```r
#net1 <- network(Adj, directed=F)

### library(latentnet)
#net2 <- ergmm(net1~euclidean(d=2), tofit=c("mle"))

#coords <- net2$mle$Z

### library(MCMCpack)
#coords = procrustes(coords, Z, translation = T, dilation = F)$X.new

#plot(coords, xlab = "Z_1", ylab = "Z_2", type = "p")
#points(Z, col = "red")   # true position
```

(case 2. dilation = T)

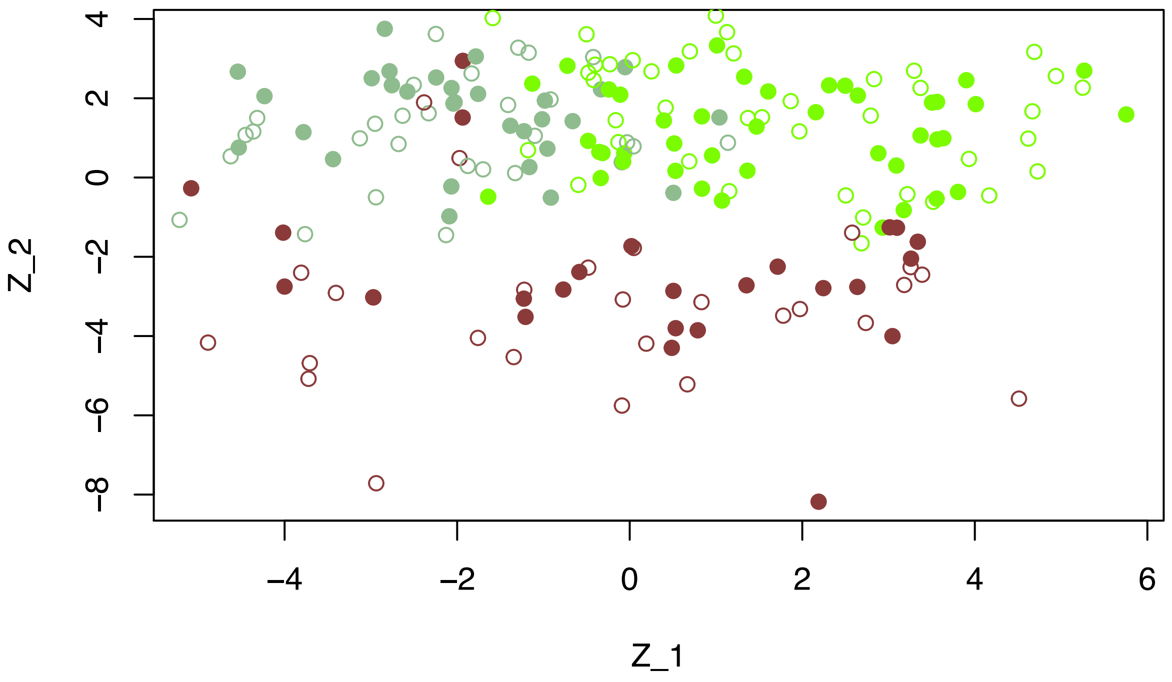
```r
net1 <- network(Adj, directed=F)

# library(latentnet)
net2 <- ergmm(net1~euclidean(d=2), tofit=c("mle"))

coords <- net2$mle$Z

# library(MCMCpack)
# coords = procrustes(coords, Z, translation=T, dilation=F)$X.new
coords = procrustes(coords, Z, translation=T, dilation=T)$X.new
```

```r
plot(coords, xlab = "Z_1", ylab = "Z_2", col = cl, pch = 19)
points(Z, col = cl)   # true position
```



```r
msq_dev1 = sum((coords - Z)^2) / N
msq_dev1
```

sum of squared deviations from the truth:

```
## [1] 4.784991
```

Implement JAGS:

