Introduction to the precision R package

Richard Wilkinson

2016-07-14

Installation

The easiest way to install is to use devtools to install directly from github.

```
devtools::install_github('rich-d-wilkinson/precision')
```

Alternatively, download the package from https://github.com/rich-d-wilkinson/precision and install manually.

Data

All of the datasets used in the paper are included in the R package.

```
library(precision)
data(package='precision')$results[,c('Item')]
```

```
##  [1] "CflorusPrimary"                  "CflorusSecondary"
##  [3] "GlegneriPrimary"                 "GlegneriSecondary"
##  [5] "GthailandensisSecondary"         "MluteolusPrimary"
##  [7] "MluteolusSecondary"              "hyper (CflorusSecondary)"
##  [9] "hyper (GlegneriSecondary)"       "hyper (GthailandensisSecondary)"
## [11] "hyper (MluteolusSecondary)"
```

For example, the C. florus secondary dataset is

```
data(CflorusSecondary)
tail(CflorusSecondary)
```

```
##        n  m
## [48,] 15  6
## [49,] 19  4
## [50,] 21  3
## [51,] 22  7
## [52,] 23 10
## [53,] 24  5
```

To use your own dataset, specify a $C \times 2$ matrix, with the first column containing the clutch size, and the second the number of males. It is necessary to label your columns as n and m.

```
my_data <- matrix(c(3,2,4,3,5,1,6,2,7,1,7,1), nc=2, byrow=TRUE)
colnames(my_data) <- c('n', 'm')
my_data
```

Standard Analyses

The pre-existing analysis methods are all built into the R functions `Meelis.test` and `James.test`. For example,

```
(meelis.out <- Meelis.test(CflorusSecondary, TwoSided = TRUE))
```

```
## $vals
##       clutch size no. clutches     p hat binom var   obs var         R
##  [1,]           1            8 0.0000000 0.0000000 0.0000000       NaN
##  [2,]           2            6 0.4166667 0.4861111 0.1666667 0.3428571
##  [3,]           3            3 0.3333333 0.6666667 1.0000000 1.5000000
##  [4,]           4            5 0.2000000 0.6400000 0.2000000 0.3125000
##  [5,]           5            6 0.3000000 1.0500000 0.3000000 0.2857143
##  [6,]           6            2 0.4166667 1.4583333 0.5000000 0.3428571
##  [7,]           7            2 0.1428571 0.8571429 0.0000000 0.0000000
##  [8,]           8            2 0.2500000 1.5000000 2.0000000 1.3333333
##  [9,]           9            4 0.5833333 2.1875000 2.2500000 1.0285714
## [10,]          10            2 0.2500000 1.8750000 0.5000000 0.2666667
## [11,]          12            3 0.4722222 2.9907407 5.3333333 1.7832817
## [12,]          13            1 0.3076923 2.7692308 0.0000000        NA
## [13,]          14            1 0.7142857 2.8571429 0.0000000        NA
## [14,]          15            3 0.2666667 2.9333333 7.0000000 2.3863636
## [15,]          19            1 0.2105263 3.1578947 0.0000000        NA
## [16,]          21            1 0.1428571 2.5714286 0.0000000        NA
## [17,]          22            1 0.3181818 4.7727273 0.0000000        NA
## [18,]          23            1 0.4347826 5.6521739 0.0000000        NA
## [19,]          24            1 0.2083333 3.9583333 0.0000000        NA
##                M          V          U    p value
##  [1,]   0.000000  0.0000000        NaN        NaN
##  [2,]   6.818182  1.5426997 -1.4638501 0.07161745
##  [3,]   4.500000  1.6071429  0.3944053 0.65335909
##  [4,]   5.894737  2.5028516 -1.1976540 0.11552588
##  [5,]  18.931034  9.5124851 -1.2745587 0.10123273
##  [6,]  14.090909  4.6280992 -0.5070926 0.30604494
##  [7,]   2.923077  0.9940828 -0.9258201 0.17726974
##  [8,]   9.600000  4.3323077  0.1921765 0.57619803
##  [9,] 117.000000 27.7219251  0.0000000 0.50000000
## [10,]  14.473684  6.8437347 -0.5633235 0.28660732
## [11,] 102.485714 35.7355102  0.7551601 0.77492354
## [12,]  16.000000  0.0000000        NaN        NaN
## [13,] 100.000000  0.0000000        NaN        NaN
## [14,]  54.000000 33.4883721  1.3824294 0.91658006
## [15,]  16.000000  0.0000000        NaN        NaN
## [16,]   9.000000  0.0000000        NaN        NaN
## [17,]  49.000000  0.0000000        NaN        NaN
## [18,] 100.000000  0.0000000        NaN        NaN
## [19,]  25.000000  0.0000000        NaN        NaN
##
## $R.av
## [1] 0.7532786
##
## $s2
## [1] 1.181833
```

```
##
## $U.av
## [1] -0.9672868
##
## $p.av
## [1] 0.3334007
##
## $exp.table
##
##      0 1 2 3 4 5 6 7 10
##   1  8 0 0 0 0 0 0 0  0
##   2  1 5 0 0 0 0 0 0  0
##   3  1 1 1 0 0 0 0 0  0
##   4  1 4 0 0 0 0 0 0  0
##   5  0 3 3 0 0 0 0 0  0
##   6  0 0 1 1 0 0 0 0  0
##   7  0 2 0 0 0 0 0 0  0
##   8  0 1 0 1 0 0 0 0  0
##   9  0 0 0 0 2 0 1 1  0
##   10 0 0 1 1 0 0 0 0  0
##   12 0 0 0 1 0 0 0 2  0
##   13 0 0 0 0 1 0 0 0  0
##   14 0 0 0 0 0 0 0 0  1
##   15 0 1 0 0 0 1 1 0  0
##   19 0 0 0 0 1 0 0 0  0
##   21 0 0 0 1 0 0 0 0  0
##   22 0 0 0 0 0 0 0 1  0
##   23 0 0 0 0 0 0 0 0  1
##   24 0 0 0 0 0 1 0 0  0
```

```
(james.out <- James.test(CflorusSecondary, TwoSided = TRUE))
```

```
## $U
## [1] 2.70893
##
## $p.val
## [1] 0.006750059
##
## $exp.table
##
##      0 1 2 3 4 5 6 7 10
##   1  8 0 0 0 0 0 0 0  0
##   2  1 5 0 0 0 0 0 0  0
##   3  1 1 1 0 0 0 0 0  0
##   4  1 4 0 0 0 0 0 0  0
##   5  0 3 3 0 0 0 0 0  0
##   6  0 0 1 1 0 0 0 0  0
##   7  0 2 0 0 0 0 0 0  0
##   8  0 1 0 1 0 0 0 0  0
##   9  0 0 0 0 2 0 1 1  0
##   10 0 0 1 1 0 0 0 0  0
##   12 0 0 0 1 0 0 0 2  0
##   13 0 0 0 0 1 0 0 0  0
##   14 0 0 0 0 0 0 0 0  1
```

```
##   15 0 1 0 0 0 1 1 0  0
##   19 0 0 0 0 1 0 0 0  0
##   21 0 0 0 1 0 0 0 0  0
##   22 0 0 0 0 0 0 0 1  0
##   23 0 0 0 0 0 0 0 0  1
##   24 0 0 0 0 0 1 0 0  0
```

