MiCoRe Vignette

Kevin McGregor

6/4/2020

Microbiome **Co**variance **Re**gression (**MiCoRe**) allows the estimation of how OTU co-occurrence networks vary with respect to a covariate profile using principles of covariance regression. This work was developed in the Greenwood Lab at McGill University.

Installation

MiCoRe can be installed easily from Github. Note that the name of the R package is all lowercase: `micore`.

```
if (!require(devtools)) {
  install.packages("devtools")
  library(devtools)
}
install_github("kevinmcgregor/micore", dependencies=TRUE)
```

The model

The goal of **MiCoRe** is to estimate how covariance matrices vary with repsect to a covariate profile in the context of microbiome data.

Assume that the matrix $\mathbf{Y}_{n\times(p+1)}$ contains the counts of $p+1$ taxa over n samples. Taxon $p+1$ will be used as a reference taxon, and will not be included in the estimated covariance matrices. The matrix $\mathbf{X}_{n\times q}$ contains the $q-1$ covariates over which the covariance (or precision) matrix is assumed to vary, along with an intercept column. The vector $\mathbf{x}_i = (1, x_{i1}, \dots, x_{i(q-1)})^\top$ contains the covariates for individual i.

We assume a multinomial logistic regression framework for the taxon counts. We denote the total count for individual i as $M_i = \sum_{j=1}^{p+1} \mathbf{Y}_{ij}$. We also assume that the true proportions of all the taxa in individual i's microbiome is $\boldsymbol{\pi}_i = (\pi_{i1}, \dots, \pi_{i(p+1)})$, with $0 < \pi_{ij} < 1$ for all $j \in \{1, \dots, p+1\}$ and $\sum_{j=1}^{p+1} \pi_{ij} = 1$. Then we assume the observed counts for individual i, denoted by $\mathbf{Y}_{i\cdot}$, follow a multinomial distribution. The full model is written as.

$$
\begin{aligned}
\mathbf{Y}_{i\cdot} | \boldsymbol{\eta}_{i\cdot}, \mathbf{A}, \mathbf{B}, \gamma_i, \boldsymbol{\Psi}, \boldsymbol{\Gamma} &\sim \text{Multinomial}(M_i, \boldsymbol{\pi}_i) \\
\boldsymbol{\eta}_{i\cdot} | \mathbf{A}, \mathbf{B}, \gamma_i, \boldsymbol{\Psi}, \boldsymbol{\Gamma} &\sim \text{Normal}\left([\mathbf{A} + \gamma_i \mathbf{B}]\mathbf{x}_i, \boldsymbol{\Psi}\right) \\
\mathbf{C} = (\mathbf{A}, \mathbf{B}) | \boldsymbol{\Psi}, \boldsymbol{\Gamma} &\sim \text{Matrix-Normal}\left(\mathbf{C}_0, \boldsymbol{\Psi}, \boldsymbol{\Gamma}\right) \\
\boldsymbol{\Psi} &\sim \text{inv-Wishart}\left(\nu_\Psi, \boldsymbol{\Psi}_0\right) \\
\boldsymbol{\Gamma} &\sim \text{inv-Wishart}\left(\nu_\Gamma, \boldsymbol{\Gamma}_0\right) \\
\gamma_i &\sim \text{Normal}(0, 1). \qquad (1)
\end{aligned}
$$

where the proportions $\boldsymbol{\pi}_i$ are parameterized using a matrix of latent parameters, $\boldsymbol{\eta}_{n\times p}$, whose elements are denoted by η_{ij}:

$$
\boldsymbol{\pi}_i = \left(\frac{\exp(\eta_{i1})}{1 + \sum_{j=1}^{p} \exp(\eta_{ij})}, \dots, \frac{\exp(\eta_{ip})}{1 + \sum_{j=1}^{p} \exp(\eta_{ij})}, \frac{1}{1 + \sum_{j=1}^{p} \exp(\eta_{ij})} \right).
$$

The elements of $\boldsymbol{\eta}$ can be thought of as the additive log-ratio transformed proportions with respect to the reference taxon $p+1$:

$$\boldsymbol{\eta}_{i\cdot} = \left[\log\left(\frac{\pi_{i1}}{\pi_{i(p+1)}}\right), \dots, \log\left(\frac{\pi_{ip}}{\pi_{i(p+1)}}\right)\right],$$

where $\boldsymbol{\eta}_{i\cdot}$ represents row i of $\boldsymbol{\eta}$.

