Occupancy models with detection error

Peter Solymos and Subhash Lele

July 16, 2016 — Madison, WI — NACCB Congress

Contents

Let us continue with the simple occupancy model we used previously. Most applied ecologists are aware that the occupancy and abundance surveys have some level of detection error. Even if the species is present, for various reasons we may not observe its presence. Similarly we may not be able to count all the individuals that are present at a location. Let us look at how to model such a situation. We will discuss the model and then show how it can be looked upon as a hierarchical model.

0.0.1 Notation

- W_i: this denotes the *observed* status at the location i, can be 0 or 1,
- Y_i: this denotes the true status at the location i, can be 0 or 1; this status is *unknown*.

0.0.2 Assumptions

1. The observed status depends on the true status. If there is no dependence between the two variables, obviously we cannot do any inference.
2. There are no "phantom" individuals. That is, if the true status is 0, we will observe 0 with probability 1.
3. True status at one location is independent of status of other locations.
4. Observation at one location is not affected by what we observed anywhere else (or, at other times at that location). Surveys are independent of each other.

We can extend the Bernoulli model from the introduction as follows:

- True status:
$$Y_i \sim Bernoulli(\varphi)$$
.
- Observed status:
$$(W_i \mid Y_i = y_i) \sim Bernoulli(p^{y_i}(1-p)^{1-y_i})$$
.

An important thing to note here is that we only observe W's and not the true statuses (Y) which are unknown. We can use the standard probability rules to compute:

$$P(W_i = 1) = P(W_i = 1 \mid Y_i = 1)P(Y_i = 1) + P(W_i = 1 \mid Y_i = 0)P(Y_i = 0) = p\varphi + 0 \cdot (1-\varphi) = p\varphi$$

$$P(W_i = 0) = P(W_i = 0 \mid Y_i = 1)P(Y_i = 1) + P(W_i = 0 \mid Y_i = 0)P(Y_i = 0) = 1 - p\varphi$$

This is called the marginal distribution of W. We can write down the likelihood function as a function of parameters (p, φ).

$$L(p, \varphi; w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(W_i = w_i; p, \varphi) = \prod_{i=1}^{n} (p\varphi)^{w_i}(1-p\varphi)^{1-w_i}$$

0.0.3 Cautionary note

Just because one can write down the likelihood function, it does not mean one can estimate the parameters.

This is a simple situation with two parameters and hence we can plot the likelihood function as a contour plot.

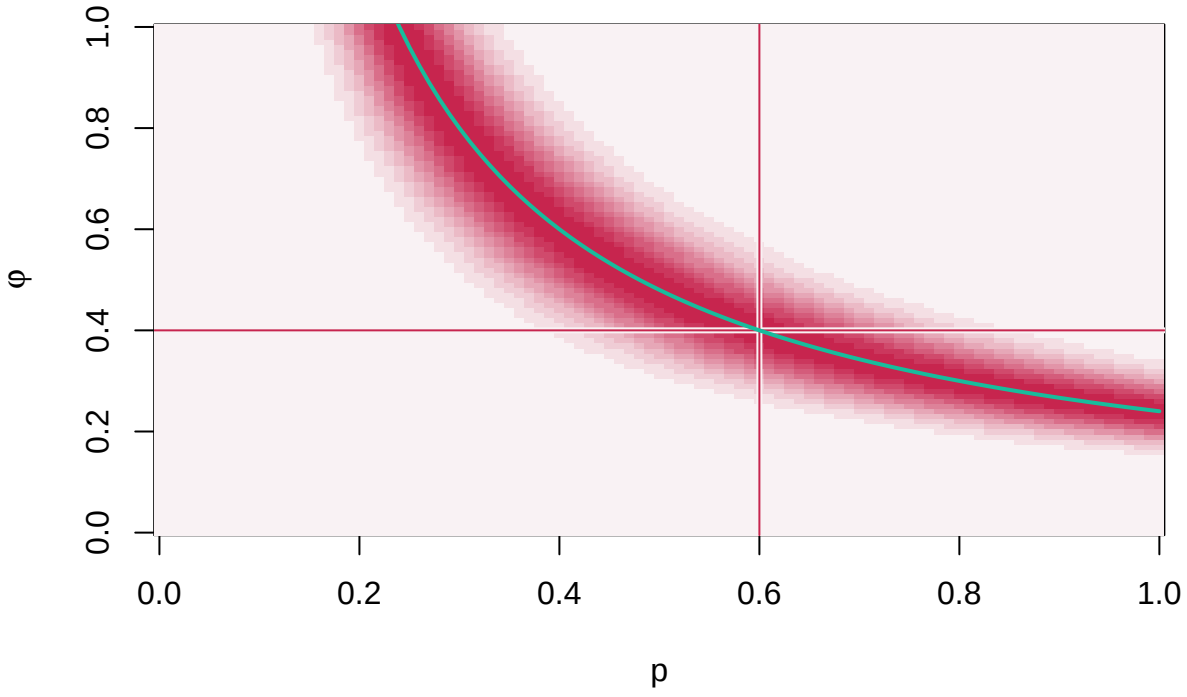
R code for data generation:

```
set.seed(4321)
n <- 100
p <- 0.6
phi <- 0.4
y <- rbinom(n = n, size = 1, prob = phi)
w <- rbinom(n = n, size = y, prob = p)
table(Y = y, W = w)
```

```
##    W
## Y    0  1
##   0 60  0
##   1 16 24
```

Given the data, plot the likelihood contours.

```
## setting up the grid for p and phi
grid <- expand.grid(p = seq(0, 1, by = 0.01),
    phi = seq(0, 1, by = 0.01),
    L = NA)
## the likelihood function
L_fun <- function(w, p, phi) {
    prod((p * phi)^w * (1 - p * phi)^(1 - w))
}
## calculating the likelihood for the grid
for (i in 1:nrow(grid)) {
    grid$L[i] <- L_fun(w = w, p = grid$p[i], phi = grid$phi[i])
}
## plot the likelihood surface
dcpal_reds <- colorRampPalette(c("#f9f2f4", "#c7254e"))
L_mat <- matrix(grid$L, sqrt(nrow(grid)))
image(L_mat,
    xlab = "p", ylab = expression(varphi),
    col = dcpal_reds(12))
abline(h = phi, v = p, col = "#f9f2f4", lwd = 3)
abline(h = phi, v = p, col = "#c7254e", lwd = 1)
curve((p * phi) / x, 0, 1, add = TRUE,
    col = "#18bc9c", lwd = 2)
```



We can see that the likelihood function looks like a mountain with a ridge tracing a curve corresponding to the product $p\varphi = c$.

```
library(rgl)
open3d()
bg3d("white")
material3d(col = "black")
dcpal_grbu <- colorRampPalette(c("#18bc9c", "#3498db"))
Col <- rev(dcpal_grbu(12))[cut(L_mat, breaks = 12)]
persp3d(L_mat / max(L_mat), col = Col,
    theta=50, phi=25, expand=0.75, ticktype="detailed",
    xlab = "p", ylab = "phi", zlab = "L")
```

- Likelihood function does not have a unique maximum. All values along this curve have equal support in the data. We can estimate the product but not the individual components of the product.
- The placement of the curve depends on the data. So there is information in the data only about the product but not the components.

When the likelihood function attains maximum at more than one parameter combination, we call the parameters *non-estimable*. There are various reasons for such non-estimability (Reference: Campbell and Lele, 2013 and a couple of references from that paper).

Structural problems with the model: it might be that the structure of the problem is such that no matter what, you cannot estimate the parameters. This is called *non-identifiability*.

Sometimes there are no structural issues but the observed data combination is such that the likelihood is problematic. This is called *non-estimability*. An example will be collinear covariates in regression.

Consequences of *non-identifiability*: management decisions can be based only on identifiable components of the model.

For models with more than two parameters, it is very difficult to plot the likelihood function. It is nearly impossible to diagnose non-identifiability and non-estimability of the parameters. Data cloning method provides a very simple approach to diagnose non-estimability for general hierarchical models.

We can skip all the mathematical details in the calculation of the likelihood function and use JAGS and MCMC to do almost all of the above analysis.

0.0.4 Bayesian model in JAGS

```
library(dclone)
```

```
## Loading required package: coda
```

```
## Loading required package: parallel
```

```
## Loading required package: Matrix
```

```
## dclone 2.1-1      2016-01-11
```

```
library(rjags)
```

```
## Linked to JAGS 4.2.0
```

```
## Loaded modules: basemod,bugs
```

