EpiBayes 2-Level Models Vignette

Matt Branan

Updated: June 23, 2015

1 Setting up the Models in R

Run this code once to load the proper packages.

```
library(epiR)  # For the BetaBuster function
library(compiler)  # To compile the larger functions for computational speed
library(coda)  # For processing Bayesian model output
library(shape)  # For nice colorbar legends
library(scales)  # For transparent colors
library(EpiBayes)  # Load our package
```

Next, we will use the hierarchical Bayesian model to investigate a 2-level sampling design in which we have one region with a single subzone of interest from which we sample a number of clusters and, from those clusters, we sample a number of subjects of interest. We then investigate the output of both the no storage model (`EpiBayes_ns`) and the storage model (`EpiBayes_s`) using the appropriate methods for the objects output by those two functions that are included in the `EpiBayes`.

We also preview ways in which the user may implement functions from the `coda` package and post-process the output of these models by hand. Additionally, we preview the use of the `epi.betabuster` function from the `epiR` package so that one may use expert opinion to elicit beta prior distributions in the Bayesian model.

2 Examples

2.1 Example 1: Simulation-based Inference

Suppose that we have a situation much like that in our mollusk application with a few alterations. We assume that we have 40 farms on which we have 100 mollusks sampled on 350 of them and 500 mollusks sampled on the remaining 5. Of these, we assume that the first ten farms were sampled in summer, the next ten in fall, the next ten in winter, and the last ten (including the 5 farms with 500 mollusks from each) in spring. These seasons had the following modes of

- Summer: 0.50
- Fall: 0.50
- Winter: 0.02
- Spring: 0.02

The disease we are studying is moderately contagious and we expect about 50% of mollusks on infected farms to have the disease with these prevalences on infected farms being relatively variable (say, 20%-80% prevalence 95% of the time). We also assume that our sensitivity is around 10% and the specificity is near 100%.

Our approach here is that we want to assume that the disease is present in the area and show that we can actually detect the disease being above a threshold (design prevalence) of 2% with 95% probability. This means that we will assume that the disease is truly in the region so we set the corresponding variable `gammat` near 100%. We also want to let the data do most of the work so we will assume a non-informative, uniform prior for the cluster - level prevalence (`taumat`).

In this first example, we will use the model as a simulation model and simulate data under the prior parameters and a specified true cluster-level prevalence, run each simulated data set through the Bayesian model, and find out how often we conclude we find the cluster-level prevalence above 2% with 95% probability (that is, we will pay attention to p4.tilde in the output).

Note that any prior distributions for beta random variables (seasonal subject-level prevalences, `gammat`, `taumat`, `mumat`, `etamat`, `thetamat`) can be computed using BetaBuster like below. Since we know that we want sensitivity (`etamat`) about 10%, we can use the code:

```
epi.betabuster(0.10, 0.95, FALSE, 0.15)

## $shape1
## [1] 15.412
##
## $shape2
## [1] 130.708
##
## $mode
## [1] 0.1
##
## $mean
## [1] 0.105475
##
## $median
## [1] 0.103675
##
## $lower
## [1] 0.06120444
##
## $upper
## [1] 0.1599439
##
## $variance
## [1] 0.0006413131
```

to get the parameters of a beta distribution with mode 10% and a 95^{th} percentile of 15%. The first two, `shape1` and `shape2`, are just those two parameters that describe such a beta distribution.

Once the prior distributions have been decided upon, we may call the actual model – we'll be using the storage model in this case just so we can check some of the posterior distributions if we would like to. The function call will look something like the following. We have included annotations next to each argument so that it is clear what each argument is and why it had been initialized as such.

```
set.seed(2015)  # To ensure reproducible results
example1.run = EpiBayes_s(
    H = 1,  # 1 subzone
    k = rep(40, 1),  # 40 farms total
    n = c(rep(100, 35), rep(500, 5)),  #100
        # mollusks sampled in 35 farms and 500 sampled in the remaining 5 farms
    seasons = rep(c(1, 2, 3, 4), each = 10),
        # Seasons corresponding to each cluster
        # (1 for  summer,  2 for  fall,  3 for  winter,  4 for  spring)
        mumodes = matrix(c(
                0.50, 0.70,
                0.50, 0.70,
                0.02, 0.50,
                0.02, 0.50
                ), 4, 2, byrow = TRUE
        ),  # Modes and 95th percentiles of
        #subject - level prevalences for  each season in order
    reps = 10,  # 10 replicated data sets in this simulation
    MCMCreps = 100,  # 100 MCMC iterations per replicated data set
        # (increasing this would be a good idea for  real data but
        # slows things down a lot)
    poi = "tau",  # Want inference on cluster-level prevalence
    y = NULL,  # Leave this as NULL if we are doing simulation and not posterior
        # inference with a particular data set
    pi.thresh = 0.10,  # Have a 10% within-cluster design prevalence
    tau.thresh = 0.02,  # Have a 2% between-cluster design prevalence
    gam.thresh = 0.10,  # Doesn't matter since we have a 2-level model
    tau.T = 0.20,  # The "true cluster - level prevalence" that we simulate our data
        # with (this means about 20% of our clusters in each replicated data set
                # will be diseased and will have a truly positive subject -
                # level prevalence)
    poi.lb = 0,  # The lower bound for  estimating the cluster - level
        # prevalence (not of interest here)
    poi.ub = 1,  # The upper bound for  estimating the cluster - level
        # prevalence (not of interest here)
    p1 = 0.95,  # The probability (used like a confidence) that we must show our
        # cluster - level prevalence is above 2% in order to count that replicated
        # data set as one in which we detected the disease
    psi = 4,  # The variability of the prevalences among infected clusters
        # within the subzone
    omegaparm = c(1000, 1),  # Prior parameters for omegamat (the probability
        # of the disease being in the region)
    gamparm = c(1000, 1),  # Prior parameters for  gammat (the subzone-level
        # prevalence)
    tauparm = c(1, 1),  # Prior parameters for  taumat (the cluster - level
        # prevalence)
    etaparm = c(15, 130),  # Prior parameters for  etamat (the diagnostic test
```

```
    # sensitivity)
  thetaparm = c(1000, 1),  # Prior parameters for  thetamat (the diagnostic
    # test specificity)
  burnin = 10  # The amount of MCMC iterations to "burn"
  )
```