Interpretations of parameters

Parameter interpretations come from marginalizing out the individual-specific term γ_i. The expected value for $\boldsymbol{\eta}_{i\cdot}$ (i.e. the additive log-ratio transformed proportions for individual i) is written as:

$$\mathbb{E}(\boldsymbol{\eta}_{i\cdot}|\mathbf{A}, \mathbf{B}, \boldsymbol{\Psi}, \boldsymbol{\Gamma}) = \mathbf{A}\mathbf{x}_i.$$

Hence, $\mathbf{A}$ characterizes how the covariates in $\mathbf{x}_i$ affect the expected value of the additive log-ratio transformed proportions for individual i, and ultimately the relative abundances of the taxa for individual i. Likewise, the covariance matrix for $\boldsymbol{\eta}_{i\cdot}$ is calculated as:

$$\begin{aligned}\text{var}(\boldsymbol{\eta}_{i\cdot}|\mathbf{A}, \mathbf{B}, \boldsymbol{\Psi}, \boldsymbol{\Gamma}) &= \boldsymbol{\Psi} + \mathbf{B}\mathbf{x}_i\mathbf{x}_i^\top\mathbf{B}^\top \\ &= \boldsymbol{\Sigma}_{\mathbf{x}_i}.\end{aligned} \tag{2}$$

The matrix $\boldsymbol{\Sigma}_{\mathbf{x}_i}$, or perhaps its corresponding correlation matrix, can then be used to define a taxon co-occurrence network for individual i based on the covariates. In this expression, $\boldsymbol{\Psi}$ can be thought of as a baseline covariance matrix and $\mathbf{B}$ describes how the covariates in $\mathbf{x}_i$ affect $\boldsymbol{\Sigma}_{\mathbf{x}_i}$.

Running MiCoRe

After installing the `micore` package, running the method is simple. Let's load in some data and run the function. Note that you need to supply the model matrix $\mathbf{X}$, and you specifically need to give it an intercept column. This can be done easily using the `model.matrix()` function. Also note that, in this example, we run only 500 burn-in and 500 MCMC samples, but in practice, you should likely run for longer. For example, the default is 4000 burn-in and 4000 MCMC samples.

We'll run `micore` and include BMI in the model matrix.

```r
# Loading in sample American Gut dataset included in package
library(micore)
data(amgut)

counts <- amgut$counts
X <- model.matrix(~BMI, amgut$clin.dat)

# Number of burn-in samples and number of MCMC samples to save
n.burn <- 500
n.samp <- 500

# Running micore
mc.fit <- micore(counts, X, n.burn = n.burn, n.samp = n.samp,
                 n.chain=4, n.cores=4, verbose=TRUE)
```

Note that the `micore` object contains one list element for each MCMC chain run. In this example, we ran 4 chains, so each chain's data can be accessed like so:

```r
# Chain 1
tmp <- mc.fit[[1]]
attributes(tmp)
```

```
## $names
##  [1] "eta"          "Psi"          "A"            "B"
##  [5] "gamma"        "eta.accepted" "sigma.zero"   "Gamma"
##  [9] "acc.probs"    "counts"       "X"
```

```r
# Chain 2
tmp <- mc.fit[[2]]
attributes(tmp)
```

```
## $names
##  [1] "eta"          "Psi"          "A"            "B"
##  [5] "gamma"        "eta.accepted" "sigma.zero"   "Gamma"
##  [9] "acc.probs"    "counts"       "X"
```

```r
# etc...
```