From this we can see the test statistics for the Meelis and James' tests, as well as the corresponding p-values. The value of R and McCullagh's s^2 are included in the output from `Meelis.test`.

Bayesian analysis

The Bayesian analysis consists of two parts. The first is finding the posterior distributions of the parameters. The second optional stage is to go on to estimate the Bayes factors. These calculations require us to run an MCMC sampler, which can be computationally intensive depending on how long it is run for. The longer it is run, the more accurate the calculations are likely to be.

The calculations all require the specification of prior distributions. The family of distributions used for each parameter is hard coded into the package, but the user is free to choose the hyper-parameters that define the mean and variance of the distribution. The priors used are

$$
\begin{aligned}
p &\sim \text{Beta}(a_p, b_p) \\
\psi &\sim N(\mu, \sigma^2) \\
\lambda &\sim \text{Gamma}(\alpha, \beta) \\
d &\sim \text{Beta}(a_d, b_d)
\end{aligned}
$$

We specify all of these through a list.

```
hyper<-list()
```

The elements of the list must use the naming convention used below. A reasonable default choice of prior for p and ψ (see paper for the rationale) is to use $p \sim U[0,1]$ and $\psi \sim N(0,1)$, which we can set as follows:

```
hyper$a.p <-  1
hyper$b.p <-  1
hyper$mu.psi <- 0
hyper$sd.psi <- 1
```