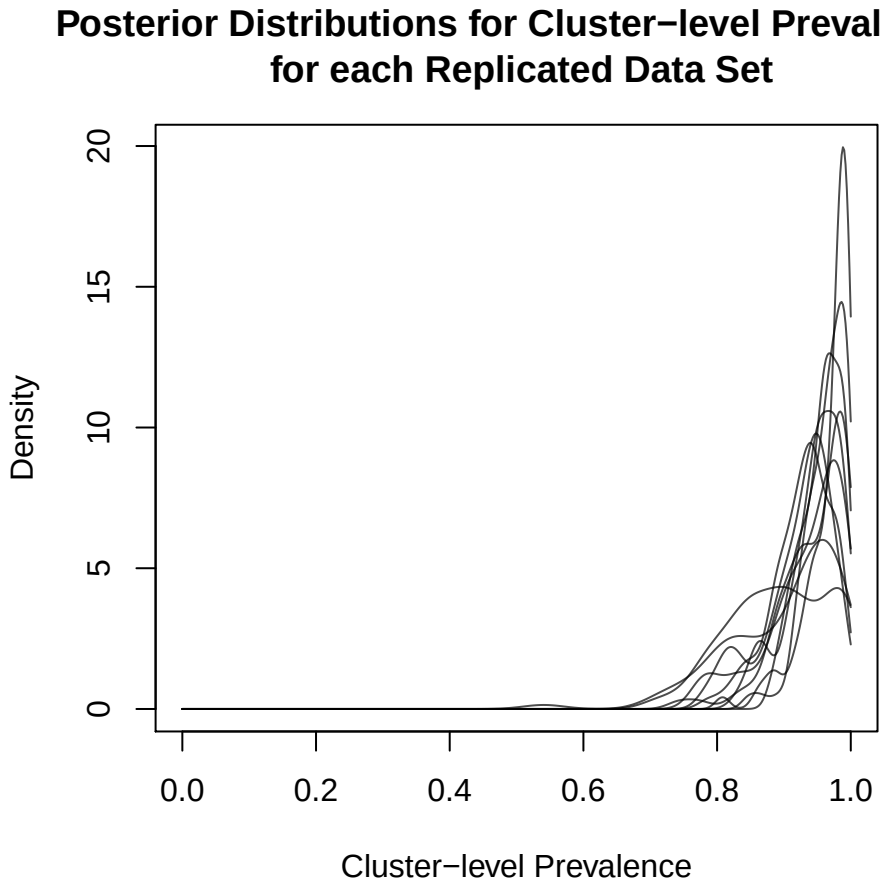
Once finished, we can look at summaries of the output. The `summary` method will return the simulation output (`p2.tilde`, `p4.tilde`, and `p6.tilde`), followed by summary values and highest posterior density (HPD) intervals for each of the parameters stored by the function. We can define how many replicated data sets we actually report summary measures for with a specification of the argument `n.output` (which defaults to 10 when `reps` is greater than 10). We can also use the `plot` method to plot the posterior distributions for the parameter the user chose to be `poi` (either the cluster-level or the subzone-level prevalence) for every replicated data set.

```
## Summary
example1.sum = summary(example1.run, n.output = 5)

## Simulation output for parameter of interest (poi)
##  *p2.tilde: Percentage of the time the disease is not detected above the disease threshold
##  *p4.tilde: Percentage of the time the disease is detected above the disease threshold
##  *p6.tilde: Percentage of the time the disease is detected between the user-supplied lower a
##
##  p2.tilde p4.tilde p6.tilde
##         0        1        1
##
##  ------------------------------------------------------------------------------
## gam: Probability of disease being in the region
##           Mean           SD     Naive SE Time-series SE Lower HPD Limit
## [1,] 0.9989854 0.0010186679 1.073770e-04   1.073770e-04       0.9967271
## [2,] 0.9990614 0.0007905371 8.332992e-05   8.332992e-05       0.9973818
## [3,] 0.9988807 0.0011317586 1.192978e-04   1.905485e-04       0.9965609
## [4,] 0.9989096 0.0009028008 9.516356e-05   9.516356e-05       0.9965480
## [5,] 0.9987604 0.0012712651 1.340031e-04   1.340031e-04       0.9956564
##      Upper HPD Limit
## [1,]       0.9999944
## [2,]       0.9999948
## [3,]       0.9999971
## [4,]       0.9999577
## [5,]       0.9999933
##
##  ------------------------------------------------------------------------------
## tau 1: Cluster-level prevalence in subzone 1
##           Mean         SD    Naive SE Time-series SE Lower HPD Limit
## [1,] 0.9280318 0.04431050 0.004670737    0.006923991       0.8332107
## [2,] 0.9025605 0.08463441 0.008921250    0.036217626       0.7408714
## [3,] 0.9644011 0.03023153 0.003186683    0.003186683       0.9136501
## [4,] 0.9334913 0.05894896 0.006213766    0.015804479       0.7874402
## [5,] 0.8966697 0.07419434 0.007820770    0.039807720       0.7701314
```

```
##      Upper HPD Limit
## [1,]       0.9906361
## [2,]       0.9995856
## [3,]       0.9998803
## [4,]       0.9969126
## [5,]       0.9969722
##
##  ------------------------------------------------------------------------------

example1.sum

## Simulation output for parameter of interest (poi)
##  *p2.tilde: Percentage of the time the disease is not detected above the disease threshold
##  *p4.tilde: Percentage of the time the disease is detected above the disease threshold
##  *p6.tilde: Percentage of the time the disease is detected between the user-supplied lower a
##
##  p2.tilde p4.tilde p6.tilde
##         0        1        1
##
##  ------------------------------------------------------------------------------
## gam: Probability of disease being in the region
##           Mean           SD     Naive SE Time-series SE Lower HPD Limit
## [1,] 0.9989854 0.0010186679 1.073770e-04   1.073770e-04       0.9967271
## [2,] 0.9990614 0.0007905371 8.332992e-05   8.332992e-05       0.9973818
## [3,] 0.9988807 0.0011317586 1.192978e-04   1.905485e-04       0.9965609
## [4,] 0.9989096 0.0009028008 9.516356e-05   9.516356e-05       0.9965480
## [5,] 0.9987604 0.0012712651 1.340031e-04   1.340031e-04       0.9956564
##      Upper HPD Limit
## [1,]       0.9999944
## [2,]       0.9999948
## [3,]       0.9999971
## [4,]       0.9999577
## [5,]       0.9999933
##
##  ------------------------------------------------------------------------------
## tau 1: Cluster-level prevalence in subzone 1
##           Mean         SD    Naive SE Time-series SE Lower HPD Limit
## [1,] 0.9280318 0.04431050 0.004670737    0.006923991       0.8332107
## [2,] 0.9025605 0.08463441 0.008921250    0.036217626       0.7408714
## [3,] 0.9644011 0.03023153 0.003186683    0.003186683       0.9136501
## [4,] 0.9334913 0.05894896 0.006213766    0.015804479       0.7874402
## [5,] 0.8966697 0.07419434 0.007820770    0.039807720       0.7701314
##      Upper HPD Limit
## [1,]       0.9906361
## [2,]       0.9995856
## [3,]       0.9998803
## [4,]       0.9969126
## [5,]       0.9969722
```

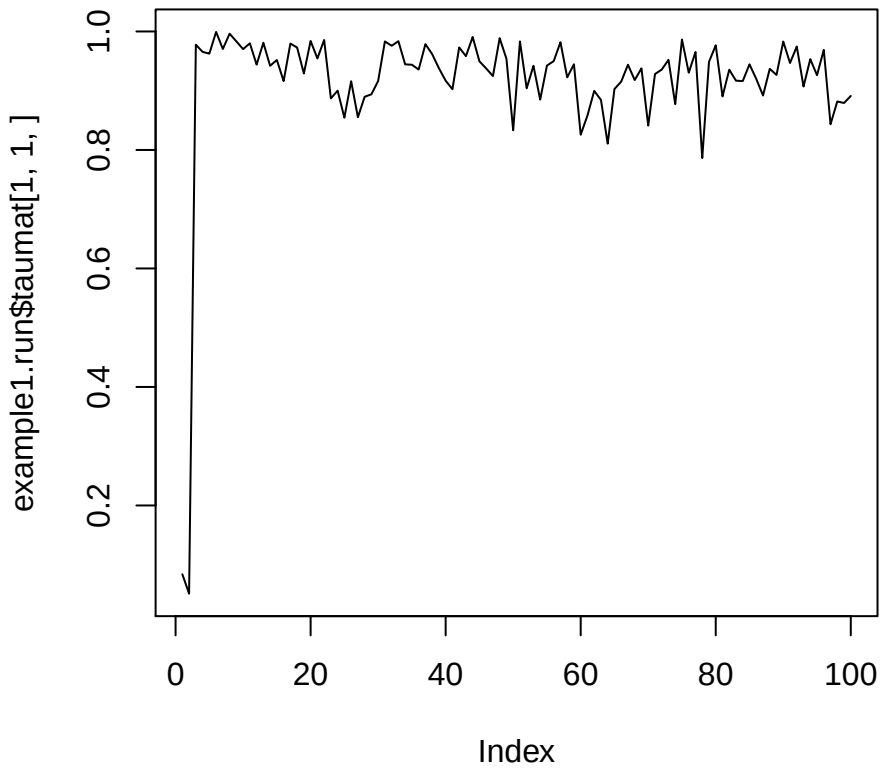
```
##
##  ------------------------------------------------------------------------------
```

```
## Plot the posterior distributions of cluster-level prevalence
plot(example1.run)
```