MCMC samples from any of the chains can be extracted from any of the parameters directly from this object. Each parameter is an array where the first dimension represents the . For example, we can extract the **B** parameter from chain 3:

```r
# Extracting the B parameter from chain 3
B.3 <- mc.fit[[3]]$B
dim(B.3)
```

```
## [1] 500  10   2
```

```r
# Get 101th sample of B in chain 3
B.3[101,,]
```

```
##             [,1]       [,2]
##  [1,] -3.180962 0.13938495
##  [2,] -4.047394 0.18379682
##  [3,] -2.175345 0.10905389
##  [4,] -1.152253 0.08639255
##  [5,] -1.693856 0.09442852
##  [6,] -2.528124 0.12236800
##  [7,] -5.662513 0.18210320
##  [8,] -2.753126 0.13536691
##  [9,] -2.362753 0.12092688
## [10,] -1.453043 0.06755437
```

The names of the parameters available to extract are:

- `eta`: $\boldsymbol{\eta}$, the additive log-ratio transformed proportions
- `Psi`: $\boldsymbol{\Psi}$, the baseline covariance matrix
- `A`: **A**, the "fixed effect" parameter
- `B`: **B**, the "random effect" parameter
- `gamma`: γ_i, $i \in 1, \ldots, n$, the individual-specific parameter (not to be confused with `Gamma` with a capital G)
- `Gamma`: $\boldsymbol{\Gamma}$ the column covariance matrix in the Matrix-Normal prior (not to be confused with `Gamma` with a lowercase g)

The MCMC samples from all chains can be merged together for a particular parameter in order to run summary statistics on all MCMC samples from the parameter:

```
# Merging all 4 chains into single array
B.merge <- mergeChains(mc.fit, par="B")
# Mean of B over all chains
apply(B.merge, 2:3, mean)
```

```
##            [,1]       [,2]
##  [1,] -2.398900 0.10476227
##  [2,] -3.405903 0.15878830
##  [3,] -1.652361 0.08226252
##  [4,] -0.965950 0.09174521
##  [5,] -1.725886 0.10078990
##  [6,] -2.436736 0.11853488
##  [7,] -4.755347 0.16659139
##  [8,] -2.762101 0.12682739
##  [9,] -2.331489 0.10848107
## [10,] -1.471495 0.06390601
```

Getting estimated OTU abundances

Though the `micore` object contains all samples from all parameters from all chains, these data structures can be difficult to work with directly. This package contains a functions to get estimates (and credible intervals) of the OTU abundances for a given covariate profile $\mathbf{x}_i$. These estimates can be done on either the additive log-ratio scale or on the proportions scale:

```
# Want to estimate OTU abundances for individual with x=1.3.
# Create the covariate profile with intercept
x.try <- matrix(c(1, 1.3), nrow=1)
# Get predicted abundances on additive log-ratio scale
p1 <- predict(mc.fit, newdata=x.try)
p1$fit
```

```
##          [,1]     [,2]      [,3]      [,4]       [,5]       [,6]      [,7]
## [1,] 2.765553 1.105678 0.2970804 -4.232377 -0.2780637 -0.4273914 -3.627995
##           [,8]     [,9]  [,10]
## [1,] -2.117715 -2.10062 -2.305
```

```
# Credible intervals
p1$quant
```

```
## $`2.5%`
##          [,1]       [,2]       [,3]      [,4]     [,5]      [,6]     [,7]
## [1,] 1.247457 -0.1675982 -0.7832183 -6.197087 -1.80567 -2.133199 -6.05111
##           [,8]      [,9]     [,10]
## [1,] -3.678743 -3.442752 -3.529878
##
## $`97.5%`
##         [,1]     [,2]     [,3]      [,4]     [,5]     [,6]      [,7]
## [1,] 4.23108 2.400815 1.384655 -2.195907 1.254917 1.348511 -1.297274
##            [,8]       [,9]    [,10]
## [1,] -0.6253585 -0.8357593 -1.15681
```

```
# Get predicted abundances on proportions scale
p2 <- predict(mc.fit, newdata=x.try, type="prop")
```

```r
p2$fit
```

```
##          [,1]      [,2]       [,3]        [,4]       [,5]       [,6]
## [1,] 0.662372 0.1349495 0.06335426 0.001081788 0.03643206 0.03205476
##             [,7]        [,8]        [,9]       [,10]      [,11]
## [1,] 0.002108189 0.006276553 0.006259068 0.005191293 0.04992049
```

```r
# Credible intervals
p2$quant
```

```
## $`2.5%`
##          [,1]       [,2]       [,3]         [,4]       [,5]        [,6]
## [1,] 0.424306 0.04890673 0.01802754 6.385309e-05 0.00971337 0.008185284
##              [,7]        [,8]        [,9]       [,10]      [,11]
## [1,] 0.0001162878 0.001213597 0.001343619 0.001066542 0.01160171
##
## $`97.5%`
##           [,1]      [,2]      [,3]        [,4]       [,5]       [,6]
## [1,] 0.8470181 0.2624134 0.1478075 0.005027361 0.08967047 0.08769129
##            [,7]       [,8]       [,9]      [,10]     [,11]
## [1,] 0.01119443 0.01876782 0.01830552 0.01505836 0.1385813
```