Step 1: Define the model using the BUGS language:

```
### JAGS, in BUGS language

func1 <- function(){

  for (i in 1:N){
        for (j in 1:N){
            dist[i,j] <- pow(pow((z[i,1] - z[j,1]), 2) + pow((z[i,2] - z[j,2]), 2), 0.5)
            M[i,j]    <- sigmasq * exp(- phi * dist[i,j])
            probit(padj[i,j]) <- a + b * abs(x[i]-x[j]) - exp(theta) * dist[i,j]
            adj[i,j]    ~ dbern(padj[i,j])
        }
  }

  for (i in 1:N){
        mu_x[i]    <- beta
        z[i, 1:2]    ~ dmnorm(mu_z[categ, 1:K], inv_kappasq[categ] * I[1:K, 1:K])
    }

    x[1:N] ~ dmnorm(mu_x[1:N], error_prec[1:N, 1:N])   ### x is the attributes


    ### Spatial Priors
    for (h in 1:H){
      mu_z[h, 1:2]        ~ dmnorm(mu_prior1[1:K], (1/mu_prior2) * I[1:K, 1:K])
      inv_kappasq[h]      ~ dgamma(kappa_prior1, kappa_prior2)
    }
  omega[1:H]              ~ ddirch(alpha[1:H])
  categ                   ~ dcat(omega[1:H])    ### categorical distribution

    beta                  ~ dnorm(0.0, 0.0001)
    inv_sigmasq           ~ dgamma(sigmasq_prior1, sigmasq_prior2)
    sigmasq              <- 1 / inv_sigmasq
    inv_tausq             ~ dgamma(tausq_prior1, tausq_prior2)
    tausq                <- 1 / inv_tausq
    phi                   ~ dunif(phi_lower, phi_upper)

    var_all[1:N, 1:N]    <- M[1:N, 1:N] + tausq * I[1:N, 1:N]
    error_prec[1:N, 1:N] <- inverse(var_all[1:N, 1:N])

    a                     ~ dnorm(amean_prior, 1/avar_prior)
    b                     ~ dnorm(bmean_prior, 1/bvar_prior)
    theta                   ~ dnorm(tmean_prior, 1/tvar_prior)
}
```

```
### specify independent priors parameters
data1 <- list(adj              = Adj,           ### adjacent matrix N by N
              x                = as.vector(x),  ### the observed N attributes s(z_i)
              mu_prior1        = c(0, 0),
              mu_prior2        = 4,
              kappa_prior1     = 2,
              kappa_prior2     = 2,
              alpha            = c(2, 2, 2),
              sigmasq_prior1 = 2,
              sigmasq_prior2 = 0.1,
              tausq_prior1     = 2,
              tausq_prior2     = 0.02,
              phi_lower        = 0.01,
              phi_upper        = 1,
              amean_prior      = 5,
              avar_prior       = 1,
              bmean_prior      = 13,
              bvar_prior       = 2,
              tmean_prior      = 1.5,
              tvar_prior       = 2,
              N = N,
              K = K,     ### latent space dim
              H = H,     ### assume N > H.
              I = diag(rep(1, N)) )

if (length(data1$alpha) != H){
  print("dimension error: alpha dimention does not comform")
}
```

Determine the parameters in the priors:

Step 2: Read in the model file using the "jags.model()" function.

This creates an object of class "jags".

```
# inits1      <- list(z = as.matrix(coords))
inits1      <- list(z = as.matrix(Z))

### write the model "func1" to the file created from the following file path:
testfile    <- file.path("/Users/linden/Desktop", "jags_test_2.txt")
write.model(func1, testfile)  # func1 is our model

# library(rjags)
jags1 <- jags.model(testfile, data1, inits = inits1)
```

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

```
##    Observed stochastic nodes: 10001
##    Unobserved stochastic nodes: 115
##    Total graph size: 170267
##
## Initializing model
```

Step 3: Update the model using the update method for "jags" objects.

This constitutes a burn-in period.

```
update(jags1, n.iter = 5000)
```

Step 4: Extract samples from the model object using the "coda.samples()" function.

```
parameters1 <- c("z", "omega", "mu_z", "inv_kappasq",
                 "tausq", "sigmasq", "phi", "beta",
                 "a", "b", "theta")

jags2       <- jags.samples(jags1, parameters1, n.iter=10000)
# jags2$...
```

PART III:

Diagnostics

```
z_res           <- jags2$z
omega_res       <- jags2$omega
mu_z_res        <- jags2$mu_z
inv_kappasq_res <- jags2$inv_kappasq
tausq_res       <- jags2$tausq
sigmasq_res     <- jags2$sigmasq
phi_res         <- jags2$phi
beta_res        <- jags2$beta
a_res           <- jags2$a
b_res           <- jags2$b
theta_res       <- jags2$theta
```

```
# dim(z_new)
# dim(omega_res)
# dim(mu_z_res)
# dim(tausq_res)
```

```
kappasq_res <- 1 / inv_kappasq_res



true_para = c(phi, sigmasq, tausq, beta,
              a, b, theta,
              omega[1], omega[2],
              kappasq[])
```

```
mcmc_para = c(mean(phi_res), mean(sigmasq_res), mean(tausq_res), mean(beta_res),
              mean(a_res), mean(b_res), mean(theta_res),
              mean(omega_res[1,,1]), mean(omega_res[2,,1]),
              mean(kappasq_res[1,,1]), mean(kappasq_res[2,,1]), mean(kappasq_res[3,,1]))


compare_para = data.frame(true_para, mcmc_para)
rownames(compare_para) <- c("phi", "sigmasq", "tausq", "beta",
                            "a", "b", "theta",
                            "omega[1]", "omega[2]",
                            "kappasq[1]", "kappasq[2]", "kappasq[3]")
compare_para
```

```
##             true_para   mcmc_para
## phi              0.50 0.635259024
## sigmasq          0.10 0.125845514
## tausq            0.02 0.009188966
## beta             1.00 1.028035571
## a                5.00 1.207795152
## b               13.00 8.347637485
## theta            1.50 0.802770004
## omega[1]         0.30 0.347527929
## omega[2]         0.40 0.287694991
## kappasq[1]       3.00 4.119292285
## kappasq[2]       4.00 1.975491796
## kappasq[3]       5.00 4.870626849
```