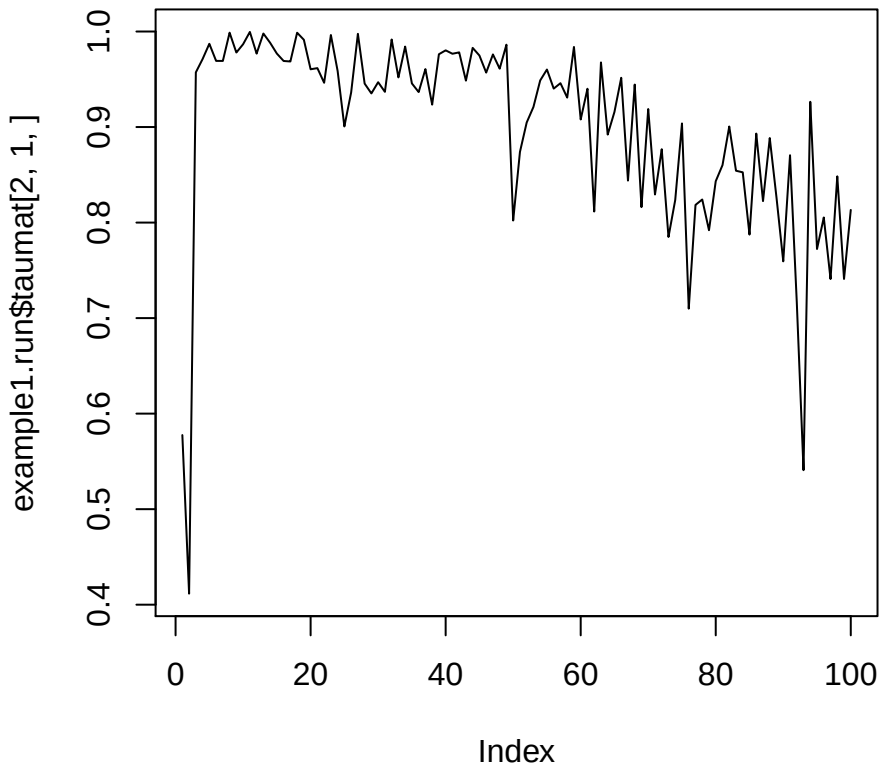


We can also look at some trace plots, ignoring burnin, for some of the `taumat` and some of the `pimat` chains.

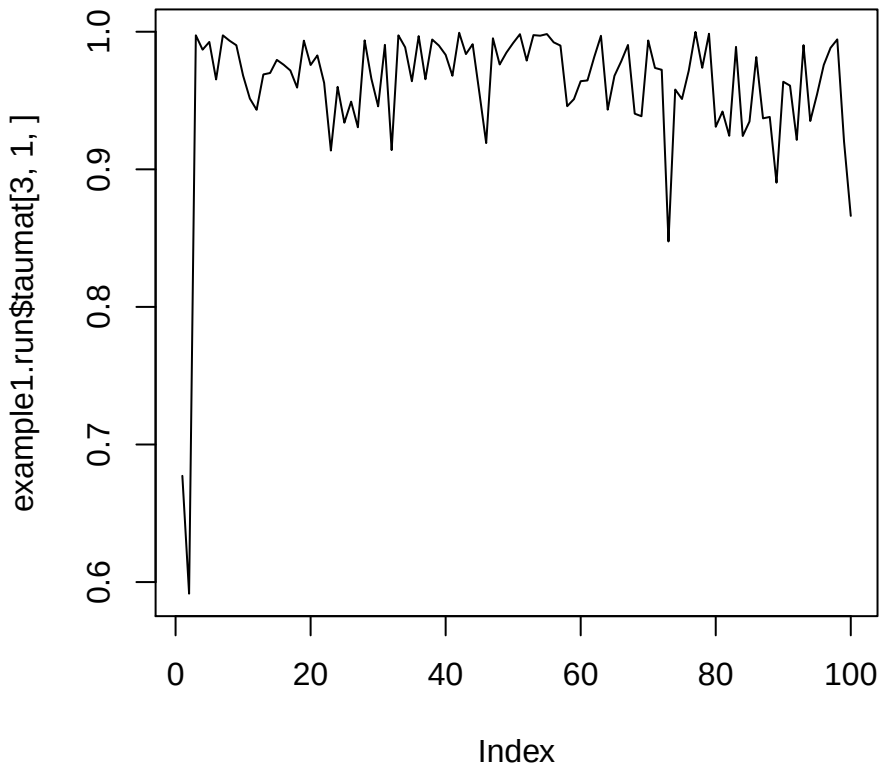
```
## Tau
        # Tau for  first replicated data set
        plot(example1.run$taumat[1, 1, ], type = "l")
```



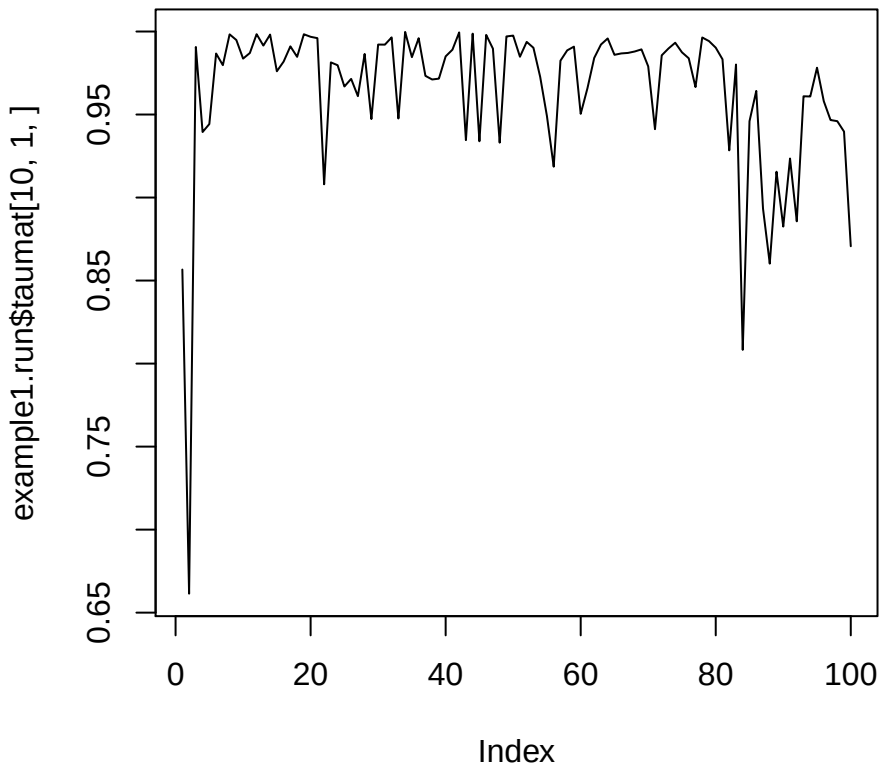

```
# Tau for  second replicated data set
plot(example1.run$taumat[2, 1, ], type = "l")
```



