Multicore Stochastic Mapping Analysis Calculator McSMAC

Dustin Lennon
`dustin.lennon [at] gmail.com`

May 2, 2012

1 Background

Multicore Stochastic Mapping Analysis Tool (McSMAC) arose out of the need to analize large, phylogenetic-like datasets with ascertainment bias. The user provides a phylogenetic tree along with tip data across a set of sites. McSMAC returns the likelihood and conditional expectations of mutational jumps and dwelling times. The underlying calculation is described in a paper by Vladimir Minin (University of Washington, USA).

The code described here is motivated by data provided by Sterling Sawaya (University of Otago, New Zealand) and Emmanuel Buschiazzo (University of California, Merced, USA). Results are described in "Measuring Microsatellite Conservation in Mammalian Evolution with a Phylogenetic Birth-Death Model," a paper to be published in the journal "Genome Biology and Evolution."

2 Installation

The McSMAC code is posted on github. It is intended to be fairly easy to install for POSIX systems. Windows users, you are out of luck. McSMAC will not run on a windows machine.

2.1 License Issues with the Multicore Workqueue

The multicore component of McSMAC depends on a workqueue implementation borrowed, without permission, from David Butenhof's text, "Programming With POSIX Threads." This code can be downloaded from the author's website, however, there is no explicit licensing documentation available. Therefore, any person compiling or using McSMAC with dependencies on the workqueue code does so at their own peril and agrees to accept any and all legal risks associated with such use. Unless you make a conscious decision to enable the Butenhof workqueue, McSMAC will run as a single thread.

2.2 Instructions to Enable the Butenhof WorkQueue

To enable the Butenhof dependencies, follow the instructions in the Makefile. You will have toenable a build flag in the Makefile, manually download the files, and apply the patch.

2.3 Building on a Mac

See the comments in the Makefile. There is a comment and corresponding build flag that can be used. This is experimental, with only light testing on Mac OS X.

3 Basic Usage

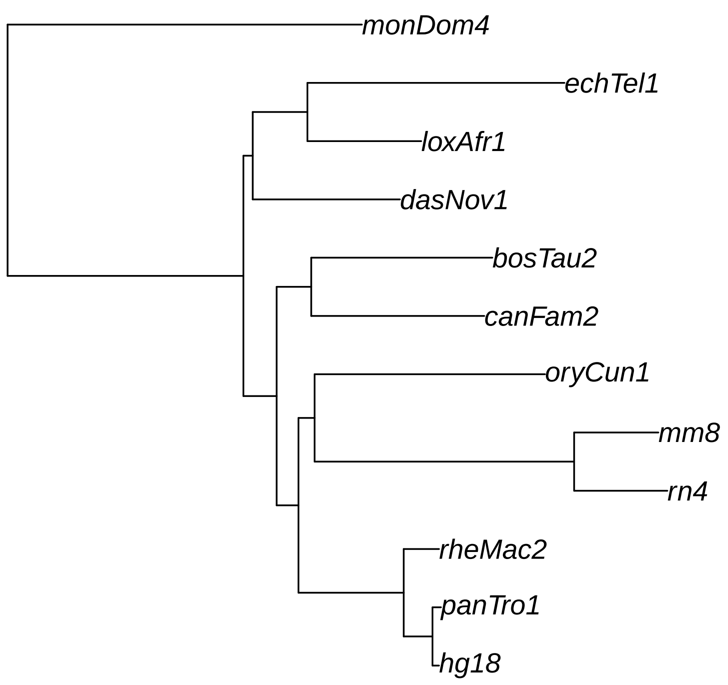
First, load the ape package and the newly built shared library. The ape package is not strictly necessary, but regular users of ape may find the use of that code in this document helpful in understanding the interfaces of McSMAC.

```
> library(ape)
> dyn.load("./src/FastMap.so")
```