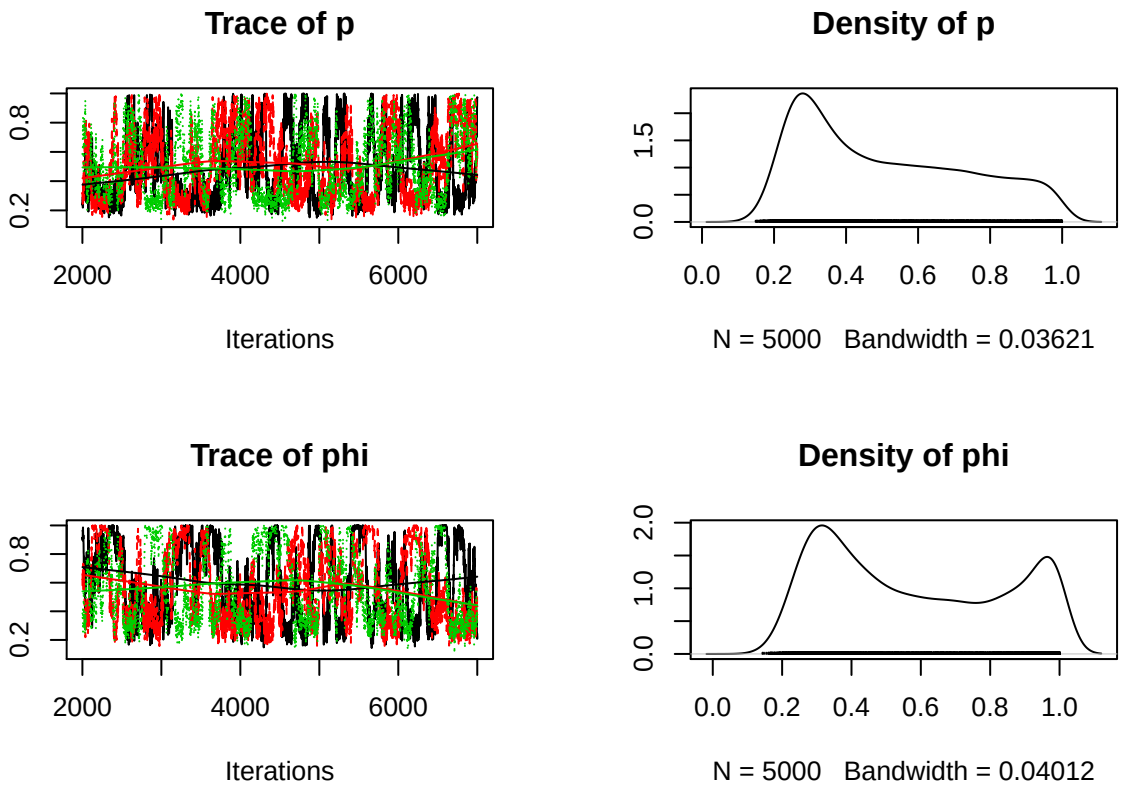
```
model <- custommodel("model {
    for (i in 1:n) {
        Y[i] ~ dbern(phi)
        W[i] ~ dbern(Y[i] * p)
    }
    #p ~ dunif(0.001, 0.999)
    #phi ~ dunif(0.001, 0.999)
    p ~ dbeta(1, 1)
    phi ~ dbeta(0.5, 0.5)
}")
dat <- list(W = w, n = n)
#ini <- list(Y = w)
ini <- list(Y = rep(1, n))
fit <- jags.fit(data = dat, params = c("p", "phi"),
    model = model, init = ini)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 102
##    Total graph size: 307
##
## Initializing model
```

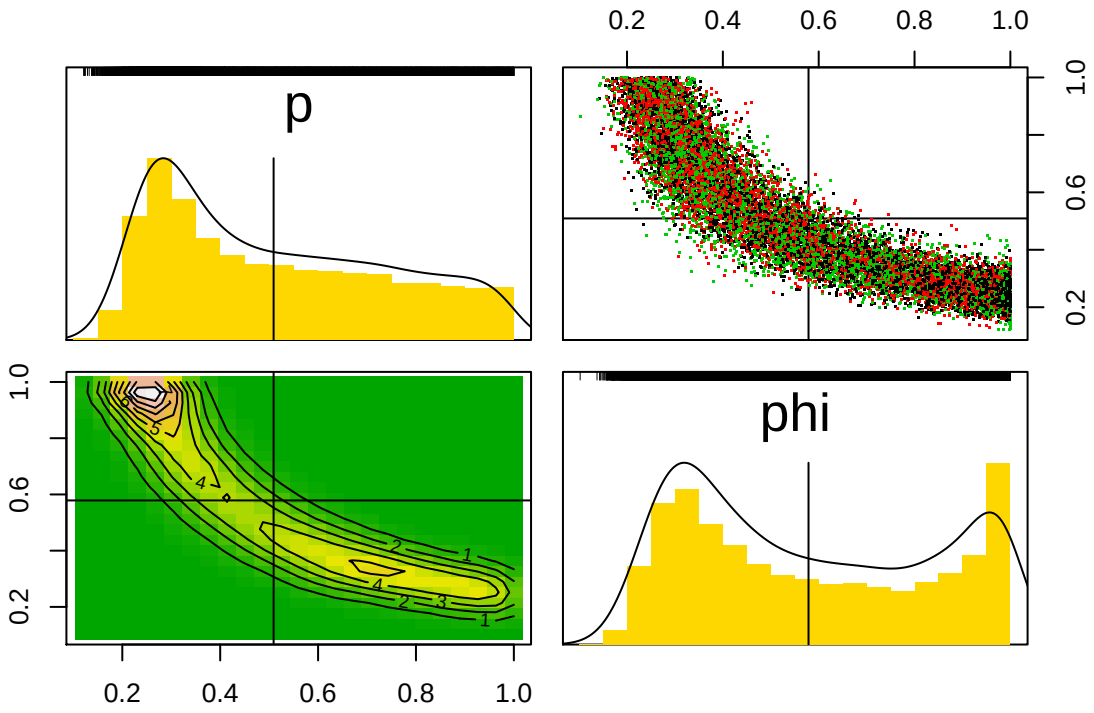
```
summary(fit)
```

```
##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##       Mean     SD Naive SE Time-series SE
## p   0.5091 0.2338 0.001909        0.01877
## phi 0.5788 0.2590 0.002115        0.02543
##
## 2. Quantiles for each variable:
##
##       2.5%    25%    50%    75%  97.5%
## p   0.2017 0.3007 0.4579 0.6986 0.9676
## phi 0.2196 0.3411 0.5284 0.8307 0.9973
```

```
plot(fit)
```



```
pairs(fit)
```



0.0.5 Bayesian inference

Observe what happens to convergence diagnostics.

0.0.6 Data cloning

To make sure that both locations and klones are independent (i.i.d.), it is safest to include and extra dimension and the corresponding loop.

```
library(dclone)
library(rjags)
model <- custommodel("model {
    for (k in 1:K) {
        for (i in 1:n) {
            Y[i,k] ~ dbern(phi)
            W[i,k] ~ dbern(Y[i,k] * p)
        }
    }
    #p ~ dunif(0.001, 0.999)
    #phi ~ dunif(0.001, 0.999)
    p ~ dbeta(1, 1)
    phi ~ dbeta(0.5, 0.5)
```

```
}")
dat <- list(W = dcdim(data.matrix(w)), n = n, K = 1)
ini <- list(Y = dcdim(data.matrix(w)))
ifun <- function(model, n.clones) {
    dclone(list(Y = dcdim(data.matrix(w))),
        n.clones)
}
dcfit <- dc.fit(data = dat, params = c("p", "phi"),
    model = model, inits = ini,
    n.clones = c(1,2,4,8), unchanged = "n", multiply = "K",
    initsfun = ifun, n.iter = 10000)
```

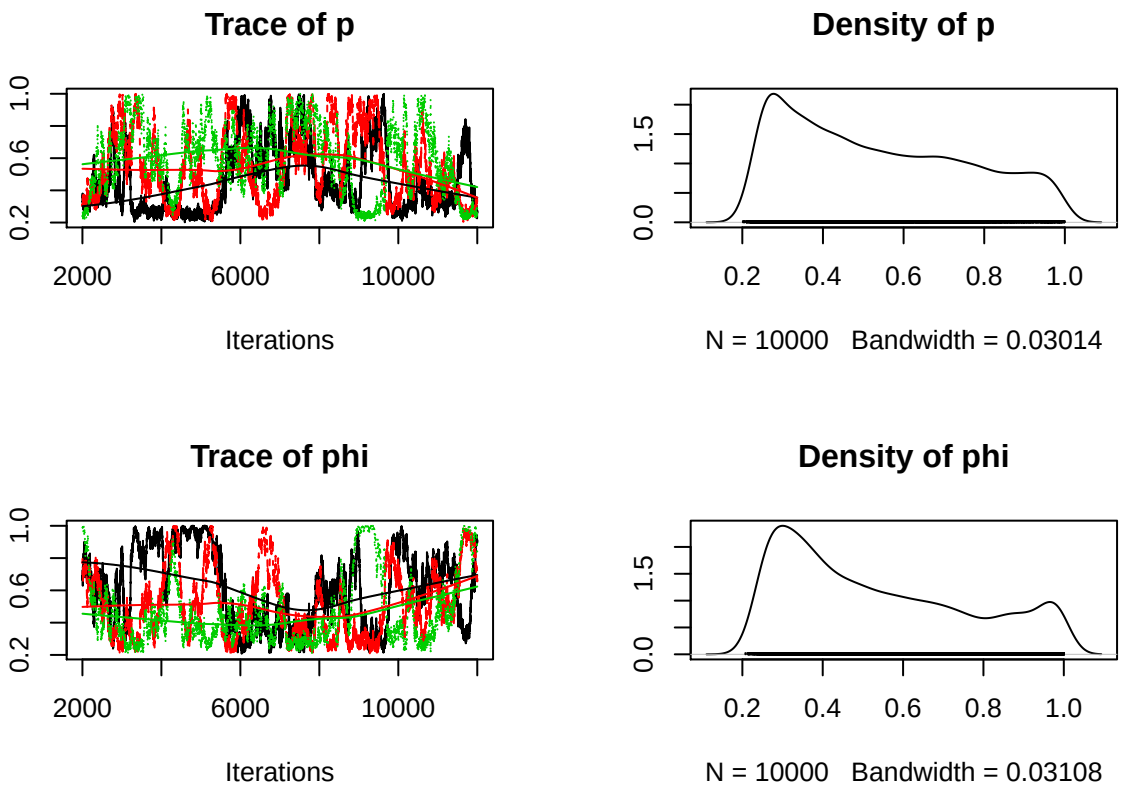
```
##
## Fitting model with 1 clone
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 102
##    Total graph size: 308
##
## Initializing model
##
##
## Fitting model with 2 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 200
##    Unobserved stochastic nodes: 202
##    Total graph size: 608
##
## Initializing model
##
##
## Fitting model with 4 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 400
##    Unobserved stochastic nodes: 402
##    Total graph size: 1208
##
## Initializing model
##
##
## Fitting model with 8 clones
```

```
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 800
##    Unobserved stochastic nodes: 802
##    Total graph size: 2408
##
## Initializing model
```

```
summary(dcfit)
```

```
##
## Iterations = 2001:12000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 10000
## Number of clones = 8
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##       Mean     SD  DC SD Naive SE Time-series SE R hat
## p   0.5379 0.2235 0.6321 0.001290        0.02855 1.010
## phi 0.5346 0.2305 0.6519 0.001331        0.03443 1.009
##
## 2. Quantiles for each variable:
##
##       2.5%    25%    50%    75%  97.5%
## p   0.2379 0.3380 0.5014 0.7175 0.9670
## phi 0.2433 0.3343 0.4789 0.7081 0.9884
```

```
plot(dcfit)
```