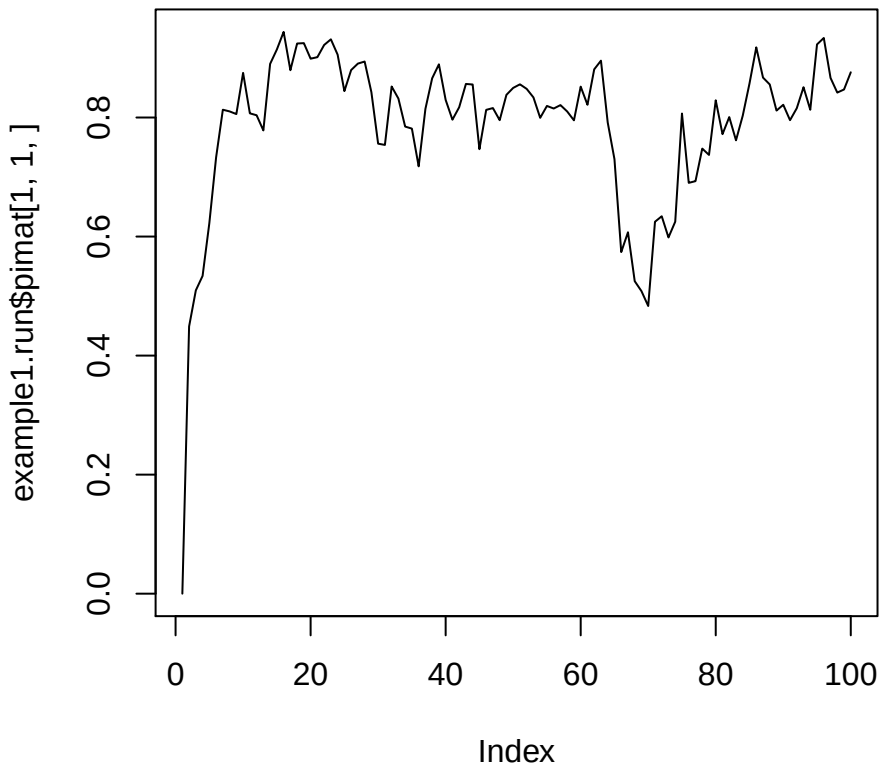

```
# Tau for  third replicated data set
plot(example1.run$taumat[3, 1, ], type = "l")
```



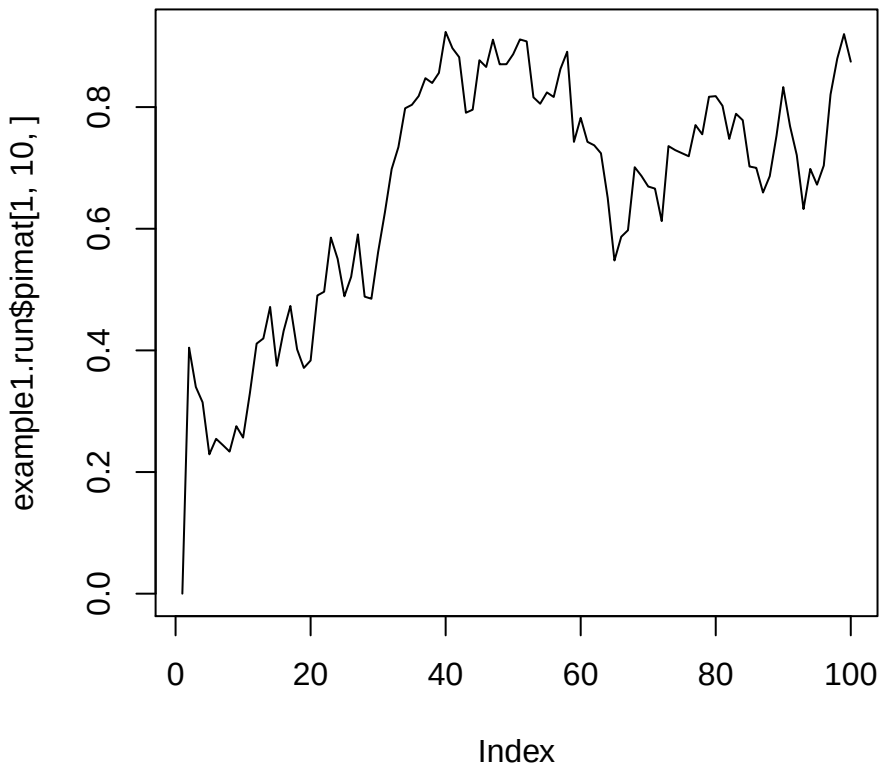

```
# Tau for  tenth replicated data set
plot(example1.run$taumat[10, 1, ], type = "l")
```



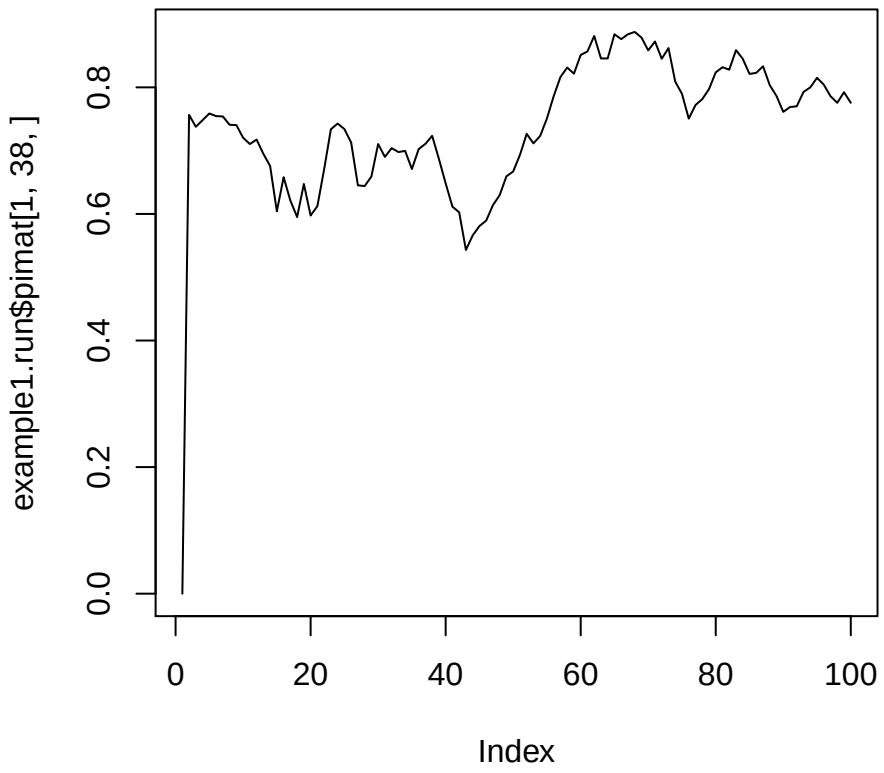

```
## Pi
        # Pi for  the first farm (100 mollusks) in the first replicated data set
        plot(example1.run$pimat[1, 1, ], type = "l")
```