Read in some data. McSMAC expects Newick format for describing the phylogenetic tree. Site data is a matrix with rows correspond to sites and columns to tips. For the Sawaya/Buschiazzo data, tips are either 1 or 2, with a 2 indicating a microsatellite that is shared with humans, the hg18 leaf node on the tree. More generally, McSMAC allows for arbitrary tip states, assumed to be numbered from 1..N.

```
> ape.tree = read.tree("data/tree12")
> tree.string = write.tree(ape.tree)
> site.data = as.matrix(
+   read.csv("data/utr3.csv", header = TRUE,
+            colClasses=rep("numeric", 12 )
+            )
+   )
> site.data = site.data + 1
> n.tips = ncol(site.data)
> n.states = 2
```

For the purposes of this example, we plot the phylogenetic tree.



3.1 Data Structures and Masks

McSMAC computes the conditional expectation of mutations as well as dwelling time. In order to do this calculation, the code requires that we set up a few data structures to describe the mutations in which we're interested. This is accomplished through masks, objects that take 0s and 1s.

3.1.1 Conditional Expected Mutuations

For example, to count jumps from state 2 to state 1, we specify the corresponding location in a transition matrix.

```
> label21.mask = matrix(
+   c(0, 0,
+     1, 0),
+   nr=2, byrow=T)
```

or, from state 1 to state 2

```
> label12.mask = matrix(
+   c(0, 1,
+     0, 0),
+   nr=2, byrow=T)
```

For the more general case, with say 4 nucleotides, several nucleotide mutations may be of interest and multiple positions in the 0-1 transition matrix may be "on." However, the diagonal must always be set to zero.

3.1.2 Conditional Expected Dwelling Time

For dwelling time, there are similar masks that are created. The first will aggregate the "time" spent in state 2. The second, the "time" spent in state 1. Here, "time" is in quotes because it is relative to—e.g. of the same dimension as—the length of the edge between two nodes in the tree.

```
> dwell2.mask = c(0,1)
> dwell1.mask = c(1,0)
```

3.1.3 Masking Tip States

Our data is characterized by the human tip always being in state 2. This is the source of the ascertainment bias. In order to make inference on mutational rates, say, one needs to incorporate this bias into a conditional likelihood. This will be further developed later, but the underlying change involves a need to compute the probability that a tip is in a specific state (or subset of states). Again, we rely on a masking paradigm.

We create a 0-1 matrix. The number of rows corresponds to the number of possible tip states, two in this case. Columns are associated with tips, as in the site.data variable defined earlier. Thus, tip (column) i will have a 1 in any "allowed" state and a 0 in any "forbidden" state.

```
> T1.mask = matrix(rep(1, n.states*n.tips), nr=n.states, nc=n.tips)
> T1.mask[1,1] = 0
> T1.mask

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[1,]    0    1    1    1    1    1    1    1    1     1     1     1
[2,]    1    1    1    1    1    1    1    1    1     1     1     1
```

This is a matrix corresponding to a tip vector where the human state must be 2, but all other tips are allowed be in state 1 or state 2. We'll eventually use this data structure to compute $\mathbb{P}(X_{hg18} = 2)$, where X_{hg18} denotes the event that we have a tip-string with the human state equal to 2.

4 Ascertainment Bias Likelihood

Using the data structures established earlier, we can compute an ascertainment bias likelihood. This arises from the standard phylogenetic model, here with two states. For notation, define the rate matrix

$$\Lambda = \begin{pmatrix} -\lambda & \lambda \\ \mu & -\mu \end{pmatrix}$$

Let X denote the random vector corresponding to the tip-string, and assume that for site i, the observed tip_vector is denoted by D_i. Then, the uncorrected (for ascertainment bias) likelihood is

$$L_i^u(\Lambda) = \mathbb{P}[X = D_i; \Lambda]$$

and the corrected likelihood is

$$L_i(\Lambda) = \frac{\mathbb{P}[X = D_i; \Lambda]}{\mathbb{P}(X_{hg18} = 2; \Lambda)} \tag{1}$$

Assuming independence and aggregating across all S sites yields

$$L(\Lambda) = \prod_{i=1}^{S} L_i(\Lambda)$$

It is well known that L_i^u can be computed with Joe Felsenstein's tree-pruning algorithm.

