Analysing data with spatial dependence

Peter Solymos and Subhash Lele

July 16, 2016 — Madison, WI — NACCB Congress

Kriging example

Exponential decay is used ($e^{-\lambda D}$). Try different values for λ (0, 0.1, 0.5). Try modifying it to half-Normal ($e^{-(\lambda D)^2}$).

The models is defined as:

$$Y \sim MVN(\mu, \mathbf{\Sigma})$$

where μ is the mean vector for the Multivariate Normal (MVN) distribution, and $\mathbf{\Sigma}$ is the variance-covariance matrix with a spatial dependence structure. $\mathbf{\Sigma}$ is defined as $\sigma^2 exp(-\lambda \mathbf{D})$, where $\mathbf{D}$ is an n-by-n spatial distance matrix (we use Euclidean distances).

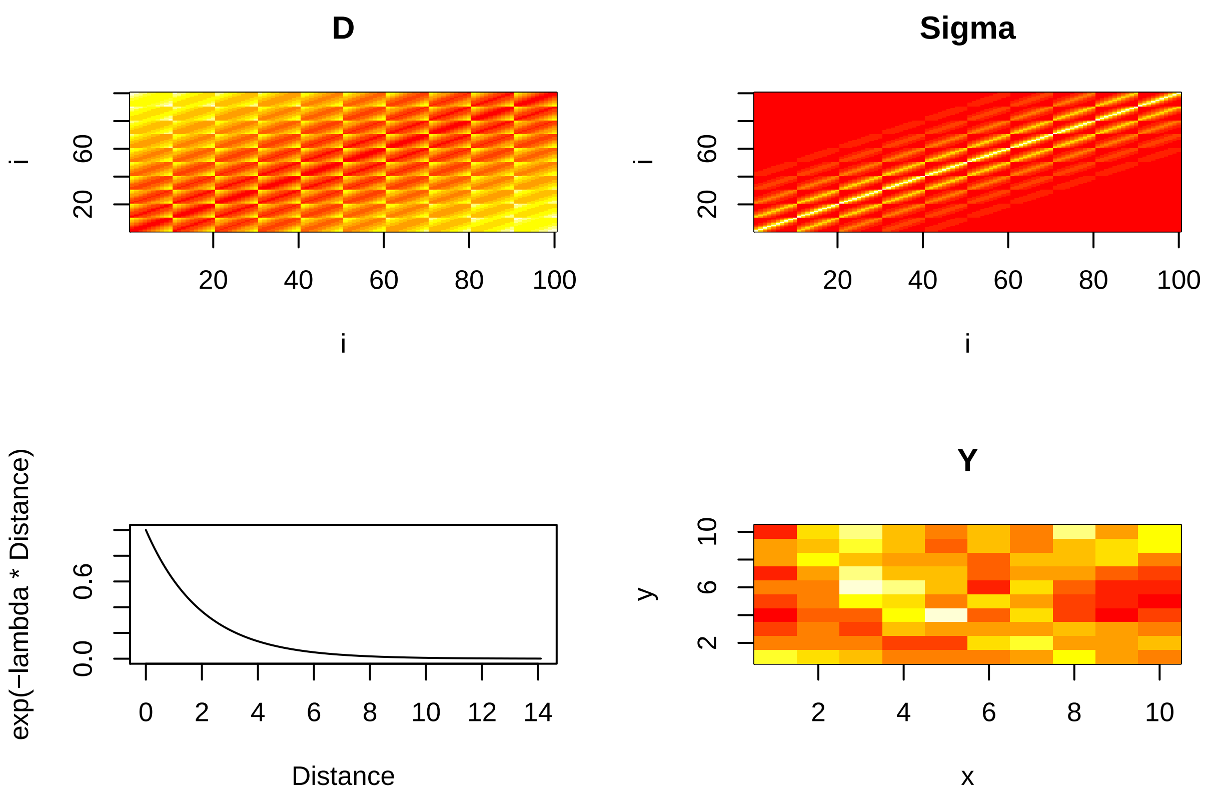
```r
set.seed(2345)
library(MASS)
mu <- 5 # global mean
sigma_sq <- 1 # global variance
lambda <- 0.5

## set up an m x m square lattice
m <- 10
xy <- expand.grid(x=seq_len(m), y=seq_len(m))
n <- nrow(xy)
D <- as.matrix(dist(xy))

Sigma <- sigma_sq * exp(-lambda*D)

Y <- mvrnorm(1, rep(mu, n), Sigma)

op <- par(mfrow = c(2, 2))
image(seq_len(n), seq_len(n), D, main = "D", ylab="i", xlab="i")
image(seq_len(n), seq_len(n), Sigma, main="Sigma", ylab="i", xlab="i")
Distance <- seq(0, m * sqrt(2), by = 0.1)
plot(Distance, exp(-lambda*Distance), type = "l")
image(seq_len(m), seq_len(m), matrix(Y, m, m), main = "Y", ylab="y", xlab="x")
```



```
par(op)
```