Getting estimated covariance matrix (or precision, correlation, partial correaltion matrix)

It's also possible to directly extract the estimated covariance matrix based on a covariate profile $\mathbf{x}_i$. The function `getPredCov()` allows the user to provide a covariate profile and will return the corresponding estimated covariance, precision, correlation, or partial correlation matrix along with interval estimates. Note that the returned object saves the estimate value in and array in the `fit` slot, and the first dimension corresponds to the individuals that covariances matrices are being calculated for.

```r
# Get estimated covariance matrix using covariate profile defined earlier...
cov1 <- getPredCov(mc.fit, newdata=x.try)
# Extract the covariance matrix for the first individual in x.try
cov1$fit[1,,]
```

```
##            [,1]      [,2]     [,3]     [,4]     [,5]      [,6]      [,7]
##  [1,]  8.578111  8.775127 4.389598 2.192009 5.507852  7.616095 11.664763
##  [2,]  8.775127 11.829767 6.135043 3.331624 5.917566  8.495672 15.912870
##  [3,]  4.389598  6.135043 3.842699 1.472380 2.798592  4.175088  7.635183
##  [4,]  2.192009  3.331624 1.472380 5.993670 1.729669  1.963854  6.966109
##  [5,]  5.507852  5.917566 2.798592 1.729669 5.878297  5.728154  9.197725
##  [6,]  7.616095  8.495672 4.175088 1.963854 5.728154  9.507258 12.388722
##  [7,] 11.664763 15.912870 7.635183 6.966109 9.197725 12.388722 28.984417
##  [8,]  7.104289  9.405264 5.003858 3.012447 5.070609  6.891021 13.375143
##  [9,]  6.120987  8.143784 4.232538 2.480711 3.872579  5.439692 10.941415
## [10,]  3.662268  5.289784 3.208313 1.261582 2.437234  3.510379  6.826279
##            [,8]      [,9]    [,10]
##  [1,]  7.104289  6.120987 3.662268
##  [2,]  9.405264  8.143784 5.289784
##  [3,]  5.003858  4.232538 3.208313
##  [4,]  3.012447  2.480711 1.261582
##  [5,]  5.070609  3.872579 2.437234
##  [6,]  6.891021  5.439692 3.510379
```

```
##  [7,] 13.375143 10.941415 6.826279
##  [8,]  9.773817  6.549634 4.127612
##  [9,]  6.549634  7.456124 3.861535
## [10,]  4.127612  3.861535 4.003116
```

```
# Credible intervals
cov1$quant$`2.5%`[1,,]
```

```
##             [,1]        [,2]        [,3]      [,4]        [,5]        [,6]
##  [1,]  1.6872979  0.57586334  0.40401610 -3.049793  0.64002065  1.35333611
##  [2,]  0.5758633  4.24740801  1.64046757 -3.606901  0.08522271  1.55263253
##  [3,]  0.4040161  1.64046757  1.24477562 -2.088181 -0.02409523  0.60135646
##  [4,] -3.0497926 -3.60690121 -2.08818084  2.373370 -2.14734016 -3.52401691
##  [5,]  0.6400206  0.08522271 -0.02409523 -2.147340  1.78980386  1.10970710
##  [6,]  1.3533361  1.55263253  0.60135646 -3.524017  1.10970710  3.09314573
##  [7,]  1.0331319  4.20657176  