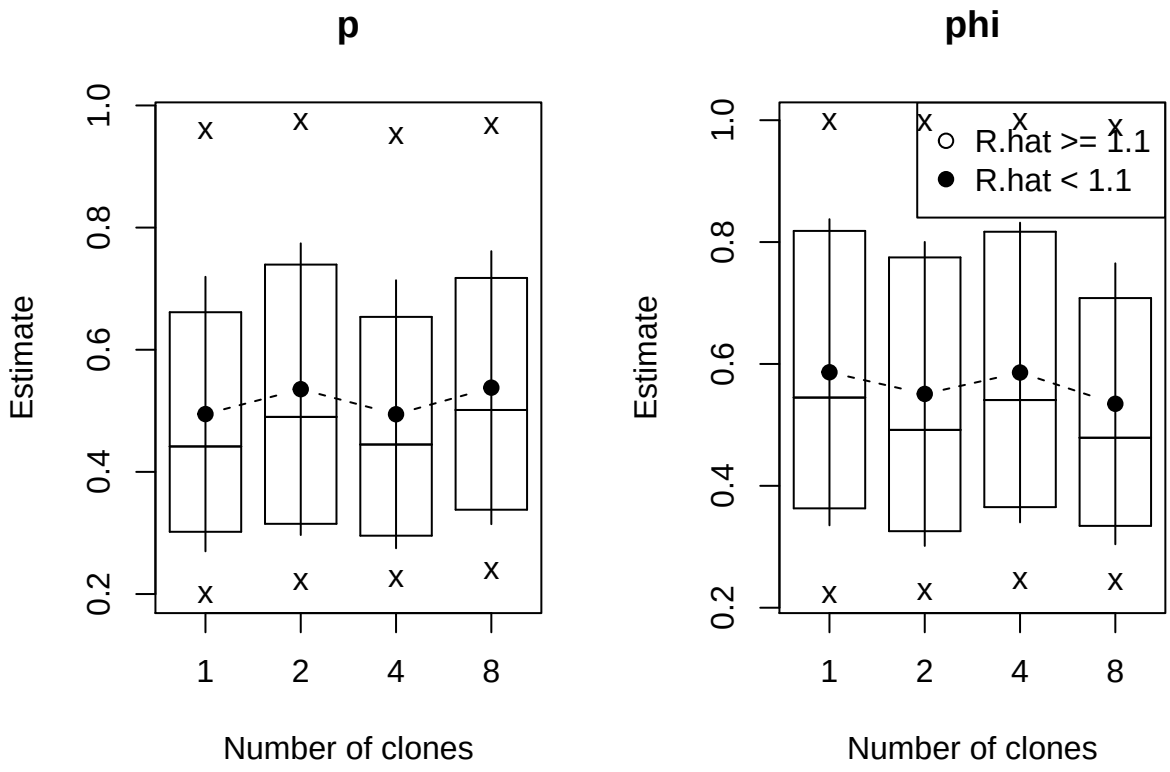



```
dctable(dcfit)
```

```
## $p
##   n.clones      mean        sd      2.5%       25%       50%       75%
## 1        1 0.4946698 0.2247542 0.1997395 0.3018791 0.4416387 0.6614126
## 2        2 0.5354086 0.2388063 0.2204358 0.3149455 0.4900126 0.7394300
## 3        4 0.4943648 0.2194045 0.2257077 0.2955147 0.4447927 0.6537955
## 4        8 0.5378894 0.2234852 0.2379364 0.3380359 0.5013558 0.7175418
##       97.5%    r.hat
## 1 0.9589935 1.016549
## 2 0.9740501 1.018516
## 3 0.9504233 1.031947
## 4 0.9670069 1.010221
##
## $phi
##   n.clones      mean        sd      2.5%       25%       50%       75%
## 1        1 0.5863918 0.2511758 0.2221881 0.3630389 0.5447969 0.8183098
## 2        2 0.5508284 0.2493873 0.2262095 0.3255760 0.4917713 0.7748121
## 3        4 0.5859401 0.2458590 0.2442072 0.3650657 0.5408028 0.8169753
## 4        8 0.5346243 0.2304716 0.2433300 0.3342821 0.4788511 0.7081220
##       97.5%    r.hat
## 1 0.9981934 1.015673
## 2 0.9953908 1.028383
## 3 0.9968277 1.056011
## 4 0.9884346 1.008668
##
```

```
## attr(,"class")
## [1] "dctable"
```

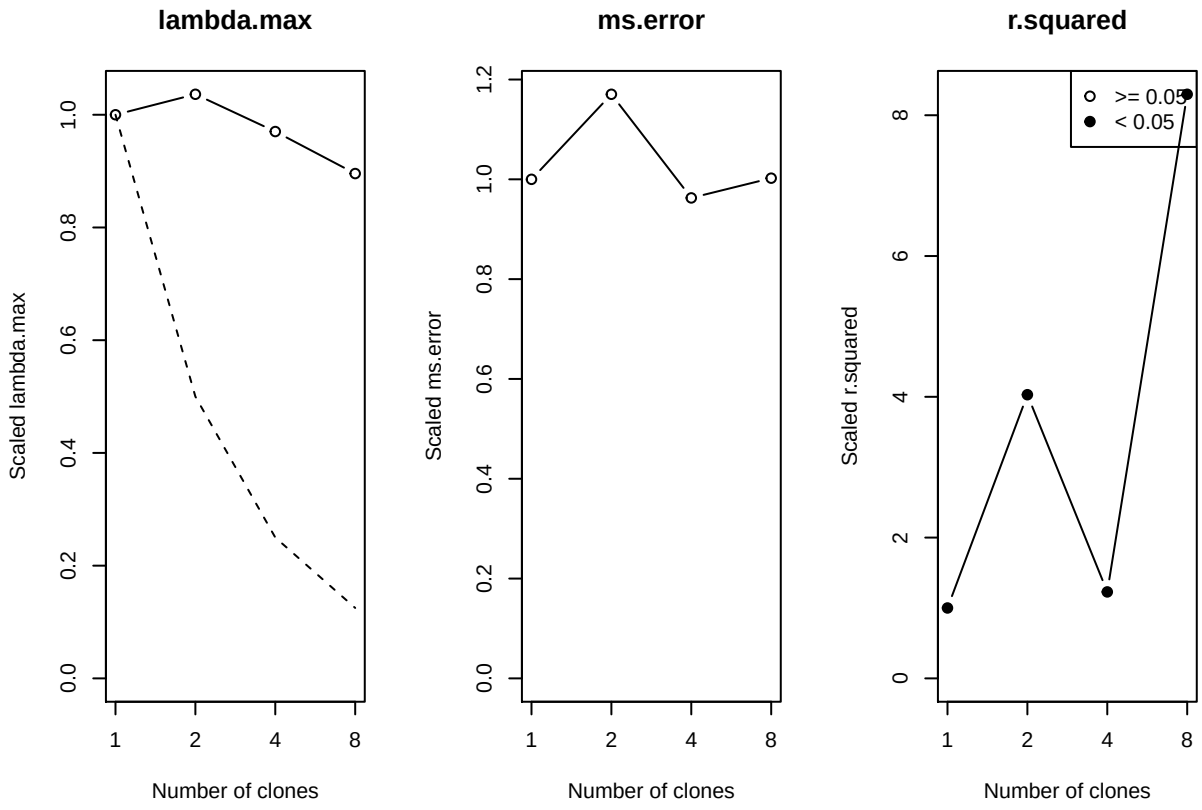
```
plot(dctable(dcfit))
```



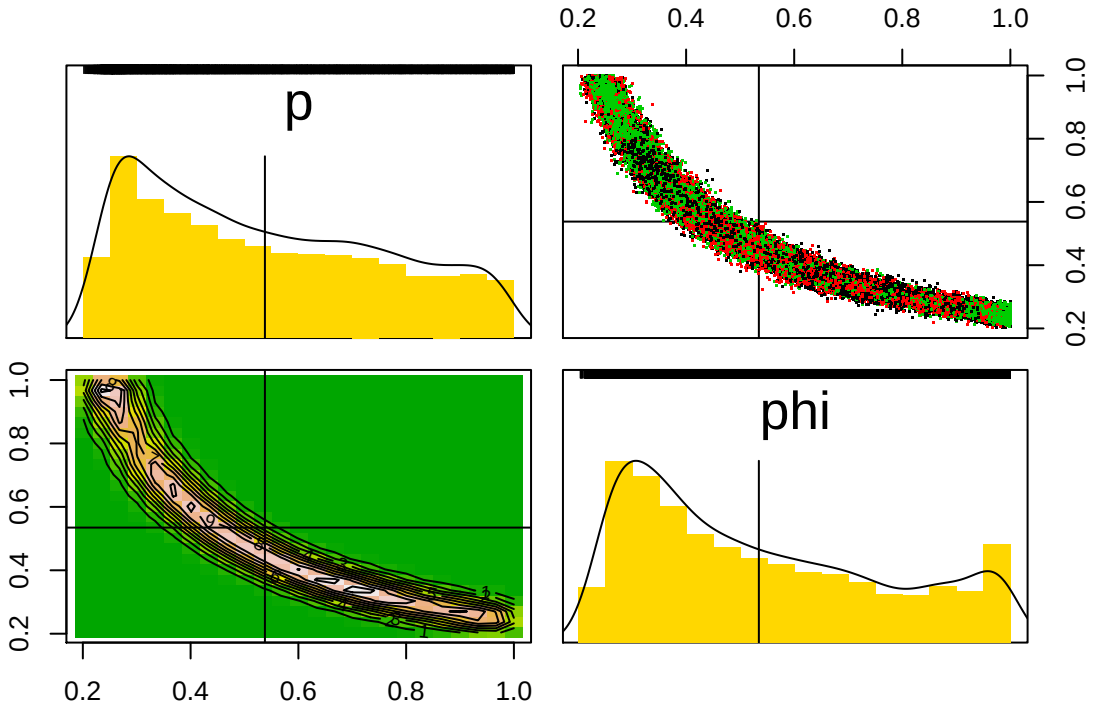

```
dcdiag(dcfit)
```

```
##   n.clones lambda.max ms.error   r.squared r.hat
## 1        1  0.1689125 1.797682 0.005264228    NA
## 2        2  0.1750505 2.104081 0.021213763    NA
## 3        4  0.1638787 1.730725 0.006465682    NA
## 4        8  0.1512871 1.801889 0.043681980    NA
```

```
plot(dcdiag(dcfit))
```


lambda.max
Scaled lambda.max
0.0
0.2
0.4
0.6
0.8
1.0
1
2
4
8
Number of clones
ms.error
Scaled ms.error
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1
2
4
8
Number of clones
r.squared
Scaled r.squared
0
2
4
6
8
>= 0.05
< 0.05
1
2
4
8
Number of clones

```
pairs(dcfit)
```