Bayesian analysis:

```
library(dclone)
```

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

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

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

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

```
model <- custommodel("model {
    for (i in 1:n) {
        for (j in 1:n) {
            Sigma[i,j] <- sigma_sq * exp(-lambda*D[i,j])
        }
        mu_vec[i] <- mu
    }
    Y[1:n] ~ dmnorm(mu_vec, invSigma)
    invSigma <- inverse(Sigma)
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
    lambda ~ dgamma(1, 0.1)
}")
dat <- list(Y = Y, n = n, D = D)
fit <- jags.fit(data = dat, params = c("mu", "sigma_sq","lambda"),
    model = model, n.iter = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 3
##    Total graph size: 10172
##
## Initializing model
```

```
summary(fit)
```

```
##
## Iterations = 2001:3000
## Thinning interval = 1
## Number of chains = 3
## 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
## lambda   0.7534 0.3093 0.005647        0.02592
## mu       5.3434 0.4256 0.007770        0.05203
## sigma_sq 0.8470 0.7244 0.013226        0.14267
##
## 2. Quantiles for each variable:
##
##            2.5%    25%    50%    75% 97.5%
## lambda   0.1200 0.5606 0.7429 0.9476 1.372
## mu       4.1510 5.2322 5.3995 5.5430 5.950
## sigma_sq 0.4248 0.5449 0.6479 0.8113 3.223
```

DC: replicating the whole experiment K times (i.e. clones are independent, and identical in terms of within-clone dependence structure)

```
library(dclone)
model <- custommodel("model {
    for (k in 1:K) {
        for (i in 1:n) {
            for (j in 1:n) {
                Sigma[i,j,k] <- sigma_sq * exp(-lambda*D[i,j])
            }
            mu_vec[i,k] <- mu # mu_vec does not really require cloning
        }
        Y[1:n,k] ~ dmnorm(mu_vec[1:n,k], invSigma[1:n,1:n,k])
        invSigma[1:n,1:n,k] <- inverse(Sigma[1:n,1:n,k])
    }
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
    lambda ~ dgamma(1, 0.1)
}")
dat <- list(Y = dcdim(data.matrix(Y)), n = n, D = D, K=1)
dcfit <- dc.fit(data = dat, params = c("mu", "sigma_sq","lambda"),
    model = model, n.iter = 1000,
    n.clones=c(1,2),
    multiply="K", unchanged=c("n", "D"))
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 3
##    Total graph size: 10173
##
## Initializing model
##
##
## Fitting model with 2 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2
##    Unobserved stochastic nodes: 3
##    Total graph size: 10177
##
## Initializing model
```

```
summary(dcfit)
```

```
##
## Iterations = 2001:3000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
## Number of clones = 2
##
## 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
## lambda   0.8286 0.1783 0.2521 0.003254       0.009555 1.0012
## mu       5.4232 0.1491 0.2108 0.002722       0.002788 0.9999
## sigma_sq 0.6153 0.1171 0.1656 0.002138       0.007414 1.0082
##
## 2. Quantiles for each variable:
##
##            2.5%    25%    50%    75% 97.5%
## lambda   0.4904 0.7096 0.8261 0.9440 1.185
## mu       5.1263 5.3344 5.4264 5.5173 5.721
## sigma_sq 0.4533 0.5384 0.5945 0.6668 0.894
```

```r
dcdiag(dcfit)
```

```
##   n.clones lambda.max  ms.error r.squared    r.hat
## 1        1 0.13317493 21.914538 0.3896628 1.009737
## 2        2 0.04148734  8.439124 0.3168109 1.002144
```

Inverse Wishart prior, σ^2 and λ is hard to recover (it requires post processing the posterior estimates, i.e. monitor the whole `invSigma` matrix or its inverse):

```r
library(dclone)
model <- custommodel("model {
    for (i in 1:n) {
        mu_vec[i] <- mu
    }
    Y[1:n] ~ dmnorm(mu_vec, invSigma)
    invSigma[1:n,1:n] ~ dwish(R[1:n,1:n], n)
    mu ~ dnorm(0, 0.1)
}")
dat <- list(Y = Y, n = n, R = diag(1, n, n))
fit <- jags.fit(data = dat, params = "mu",
    model = model, n.iter = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 2
##    Total graph size: 10008
##
## Initializing model
```

```r
summary(fit)
```

```
##
## Iterations = 1001:2000
## Thinning interval = 1
## Number of chains = 3
## 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
##       5.441458       0.072554       0.001325       0.009116
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
## 5.294 5.395 5.441 5.485 5.597
```