The independence also implies that unique strings and their respective counts are sufficient statistics, and we can use the following computational shortcut:

```
> ## create unique tip vectors and counts
> tip.strings = apply(site.data, 1, paste, collapse="")
> tmp = rle(sort(tip.strings))
> uniq.tip.strings = tmp$values
> uniq.tip.vecs = t(sapply(strsplit(uniq.tip.strings, ''), function(x) as.numeric(x)))
> uniq.tip.counts = tmp$lengths
> ## copy over the dimnames, as these are used internally
> dimnames(uniq.tip.vecs) = dimnames(site.data)
> ## reverse lookup:  tip.strings[ii] == uniq.tip.strings[rev.idx[ii]]
> rev.idx = match(tip.strings, uniq.tip.strings)
```

The code defining the ascertainment bias log-likelihood function described by equation 1 is relatively straightforward.

```
> abc.lhood = function(params, my.data, counts)
+   {
+     mu = params[1]   ## death rate
+     lam = params[2]  ## birth rate
+
+     ## transition rate matrix
+     Q = matrix(
+       c(-lam, lam,
+         mu,  -mu),
+       nr=2, nc=2,
+       byrow=T
+       )
+
+     ## likelihood calculator & stochastic map quantities (multicore)
+     out.multi = .Call(
+       "MultiMap_RAPI",
+       tree.string,
+       Q,
+       label21.mask,  ## compute cond. exp. jumps from state 2 to 1
+       dwell2.mask,   ## compute cond. exp. dwelling time in state 2
+       my.data,
+       NULL,    ## root distribution (stationary dist is used if NULL)
+       FALSE    ## shortcut bool (TRUE will omit exp.jumps & exp.dwell_time)
+       )
+
+     ## compute P(X_hg18 == 1):  in our case, this reduces to lam / (mu + lam)
+     ## However, OneMap_RAPI is general enough to handle the cases without
+     ## a closed form solution as well.
+     B.evt = .Call(
+       "OneMap_RAPI",
+       tree.string,
+       Q,
+       label21.mask,
+       dwell2.mask,
+       T1.mask
+       )
+
+     ## extract uncorrected log-likelihood and marginal likelihood
+     lhood.uncorr = out.multi[,1]
+     lhood.marg = B.evt[1]
+
+     ## log-likelihood, after correcting for ascertainment bias
+     objval = ( log(lhood.uncorr) - log(lhood.marg) ) %*% counts
+
+     return(objval)
+   }
```

4.1 Ascertainment Bias Corrected MLE

Optimizing the "ABC" MLE is a simple matter of calling R's optim function.

```
> obj.fn = function(params, my.data, counts) -abc.lhood(params, my.data, counts)
> opt.soln = optim(
+   par = c(5,5),
+   obj.fn,
+   my.data=uniq.tip.vecs,
+   counts = uniq.tip.counts,
+   lower=c(1e-4, 1e-4),
+   method="L-BFGS-B",
+   hessian=TRUE
+   )

> opt.soln$par

[1] 4.9010130 0.2641966
```

Hence, the death rate $\hat{\mu} = 4.90$, and the birth rate $\hat{\lambda} = 0.26$.

Asymptotic, 95 percent confidence intervals can be extracted from the hessian, which is just the Fisher Information matrix.

```
> sig = sqrt(diag(solve(opt.soln$hessian)))
> opt.soln$par[1] + c(-1,1) * 1.96 * sig[1]

[1] 4.816359 4.985667

> opt.soln$par[2] + c(-1,1) * 1.96 * sig[2]

[1] 0.2435698 0.2848235
```

5 Site-specific Birth and Death Rates

We return briefly to the uncorrected likelihood, noting that it can be written in terms of unobserved sufficient statistics:

$$L_i^u(\Lambda) = \lambda^{N^{12}} \mu^{N^{21}} e^{-W_1 \lambda} e^{-W_2 \mu}$$

where

- N^{12} is the number of jumps from state 1 to 2
- N^{21} the number of jumps from state 2 to 1
- W_1 the dwelling time in state 1
- W_2 the dwelling time in state 2.

