

moveIndex = 0

moves[moveIndex++] = mvSlice(A)

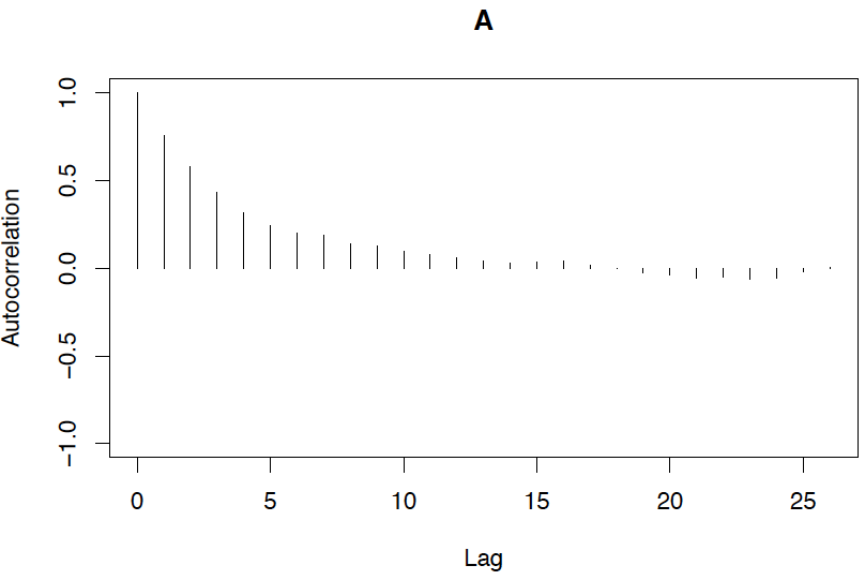
moves[moveIndex++] = mvSlice(lalpha)

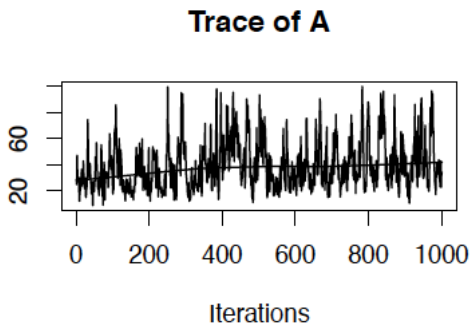
moves[moveIndex++] = mvSlide(A)

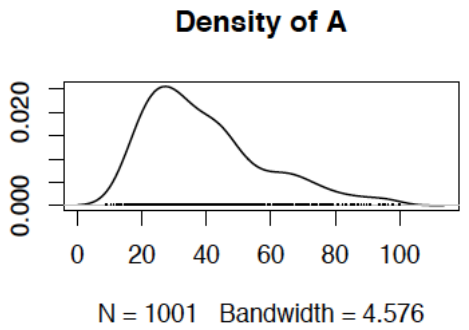
moves[moveIndex++] = mvSlide(lalpha)

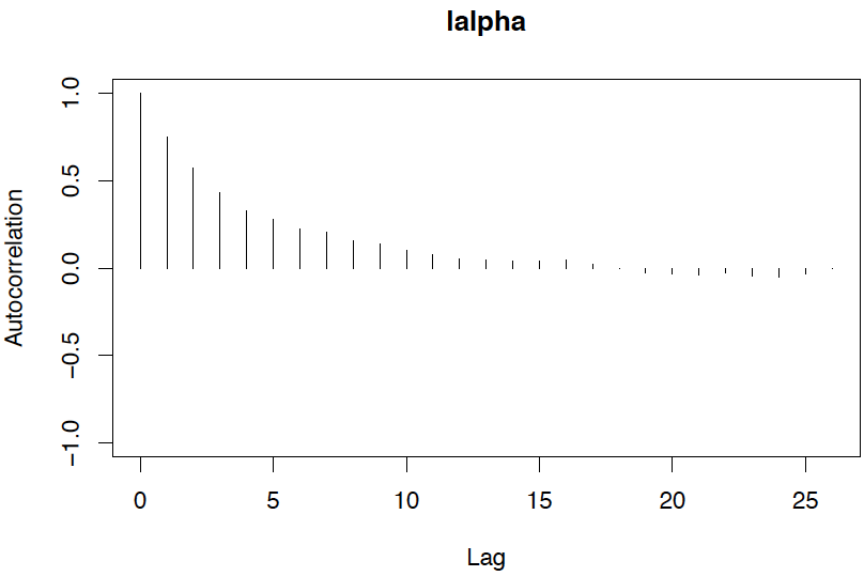
moves[moveIndex++] = mvScale(A)