DC for inverse Wishart parametrization:

```
library(dclone)
model <- custommodel("model {
    for (i in 1:n) {
        mu_vec[i] <- mu
    }
    for (k in 1:K) {
        Y[1:n,k] ~ dmnorm(mu_vec[1:n], invSigma[1:n,1:n,k])
        invSigma[1:n,1:n,k] ~ dwish(R[1:n,1:n], n)
    }
    mu ~ dnorm(0, 0.1)
}")
dat <- list(Y = dcdim(data.matrix(Y)), n = n, R = diag(1, n, n), K=1)
dcfit <- dc.fit(data = dat, params = "mu",
    model = model, n.iter = 1000,
    n.clones=c(1,2),
    multiply="K", unchanged=c("n", "R"))
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 2
##    Total graph size: 10009
##
## Initializing model
##
##
## Fitting model with 2 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2
##    Unobserved stochastic nodes: 3
##    Total graph size: 10011
```

```
##
## Initializing model

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

```
summary(dcfit)
```

```
##
## Iterations = 1001:2000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
## Number of clones = 2
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##     Mean     SD DC SD.mu Naive SE Time-series SE R hat
## mu 5.468 0.0559  0.07905 0.001021       0.009945 1.134
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
## 5.346 5.431 5.464 5.507 5.579
```

```
dcdiag(dcfit)
```

```
##   n.clones  lambda.max   ms.error   r.squared    r.hat
## 1        1 0.005035764 0.01438904 0.005440831 1.077591
## 2        2 0.003124331 0.04229511 0.021034054 1.134177
```

Correlation (ρ) based parametrization. Try different values for ρ.

Same model as above, but now $\mathbf{\Sigma}$ is defined as $\sigma^2\rho^{\mathbf{D}}$).

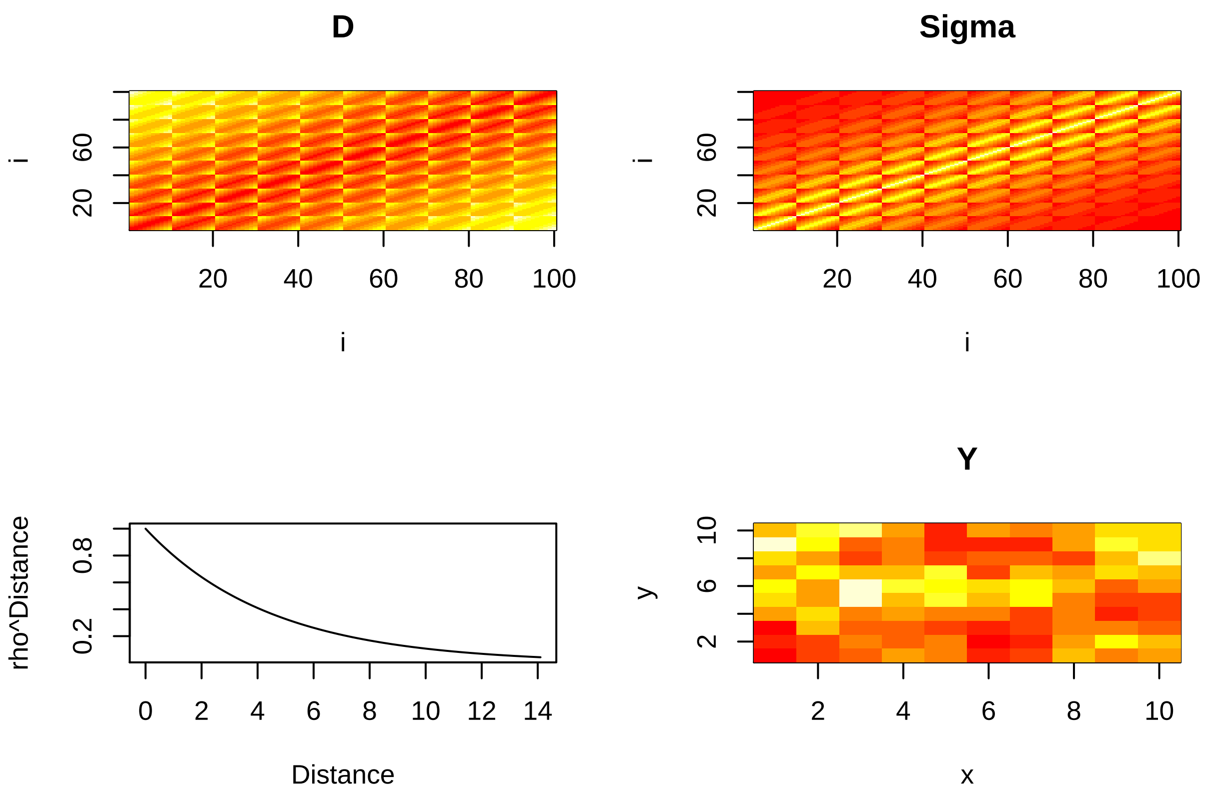
```
set.seed(2345)
library(MASS)
mu <- 5
sigma_sq <- 1
rho <- 0.8