0.0.6.1 Modification If locations are treated as i.i.d., it is possible to replicate the vector, so that length becomes `n * K`.

```
model <- custommodel("model {
    for (i in 1:n) {
        Y[i] ~ dbern(p)
        W[i] ~ dbern(Y[i] * phi)
    }
    p ~ dunif(0.001, 0.999)
    phi ~ dunif(0.001, 0.999)
}")
dat <- list(W = w, n = n)
ini <- list(Y = w)
ifun <- function(model, n.clones) {
    dclone(list(Y = w), n.clones)
}
dcfit <- dc.fit(data = dat, params = c("p", "phi"),
    model = model, inits = ini,
    n.clones = c(1,2,4,8), multiply = "n",
    initsfun = ifun)
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
```

```
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 102
##    Total graph size: 307
##
## Initializing model
##
##
## Fitting model with 2 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 200
##    Unobserved stochastic nodes: 202
##    Total graph size: 607
##
## Initializing model
##
##
## Fitting model with 4 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 400
##    Unobserved stochastic nodes: 402
##    Total graph size: 1207
##
## Initializing model
##
##
## Fitting model with 8 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 800
##    Unobserved stochastic nodes: 802
##    Total graph size: 2407
##
## Initializing model

## Warning in dclone::.dcFit(data, params, model, inits, n.clones, multiply =
## multiply, : chains convergence problem, see R.hat values
```

0.0.7 Data cloning inference

Observe what happens to the standard errors as we increase the number of clones. It does not converge to 0 as it did before. This indicates non-estimabilty of the parameters.

0.0.8 Can we do something about this non-identifiability?

Suppose we go to the same location more than once, say T times. Then sometimes we will observe the species and sometimes we will not. These changes may help us learn about the detection error process.

The occupancy model with replicate visits is:

- True status:
$$Y_i \sim Bernoulli(\varphi)$$
.
- Observed status:
$$(W_{i,t} \mid Y_i = 1) \sim Bernoulli(p)$$
and
$$W_{i,t} \mid Y_i = 0$$
equals 0 with probability 1.

The likelihood function is:

$$L(p, \varphi; w_{1,1}, \ldots, w_{n,T}) = \prod_{i=1}^{n} \left[\varphi \left(\binom{Y}{w_{i\cdot}} p^{w_{i\cdot}} (1-p)^{T-w_{i\cdot}} \right) + (1-\varphi) I(w_{i\cdot} = 0) \right]$$

where

$$w_{i\cdot} = \sum_{T}^{t=1} w_{i,t}$$

and

$$I(w_{i\cdot} = 0)$$

is an indicator function that is equal to one if

$$w_{i\cdot} = 0$$

.

0.0.9 Assumptions

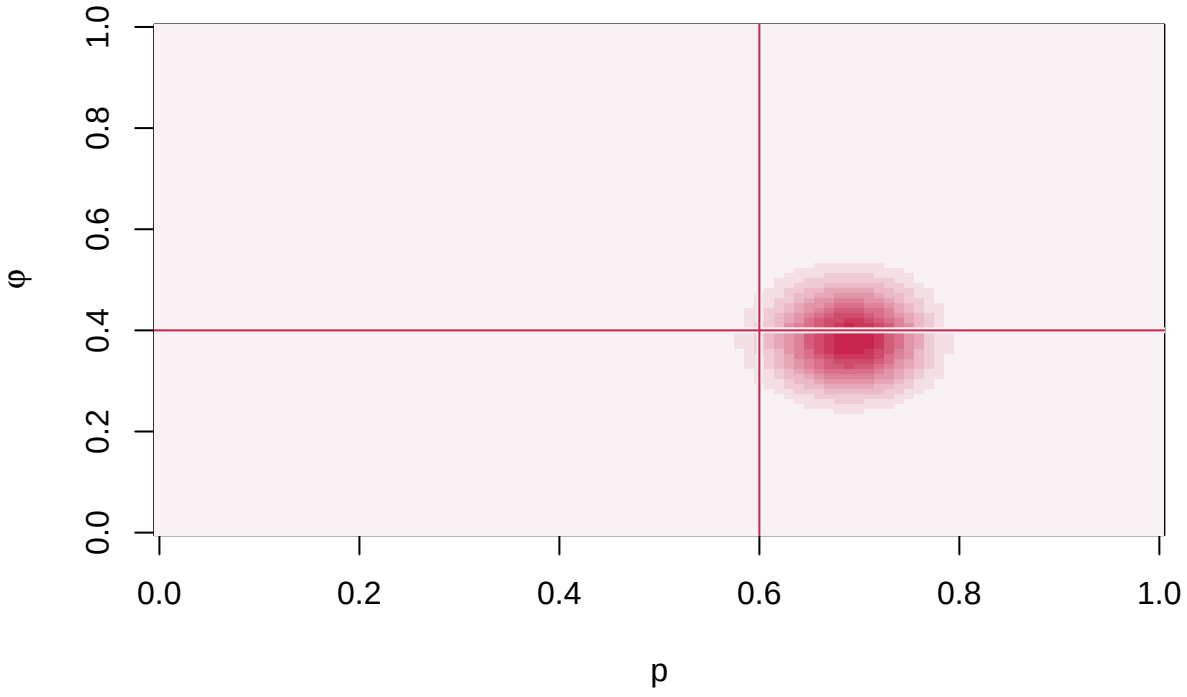
1. Closed population assumption: there is colonization or extinction, that is the true status remains the same over the visits.
2. Independent survey assumption: replicate visits are independent of each other.

R code for data generation:

```r
set.seed(1234)
n <- 50
T <- 5
p <- 0.6
phi <- 0.4
y <- rbinom(n = n, size = 1, prob = phi)
w <- matrix(NA, n, T)
for (t in 1:T)
    w[,t] <- rbinom(n = n, size = y, prob = p)
```

Given the data, plot the likelihood contours.

```r
## setting up the grid for p and phi
grid <- expand.grid(p = seq(0, 1, by = 0.01),
    phi = seq(0, 1, by = 0.01),
    L = NA)
## the likelihood function
L_fun <- function(w, p, phi) {
    wdot <- rowSums(w)
    T <- ncol(w)
    prod(phi * (choose(T, wdot) * p^wdot * (1 - p)^(T - wdot)) +
        (1 - phi) * (wdot == 0))
}
## calculating the likelihood for the grid
for (i in 1:nrow(grid)) {
    grid$L[i] <- L_fun(w = w, p = grid$p[i], phi = grid$phi[i])
}
## plot the likelihood surface
dcpal_reds <- colorRampPalette(c("#f9f2f4", "#c7254e"))
L_mat <- matrix(grid$L, sqrt(nrow(grid)))
image(L_mat,
    xlab = "p", ylab = expression(varphi),
    col = dcpal_reds(12))
abline(h = phi, v = p, col = "#f9f2f4", lwd = 3)
abline(h = phi, v = p, col = "#c7254e", lwd = 1)
```




```
library(rgl)
open3d()
bg3d("white")
material3d(col = "black")
dcpal_grbu <- colorRampPalette(c("#18bc9c", "#3498db"))
Col <- rev(dcpal_grbu(12))[cut(L_mat, breaks = 12)]
persp3d(L_mat / max(L_mat), col = Col,
    theta=50, phi=25, expand=0.75, ticktype="detailed",
    ylab = "p", xlab = "phi", zlab = "L")
```

0.0.10 Bayesian model in JAGS

```
library(dclone)
library(rjags)
model <- custommodel("model {
    for (i in 1:n) {
        Y[i] ~ dbern(phi)
        for (t in 1:T) {
            W[i,t] ~ dbern(Y[i] * p)
        }
    }
    p ~ dunif(0.001, 0.999)
    phi ~ dunif(0.001, 0.999)
}")
```