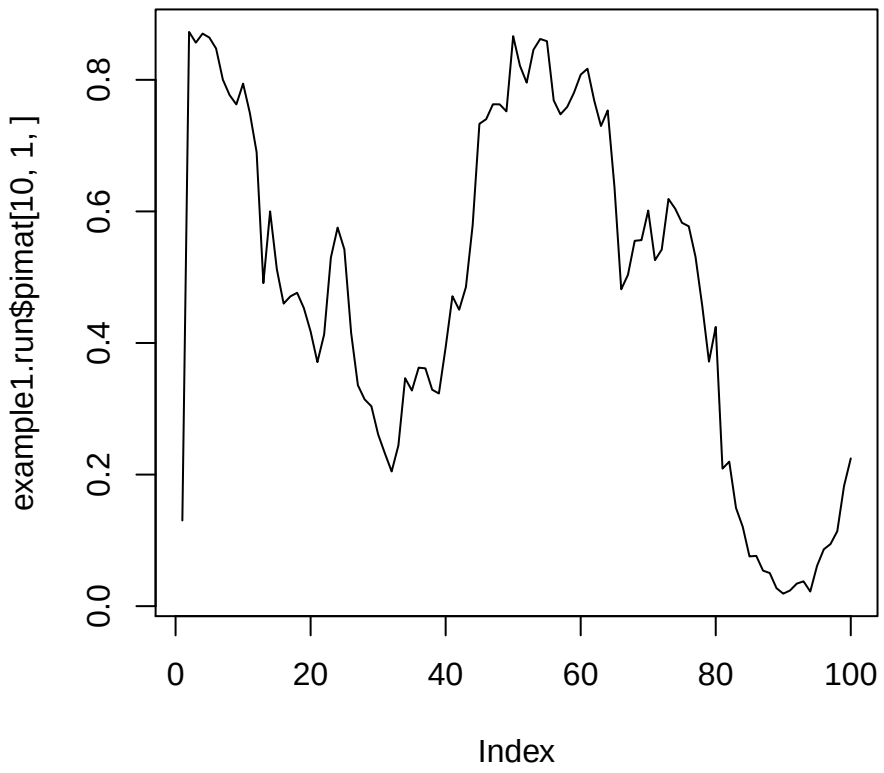

```
# Pi for  the tenth farm (100 mollusks) in the first replicated data set
plot(example1.run$pimat[1, 10, ], type = "l")
```



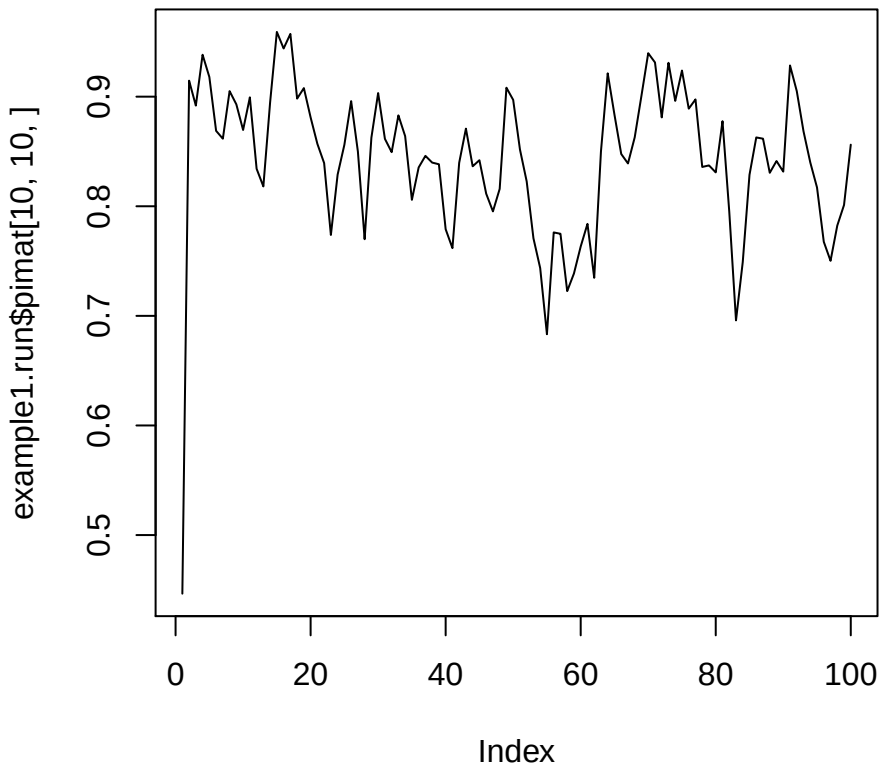

```
# Pi for  the thirty-eighth farm (500 mollusks) in the first replicated data set
plot(example1.run$pimat[1, 38, ], type = "l")
```




```
# Pi for  the first farm (100 mollusks) in the tenth replicated data set
plot(example1.run$pimat[10, 1, ], type = "l")
```

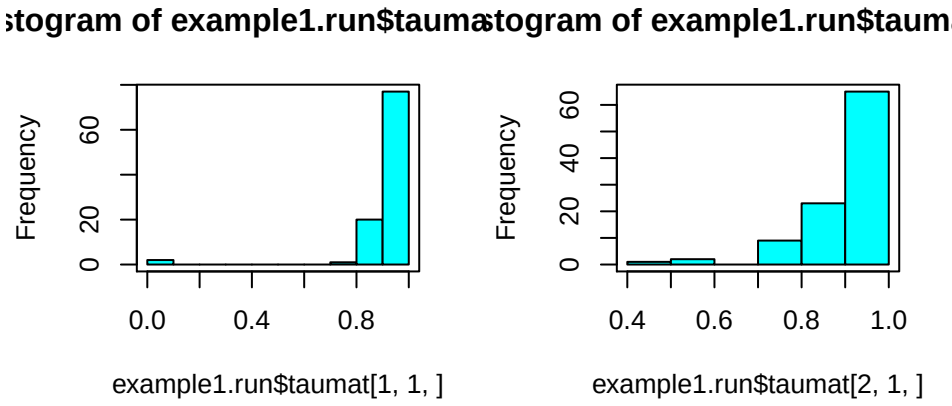


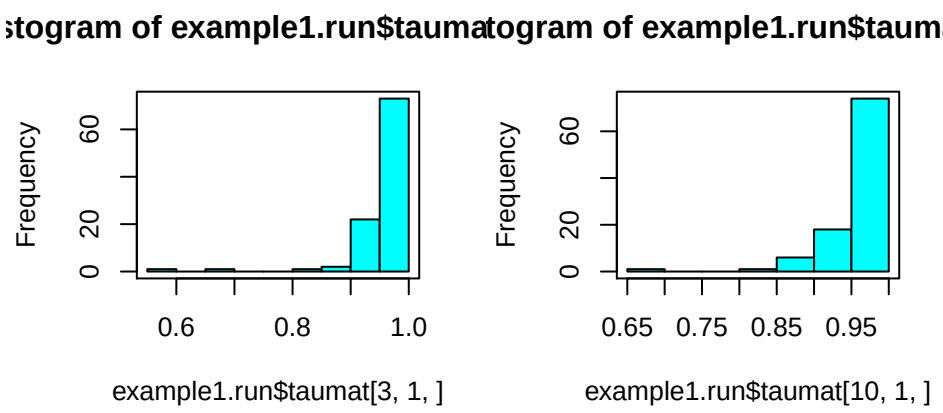

```
# Pi for  the tenth farm (100 mollusks) in the tenth replicated data set
plot(example1.run$pimat[10, 10, ], type = "l")
```



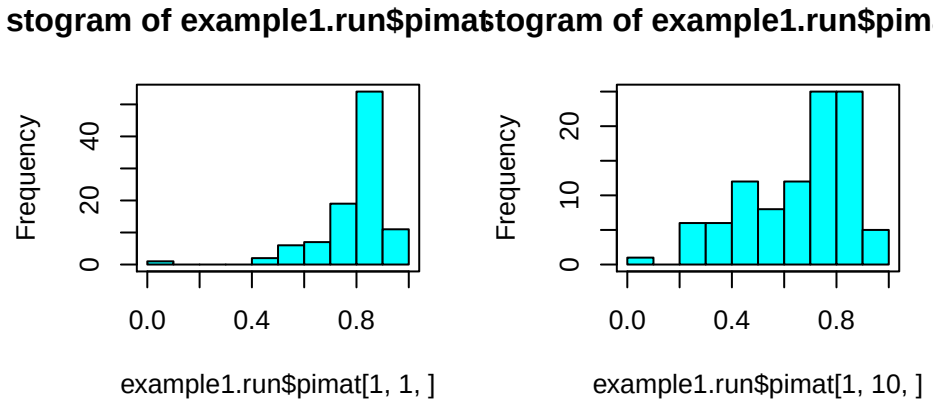
Let's also look at some histograms of the same variables as above.