## set up an m x m square lattice
m <- 10
xy <- expand.grid(x=seq_len(m), y=seq_len(m))
n <- nrow(xy)
D <- as.matrix(dist(xy))

Sigma <- sigma_sq * rho^D

Y <- mvrnorm(1, rep(mu, n), Sigma)
```

```
op <- par(mfrow = c(2, 2))
image(seq_len(n), seq_len(n), D, main = "D", ylab="i", xlab="i")
image(seq_len(n), seq_len(n), Sigma, main="Sigma", ylab="i", xlab="i")
Distance <- seq(0, m * sqrt(2), by = 0.1)
plot(Distance, rho^Distance, type = "l")
image(seq_len(m), seq_len(m), matrix(Y, m, m), main = "Y", ylab="y", xlab="x")
```



```
par(op)
```

Bayesian analysis:

```
library(dclone)
model <- custommodel("model {
    for (i in 1:n) {
        for (j in 1:n) {
            Sigma[i,j] <- sigma_sq * rho^D[i,j]
        }
        mu_vec[i] <- mu
    }
    Y[1:n] ~ dmnorm(mu_vec, invSigma)
    invSigma <- inverse(Sigma)
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
    rho ~ dunif(0, 0.999)
}")
```

```
dat <- list(Y = Y, n = n, D = D)
fit <- jags.fit(data = dat, params = c("mu", "sigma_sq","rho"),
    model = model, n.iter = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 3
##    Total graph size: 10120
##
## Initializing model
```

```
summary(fit)
```

```
##
## Iterations = 2001:3000
## Thinning interval = 1
## Number of chains = 3
## 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
## mu       5.6865 0.3494 0.006378        0.01442
## rho      0.6155 0.1301 0.002375        0.01122
## sigma_sq 0.5253 0.3664 0.006690        0.05099
##
## 2. Quantiles for each variable:
##
##            2.5%    25%    50%    75%  97.5%
## mu       4.8923 5.5459 5.7102 5.8604 6.3088
## rho      0.3789 0.5217 0.6058 0.7045 0.8851
## sigma_sq 0.2468 0.3372 0.4231 0.5698 1.5836
```

DC:

```
model <- custommodel("model {
    for (k in 1:K) {
        for (i in 1:n) {
            for (j in 1:n) {
                Sigma[i,j,k] <- sigma_sq * rho^D[i,j]
            }
            mu_vec[i,k] <- mu # mu_vec does not really require cloning
        }
        Y[1:n,k] ~ dmnorm(mu_vec[1:n,k], invSigma[1:n,1:n,k])
        invSigma[1:n,1:n,k] <- inverse(Sigma[1:n,1:n,k])
    }
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
    rho ~ dunif(0, 0.999)
}")
dat <- list(Y = dcdim(data.matrix(Y)), n = n, D = D, K=1)
dcfit <- dc.fit(data = dat, params = c("mu", "sigma_sq","rho"),
    model = model, n.iter = 1000,
    n.clones=c(1,2),
    multiply="K", unchanged=c("n", "D"))
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 3
##    Total graph size: 10121
##
## Initializing model
##
##
## Fitting model with 2 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2
##    Unobserved stochastic nodes: 3
##    Total graph size: 10125
##
## Initializing model
```

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

```
summary(dcfit)
```

```
##
## Iterations = 2001:3000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
## Number of clones = 2
##
## 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
## mu       5.7260 0.17593 0.2488 0.003212       0.004198 1.071
## rho      0.5629 0.09329 0.1319 0.001703       0.009024 1.144
## sigma_sq 0.4039 0.18722 0.2648 0.003418       0.032499 1.309
##
## 2. Quantiles for each variable:
##
##            2.5%    25%    50%    75%  97.5%
## mu       5.4097 5.6337 5.7322 5.8211 6.0278
## rho      0.4102 0.4998 0.5519 0.6128 0.8099
## sigma_sq 0.2610 0.3190 0.3631 0.4236 0.9161
```