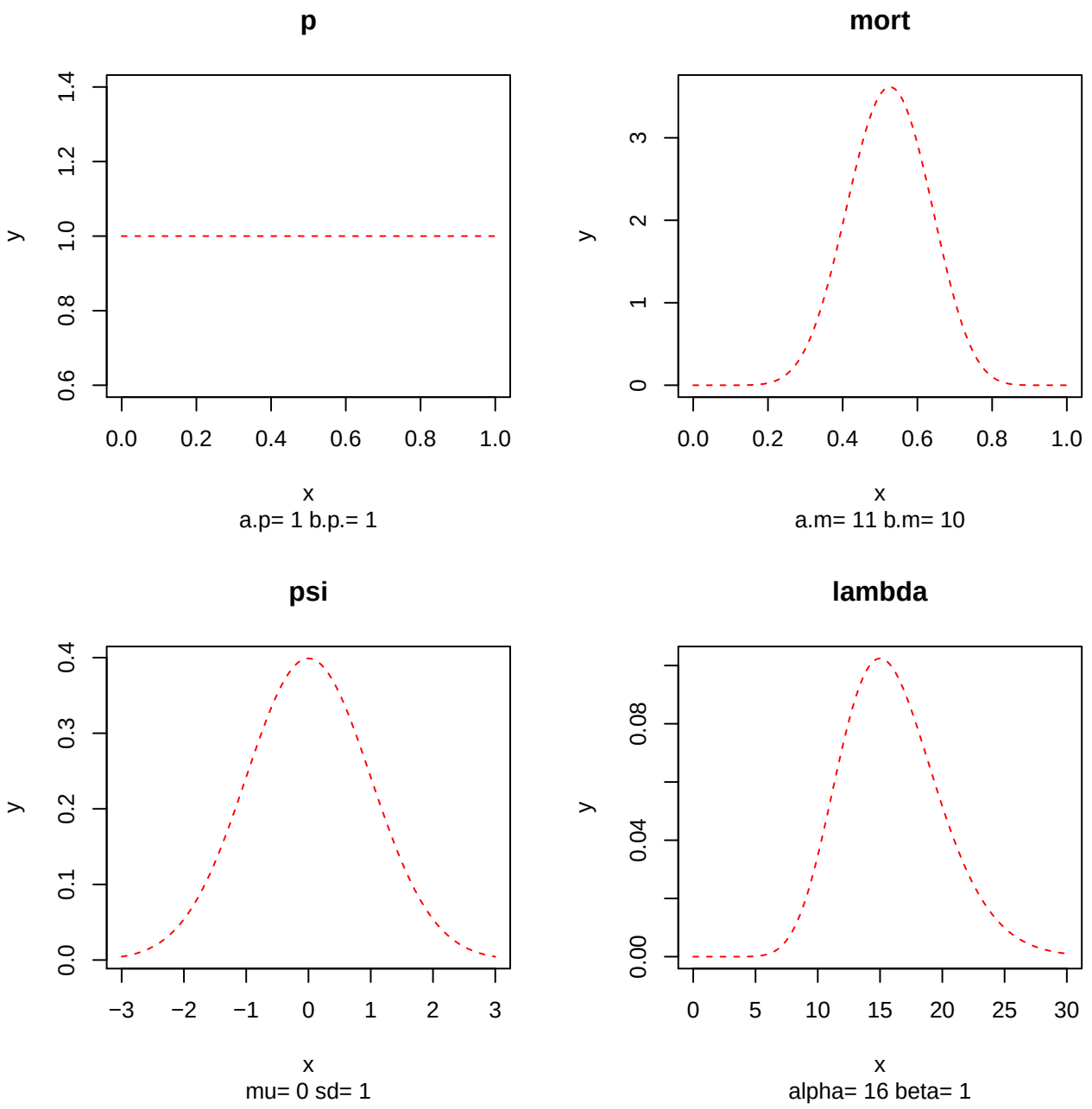
For C. florus, previous work has reported a mortality rate of 57% and an average clutch size of 7.4. Some experimentation with the values, and recalling that the mean of a Gamma(α, β) distribution is α/β and the variance is α/β^2, led us to use

```
hyper$a.m <- 11
hyper$b.m <-  10

hyper$alpha.lambda <- 16
hyper$beta.lambda  <- 1
```

It is a good idea to plot the prior distributions, to check that they agree with prior beliefs. This can be done as follows:

```
plot.prior(hyper=hyper, show=TRUE, family="multbinom")
```




```
## pdf
##   2
```

Posteriors

To calculate the posterior distribution, we have to run an MCMC sampler for a larger number of iterations. The longer we run the sampler, the better the posterior estimates will be. We would suggest a minimum of 10^5 iterations to get a reasonable estimate of the posteriors, and that 10^6 iterations should be more than sufficient. If Bayes factors are to be estimated, we would err towards the higher end of that range. The run

time will depend upon both the number of MCMC iterations used, and the number of clutches in the dataset (as the MCMC algorithm samples the unobserved primary counts). To do 10^6 iterations with the C. florus dataset, you should expect to wait about an hour, depending on processor speed, for each set of MCMC results.

```
nbatch <- 1*10^6
```

Binomial Model

The proceedure for fitting each of the three models (binomial, multiplicative binomial, and double binomial) is the same, and each can be done independently (on different cores if possible). To begin with, we choose a start point for the MCMC chain. The chains mix well and so a random value chosen from the prior works well here. It is necessary to label the parameters in the parameter matrix

```
b.theta0 <-c("lambda"=10, "p"=0.1, "mort"=0.5)
```

To run the code, we then just call the `MCMCWithinGibbs` function. Note that you can specify whether to keep the imputed missing primary values (the N and M values).