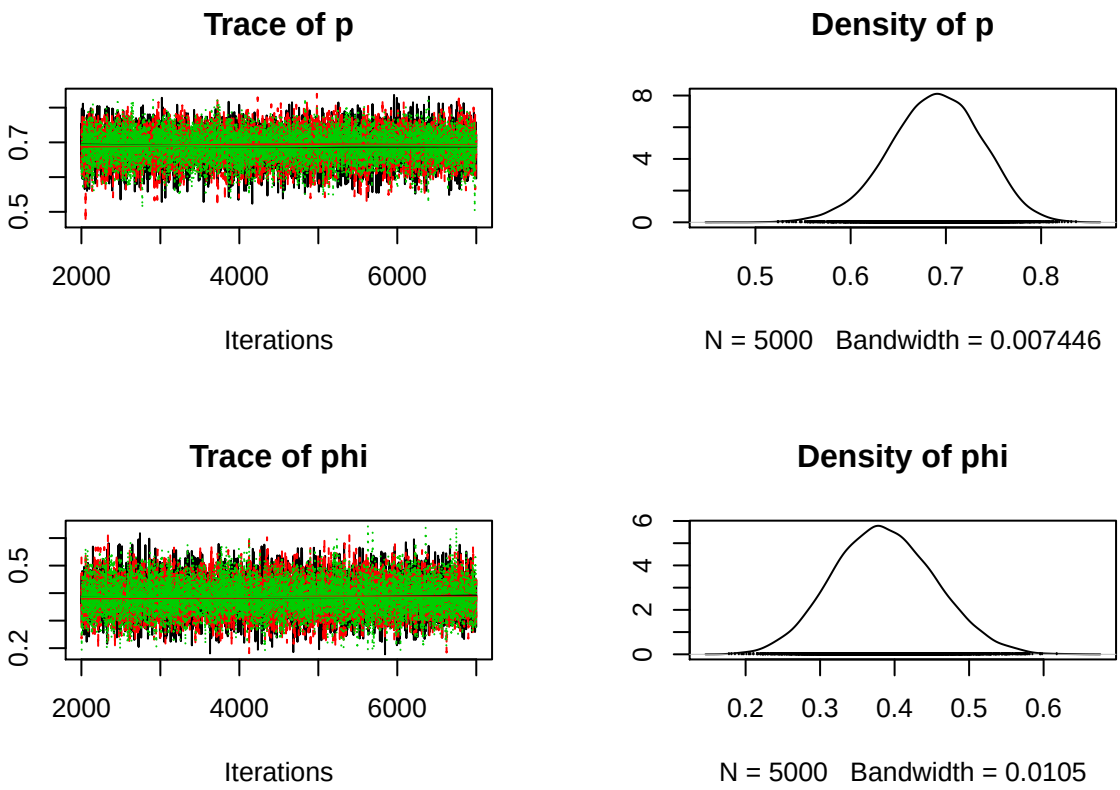
```
dat <- list(W = w, n = n, T = T)
ini <- list(Y = ifelse(rowSums(w) > 0, 1, 0))
fit <- jags.fit(data = dat, params = c("p", "phi"),
    model = model, inits = ini)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 250
##    Unobserved stochastic nodes: 52
##    Total graph size: 358
##
## Initializing model
```

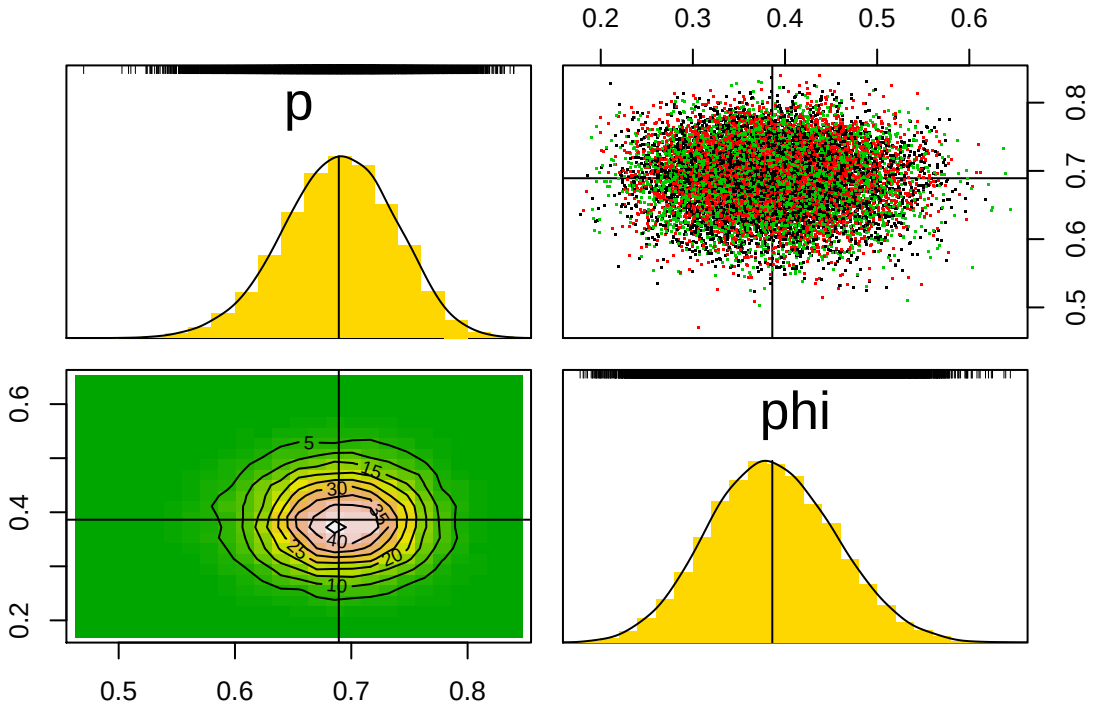
```
summary(fit)
```

```
##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##       Mean      SD  Naive SE Time-series SE
## p   0.6893 0.04806 0.0003924      0.0005077
## phi 0.3862 0.06778 0.0005534      0.0006919
##
## 2. Quantiles for each variable:
##
##       2.5%    25%    50%    75%  97.5%
## p   0.5899 0.6574 0.6906 0.7225 0.7786
## phi 0.2579 0.3387 0.3839 0.4320 0.5222
```

```
plot(fit)
```


Trace of p
0.5
0.7
2000
4000
6000
Iterations
Density of p
0
4
8
0.5
0.6
0.7
0.8
N = 5000 Bandwidth = 0.007446
Trace of phi
0.2
0.5
2000
4000
6000
Iterations
Density of phi
0
2
4
6
0.2
0.3
0.4
0.5
0.6
N = 5000 Bandwidth = 0.0105

```
pairs(fit)
```



0.0.11 Bayesian inference

Effect of priors on the estimation and prediction of the occupancy proportion:

```
model <- custommodel("model {
    for (i in 1:n) {
        Y[i] ~ dbern(p)
        for (t in 1:T) {
            W[i,t] ~ dbern(Y[i] * phi)
        }
    }
    p <- ilogit(logit_p)
    phi <- ilogit(logit_phi)
    logit_p ~ dnorm(-2, 0.01)
    logit_phi ~ dnorm(2, 0.01)
}")
dat <- list(W = w, n = n, T = T)
ini <- list(Y = ifelse(rowSums(w) > 0, 1, 0))
fit2 <- jags.fit(data = dat, params = c("p", "phi"),
    model = model, inits = ini)
```

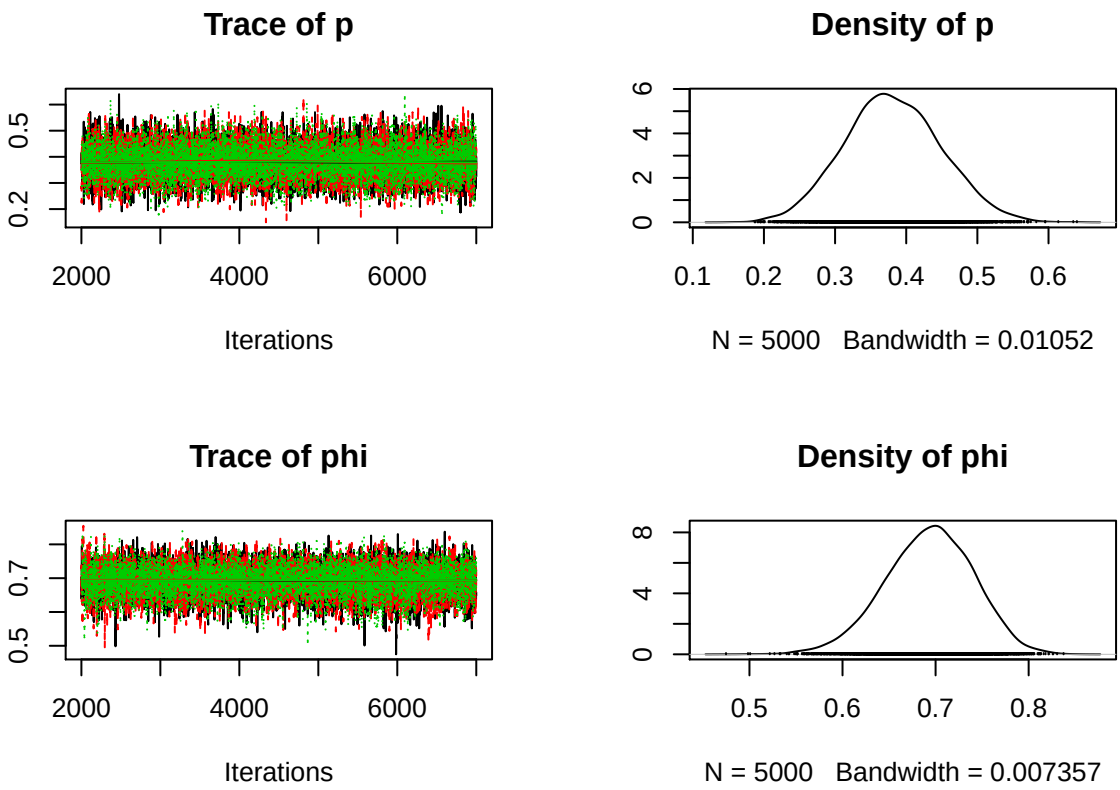
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
```