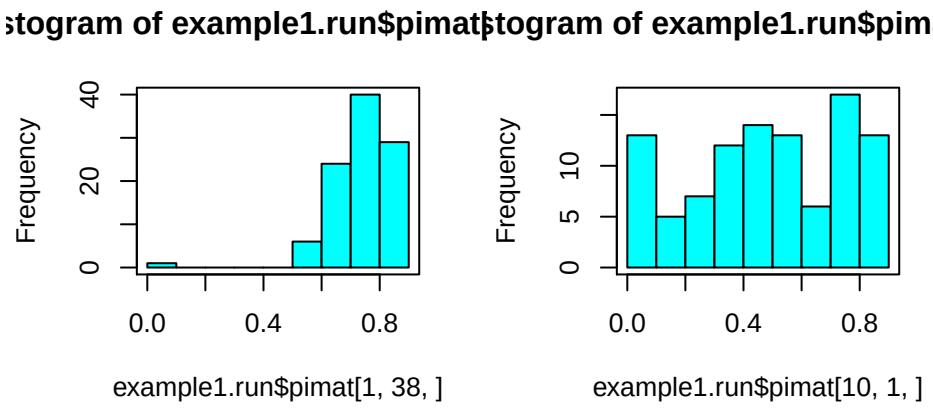
```
par(mfrow=c(2,2))
## Tau
        # Tau for  first replicated data set
        hist(example1.run$taumat[1, 1, ], col = "cyan");box("plot")
        # Tau for  second replicated data set
        hist(example1.run$taumat[2, 1, ], col = "cyan");box("plot")
        # Tau for  third replicated data set
        hist(example1.run$taumat[3, 1, ], col = "cyan");box("plot")
        # Tau for  tenth replicated data set
        hist(example1.run$taumat[10, 1, ], col = "cyan");box("plot")
```



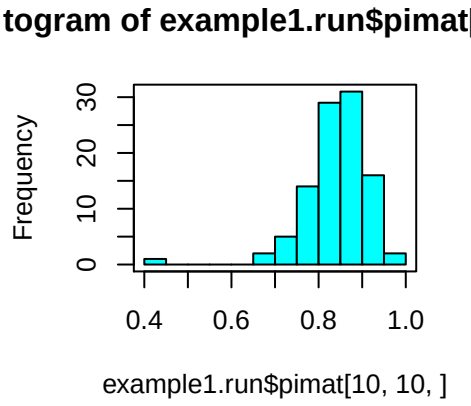



```
## Pi
        # Pi for  the first farm (100 mollusks) in the first replicated data set
        hist(example1.run$pimat[1, 1, ], col = "cyan");box("plot")
        # Pi for  the tenth farm (100 mollusks) in the first replicated data set
        hist(example1.run$pimat[1, 10, ], col = "cyan");box("plot")
        # Pi for  the thirty-eighth farm (500 mollusks) in the first replicated data set
        hist(example1.run$pimat[1, 38, ], col = "cyan");box("plot")
        # Pi for  the first farm (100 mollusks) in the tenth replicated data set
        hist(example1.run$pimat[10, 1, ], col = "cyan");box("plot")
```






```
# Pi for  the tenth farm (100 mollusks) in the tenth replicated data set
hist(example1.run$pimat[10, 10, ], col = "cyan");box("plot")
```



2.2 Example 2: Posterior Inference with All Zeros

Suppose that we have a the same situation as in Example 1 but now we have a different objective. Instead of simulating the sample data, we will say that we have observed all zeroes in our sample and will use that single set of sample data.

Not too much will change with our call to the model, but the big things are: (1) we will have only 1 replication, (2) we can increase our MCMC iterations easily since we have only 1 replication, and (3) our argument to our model `y` will be supplied by us and it must be a matrix with dimensions (`reps` x `k`) - - from the description of the variables above the model code.

The code for calling the storage model under these specifications is below. We use the no storage model, which is a bit faster than the storage model, but doesn't store as much output that could potentially be used to diagnose model fit or convergence issues. Highlight the following code and run it.

```
set.seed(2015) # To ensure reproducible results
example2.run = EpiBayes_ns(
   H = 1,
    k = rep(40, 1),
    n = c(rep(100, 35), rep(500, 5)),
    seasons = rep(c(1, 2, 3, 4), each = 10),
```

```
    mumodes = matrix(c(
            0.50, 0.70,
            0.50, 0.70,
            0.02, 0.50,
            0.02, 0.50
            ), 4, 2, byrow = TRUE
    ),
    reps = 1,  # Only 1 replication this time
    MCMCreps = 1000,  # 1000 MCMC iterations now that we have only 1 replication
    poi = "tau",  # Want inference on cluster-level prevalence
    y = matrix(0, nrow = 1, ncol = 40),  # Set so we have all negative diagnostic
        # test results
    pi.thresh = 0.10,  # Have a 10% within-cluster design prevalence
    tau.thresh = 0.02,  # Doesn't matter since we aren't simulating
    gam.thresh = 0.10,  # Doesn't matter since we aren't simulating
    tau.T = 0.20,  # Doesn't matter since we aren't simulating any data in this
        # case (we'll leave it the same here)
    poi.lb = 0,  # Doesn't matter since we aren't simulating
    poi.ub = 1,  # Doesn't matter since we aren't simulating
    p1 = 0.95,  # Doesn't matter since we aren't simulating
    psi = 4,
    omegaparm = c(1000, 1),
    gamparm = c(1000, 1),
    tauparm = c(1, 1),
    etaparm = c(15, 130),
    thetaparm = c(1000, 1),
    burnin = 10
    )
```

Again, we can look at some quick summary measures using the `summary` and `plot` methods. Note that since we only have one replication here, the simulation statistics are not of much interest to us and we only have one posterior distribution in the plot.