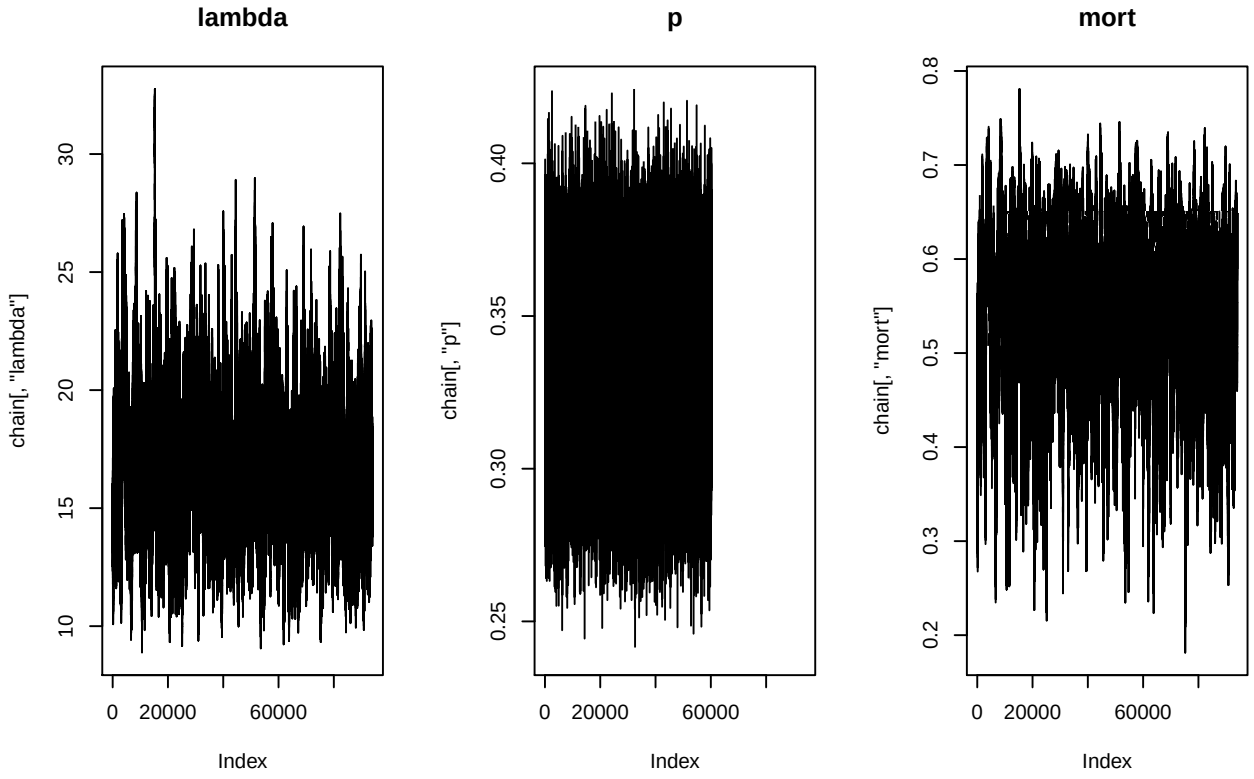
```
b.mcmc.out <- MCMCWithinGibbs(theta0=b.theta0,  data=CflorusSecondary, hyper=hyper, nbatch=nbatch, fami
```

Finally, it can often be a good idea to thin the MCMC output (by only keeping every 10th value for example) and to discard an initial 'burn-in' period.

```
b.mcmc.out.t  <- ThinChain(b.mcmc.out, thinby=10, burnin=10^5)
```

The trace plots are useful to ensure that the chains have converged, and that they are mixing well.