1.57631769 -2.293361  1.05494365  2.30712143
##  [8,]  0.7170254  2.66966254  1.18208444 -2.679309  0.24102588  1.16602075
##  [9,] -0.1300964  0.33040418  0.43561697 -2.529672 -0.63934962 -0.37439328
## [10,] -0.0270440  0.43191481  0.71753131 -2.013831 -0.32850697 -0.08083865
##             [,7]       [,8]       [,9]       [,10]
##  [1,]  1.0331319  0.7170254 -0.1300964 -0.02704400
##  [2,]  4.2065718  2.6696625  0.3304042  0.43191481
##  [3,]  1.5763177  1.1820844  0.4356170  0.71753131
##  [4,] -2.2933610 -2.6793089 -2.5296715 -2.01383065
##  [5,]  1.0549437  0.2410259 -0.6393496 -0.32850697
##  [6,]  2.3071214  1.1660207 -0.3743933 -0.08083865
##  [7,]  9.2064288  3.0815243  0.3655320  0.33920952
##  [8,]  3.0815243  2.9050857  0.3637182  0.23853377
##  [9,]  0.3655320  0.3637182  1.4322689  0.32694107
## [10,]  0.3392095  0.2385338  0.3269411  1.47604909
```

```
cov1$quant$`97.5%`[1,,]
```

```
##            [,1]     [,2]      [,3]      [,4]      [,5]      [,6]     [,7]
##  [1,] 19.997011 19.75444 11.082215 10.700120 15.944140 19.861520 27.71016
##  [2,] 19.754438 23.84549 13.984568 14.164409 15.840458 20.209338 33.34042
##  [3,] 11.082215 13.98457  9.177455  6.960473  8.545373 10.821335 17.71360
##  [4,] 10.700120 14.16441  6.960473 15.437876  9.540200 10.506225 24.62475
##  [5,] 15.944140 15.84046  8.545373  9.540200 17.767647 17.105254 26.49623
##  [6,] 19.861520 20.20934 10.821335 10.506225 17.105254 23.761437 29.86216
##  [7,] 27.710159 33.34042 17.713595 24.624755 26.496230 29.862157 65.37488
##  [8,] 17.855512 20.16462 11.851311 12.188797 15.370417 18.312721 31.56215
##  [9,] 15.727003 18.60602 11.005880 11.003841 12.713176 15.147882 26.52831
## [10,]  9.709656 12.61915  8.303179  6.392766  7.524335  9.964108 16.44568
##           [,8]     [,9]     [,10]
##  [1,] 17.85551 15.72700  9.709656
##  [2,] 20.16462 18.60602 12.619146
##  [3,] 11.85131 11.00588  8.303179
##  [4,] 12.18880 11.00384  6.392766
##  [5,] 15.37042 12.71318  7.524335
##  [6,] 18.31272 15.14788  9.964108
##  [7,] 31.56215 26.52831 16.445681
##  [8,] 22.90306 16.64913 10.657662
##  [9,] 16.64913 17.83423 10.246928
## [10,] 10.65766 10.24693  9.957400
```

```r
# Partial correlation instead
pc1 <- getPredCov(mc.fit, newdata=x.try, type="pcor")
# Extract the partial correlation matrix for the first individual in x.try
pc1$fit[1,,]
```

```