```
dcdiag(dcfit)
```

```
##   n.clones lambda.max ms.error r.squared    r.hat
## 1        1 0.43500305 88.32530 0.5809173 1.084551
## 2        2 0.04311904 99.68115 0.6495776 1.095697
```

Cluster sampling design, Poisson-Lognormal GLMM

We assume that points within clusters are dependent (i.e. close to each other to assume high dependence), but clusters are independent (i.e. far apart to assume independence). We assume each cluster has same number of points for simplicity, but number of points within cluster can vary.

The model for any given cluster is defined as: $(Y_{ij} \mid \lambda_i) \sim Poisson(\lambda_i)$, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$, $log(\lambda_i) = \alpha_i + \mu$, and $\alpha_\mathbf{i} \sim MVN(\mathbf{0}, \mathbf{\Sigma})$.

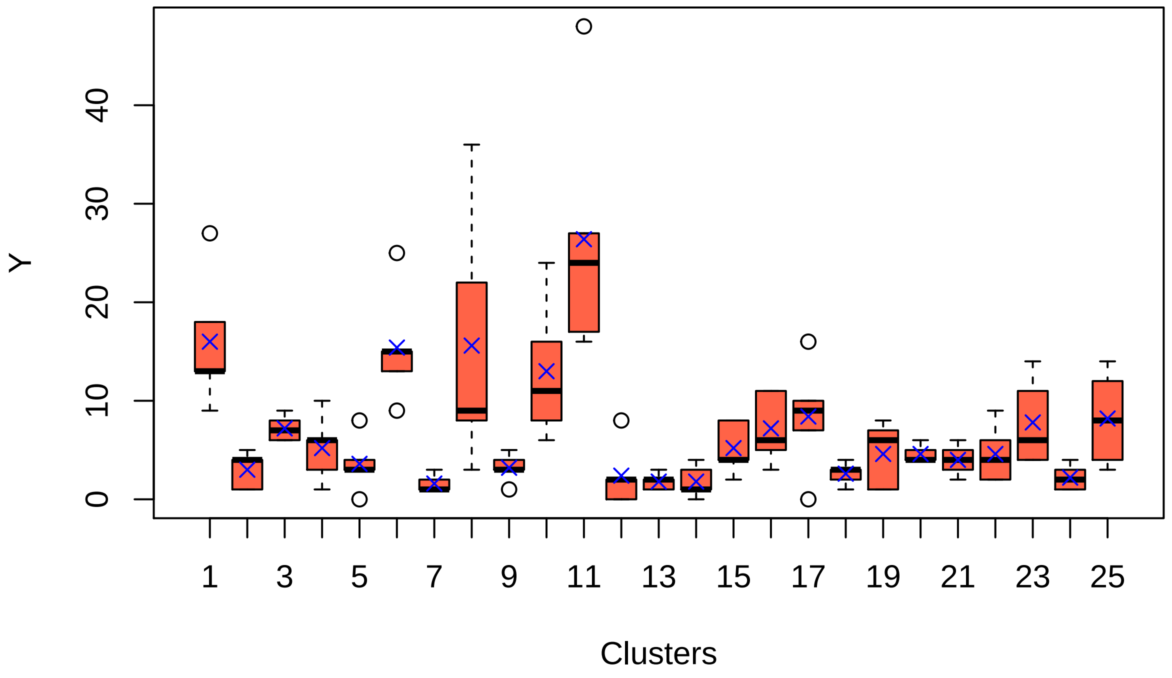
$\mathbf{\Sigma}$ is the m-by-m variance covariance matrix for cluster i. Diagonal elements are defined as σ^2, off-diagonal elements are defined as $\sigma^2\rho$.

```
set.seed(1234)
library(MASS)
## total sample size is n x m
m <- 5 # number of points in a cluster
n <- 25 # number of clusters
mu <- 1.6 # global mean on log scale
sigma_sq <- 1 # global variance
rho <- 0.8