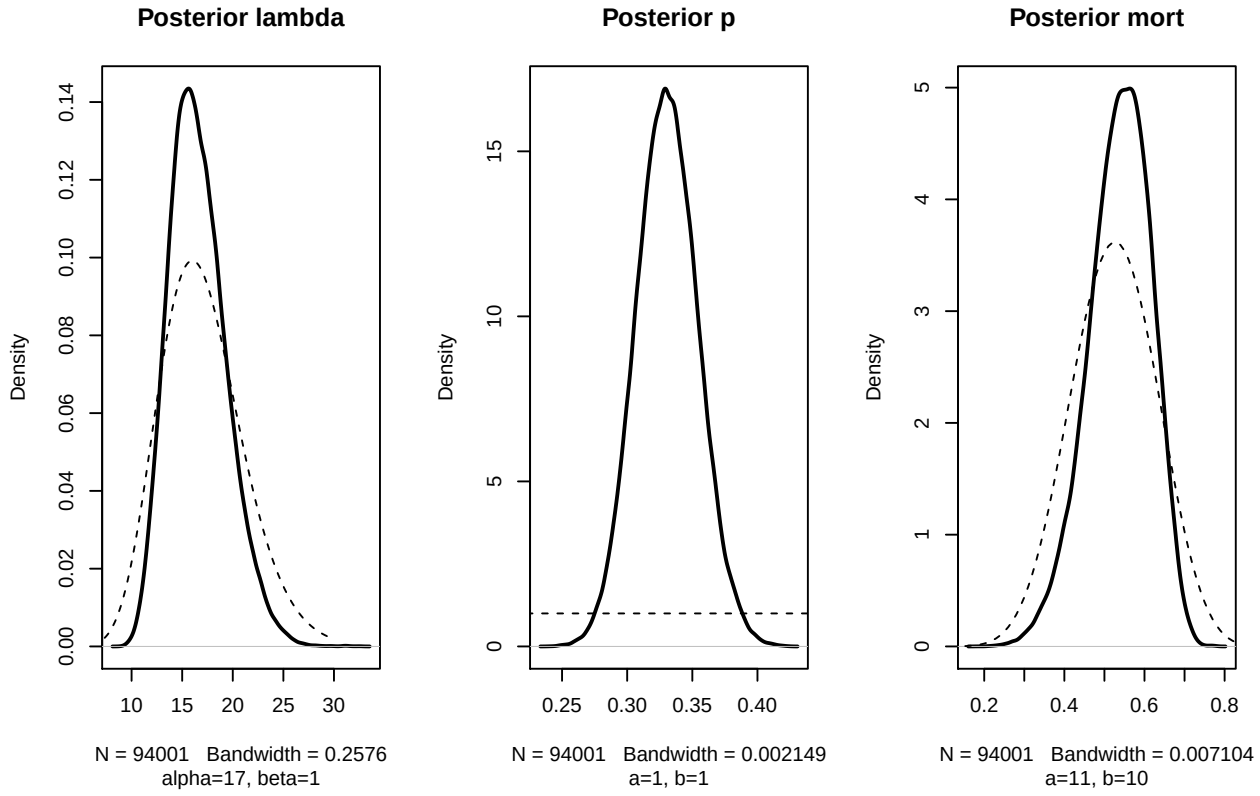
```
plot.trace(chain=b.mcmc.out.t$chain, show=T, family="binomial")
```



```
## pdf
##   2
```

These all look fine, and so we can plot the posteriors and draw conclusions:

```
plot.posterior(chain=b.mcmc.out.t$chain, hyper=hyper, show=T, family="binomial")
```




```
## pdf
##   2
```

Multiplicative and Double Binomial Models

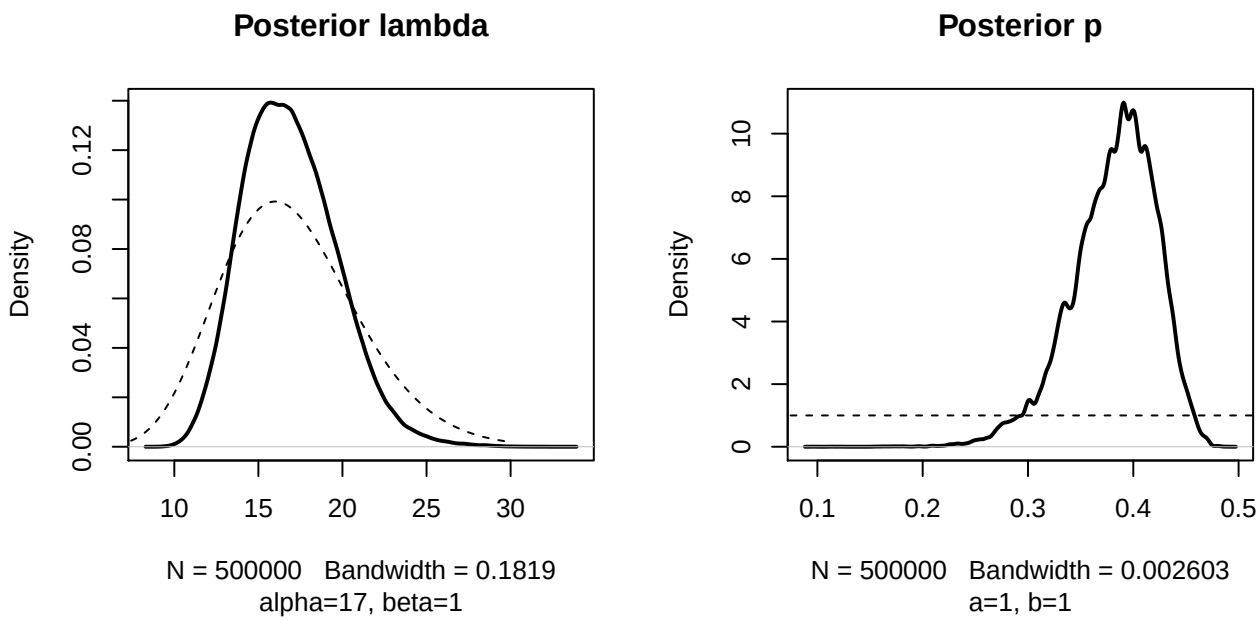
The process for fitting the other models is very similar. However now we are forced to use Metropolis-Hastings as well as a Gibbs sampler, and so we need to specify the Metropolis-Hastings random walk step size.

```
m.step.size<-c('p.logit'=0.3, 'psi'=0.2)
```