```
##    Observed stochastic nodes: 250
##    Unobserved stochastic nodes: 52
##    Total graph size: 361
##
## Initializing model
```

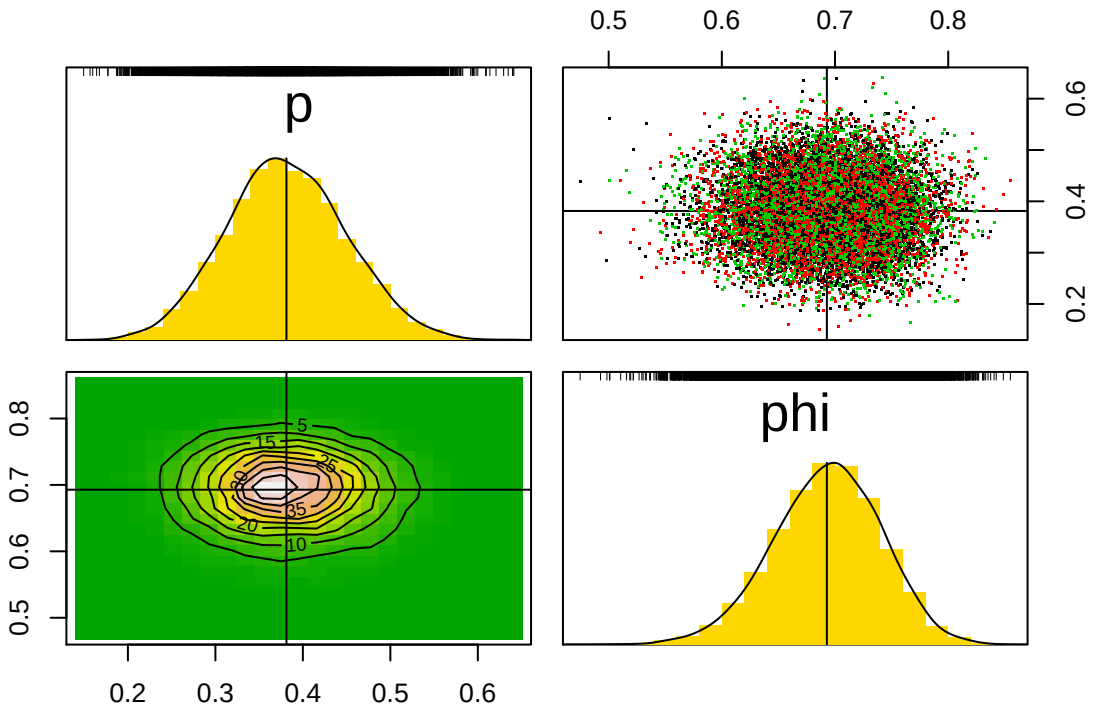
```
summary(fit2)
```

```
##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##       Mean      SD  Naive SE Time-series SE
## p   0.3812 0.06792 0.0005545      0.0007249
## phi 0.6928 0.04749 0.0003878      0.0005108
##
## 2. Quantiles for each variable:
##
##       2.5%    25%    50%   75%  97.5%
## p   0.2520 0.3352 0.3795 0.427 0.5166
## phi 0.5953 0.6615 0.6948 0.726 0.7797
```

```
plot(fit2)
```


Trace of p
0.2
0.5
2000
4000
6000
Iterations
Density of p
0
2
4
6
0.1
0.2
0.3
0.4
0.5
0.6
N = 5000 Bandwidth = 0.01052
Trace of phi
0.5
0.7
2000
4000
6000
Iterations
Density of phi
0
4
8
0.5
0.6
0.7
0.8
N = 5000 Bandwidth = 0.007357

```
pairs(fit2)
```



0.0.12 Data cloning

Frequentist inference: Identifiability check, independence from the specification of the prior check, confidence intervals and predictions for the occupancy proportion.

```
library(dclone)
library(rjags)
model <- custommodel("model {
    for (k in 1:K) {
    for (i in 1:n) {
        Y[i,k] ~ dbern(phi)
        for (t in 1:T) {
            W[i,t,k] ~ dbern(Y[i,k] * p)
        }
    }
    }
    p ~ dunif(0.001, 0.999)
    phi ~ dunif(0.001, 0.999)
}")
dat <- list(W = dcdim(array(w, c(n,T,1))), n = n, T = T, K = 1)
ini <- list(Y = data.matrix(rep(1, n)))
ifun <- function(model, n.clones) {
    list(Y = dclone(dcdim(data.matrix(rep(1, n))), n.clones))
}
dcfit <- dc.fit(data = dat, params = c("p", "phi"),
```

```
    model = model, inits = ini,
    n.clones = c(1,2,4,8), multiply = "K", unchanged = c("n","T"),
    initsfun = ifun)
```

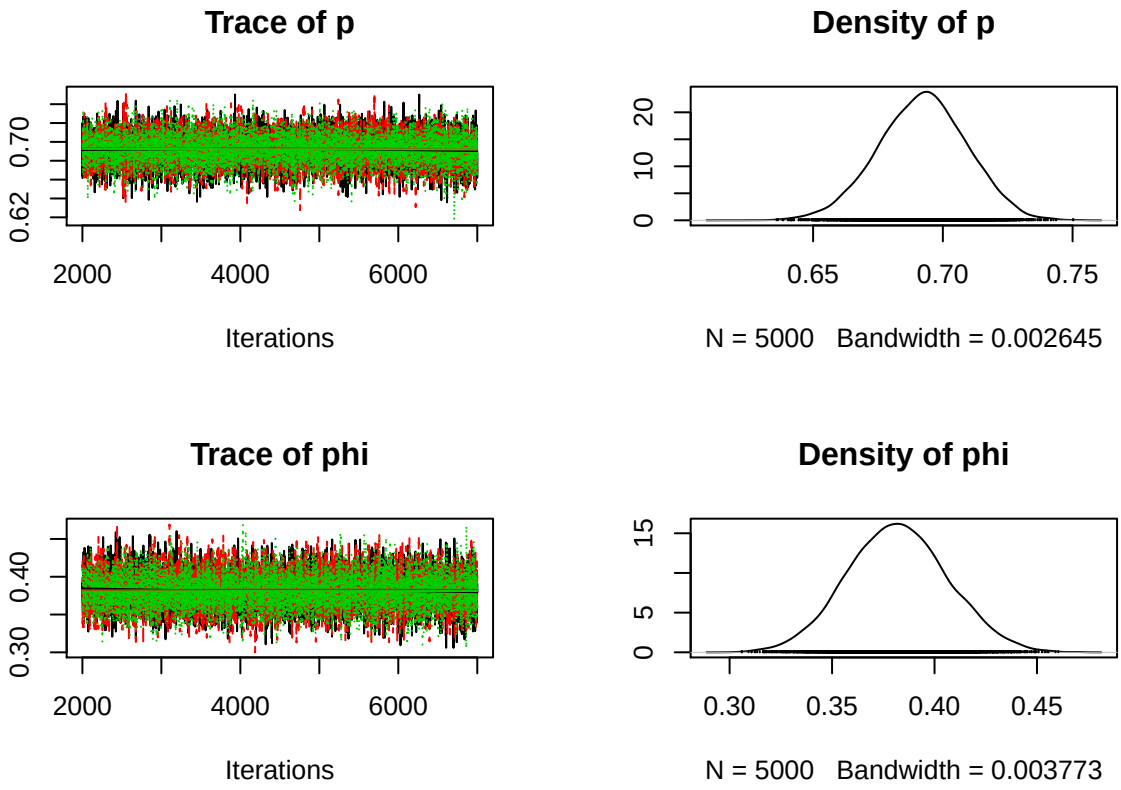
```
##
## Fitting model with 1 clone
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 250
##    Unobserved stochastic nodes: 52
##    Total graph size: 359
##
## Initializing model
##
##
## Fitting model with 2 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 500
##    Unobserved stochastic nodes: 102
##    Total graph size: 709
##
## Initializing model
##
##
## Fitting model with 4 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 202
##    Total graph size: 1409
##
## Initializing model
##
##
## Fitting model with 8 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2000
##    Unobserved stochastic nodes: 402
##    Total graph size: 2809
```

```
##
## Initializing model
```

```
summary(dcfit)
```

```
##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
## Number of clones = 8
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##       Mean      SD   DC SD  Naive SE Time-series SE R hat
## p   0.6922 0.01707 0.04829 0.0001394      0.0001837 1.000
## phi 0.3821 0.02436 0.06889 0.0001989      0.0002542 1.001
##
## 2. Quantiles for each variable:
##
##       2.5%    25%    50%    75%  97.5%
## p   0.6581 0.6808 0.6925 0.7037 0.7249
## phi 0.3353 0.3653 0.3819 0.3983 0.4306
```

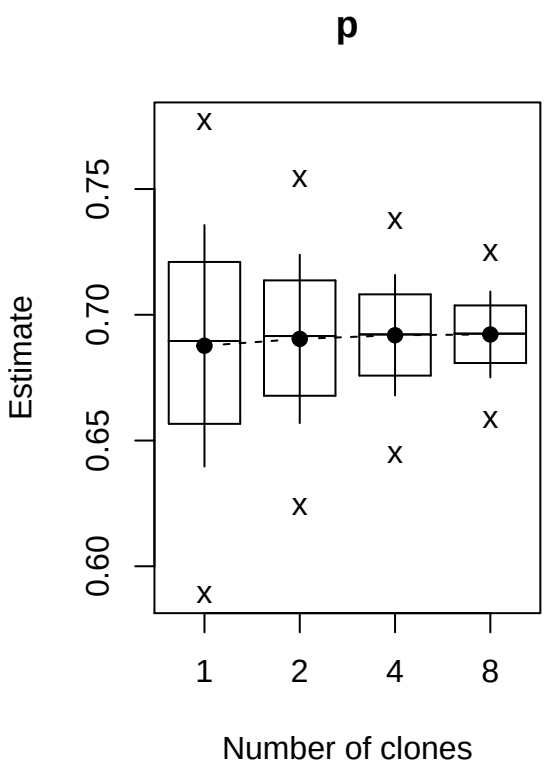
```
plot(dcfit)
```

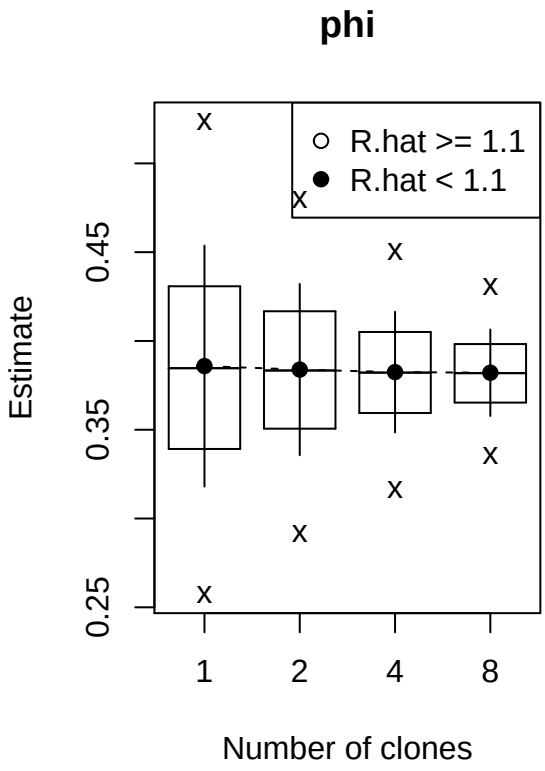