Taking logs and derivatives yields the uncorrected MLEs

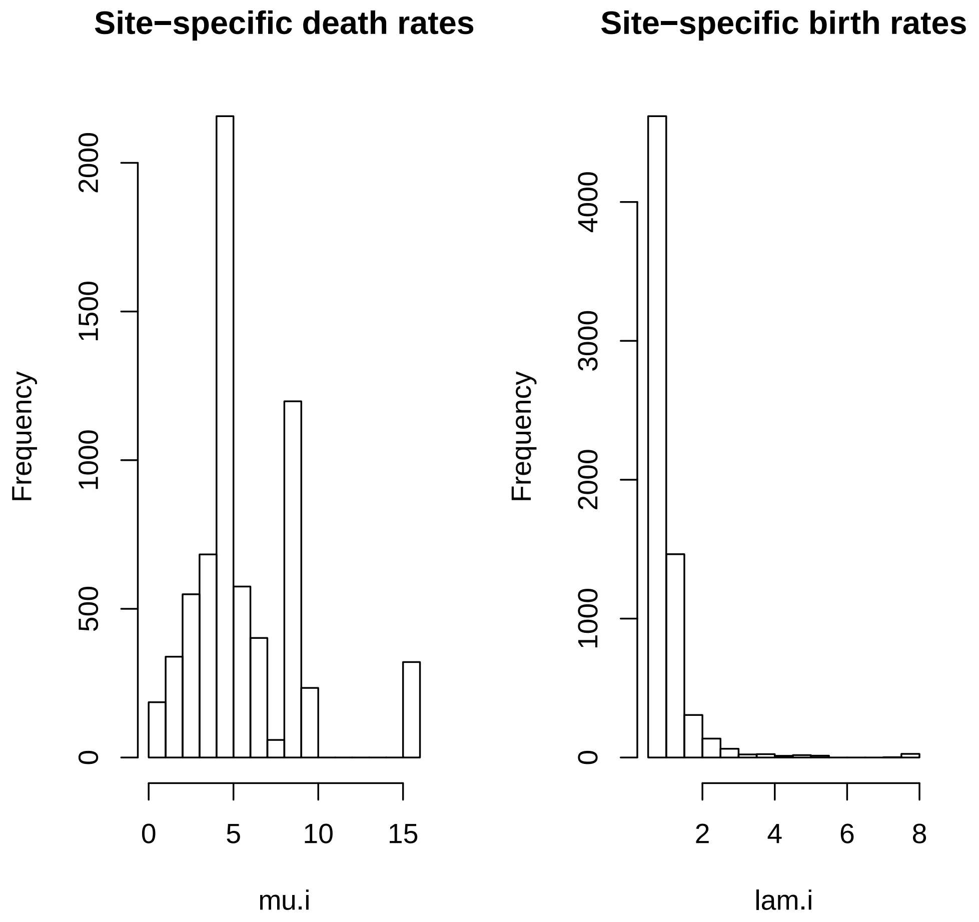
$$\mu^* = \frac{N^{21}}{W_2}, \quad \lambda^* = \frac{N^{12}}{W_1}.$$

We can compute the ABC MLE to obtain $(\hat{\mu}, \hat{\lambda})$ and use this in conjunction with the stochastic mapping quantities in an analogous way:

$$\tilde{\mu}_i = \frac{\mathbb{E}\left[N^{21}|D_i\right]}{\mathbb{E}\left[W_2|D_i\right]}, \quad \tilde{\lambda}_i = \frac{\mathbb{E}\left[N^{12}|D_i\right]}{\mathbb{E}\left[W_1|D_i\right]} \tag{2}$$