Note again that it is necessary to name the elements in this vector to avoid ambiguity. The rest of the code is the same as for the binomial model:

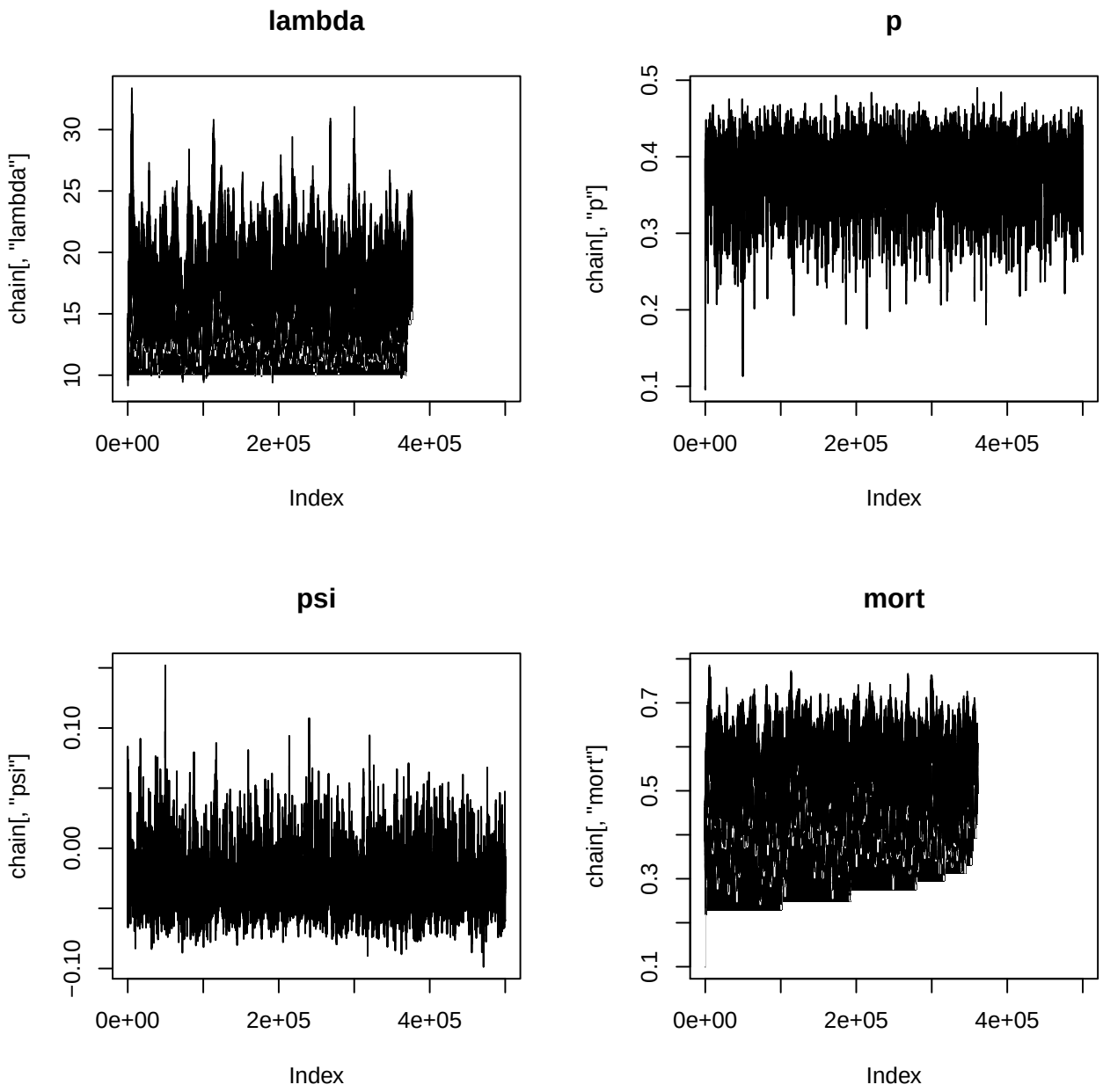
```
m.theta0 <-c('lambda'=10, 'p'=0.1,'psi'=0, 'mort'=0.1)
m.mcmc.out <- MCMCWithinGibbs( theta0=m.theta0,  data=GlegneriSecondary, hyper=hyper, nbatch=nbatch,  fa
m.mcmc.out.t  <- ThinChain(m.mcmc.out, thinby=10, burnin=10^5)
```

```
plot.posterior(chain=m.mcmc.out.t$chain, hyper=hyper, show=T,   family="multbinom")
```




```
## pdf
##   2
```