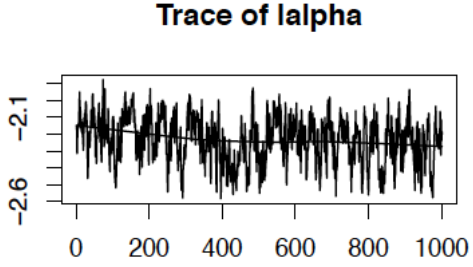
# Convergence plots with coda

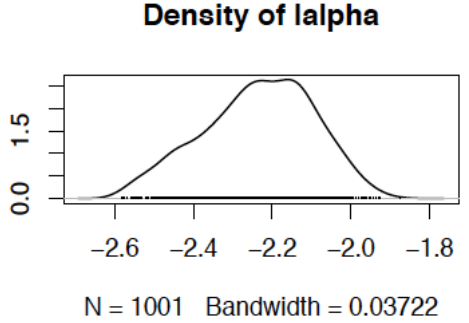


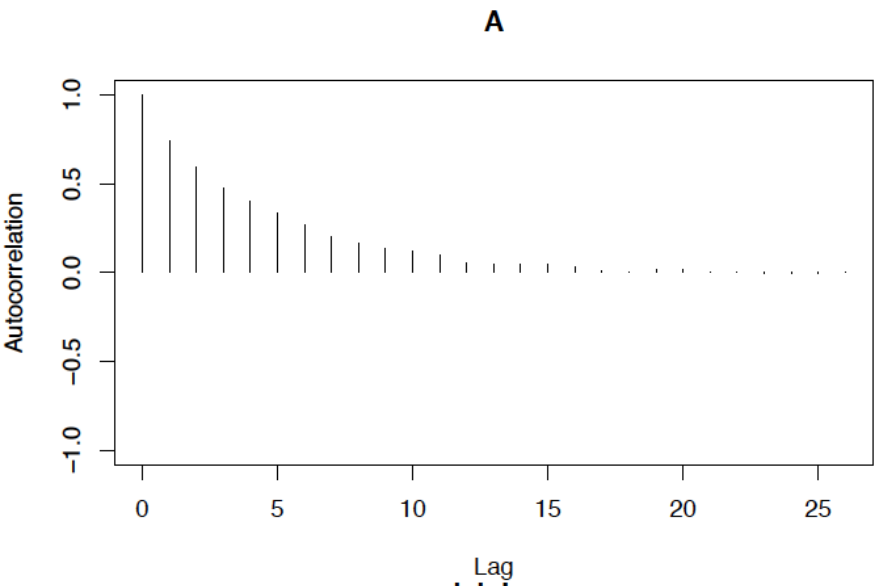



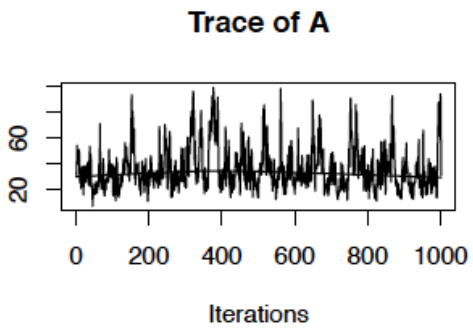



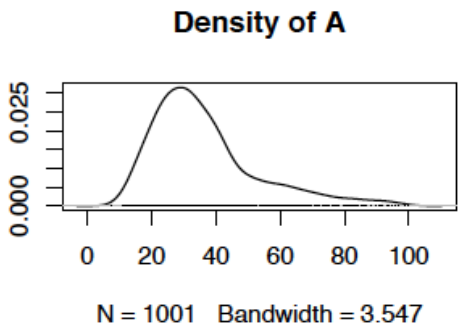
# Combining moves and using MCMCMC

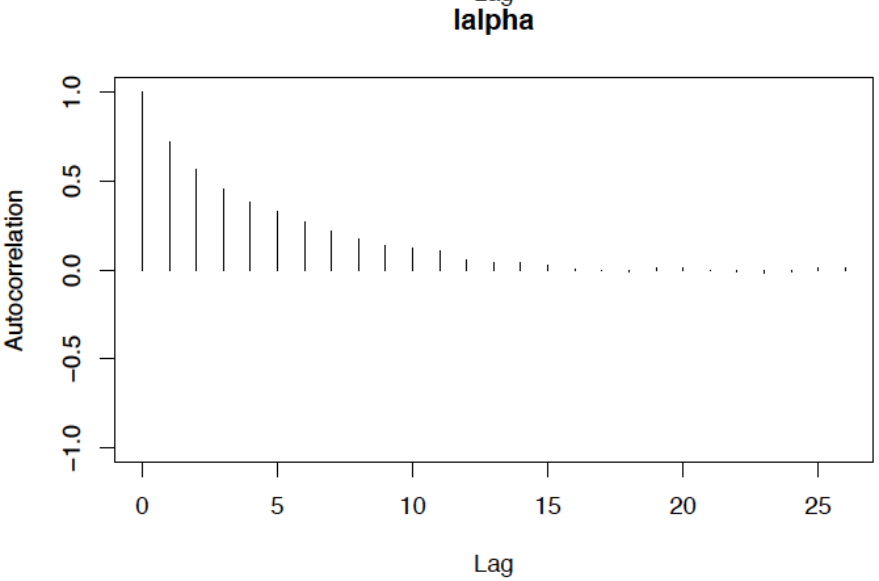
```
moveIndex = 0
moves[moveIndex++] = mvSlice(A)
moves[moveIndex++] = mvSlice(lalpha)
moves[moveIndex++] = mvSlide(A)
moves[moveIndex++] = mvSlide(lalpha)
moves[moveIndex++] = mvScale(A)
#...
mymcmc = mcmcmc(mymodel, monitors, moves,
   moveschedule="random", nchains=4, nruns=2)
```