compare mu:

```
mu
```

```
##      [,1] [,2]
## [1,]   -2    2
## [2,]    2    2
## [3,]    0   -3
```

```
mu_z_res
```

```
## mcarray:
##            [,1]        [,2]
## [1,]  0.5567184 0.012135555
## [2,] -0.0199149 0.005530977
## [3,]  0.3726574 0.225281425
##
## Marginalizing over: iteration(10000),chain(1)
```

compare Z:

```
z_new <- array(NA, dim = dim(z_res))

for(i in 1:dim(z_res)[3]){
    z_new[, , i, 1] <- procrustes(z_res[,,i,1], Z, translation = T,dilation = T)$X.new
}
```

```r
dim(z_new)
```

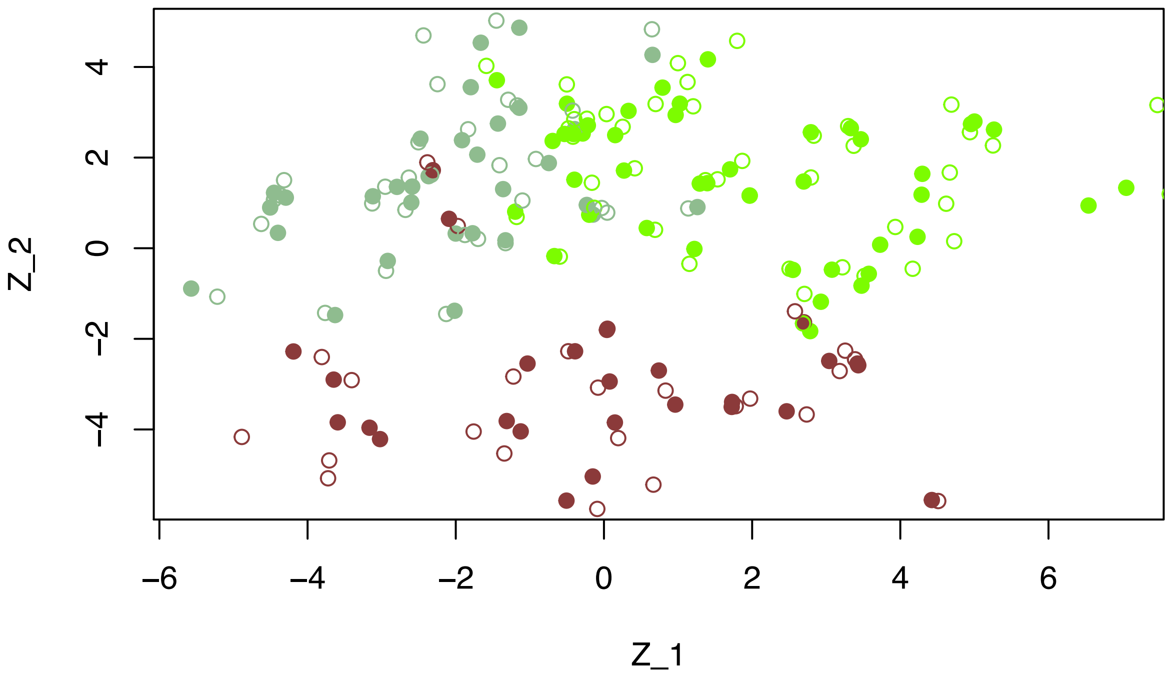
Procrustes transformations on locations Z:

```
##                     iteration     chain
##       100         2     10000         1
```

```r
z_post_mean = apply(z_new[, , , 1], 1:2, mean)
z_post_mean
```

```
##              [,1]        [,2]
##   [1,]  1.96755262  1.16053786
##   [2,] -0.66920957 -0.17380971
##   [3,]  0.79104920  3.54408877
##   [4,]  0.96240748 -3.45116231
##   [5,] -0.15101875 -5.03770368
##   [6,] -3.11935599  1.14650434
##   [7,]  3.07685335 -0.47421366
##   [8,] -2.58795722  1.36059286
##   [9,] -2.09114949  0.64883743
##  [10,]  1.29338431  1.42641583
##  [11,]  3.46666331  2.40372788
##  [12,] -2.79387484  1.35409349
##  [13,] -0.28666017  2.53187221
##  [14,] -3.64765577 -2.89982811
##  [15,] -1.70902041  2.06482950
##  [16,]  4.23219389  0.25159089
##  [17,] -3.62858893 -1.47488869
##  [18,]  6.54042520  0.94182935
##  [19,] -1.91578192  2.38386619
##  [20,] -1.43086918  2.75110072
##  [21,] -0.39413128  2.62859808
##  [22,] -0.14773322  0.74182166
##  [23,]  1.02432918  3.19094134
##  [24,]  0.57764144  0.44573109
##  [25,]  3.72768599  0.07738893
##  [26,] -4.29343865  1.11631425
##  [27,]  1.72446216 -3.50269136
##  [28,] -0.21718160  2.71031145
##  [29,] -1.66210211  4.53507784
##  [30,]  1.73003510 -3.39302616
##  [31,]  1.22114705 -0.01653950
##  [32,]  4.29706338  1.64148450
##  [33,] -2.47686287  2.41749243
##  [34,] -2.01599852 -1.38061029
##  [35,]  1.26212648  0.90776816
##  [36,]  3.04049356 -2.48757164
##  [37,]  4.28701818  1.18101286
##  [38,] -2.91756738 -0.28009995
##  [39,] -4.19059228 -2.27890674
##  [40,]  3.47616809 -0.82657554
##  [41,] -3.02182371 -4.21277527
##  [42,] -2.36649683  1.59006289
##  [43,] -4.39958297  0.33925877
##  [44,] -1.44527720  3.70768320
##  [45,]  0.96791423  2.94141590