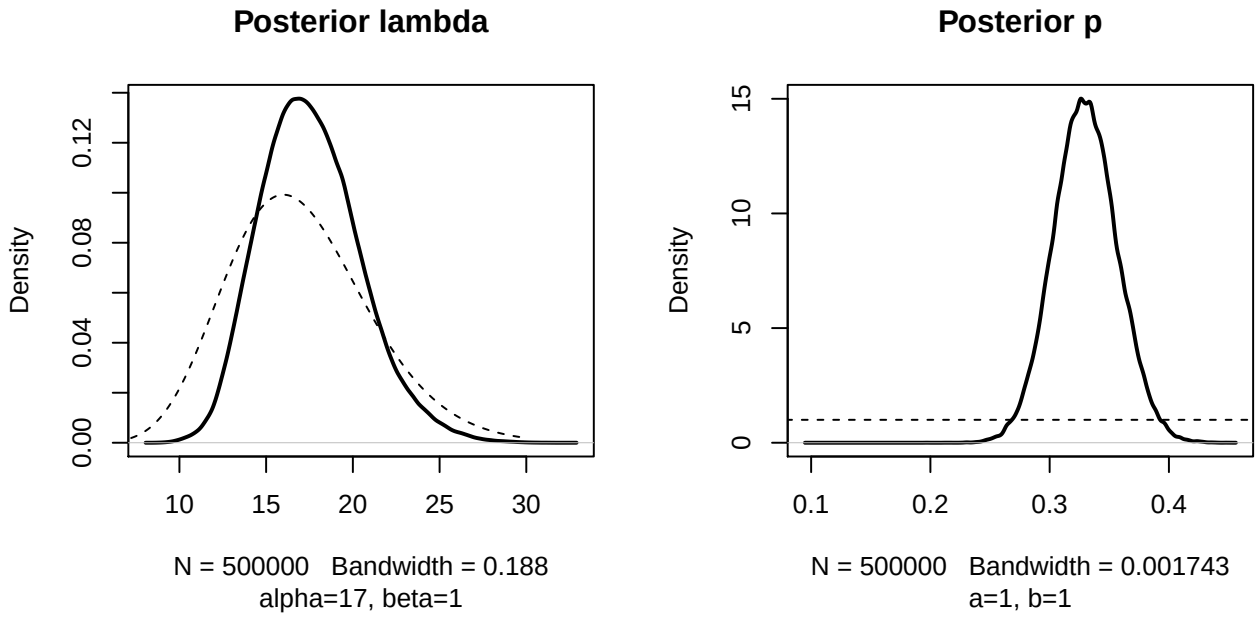
```
plot.trace(chain=m.mcmc.out.t$chain, show=T, family="multbinom")
```




```
## pdf
##   2
```

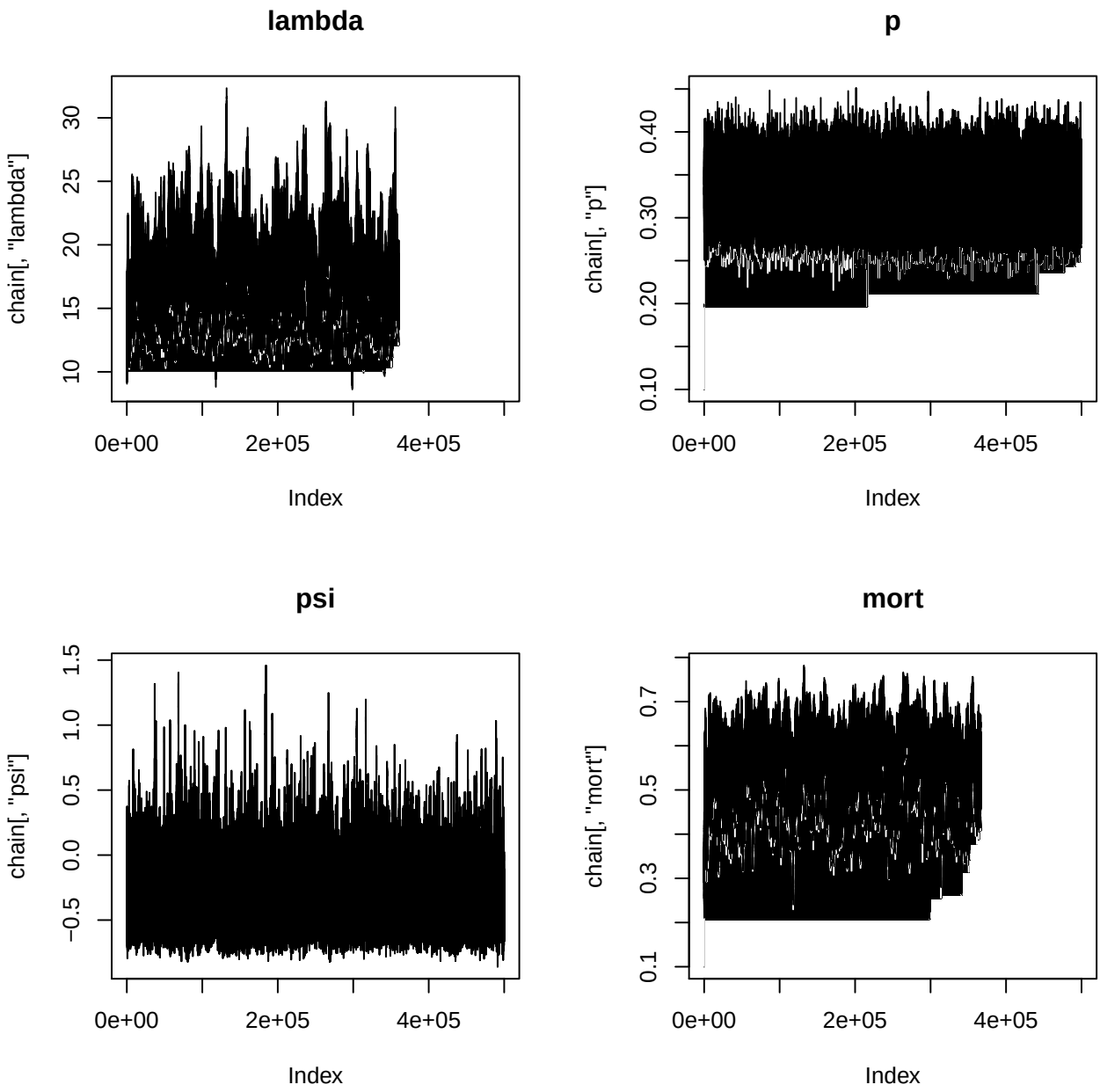
```
d.step.size<-c('p.logit'=0.3, 'psi'=0.2)
d.theta0 <-c('lambda'=10, 'p'=0.1,'psi'=0, 'mort'=0.1)
d.mcmc.out <- MCMCWithinGibbs( theta0=d.theta0,  data=GlegneriSecondary, hyper=hyper, nbatch=nbatch,  fa
d.mcmc.out.t  <- ThinChain(d.mcmc.full, thinby=1, burnin=10^4)
```

```
plot.posterior(chain=d.mcmc.out.t$chain, hyper=hyper, show=TRUE, family="doublebinom")
```




```
## pdf
##   2
```

```
plot.trace(chain=d.mcmc.out.t$chain, show=TRUE, family="doublebinom")
```




```
## pdf
##   2
```