## variance-covariance matrix for a single cluster
Sigma1 <- sigma_sq * diag(1, m, m) + sigma_sq * rho * (1 - diag(1, m, m))
## the full variance covariance matrix is 'block-diagonal'
## which means it is filled with zeros across clusters
alpha <- matrix(0, n, m)
for (i in 1:n) {
    alpha[i,] <- mvrnorm(1, rep(0, m), Sigma1)
}
Y <- rpois(n*m, exp(as.numeric(alpha) + mu))
dim(Y) <- dim(alpha)
Y
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]   18   13    9   13   27
##  [2,]    1    5    4    4    1
##  [3,]    7    6    8    9    6
##  [4,]    6    1    3   10    6
##  [5,]    0    4    3    8    3
##  [6,]   15   25   13   15    9
##  [7,]    2    3    1    1    1
##  [8,]    3    8   22   36    9
##  [9,]    1    3    3    5    4
## [10,]    6    8   11   24   16
## [11,]   16   24   17   48   27
## [12,]    0    8    2    0    2
## [13,]    1    3    2    2    1
## [14,]    1    1    4    0    3
## [15,]    8    2    8    4    4
## [16,]    5    3   11   11    6
## [17,]    7    9   10   16    0
## [18,]    3    4    1    3    2
## [19,]    1    6    7    1    8
## [20,]    6    4    5    4    4
## [21,]    4    3    2    5    6
## [22,]    4    2    2    9    6
## [23,]    4   11   14    6    4
## [24,]    1    2    3    4    1
## [25,]    8    4   14    3   12
```

```
boxplot(t(Y), xlab="Clusters", ylab="Y", col="tomato")
points(1:n, rowMeans(Y), pch=4, col=4)
```



Bayesian analysis:

```
library(dclone)
model <- custommodel("model {
    for (i in 1:n) {
        for (j in 1:m) {
            Y[i,j] ~ dpois(lambda[i,j])
            lambda[i,j] <- exp(alpha[i,j] + mu)
        }
        alpha[i,1:m] ~ dmnorm(zeros, invSigma)
    }
    Sigma <- sigma_sq * R + sigma_sq * rho * (1 - R)
    invSigma <- inverse(Sigma)
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
    rho ~ dunif(0, 0.999)
}")
dat <- list(Y = Y, n = n, m = m, R = diag(1, m, m),
    zeros = rep(0, m))
fit <- jags.fit(data = dat, params = c("mu", "sigma_sq","rho"),
    model = model, n.iter = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 125
##    Unobserved stochastic nodes: 28
##    Total graph size: 579
##
## Initializing model
```

```
summary(fit)
```

```
##
## Iterations = 2001:3000
## Thinning interval = 1
## Number of chains = 3
## 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
## mu       1.6769 0.12608 0.002302       0.021077
## rho      0.7088 0.09034 0.001649       0.007754
## sigma_sq 0.6918 0.18146 0.003313       0.014935
##
## 2. Quantiles for each variable:
##
##            2.5%    25%    50%    75%  97.5%
## mu       1.3991 1.5999 1.6977 1.7614 1.8883
## rho      0.5014 0.6572 0.7185 0.7724 0.8596
## sigma_sq 0.4192 0.5621 0.6680 0.7901 1.1643
```

DC: cloning the whole dat set which means more independent clusters (i.e. not increased levels of replication within cluster)

```r
library(dclone)
model <- custommodel("model {
    for (k in 1:K) {
        for (i in 1:n) {
            for (j in 1:m) {
                Y[i,j,k] ~ dpois(lambda[i,j,k])
                lambda[i,j,k] <- exp(alpha[i,j,k] + mu)
            }
            alpha[i,1:m,k] ~ dmnorm(zeros, invSigma)
        }
    }
    Sigma <- sigma_sq * R + sigma_sq * rho * (1 - R)
    invSigma <- inverse(Sigma)
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
    rho ~ dunif(0, 0.999)
}")
dat <- list(Y = dcdim(array(Y, c(dim(Y), 1))), n = n, m = m, R = diag(1, m, m),
    zeros = rep(0, m), K=1)
dcfit <- dc.fit(data = dat, params = c("mu", "sigma_sq","rho"),
    model = model, n.iter = 1000,
    n.clones=c(1,2),
    unchanged = c("n", "m", "R", "zeros"), multiply="K")
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 125
##    Unobserved stochastic nodes: 28
##    Total graph size: 580
##
## Initializing model

## Warning in rjags::jags.model(model, data, n.chains = n.chains, n.adapt =
## n.adapt, : Adaptation incomplete

## NOTE: Stopping adaptation
##
##
##
## Fitting model with 2 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 250
##    Unobserved stochastic nodes: 53
##    Total graph size: 1105
##
## Initializing model

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

```
summary(dcfit)
```

```
##
## Iterations = 2001:3000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
## Number of clones = 2
##
## 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
## mu       1.6003 0.09680 0.1369 0.001767       0.029319 1.112
## rho      0.7156 0.05968 0.0844 0.001090       0.004204 1.040
## sigma_sq 0.6941 0.12079 0.1708 0.002205       0.007340 1.007
##
## 2. Quantiles for each variable:
##
##            2.5%    25%    50%    75%  97.5%
## mu       1.4186 1.5253 1.6044 1.6753 1.7750
## rho      0.5827 0.6776 0.7215 0.7599 0.8146
## sigma_sq 0.4960 0.6064 0.6786 0.7670 0.9681
```

```
dcdiag(dcfit)
```

```
##   n.clones lambda.max   ms.error   r.squared    r.hat
## 1        1 0.03333722 0.13408288 0.014524276 1.123135
## 2        2 0.01555943 0.04335546 0.007042174 1.104309
```