##  [46,] -1.31250825 -3.81426310
##  [47,] -0.69225782  2.36661872
##  [48,]  2.46600389 -3.59975774
##  [49,]  4.42508074 -5.55735478
##  [50,] -0.38975192 -2.27490219
##  [51,] -4.50465357  0.89817945
##  [52,]  3.42215783 -2.54479796
##  [53,]  1.40297144  4.16825048
##  [54,] -3.59276472 -3.84240851
##  [55,]  3.57464847 -0.56488120
##  [56,] -2.00060067  0.32370660
##  [57,]  1.39491758  1.43509089
##  [58,] -0.49906785  3.18965560
##  [59,] -0.39858660  1.51174137
##  [60,] -0.41086495  2.51974014
##  [61,]  5.26457230  2.61761517
##  [62,]  2.79253915  2.55934985
##  [63,] -1.14143852  3.09967485
##  [64,]  0.03992572 -1.80309706
##  [65,]  0.33202026  3.02981404
##  [66,]  5.00033253  2.79599266
##  [67,] -0.53253916  2.52136127
##  [68,] -0.19836617  0.73925825
##  [69,]  7.04981160  1.33354187
##  [70,]  0.14775805 -3.84662342
##  [71,]  2.55064807 -0.47856544
##  [72,]  3.43263368 -2.58313021
##  [73,]  0.65536297  4.26712993
##  [74,] -5.57098151 -0.89088271
##  [75,] -2.30959731  1.72189301
##  [76,]  0.27041317  1.71374308
##  [77,] -1.14209806  4.86602689
##  [78,]  1.70216082  1.74077294
##  [79,] -1.77271104  0.33009723
##  [80,] -1.03138653 -2.54570527
##  [81,] -4.45250046  1.22297006
##  [82,] -1.79757568  3.55362473
##  [83,] -1.32895936  0.17447894
##  [84,]  2.78370387 -1.83000161
##  [85,]  4.95384277  2.74057737
##  [86,]  0.74118147 -2.69960676
##  [87,]  0.07694291 -2.94185121
##  [88,]  2.70005121 -1.63455583
##  [89,] -3.16366033 -3.96174108
##  [90,]  3.33262344  2.64955624
##  [91,]  2.92796602 -1.18154135
##  [92,] -1.36075485  1.30416847
##  [93,] -0.23145893  0.95503036
##  [94,] -1.19771491  0.80663606
##  [95,] -1.12337553 -4.04291223
##  [96,] -0.50577405 -5.57026824
##  [97,]  2.69591149  1.47211401
##  [98,]  0.15211075  2.49627598
##  [99,] -2.59965803  1.01018427
## [100,] -0.74378214  1.87956379
```

```
plot(z_post_mean, xlab = "Z_1", ylab = "Z_2", col = cl, pch = 19)
points(Z, col = cl)    # true position
```



```
msq_dev2 = sum((z_post_mean - Z)^2) / N
msq_dev2
```

sum of squared deviations from the truth "Z":

```
## [1] 0.519649
```

```
### compare with latent position model without spatial effect:
msq_dev1
```

```
## [1] 4.784991
```

Markov chain trace plot:

```
# dim(phi_res)
phi_mcmc <- phi_res[1, , 1]
plot(phi_mcmc, type = "l")
```