```
dctable(dcfit)
```

```
## $p
##   n.clones      mean         sd      2.5%       25%       50%       75%
## 1        1 0.6876796 0.04799556 0.5888815 0.6566348 0.6895772 0.7210128
## 2        2 0.6904123 0.03346959 0.6235265 0.6678034 0.6915510 0.7136860
## 3        4 0.6918760 0.02395590 0.6443313 0.6758134 0.6921874 0.7081302
## 4        8 0.6921694 0.01707297 0.6581475 0.6808260 0.6924950 0.7037301
##       97.5%    r.hat
## 1 0.7769193 1.000069
## 2 0.7539874 1.000599
## 3 0.7375322 1.001096
## 4 0.7249083 1.000328
##
## $phi
##   n.clones      mean         sd      2.5%       25%       50%       75%
## 1        1 0.3858494 0.06783285 0.2573580 0.3391781 0.3846634 0.4308015
## 2        2 0.3838812 0.04821476 0.2914532 0.3506055 0.3833661 0.4167206
## 3        4 0.3824623 0.03398936 0.3163785 0.3594094 0.3821711 0.4050521
## 4        8 0.3821071 0.02435574 0.3352866 0.3652863 0.3818664 0.3982514
##       97.5%    r.hat
## 1 0.5236792 1.000894
## 2 0.4794039 1.000460
## 3 0.4505861 1.001489
## 4 0.4305690 1.001195
##
## attr(,"class")
## [1] "dctable"
```

```
plot(dctable(dcfit))
```

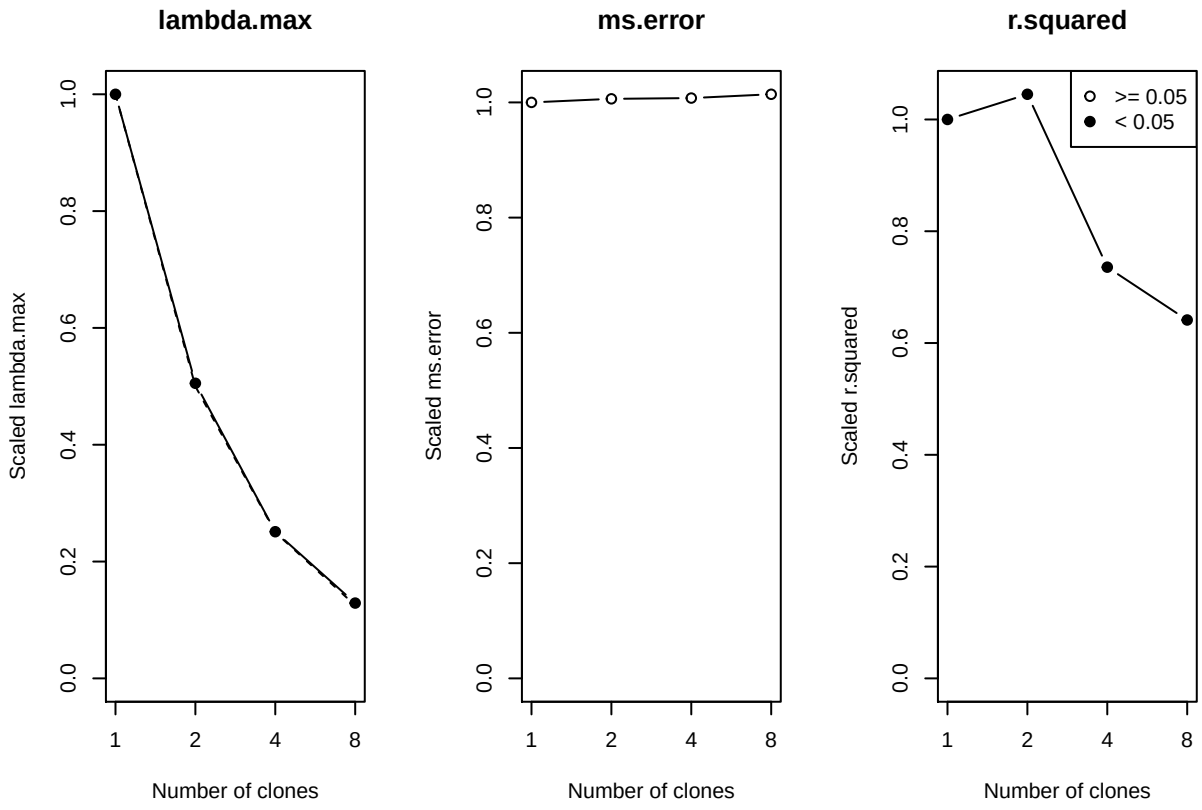





```
dcdiag(dcfit)
```

```
##   n.clones  lambda.max ms.error   r.squared    r.hat
## 1        1 0.009203420 1.234163 0.006270337 1.000608
## 2        2 0.004649645 1.241814 0.006552996 1.000753
## 3        4 0.002310627 1.243466 0.004613258       NA
## 4        8 0.001186438 1.251609 0.004020593       NA
```

```
plot(dcdiag(dcfit))
```




```
#pairs(dcfit)
```

0.0.13 Generalization to take into account covariates

p and φ can be a function of independent variables with values varying across the n location, for example:

- $$p_i = \frac{exp(\theta_0 + \theta_1 z_i)}{1 + exp(\theta_0 + \theta_1 z_i)}$$

  .
- $$\varphi_i = \frac{exp(\beta_0 + \beta_1 x_i)}{1 + exp(\beta_0 + \beta_1 x_i)}$$

  ,

R code for data generation:

```
set.seed(1234)
n <- 1000
x <- rnorm(n)
z <- rnorm(n)
beta <- c(0, 1)
theta <- c(0.2, -0.5)
p <- exp(theta[1] + theta[2] * z) / (1 + exp(theta[1] + theta[2] * z))
```

```r
phi <- exp(beta[1] + beta[2] * x) / (1 + exp(beta[1] + beta[2] * x))
#p <- plogis(model.matrix(~z) %*% theta)
#phi <- plogis(model.matrix(~x) %*% beta)
y <- rbinom(n = n, size = 1, prob = phi)
w <- rbinom(n = n, size = y, prob = p)
table(Y = y, W = w)
```

```
##    W
## Y     0   1
##   0 507   0
##   1 209 284
```

```r
naive <- glm(w ~ x, family = binomial("logit"))
summary(naive)
```

```
##
## Call:
## glm(formula = w ~ x, family = binomial("logit"))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6023  -0.8300  -0.6679   1.1997   2.3628
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.98210    0.07425 -13.227  < 2e-16 ***
## x            0.60750    0.07803   7.785 6.95e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1193.4  on 999  degrees of freedom
## Residual deviance: 1126.3  on 998  degrees of freedom
## AIC: 1130.3
##
## Number of Fisher Scoring iterations: 4
```

```r
library(detect)
```

```
## Loading required package: Formula
```

```
## Loading required package: stats4
```

```
## Loading required package: pbapply
```

```
## detect 0.4-0      2016-03-02
```

```r
m <- svocc(w ~ x | z)
summary(m)
```

```
##
## Call:
## svocc(formula = w ~ x | z)
##
##
## Single visit site-occupancy model
## Maximum Likelihood estimates (optim method)
##
## Occupancy model coefficients with cloglog link:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.1876     0.3403  -0.551    0.581
## x             0.8164     0.2062   3.959 7.53e-05 ***
## Detection model coefficients with logit link:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.03483    0.36747   0.095  0.92450
## z           -0.40339    0.12791  -3.154  0.00161 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood:  -554 on 4 Df
## AIC =  1116
```

```
model <- custommodel("model {
    for (i in 1:n) {
        W[i] ~ dbin(p[i] * phi[i], K)
        logit(p[i]) <- inprod(Z[i,], theta)
        logit(phi[i]) <- inprod(X[i,], beta)
    }
    beta[1] ~ dnorm(0, 0.001)
    beta[2] ~ dnorm(0, 0.001)
    theta[1] ~ dnorm(0, 0.001)
    theta[2] ~ dnorm(0, 0.001)
}")
dat <- list(W = w, n = n, K = 1,
    X = model.matrix(~x), Z = model.matrix(~z))
dcfit <- dc.fit(data = dat,
    params = c("beta", "theta"), model = model,
    n.clones = c(1, 10), n.iter = 2000,
    unchanged = c("W", "n", "X", "Z"), multiply = "K")
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 4
##    Total graph size: 12016
##
## Initializing model
##
##
```

```
## Fitting model with 10 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 4
##    Total graph size: 12016
##
## Initializing model

## Warning in dclone::.dcFit(data, params, model, inits, n.clones, multiply =
## multiply, : chains convergence problem, see R.hat values
```

```
summary(dcfit)
```

```
##
## Iterations = 2001:4000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 2000
## Number of clones = 10
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean     SD  DC SD Naive SE Time-series SE R hat
## beta[1]  -0.9759 1.3522 4.2760 0.017457        0.39705 1.601
## beta[2]   0.8181 0.3586 1.1338 0.004629        0.05165 1.317
## theta[1] -1.7899 1.4921 4.7184 0.019263        0.41119 1.721
## theta[2] -0.2995 0.2326 0.7355 0.003002        0.04166 1.536
##
## 2. Quantiles for each variable:
##
##             2.5%     25%     50%     75%    97.5%
## beta[1]  -3.4532 -2.1277 -0.6380  0.1164  1.00808
## beta[2]   0.3628  0.5143  0.7430  1.0640  1.62877
## theta[1] -3.1407 -2.8490 -2.4579 -1.0966  2.17182
## theta[2] -1.0575 -0.3072 -0.2325 -0.1798 -0.09424
```