Binomial GLMM for clustered data

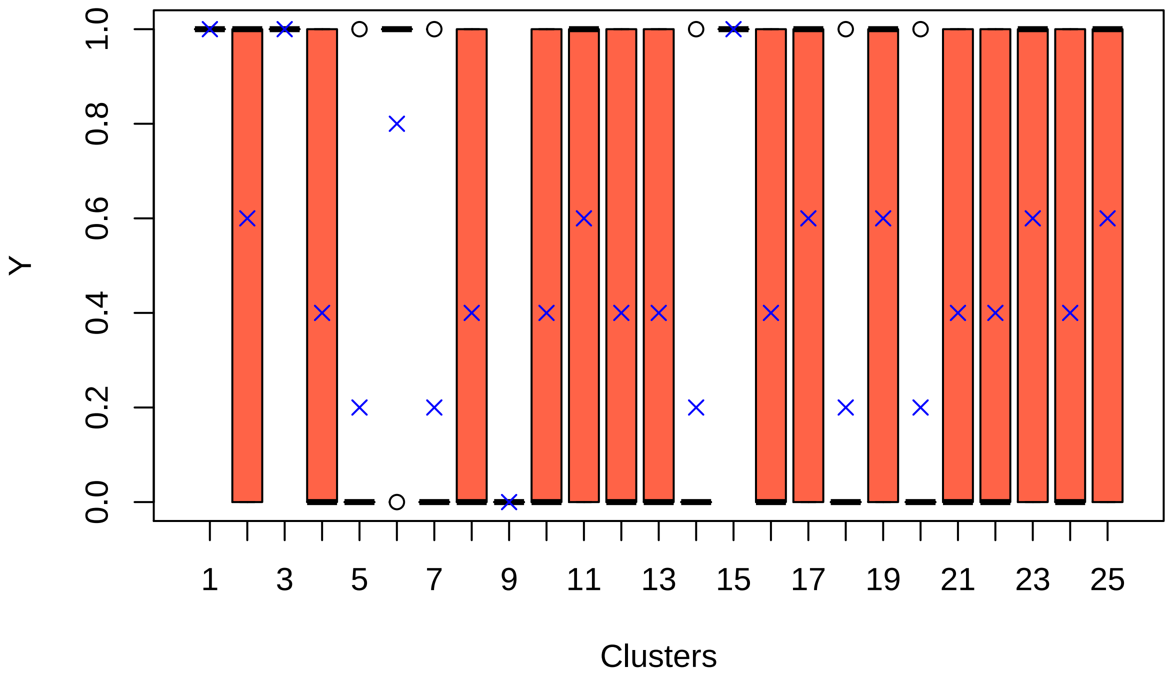
```
set.seed(1234)
library(MASS)
## total sample size is n x m
m <- 5 # number of points in a cluster
n <- 25 # number of clusters
mu <- 0 # global mean on logit scale
sigma_sq <- 1 # global variance
rho <- 0.8

## variance-covariance matrix for a single cluster
Sigma1 <- sigma_sq * diag(1, m, m) + sigma_sq * rho * (1 - diag(1, m, m))
```

```
## the full variance covariance matrix is 'block-diagonal'
## which means it is filled with zeros across clusters
alpha <- matrix(0, n, m)
for (i in 1:n) {
    alpha[i,] <- mvrnorm(1, rep(0, m), Sigma1)
}
Y <- rbinom(n*m, 1, plogis(as.numeric(alpha) + mu))
dim(Y) <- dim(alpha)
Y
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    1    1    1    1    1
##  [2,]    0    1    1    0    1
##  [3,]    1    1    1    1    1
##  [4,]    1    0    0    0    1
##  [5,]    0    0    0    0    1
##  [6,]    1    1    1    1    0
##  [7,]    0    0    0    1    0
##  [8,]    1    0    0    0    1
##  [9,]    0    0    0    0    0
## [10,]    1    0    0    1    0
## [11,]    0    1    0    1    1
## [12,]    0    1    1    0    0
## [13,]    1    1    0    0    0
## [14,]    0    0    0    1    0
## [15,]    1    1    1    1    1
## [16,]    0    1    1    0    0
## [17,]    0    0    1    1    1
## [18,]    0    1    0    0    0
## [19,]    0    1    1    1    0
## [20,]    0    0    0    0    1
## [21,]    0    1    0    0    1
## [22,]    1    1    0    0    0
## [23,]    1    0    1    0    1
## [24,]    1    0    0    1    0
## [25,]    1    0    1    0    1
```

```
boxplot(t(Y), xlab="Clusters", ylab="Y", col="tomato")
points(1:n, rowMeans(Y), pch=4, col=4)
```