The following code implements equation 2. The key call is to the MultiMap_RAPI function, passing the appropriate masks and extracting the edge-wise jump and dwelling time estimates. The column structure of the matrix returned by MultiMap_RAPI is the likelihood, the jumps, and finally, the dwelling times.

```
> ## set mu and lam to the ABC MLE
> mu = opt.soln$par[1]
> lam = opt.soln$par[2]
> ## Set the transition rate matrix
> Q = matrix(
+   c(-lam, lam,
+     mu,  -mu),
+   nr=2, nc=2, byrow=T)
> ## compute stochastic map quantities, use unique tip vectors
> stmap.mu = .Call(
+   "MultiMap_RAPI",
+   tree.string,Q,
+   label21.mask, dwell2.mask,
+   uniq.tip.vecs,
+   NULL, FALSE
+   )
> stmap.lam = .Call(
+   "MultiMap_RAPI",
+   tree.string,Q,
+   label12.mask,dwell1.mask,
+   uniq.tip.vecs,
+   NULL,FALSE
+   )
> n.edges = (ncol(stmap.mu) - 1)/2
> n.sites = nrow(stmap.mu)
> ## explicitly define the column indices corresponding to
> ## jumps and dwelling time
> jump.col.idx = seq(2,len=n.edges)
> dwell.col.idx = seq(2+n.edges, len=n.edges)
> ## compute site specific birth/death rates
> lam.i = numeric(n.sites)
> mu.i = numeric(n.sites)
> for(siteidx in 1:n.sites)
+   {
+     N21 = sum(stmap.mu[siteidx,jump.col.idx])
+     W1 = sum(stmap.mu[siteidx,dwell.col.idx])
+     mu.i[siteidx] = N21/W1
+
+     N12 = sum(stmap.lam[siteidx,jump.col.idx])
+     W2 = sum(stmap.lam[siteidx,dwell.col.idx])
+     lam.i[siteidx] = N12/W2
+   }
> ## Use the reverse index to move back to original problem
> mu.i = mu.i[rev.idx]
> lam.i = lam.i[rev.idx]
```



6 A Discrete Gamma Model

Unfortunately, the treatment of site-specific birth and death rates in the previous section is a bit too ad-hoc for many statisticians. To address the inherently approximate nature of that approach, here we describe an alternative, fully model-based approach.

The basic idea is to model birth and death rates with a discretized gamma distribution. To fix ideas, we assume a model with three possible death rates and a single birth rate. The death rates are denoted $\mu_i = \mu R_i$, and R_i follows a discrete gamma distribution which we define below.