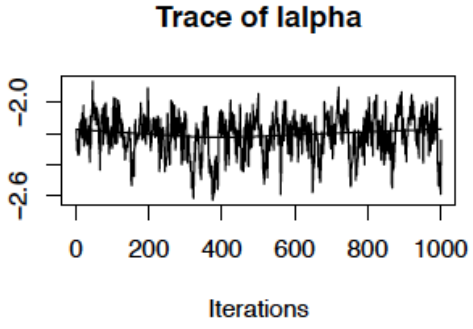
# Convergence plots with coda

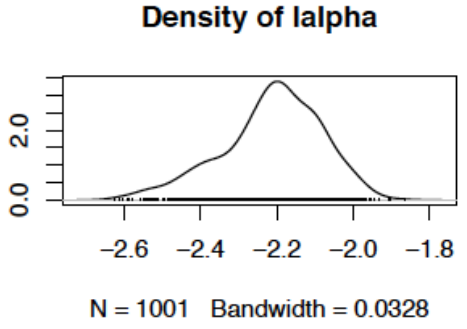


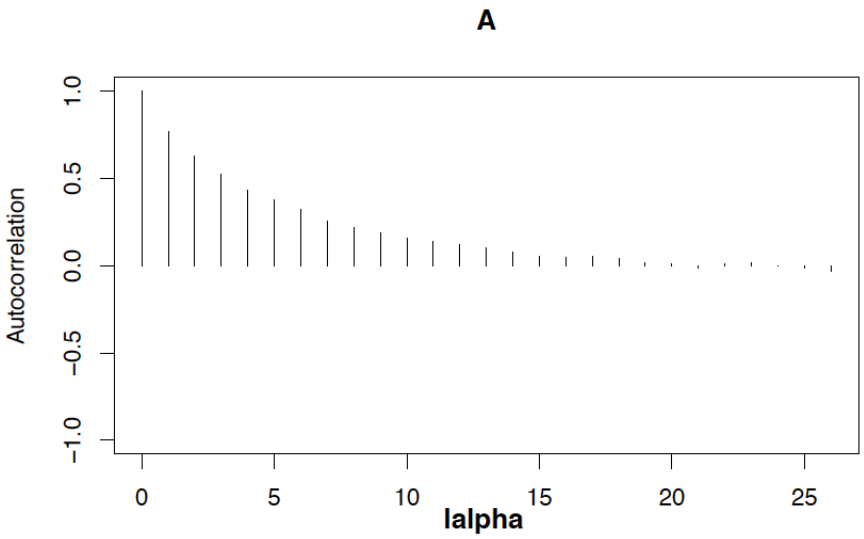