Bayes factors

The posterior distributions give much of the information about how much under-dispersion there is in the data. However, often we will want to also calculate the Bayes factor to see whether the data support one model over the others. To do this, we have to run additional MCMC chains fixing some of the parameters (and so this step is also computationally costly).

```
b.log.evidence <- CalculateEvidence(mcmc.out=b.mcmc.out.t, data=GlegneriSecondary,  hyper=hyper, family=
m.log.evidence <- CalculateEvidence(mcmc.out=m.mcmc.out.t, data=GlegneriSecondary,  hyper=hyper, family=
d.log.evidence <- CalculateEvidence(mcmc.out=d.mcmc.out.t, data=GlegneriSecondary,  hyper=hyper, family=
```

Finally, we can put all the information together in a nice format as follows:

```
log.evidence <- c(b.log.evidence, m.log.evidence, d.log.evidence)
BF<-CalcBF(log.evidence)
chib.out <- list(BF=BF$BF, probH0 = BF$probH0 ,
                 ProbPosPsi = c("multbinom"=sum((m.mcmc.out.t$chain[,"psi"]>0))/length(m.mcmc.out.t$cha
                 log.BF=log(BF$BF), log.evidence=log.evidence,
                 R= c("R"=meelis.out$R.av),
                 s2 = c(meelis.out$s2),
                 meelis = c("U"=meelis.out$U.av, "p"=meelis.out$p.av,  "conclusion"=ifelse(meelis.out$p
                 james = c("U"=james.out$U, "p"=james.out$p.val,  "conclusion" = ifelse(james.out$p.val
print(chib.out)
```

```
## $BF
##        mb        db        dm
## 0.3620485 0.2687031 0.7421744
##
## $probH0
##    binomial   multbinom doublebinom
##   0.6132142   0.2220133   0.1647726
##
## $ProbPosPsi
##   multbinom doublebinom
##    0.075386    0.089588
##
## $log.BF
##        mb        db        dm
## -1.015977 -1.314148 -0.298171
##
## $log.evidence
## [1] -307.7081 -308.7241 -309.0222
##
## $R
##         R
## 0.7532786
##
## $s2
## [1] 1.181833
##
## $meelis
##                     U                     p           conclusion
## "-0.967286848797403"  "0.333400656210468"           "AcceptH0"
##
## $james
##                      U                      p            conclusion
##    "2.70892995669601" "0.00675005884011858"            "RejectH0"
```

From this we can read off the Bayes factors (which show that the binomial model is slightly favoured here), the posterior probabilities of each model, the posterior probability that ψ is positive (which is only 0.075 and 0.090 for the multiplicative and double binomial models respectively), as well as the other descriptive statistics previously used.