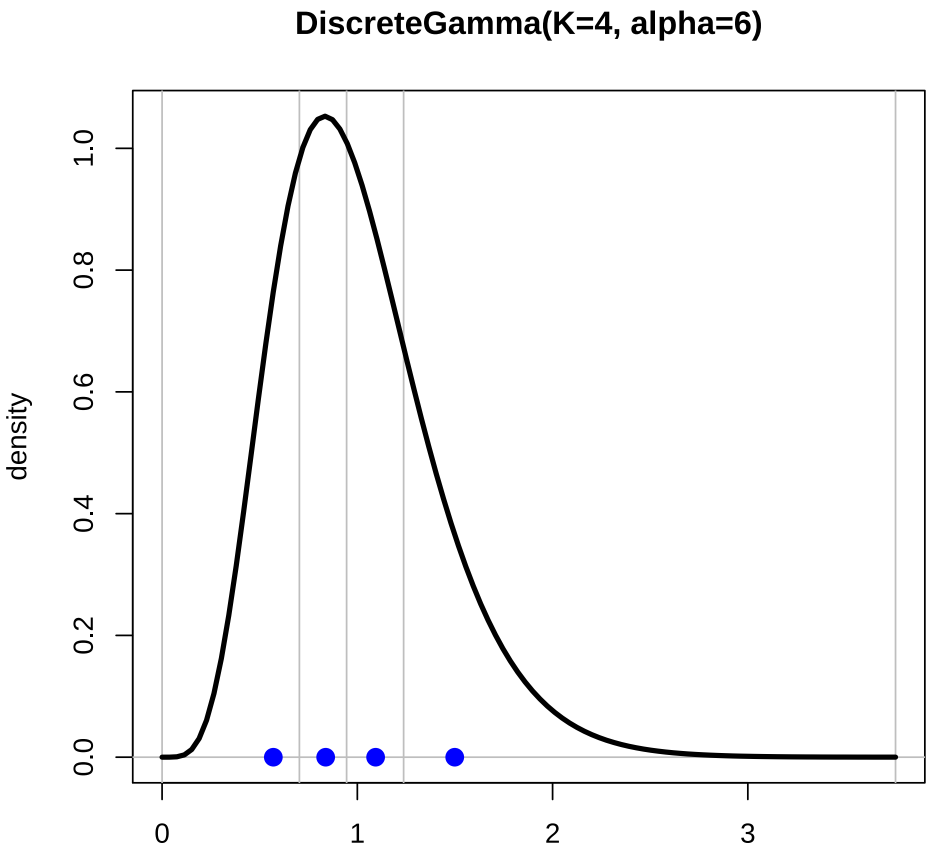
$$R_i \sim \text{DiscreteGamma}(\alpha, \alpha)$$

for $i \in \{1, ..., \text{n.sites}\}$, and α describes both the shape and rate parameters.

A discrete gamma with K bins is formed by computing the conditional median of R_i, given that r_i is between the i^{th} and $(i + 1)^{st}$ K-quantile.

```
> dg.support = function(K, alpha)
+   {
+     eps = 1e-6
+     pct = c(seq(0,K-1) / K, 1-eps)
+     med.pct = pct[1:K] + diff(pct) / 2
+     cond.median = qgamma(med.pct, shape=alpha, rate=alpha)
+     cond.median = cond.median / mean(cond.median)
+     return(cond.median)
+   }
```

For example, the next figure shows the construction for a DiscreteGamma with $K = 4$ and $\alpha = 6$. The black curve plots the usual gamma(6,6) density, and the vertical gray lines show the quartile boundaries. The conditional medians, shown as blue points on the axis, are the support of the corresponding discrete gamma distribution and are assigned equal probabilities.



6.1 Model Definition

The discrete gamma probabilistic model is described by three unknown parameters, λ, μ, and α. We modify the ABC Likelihood in equation 1 to incorporate the death rate mixing:

$$\begin{aligned} L_i(\Lambda, \alpha) &= \frac{\mathbb{P}\left[X = D_i; \Lambda, \alpha\right]}{\mathbb{P}\left(X_{hg18} = 2; \Lambda, \alpha\right)} \\ &= \frac{\sum_{k=1}^{3} \mathbb{P}\left[X_i = D_i | R_i = r_k\right] \mathbb{P}\left[R_i = r_k\right]}{\sum_{k=1}^{3} \mathbb{P}\left[X_{ih} = 2 | R_i = r_k\right] \mathbb{P}\left[R_i = r_k\right]} \\ &= \frac{\sum_{k=1}^{3} \mathbb{P}\left[X_i = D_i | R_i = r_k\right]}{\sum_{k=1}^{3} \mathbb{P}\left[X_{ih} = 2 | R_i = r_k\right]} \end{aligned} \quad (3)$$