##             [,1]        [,2]          [,3]        [,4]        [,5]
##  [1,]  1.00000000  0.41621844  0.0856928189  0.34517917  0.53878618
##  [2,]  0.41621844  1.00000000  0.4517814259 -0.25952007 -0.24355815
##  [3,]  0.08569282  0.45178143  1.0000000000 -0.09022706 -0.14945830
##  [4,]  0.34517917 -0.25952007 -0.0902270567  1.00000000 -0.37864479
##  [5,]  0.53878618 -0.24355815 -0.1494583019 -0.37864479  1.00000000
##  [6,]  0.53443627 -0.02618654 -0.1018654253 -0.31170047 -0.05195328
##  [7,] -0.48912736  0.52572197 -0.0730274350  0.65449847  0.49343474
##  [8,]  0.07919880  0.11708760  0.2075893652  0.01020317  0.05644247
##  [9,]  0.02850736  0.26031529  0.0007790436  0.01750511 -0.02686515
## [10,] -0.15495129  0.09712641  0.4062570210  0.05620966  0.12592362
##             [,6]        [,7]        [,8]          [,9]       [,10]
##  [1,]  0.53443627 -0.48912736  0.07919880  0.0285073607 -0.15495129
##  [2,] -0.02618654  0.52572197  0.11708760  0.2603152855  0.09712641
##  [3,] -0.10186543 -0.07302743  0.20758937  0.0007790436  0.40625702
##  [4,] -0.31170047  0.65449847  0.01020317  0.0175051069  0.05620966
##  [5,] -0.05195328  0.49343474  0.05644247 -0.0268651478  0.12592362
##  [6,]  1.00000000  0.37326032 -0.06012928 -0.0665343136  0.04609372
##  [7,]  0.37326032  1.00000000  0.12810128  0.0224492411 -0.04812089
##  [8,] -0.06012928  0.12810128  1.00000000  0.0795007857 -0.10063633
##  [9,] -0.06653431  0.02244924  0.07950079  1.0000000000  0.15223493
## [10,]  0.04609372 -0.04812089 -0.10063633  0.1522349258  1.00000000
```

```r
# Credible intervals
pc1$quant$`2.5%`[1,,]
```

```
##             [,1]        [,2]        [,3]       [,4]        [,5]
##  [1,]  1.00000000 -0.51235494 -0.54434847 -0.4585399 -0.09566078
##  [2,] -0.51235494  1.00000000 -0.02031701 -0.7193249 -0.68574089
##  [3,] -0.54434847 -0.02031701  1.00000000 -0.6492344 -0.65070281
##  [4,] -0.45853992 -0.71932492 -0.64923444  1.0000000 -0.74670566
##  [5,] -0.09566078 -0.68574089 -0.65070281 -0.7467057  1.00000000
##  [6,]  0.05126359 -0.55281050 -0.54236449 -0.7217342 -0.53976824
##  [7,] -0.81636545 -0.05738028 -0.71437094  0.2868984 -0.01388979
##  [8,] -0.43756764 -0.36923967 -0.22159281 -0.5132335 -0.42349742
##  [9,] -0.48776532 -0.42819083 -0.44592057 -0.5577747 -0.59139008
## [10,] -0.64117644 -0.33068866 -0.04078722 -0.4217755 -0.32478283
##             [,6]        [,7]       [,8]       [,9]       [,10]
##  [1,]  0.05126359 -0.81636545 -0.4375676 -0.4877653 -0.64117644
##  [2,] -0.55281050 -0.05738028 -0.3692397 -0.4281908 -0.33068866
##  [3,] -0.54236449 -0.71437094 -0.2215928 -0.4459206 -0.04078722
##  [4,] -0.72173423  0.28689839 -0.5132335 -0.5577747 -0.42177548
##  [5,] -0.53976824 -0.01388979 -0.4234974 -0.5913901 -0.32478283
##  [6,]  1.00000000 -0.28747334 -0.4960360 -0.5284953 -0.33865870
##  [7,] -0.28747334  1.00000000 -0.4455271 -0.6360222 -0.61811762
##  [8,] -0.49603596 -0.44552714  1.0000000 -0.2695615 -0.45464027
##  [9,] -0.52849528 -0.63602224 -0.2695615  1.0000000 -0.19672777
## [10,] -0.33865870 -0.61811762 -0.4546403 -0.1967278  1.00000000
```