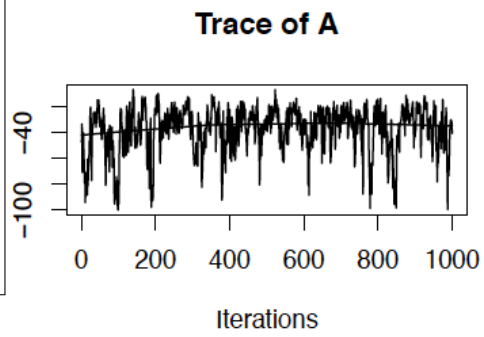



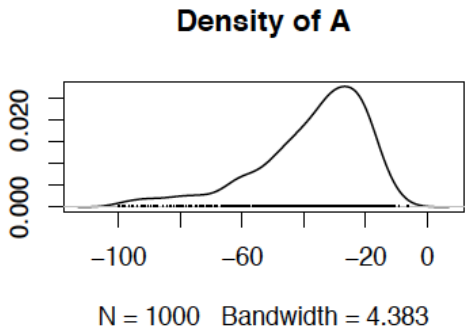



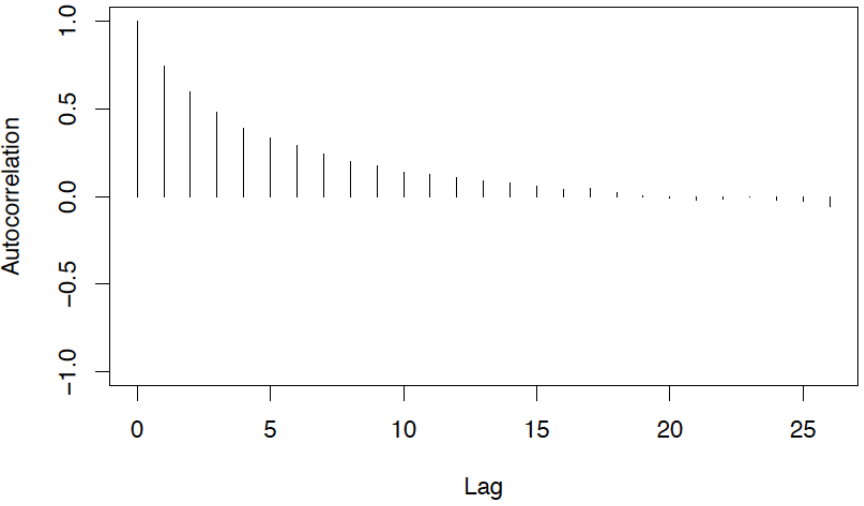



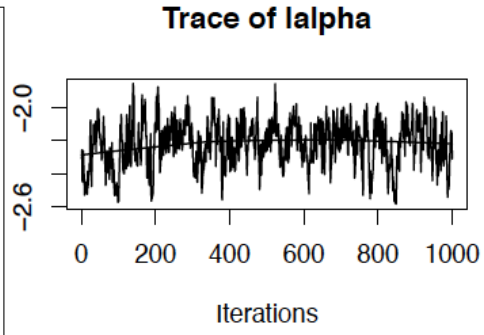