Code to implement equation 3 follows:

```
> abcdg.lhood = function(params, K, my.data, counts)
+   {
+     mu = params[1]
+     lam = params[2]
+     alpha = params[3]
+
+     mu.vec = dg.support(K, alpha)*mu
+
+     num = vector("list", K)
+     den = vector("list", K)
+     for(ii in 1:K)
+       {
+         mu = mu.vec[ii]
+
+         Q = matrix(
+           c(-lam, lam,
+             mu,  -mu),
+           nr=2, nc=2, byrow=T
+           )
+
+         ## compute the probabilities in the numerators
+         num[[ii]] = .Call(
+           "MultiMap_RAPI",
+           tree.string,Q,
+           label21.mask, dwell2.mask,
+           my.data,
+           NULL, FALSE
+           )[,1]
+
+         ## compute the probabilities in the denominators using
+         ## the shortcut quoted in the abc.lhood code comment
+         den[[ii]] = rep(lam / (mu + lam), nrow(my.data))
+       }
+
+     num.sum = apply(do.call(cbind, num), 1, sum)
+     den.sum = apply(do.call(cbind, den), 1, sum)
+
+     lhood = (log(num.sum/den.sum)) %*% counts
+     return(lhood)
+   }
```

Optimization is done through another simple call to optim:

```
> dg.obj.fn = function(params, K, my.data, counts) -abcdg.lhood(params, K, my.data, counts)
> dg.opt.soln = optim(
+   par = c(opt.soln$par, 5),
+   dg.obj.fn,
+   K = 3,
+   my.data=uniq.tip.vecs,
+   counts = uniq.tip.counts,
+   lower=c(1e-4, 1e-4, 0.1),
+   method="L-BFGS-B",
+   hessian=TRUE
+   )
```

```
> ## Optimal parameter vector (mu, lam, alpha)
> dg.opt.soln$par

[1] 10.7562059  0.4296545  1.7933456

> ## Asymptotic 95% confidence intervals
> sig = sqrt(diag(solve(dg.opt.soln$hessian)))
> dg.opt.soln$par[1] + c(-1,1) * 1.96 * sig[1]

[1] 10.07595 11.43646

> dg.opt.soln$par[2] + c(-1,1) * 1.96 * sig[2]

[1] 0.3990288 0.4602802

> dg.opt.soln$par[3] + c(-1,1) * 1.96 * sig[3]

[1] 1.652450 1.934241
```

6.1.1 A comparison with site-specific rates

For comparison with the site-specific birth and death rates, the MLE bins of the DiscreteGamma give the following "allowed" death rates

```
> dg.support(3, dg.opt.soln$par[3]) * dg.opt.soln$par[1]

[1]  3.876962  9.456549 18.935107
```

and the birth rate

```
> dg.opt.soln$par[2]

[1] 0.4296545
```

These values match up reasonably well with the modes of the histograms in the site-specific rates discussion.

Interestingly, one counter argument to the fully, model-based approach described in the present section is that while we do have a model at the rates level, it's relatively coarse as it only allows three discrete death rates. It's a typical trade off between the unknown statistical behavior of a methodological shortcut and an overly simplistic, yet fully specified, model.

7 Final Thoughts

As mentioned earlier, this Sweave document is intended to support a pending publication by Sawaya, et. al. in "Genome Biology and Evolution." In particular, the authors of that paper want to support the paradigm of reproducible research.

The author of this work, Dustin Lennon, makes no guarantee that the results quoted here are, in any way, equivalent to those in the final, peer-reviewed paper. One substantive difference is that the results in this analysis used a small, potentially non-representative subset of the full data, namely tip strings extracted from the utr3 dataset.

References

[1] D Lennon, *McSMAC source code*, http://www.github.com/dnlennonpu01/mcsmac, May 2012.

[2] VN Minin and MA Suchard, *Fast, accurate and simulation-free stochastic mapping*, Philosophical Transactions of the Royal Society B: Biological Sciences **363** (2008), 3985–3995.

[3] SM Sawaya, D Lennon, E Buschiazzo, N Gemmel, and VN Minin, *Measuring microsatellite conservation in mammalian evolution with a phylogenetic birth-death model*, Genome Biology and Evolution (submitted May 2012).