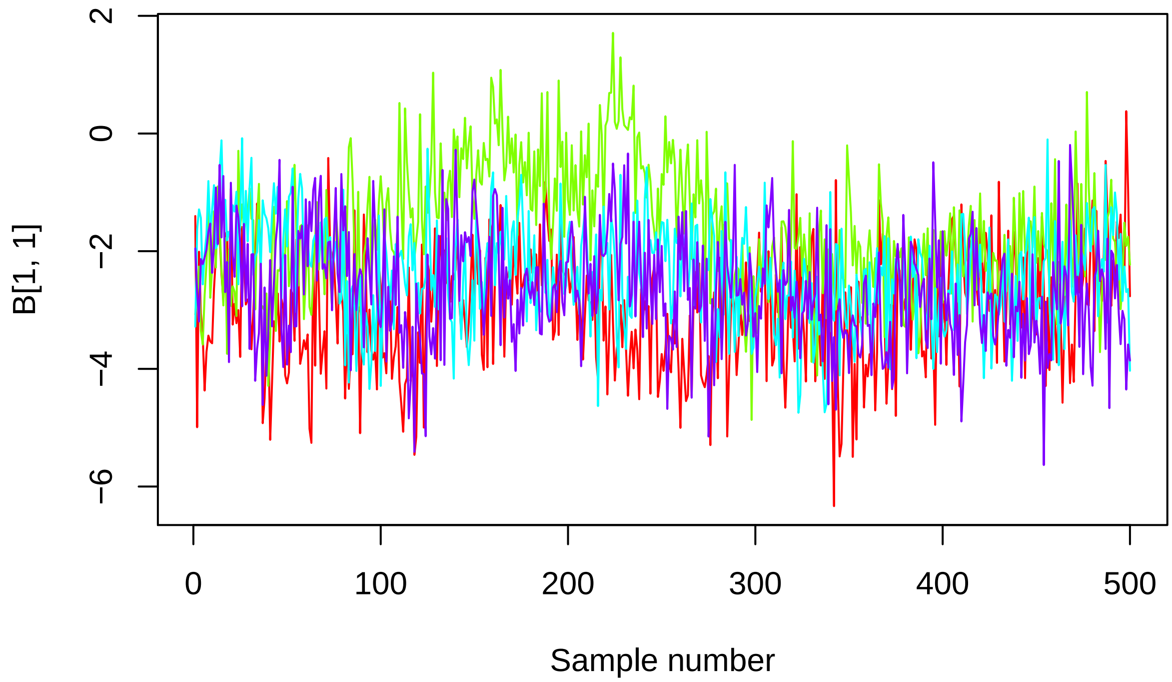
```r
pc1$quant$`97.5%`[1,,]
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
##  [1,] 1.0000000 0.8104767 0.6203881 0.7473430 0.8429574 0.8200649
##  [2,] 0.8104767 1.0000000 0.8252272 0.3389284 0.3197307 0.6041380
##  [3,] 0.6203881 0.8252272 1.0000000 0.5772463 0.4248091 0.4392941
##  [4,] 0.7473430 0.3389284 0.5772463 1.0000000 0.1569220 0.3814846
##  [5,] 0.8429574 0.3197307 0.4248091 0.1569220 1.0000000 0.5395097
##  [6,] 0.8200649 0.6041380 0.4392941 0.3814846 0.5395097 1.0000000
##  [7,] 0.4002063 0.8510129 0.5376504 0.8631801 0.7875441 0.7605729
##  [8,] 0.5974499 0.5586621 0.5769615 0.6260807 0.5635014 0.3425479
##  [9,] 0.5814319 0.7382730 0.4979821 0.6228233 0.5335024 0.4372249
## [10,] 0.2988021 0.5551981 0.6956715 0.5795748 0.6050562 0.5157605
##            [,7]      [,8]      [,9]     [,10]
##  [1,] 0.4002063 0.5974499 0.5814319 0.2988021
##  [2,] 0.8510129 0.5586621 0.7382730 0.5551981
##  [3,] 0.5376504 0.5769615 0.4979821 0.6956715
##  [4,] 0.8631801 0.6260807 0.6228233 0.5795748
##  [5,] 0.7875441 0.5635014 0.5335024 0.6050562
##  [6,] 0.7605729 0.3425479 0.4372249 0.5157605
##  [7,] 1.0000000 0.6615293 0.6406267 0.4683205
##  [8,] 0.6615293 1.0000000 0.4260342 0.2360918
##  [9,] 0.6406267 0.4260342 1.0000000 0.5092764
## [10,] 0.4683205 0.2360918 0.5092764 1.0000000
```