```
#plot(dcfit)
dctable(dcfit)
```

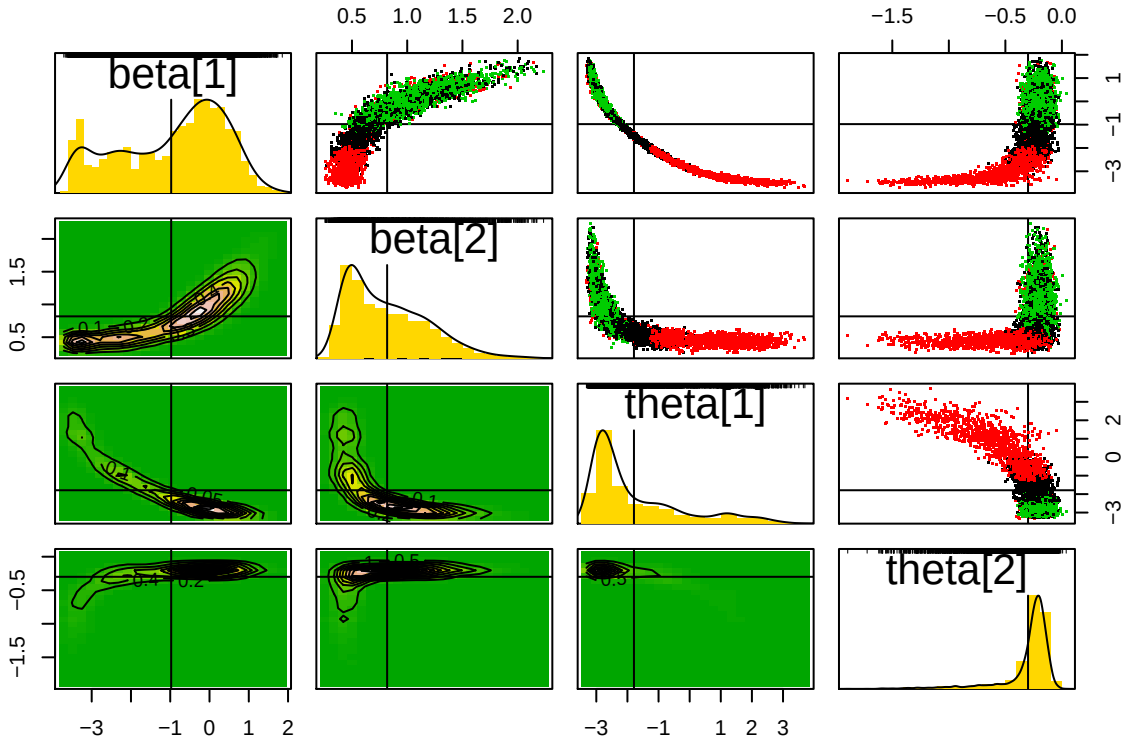
```
## $`beta[1]`
##   n.clones        mean        sd       2.5%        25%         50%
## 1        1 -0.01819852 0.4672666 -0.8150219 -0.3698013 -0.05507758
## 2       10 -0.97591612 1.3521834 -3.4532441 -2.1277067 -0.63799913
##         75%     97.5%    r.hat
## 1 0.2990124 0.9503273 1.016092
## 2 0.1164379 1.0080759 1.601027
##
```

```
## $`beta[2]`
##   n.clones      mean        sd      2.5%       25%       50%      75%
## 1        1 1.0184972 0.2616443 0.6095433 0.8264900 0.9800229 1.179661
## 2       10 0.8181394 0.3585541 0.3628013 0.5142885 0.7430445 1.064023
##      97.5%    r.hat
## 1 1.633045 1.016185
## 2 1.628767 1.316962
##
## $`theta[1]`
##   n.clones       mean        sd       2.5%        25%        50%
## 1        1  0.5741953 0.7345053 -0.3021427  0.0777142  0.3970229
## 2       10 -1.7898897 1.4920781 -3.1406978 -2.8489600 -2.4578686
##          75%    97.5%    r.hat
## 1  0.8647042 2.499524 1.026262
## 2 -1.0965956 2.171816 1.720955
##
## $`theta[2]`
##   n.clones       mean        sd      2.5%        25%        50%        75%
## 1        1 -0.5557422 0.2589582 -1.244491 -0.6556426 -0.4991687 -0.3896529
## 2       10 -0.2994957 0.2325716 -1.057509 -0.3071745 -0.2324525 -0.1797858
##         97.5%    r.hat
## 1 -0.22785273 1.022989
## 2 -0.09423834 1.535758
##
## attr(,"class")
## [1] "dctable"
```

```
#plot(dctable(dcfit))
dcdiag(dcfit)
```

```
##   n.clones lambda.max  ms.error r.squared    r.hat
## 1        1  0.7942122 34.777607 0.4952484 1.019778
## 2       10  4.0406656  7.125958 0.1681139 1.438278
```

```
#plot(dcdiag(dcfit))
pairs(dcfit)
```



For a quasi-Bayesian approach, see here how to utilize the naive estimator to stabilize single visit based estimates:

```
model <- custommodel("model {
    for (i in 1:n) {
        W[i] ~ dbin(p[i] * phi[i], K)
        logit(p[i]) <- inprod(Z[i,], theta)
        logit(phi[i]) <- inprod(X[i,], beta)
    }
    beta[1] ~ dnorm(naive[1], penalty)
    beta[2] ~ dnorm(naive[2], penalty)
    theta[1] ~ dnorm(0, 0.001)
    theta[2] ~ dnorm(0, 0.001)
}")
dat <- list(W = w, n = n, K = 1,
    X = model.matrix(~x), Z = model.matrix(~z),
    naive = coef(naive), penalty = 0.5)
dcfit <- dc.fit(data = dat,
    params = c("beta", "theta"), model = model,
    n.clones = c(1, 10, 100),
    n.update = 5000, n.iter = 2000,
    unchanged = c("W", "n", "X", "Z", "naive", "penalty"),
    multiply = "K")
```

```
##
## Fitting model with 1 clone
```

```
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 4
##    Total graph size: 12015
##
## Initializing model
##
##
## Fitting model with 10 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 4
##    Total graph size: 12015
##
## Initializing model
##
##
## Fitting model with 100 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 4
##    Total graph size: 12015
##
## Initializing model
```

```
summary(dcfit)
```

```
##
## Iterations = 6001:8000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 2000
## Number of clones = 100
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean      SD  DC SD  Naive SE Time-series SE  R hat
## beta[1]  -0.2208 0.66191 6.6191 0.0085453       0.076249 1.0195
## beta[2]   0.9517 0.27991 2.7991 0.0036137       0.023620 1.0142
## theta[1] -5.0092 0.35928 3.5928 0.0046383       0.049949 1.0263
## theta[2] -0.1927 0.06036 0.6036 0.0007793       0.001075 0.9998
```

```
##
## 2. Quantiles for each variable:
##
##              2.5%     25%     50%     75%    97.5%
## beta[1]  -1.6928 -0.6128 -0.1755  0.2408  0.95381
## beta[2]   0.4894  0.7470  0.9295  1.1308  1.54438
## theta[1] -5.4752 -5.2597 -5.0863 -4.8501 -4.06869
## theta[2] -0.3133 -0.2339 -0.1920 -0.1506 -0.07762
```

```
#plot(dcfit)
dctable(dcfit)
```

```
## $`beta[1]`
##   n.clones       mean        sd       2.5%        25%        50%
## 1        1 -0.1599874 0.4228195 -0.9460069 -0.4427936 -0.1776284
## 2       10 -0.3740762 0.8930892 -2.8595797 -0.7420439 -0.2107851
## 3      100 -0.2208288 0.6619149 -1.6928089 -0.6127773 -0.1755120
##          75%     97.5%    r.hat
## 1 0.08845285 0.7773330 1.012669
## 2 0.19812343 0.9555311 1.208522
## 3 0.24076180 0.9538128 1.019516
##
## $`beta[2]`
##   n.clones      mean        sd      2.5%       25%       50%      75%
## 1        1 0.9534600 0.2295623 0.5897078 0.7892177 0.9267381 1.086907
## 2       10 0.9196059 0.2958398 0.4309286 0.7028744 0.8996803 1.109365
## 3      100 0.9516845 0.2799131 0.4894487 0.7470253 0.9294549 1.130779
##      97.5%    r.hat
## 1 1.489567 1.003902
## 2 1.529192 1.053449
## 3 1.544385 1.014174
##
## $`theta[1]`
##   n.clones       mean        sd       2.5%        25%        50%       75%
## 1        1  0.9251492 1.5638217 -0.2088942  0.2478628  0.5492603  1.014738
## 2       10 -2.1884669 3.8676936 -3.1364383 -2.8962358 -2.7080118 -2.390118
## 3      100 -5.0091513 0.3592842 -5.4751813 -5.2597290 -5.0862733 -4.850121
##        97.5%    r.hat
## 1  7.0726039 1.270178
## 2 -0.1155289 1.314645
## 3 -4.0686929 1.026340
##
## $`theta[2]`
##   n.clones       mean         sd       2.5%        25%        50%
## 1        1 -0.6176311 0.43672588 -1.6152989 -0.6883216 -0.5391786
## 2       10 -0.2694694 1.00873682 -0.4278817 -0.2583411 -0.2114397
## 3      100 -0.1927470 0.06036376 -0.3133123 -0.2339495 -0.1920441
##          75%       97.5%     r.hat
## 1 -0.4167900 -0.23190698 1.2004089
## 2 -0.1653971 -0.08087662 1.2958771
## 3 -0.1505965 -0.07761668 0.9998176
##
## attr(,"class")
## [1] "dctable"
```

```
#plot(dctable(dcfit))
dcdiag(dcfit)
```

```
##   n.clones lambda.max   ms.error  r.squared    r.hat
## 1        1  2.6116874  87.548887 0.50980303 1.068480
## 2       10 15.5107642 731.917051 0.81960988 1.105316
## 3      100  0.6146943   2.085642 0.07763422 1.014166
```

```
#plot(dcdiag(dcfit))
pairs(dcfit)
```