Bayesian analysis:

```
library(dclone)
model <- custommodel("model {
    for (i in 1:n) {
        for (j in 1:m) {
            Y[i,j] ~ dbern(p[i,j])
            p[i,j] <- ilogit(alpha[i,j] + mu)
        }
        alpha[i,1:m] ~ dmnorm(zeros, invSigma)
    }
    Sigma <- sigma_sq * R + sigma_sq * rho * (1 - R)
    invSigma <- inverse(Sigma)
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
    rho ~ dunif(0, 0.999)
}")
dat <- list(Y = Y, n = n, m = m, R = diag(1, m, m),
    zeros = rep(0, m))
fit <- jags.fit(data = dat, params = c("mu", "sigma_sq","rho"),
    model = model, n.iter = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 125
##    Unobserved stochastic nodes: 28
##    Total graph size: 579
##
## Initializing model
## Warning in rjags::jags.model(model, data, n.chains = n.chains, n.adapt =
## n.adapt, : Adaptation incomplete

## NOTE: Stopping adaptation
```

```
summary(fit)
```

```
##
## Iterations = 2001:3000
## Thinning interval = 1
## Number of chains = 3
## 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
## mu       -0.07598 0.18469 0.003372        0.00460
## rho       0.58673 0.29002 0.005295        0.06329
## sigma_sq  0.08147 0.09218 0.001683        0.01309
##
## 2. Quantiles for each variable:
##
##               2.5%      25%      50%     75%  97.5%
## mu       -0.437938 -0.20312 -0.07623 0.05234 0.2821
## rho       0.034280  0.33899  0.65636 0.84805 0.9546
## sigma_sq  0.008969  0.01702  0.04132 0.10713 0.3467
```

DC:

```
library(dclone)
model <- custommodel("model {
    for (k in 1:K) {
        for (i in 1:n) {
            for (j in 1:m) {
                Y[i,j,k] ~ dbern(p[i,j,k])
                p[i,j,k] <- ilogit(alpha[i,j,k] + mu)
            }
            alpha[i,1:m,k] ~ dmnorm(zeros, invSigma)
        }
    }
    Sigma <- sigma_sq * R + sigma_sq * rho * (1 - R)
    invSigma <- inverse(Sigma)
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
    rho ~ dunif(0, 0.999)
}")
dat <- list(Y = dcdim(array(Y, c(dim(Y), 1))), n = n, m = m, R = diag(1, m, m),
    zeros = rep(0, m), K=1)
dcfit <- dc.fit(data = dat, params = c("mu", "sigma_sq","rho"),
    model = model, n.iter = 1000,
    n.clones=c(1,2),
    unchanged = c("n", "m", "R", "zeros"), multiply="K")
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 125
##    Unobserved stochastic nodes: 28
##    Total graph size: 580
##
## Initializing model

## Warning in rjags::jags.model(model, data, n.chains = n.chains, n.adapt =
## n.adapt, : Adaptation incomplete

## NOTE: Stopping adaptation
##
##
##
## Fitting model with 2 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 250
##    Unobserved stochastic nodes: 53
##    Total graph size: 1105
##
## Initializing model

## Warning in rjags::jags.model(model, data, n.chains = n.chains, n.adapt =
## n.adapt, : Adaptation incomplete

## NOTE: Stopping adaptation

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

```r
summary(dcfit)
```

```
##
## Iterations = 2001:3000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
## Number of clones = 2
##
## 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
## mu       -0.08141 0.12654 0.17895 0.0023103       0.002947 1.000
## rho       0.45966 0.27155 0.38403 0.0049577       0.054373 1.757
## sigma_sq  0.01654 0.02035 0.02878 0.0003715       0.002447 2.856
##
## 2. Quantiles for each variable:
##
##               2.5%       25%       50%      75%   97.5%
## mu       -0.328565 -0.165522 -0.080676 0.004349 0.16660
## rho       0.019960  0.202610  0.520727 0.700883 0.88415
## sigma_sq  0.001609  0.002983  0.006797 0.027521 0.06727
```

```
dcdiag(dcfit)
```

```
##   n.clones lambda.max  ms.error  r.squared    r.hat
## 1        1 0.05714114 0.4286154 0.02487922 2.644137
## 2        2 0.07390028 1.7165919 0.15531111 2.349080
```