# Comparison with Jags

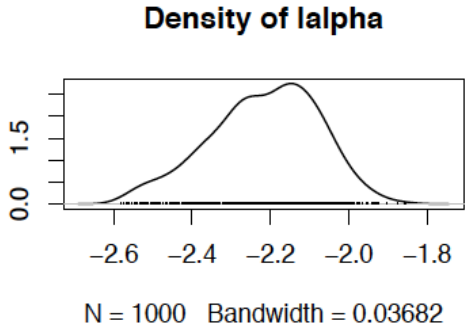


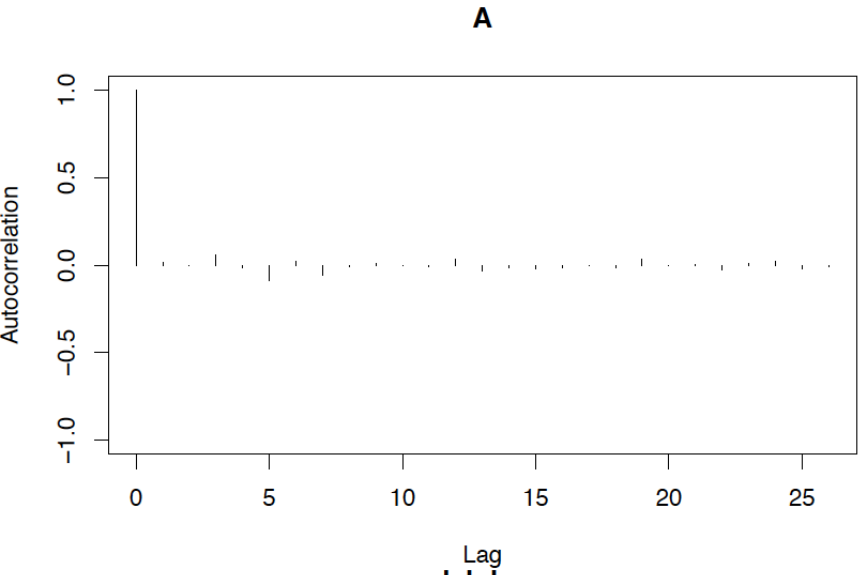



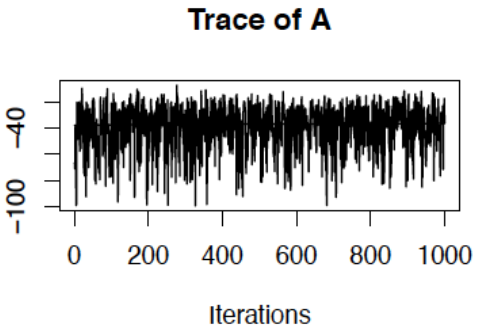



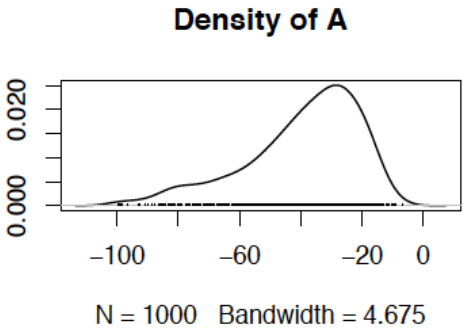