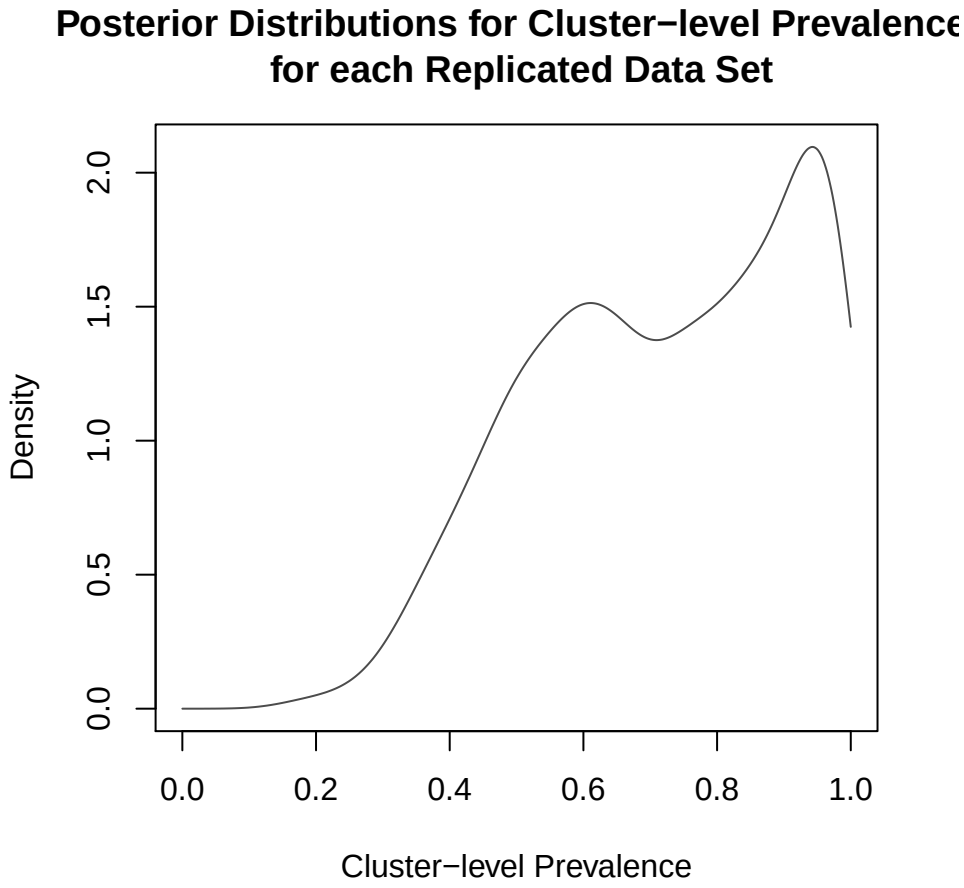
```
## Summary
example2.sum = summary(example2.run)

## Simulation output for parameter of interest (poi)
##  *p2.tilde: Percentage of the time the disease is not detected above the disease threshold
##  *p4.tilde: Percentage of the time the disease is detected above the disease threshold
##  *p6.tilde: Percentage of the time the disease is detected between the user-supplied lower a
##
##  p2.tilde p4.tilde p6.tilde
##         0        1        1
##
##  ------------------------------------------------------------------------------
## gam: Probability of disease being in the region
##           Mean           SD     Naive SE Time-series SE Lower HPD Limit
## [1,] 0.9990176 0.0009982845 3.172756e-05   3.172756e-05       0.9970507
```

```
##      Upper HPD Limit
## [1,]       0.9999999
##
##  ------------------------------------------------------------------------
## tau 1: Cluster-level prevalence in subzone 1
##           Mean        SD    Naive SE Time-series SE Lower HPD Limit
## [1,] 0.7239777 0.1946877 0.006187581      0.1247744       0.3927977
##      Upper HPD Limit
## [1,]       0.9999856
##
##  ------------------------------------------------------------------------
```

```
example2.sum
```

```
## Simulation output for parameter of interest (poi)
##  *p2.tilde: Percentage of the time the disease is not detected above the disease threshold
##  *p4.tilde: Percentage of the time the disease is detected above the disease threshold
##  *p6.tilde: Percentage of the time the disease is detected between the user-supplied lower a
##
##  p2.tilde p4.tilde p6.tilde
##         0        1        1
##
##  ------------------------------------------------------------------------
## gam: Probability of disease being in the region
##           Mean           SD     Naive SE Time-series SE Lower HPD Limit
## [1,] 0.9990176 0.0009982845 3.172756e-05   3.172756e-05       0.9970507
##      Upper HPD Limit
## [1,]       0.9999999
##
##  ------------------------------------------------------------------------
## tau 1: Cluster-level prevalence in subzone 1
##           Mean        SD    Naive SE Time-series SE Lower HPD Limit
## [1,] 0.7239777 0.1946877 0.006187581      0.1247744       0.3927977
##      Upper HPD Limit
## [1,]       0.9999856
##
##  ------------------------------------------------------------------------
```

```
## Plot the posterior distributions of cluster-level prevalence
plot(example2.run)
```



Once finished, we can make posterior inference by using the `coda` package, which is an R package that is designed to process output of Bayesian models. Of primary interest is the posterior distribution of the cluster-level prevalence, though we could do the same with any of the posterior distributions available to us.

```
## Must transform the posterior distribution of tau into an mcmc
# object for  CODA to recognize it
example2.tauposterior = as.mcmc(example2.run$taumat[1, 1, ])

## Now we can use any function in the CODA package to analyze the posterior
        # Generic summary statistics
        summary(example2.tauposterior)

##
## Iterations = 1:1000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
```

```
##
##           Mean             SD       Naive SE Time-series SE
##       0.726487       0.195327       0.006177       0.126223
##
## 2. Quantiles for each variable:
##
##   2.5%    25%    50%    75%  97.5%
## 0.3553 0.5748 0.7449 0.9047 0.9952
```

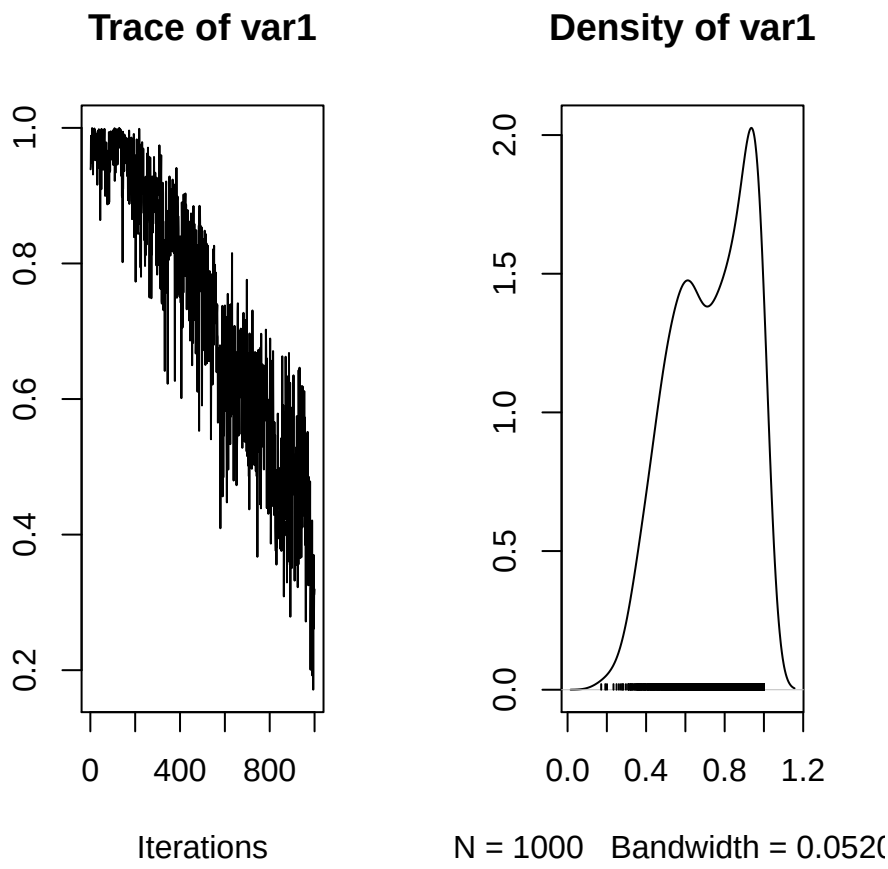
```
        # 95% HPD interval for  the cluster - level prevalence
        HPDinterval(example2.tauposterior, prob = 0.95)
```

```
##          lower     upper
## var1 0.3927977 0.9999856
## attr(,"Probability")
## [1] 0.95
```