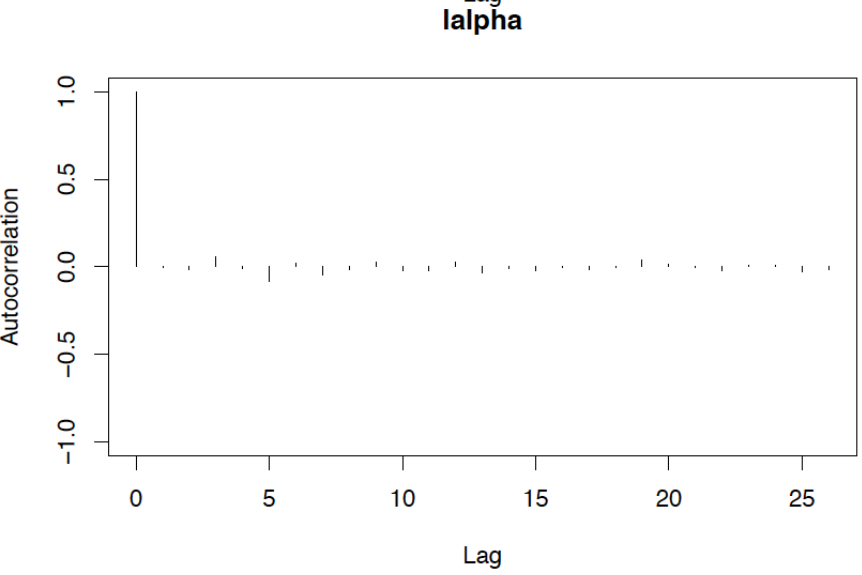



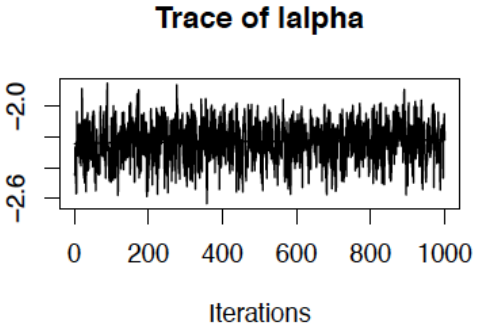
# Comparison with Stan

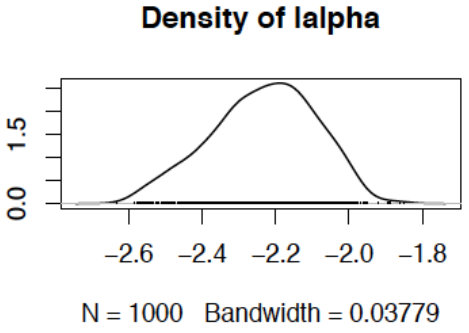


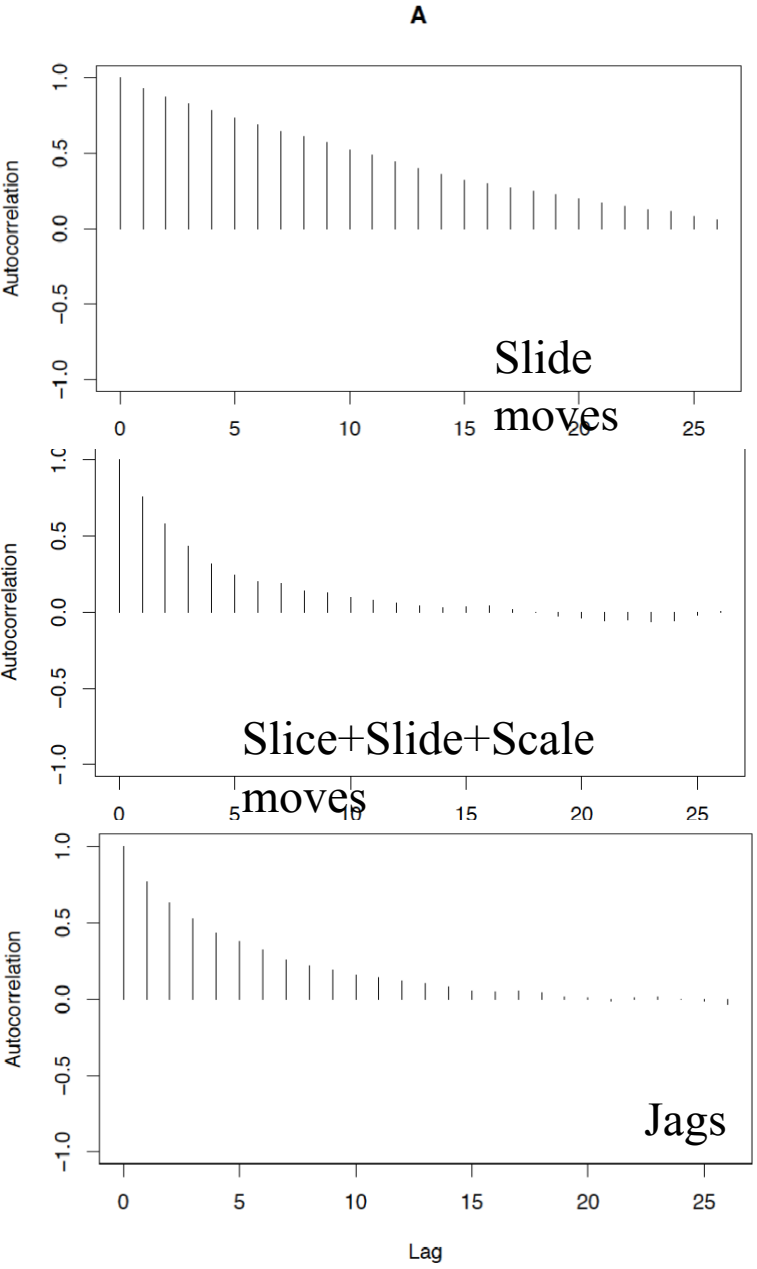



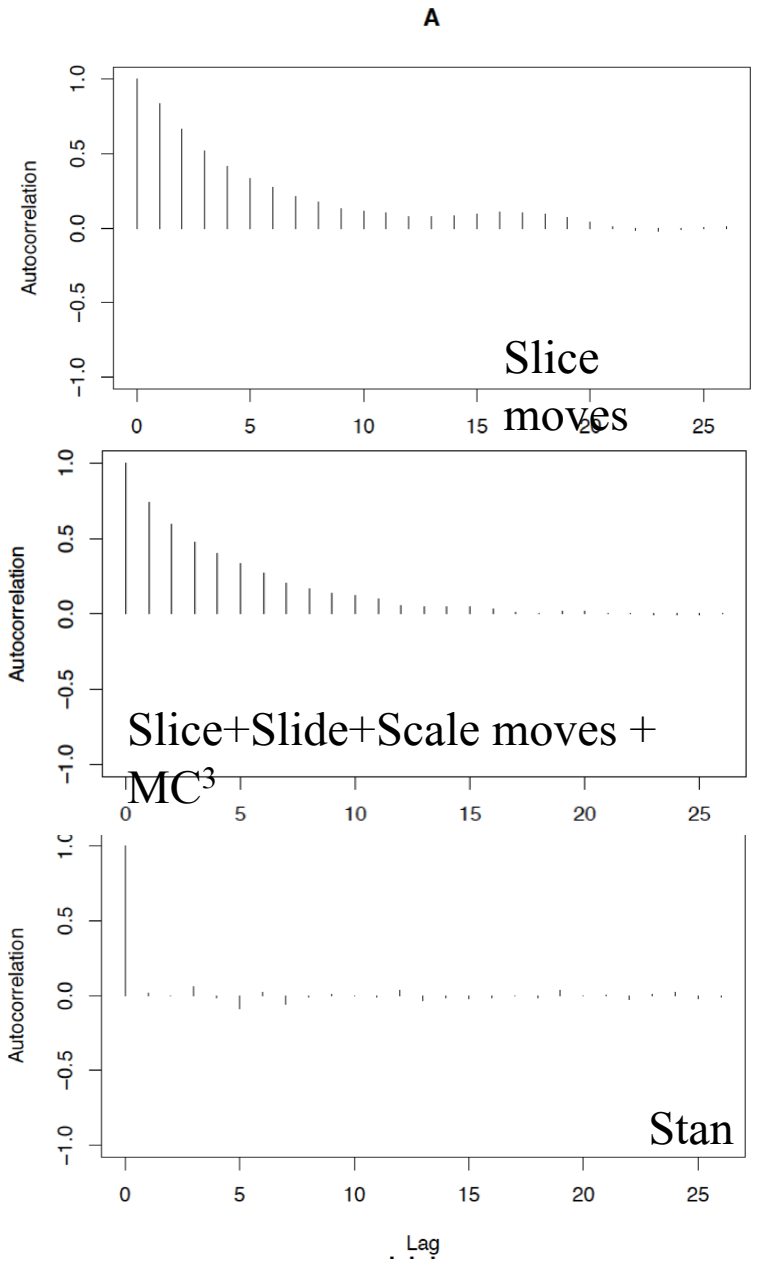



# Comparison of the lag for various moves and vs Jags and Stan



# 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