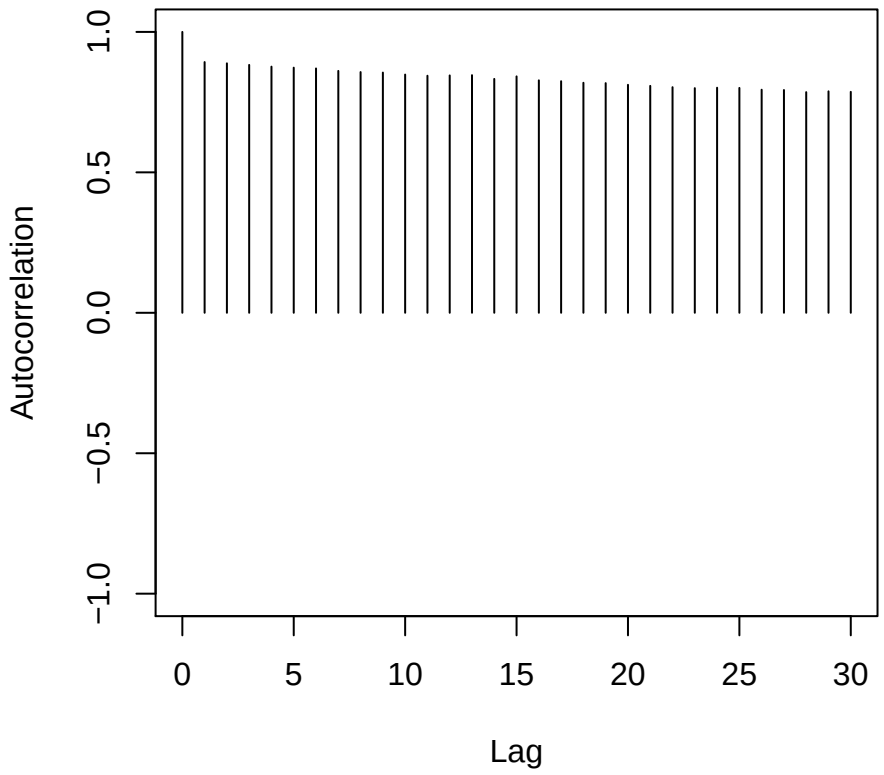
```
        # Geweke convergence statistic
        # If  it looks like the realization of a standard normal random variable
        coda::geweke.diag(example2.tauposterior)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## var1
## 10.4
```

```
        # Trace plot and density plot for  tau minus burnin
        plot(example2.tauposterior)
```




```
# Autocorrelation plot for  the chain minus burnin
autocorr.plot(example2.tauposterior)
```



2.3 Example 3: Sample Size Search

Again, suppose that our setup is the same as in Example 1, but we have yet another objective. Instead of deriving the simulation output for the cluster-level prevalence using a particular set of values for our sample sizes H, k, and n, we will derive the simulation output for all combinations of user-supplied `H`, `k`, and `n` vectors.

We will make a call to the function `EpiBayesSampleSize`, which requests vectors of possible numbers of subzones, numbers of clusters per subzone (same for all subzones), and numbers of subjects per cluster per subzone (same for all clusters). Then, all combinations of the values of these three vectors will be taken and `EpiBayes_ns` will be run on each combination. The simulation output (`p2.tilde`, `p4.tilde`, and `p6.tilde`) will be stored for each run and will be printed by the print method.

Notice that we need to supply those three vectors of sample sizes, a season, and the rest of the inputs from the `EpiBayes_ns` function. Also notice that we have simplified the flexibility of `EpiBayes_ns` a bit by forcing all subzones to have the same number of clusters and all clusters to have the same number of subjects and all sampling occasions must occur in the same season for each combination of one value from `H`, `k`, and `n`. This simplifies input by the user, though if very precise estimates are required for a particular set of `H`, `k`, and `n`, we recommend running `EpiBayes_ns` or `EpiBayes_s` on that particular set so that the results may be more fully explored. This function effectively serves as a screening for 'good' values of the sample sizes under specific conditions.

Here, we would like to determine the probability of detection (`p4.tilde`) and the probability that the prevalence falls within 0.0 and 0.1 (`p6.tilde`) amongst our replicated data sets for each combination of subzones (1 or 2 subzones), clusters (10, 20, or 30 clusters in each subzone), and numbers of subjects (100, 300, or 500 subjects in each cluster).

```
example3.run = EpiBayesSampleSize(
        H = c(1, 2),  # Allow 1 and 2 subzones
        k = c(10, 20, 30),  # Allow 10, 20, and 30 clusters per subzone
        n = c(100, 300, 500),  # Allow 100, 300, and 500 subjects per
                # cluster per subzone
        season = 3,  # Force all sampling to be done in Winter
        reps = 2,
        MCMCreps = 2,
        tau.T = 0,
        y = NULL,
        poi = "tau",
        mumodes = matrix(c(
                0.50, 0.70,
                0.50, 0.70,
                0.02, 0.50,
                0.02, 0.50
                ), 4, 2, byrow = TRUE
        ),
        pi.thresh = 0.10,
        tau.thresh = 0.02,
        gam.thresh = 0.10,
        poi.lb = 0,
        poi.ub = 0.1,
        p1 = 0.95,
        psi = 4,
        tauparm = c(1, 1),
        omegaparm = c(1000, 1),
        gamparm = c(1000, 1),
        etaparm = c(15, 130),
        thetaparm = c(100, 6),
        burnin = 1
        )
```

By default, the `print` method will return all three simulation output values. We can specifically request only the one or two in which we are interested.

```
example3.run  # Prints all three simulation statistics

## Estimated probabilities of not detecting cluster-level prevalence above for given H, k, and
##
## , , H = 1
##
##     n
```

```
## k    100 300 500
##   10 0.0 0.0   0
##   20 0.5 0.0   0
##   30 0.0 0.5   0
##
## , , H = 2
##
##     n
## k    100 300 500
##   10   0   0   0
##   20   0   0   0
##   30   0   0   0
##
## Estimated probabilities of detecting cluster-level prevalence above for given H, k, and n.
##
## , , H = 1
##
##     n
## k    100 300 500
##   10 1.0 1.0   1
##   20 0.5 1.0   1
##   30 1.0 0.5   1
##
## , , H = 2
##
##     n
## k    100 300 500
##   10   1   1   1
##   20   1   1   1
##   30   1   1   1
##
## Estimated probabilities of detecting cluster-level prevalence in the interval (0, 0.1) for g
##
## , , H = 1
##
##     n
## k    100 300 500
##   10 0.0 0.0   0
##   20 0.5 0.0   0
##   30 0.5 0.5   0
##
## , , H = 2
##
##     n
## k    100 300 500
##   10   0   0   0
##   20   0   0   0
```

```
##   30   0   0   0

# Prints only those two requested
print(example3.run, out.ptilde = c("p4.tilde", "p6.tilde"))

## Estimated probabilities of not detecting cluster-level prevalence above for given H, k, and
##
## , , H = 1
##
##     n
## k    100 300 500
##   10 0.0 0.0   0
##   20 0.5 0.0   0
##   30 0.0 0.5   0
##
## , , H = 2
##
##     n
## k    100 300 500
##   10   0   0   0
##   20   0   0   0
##   30   0   0   0
##
## Estimated probabilities of detecting cluster-level prevalence above for given H, k, and n.
##
## , , H = 1
##
##     n
## k    100 300 500
##   10 1.0 1.0   1
##   20 0.5 1.0   1
##   30 1.0 0.5   1
##
## , , H = 2
##
##     n
## k    100 300 500
##   10   1   1   1
##   20   1   1   1
##   30   1   1   1
##
## Estimated probabilities of detecting cluster-level prevalence in the interval (0, 0.1) for g
##
## , , H = 1
##
##     n
## k    100 300 500
##   10 0.0 0.0   0
```

```
##   20 0.5 0.0   0
##   30 0.5 0.5   0
##
## , , H = 2
##
##     n
## k    100 300 500
##   10   0   0   0
##   20   0   0   0
##   30   0   0   0
```

Note: The above examples are not meant to reflect reality. Notice that the `MCMCreps` in both examples are set very low. This would have been bad if executed in practice, but was set so in order to ensure quick build times for this vignette.