Model diagnostics

When running the model, you should always run multiple chains (the default is 4), to check convergence of the parameters. This can be investigated using the `trplot()` function. Let's check the traceplot for the $\mathbf{B}_{11}$ matrix element:

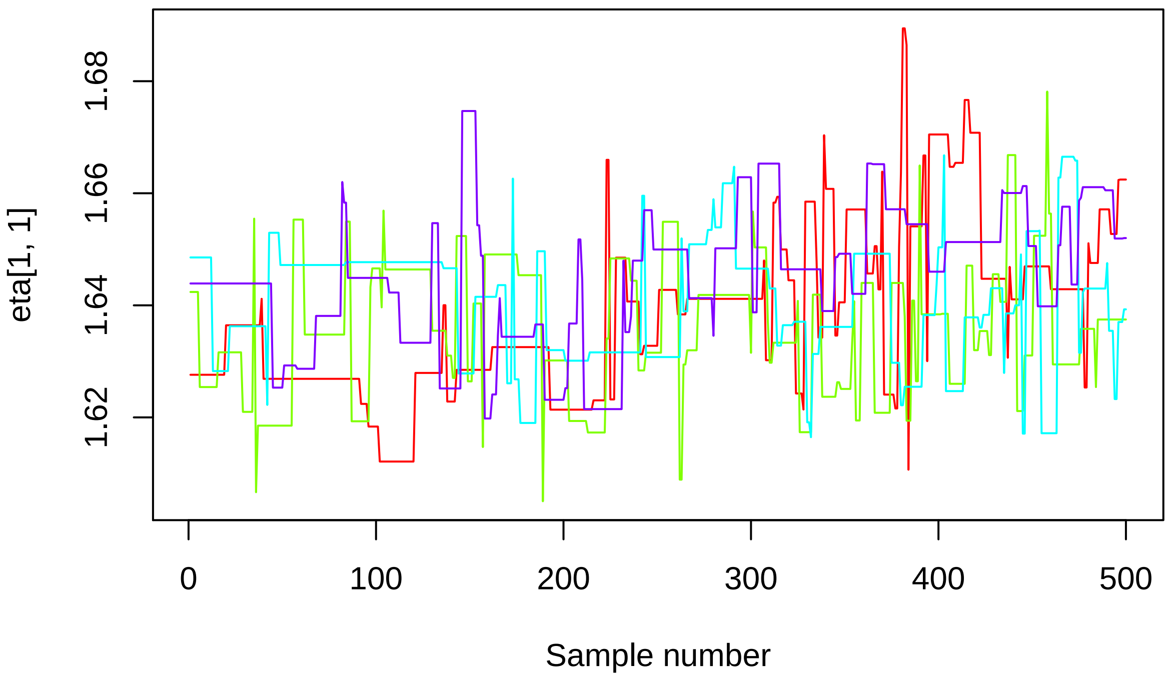
```r
trplot(mc.fit, par="B", ind=c(1,1))
```



In this example, the chains have not yet converged, meaning that it is possible that not enough MCMC samples have been run. In this case it would be wise to increase the parameters `n.burn` and `n.samp` in the `micore()` function.

It is also a good idea to check convergence of `eta`, as this parameter relies on the adaptive Metropolis step.

```
trplot(mc.fit, par="eta", ind=c(1,1))
```



Several parameters for the adaptive Metropolis sampler can be changed from their defaults; these parameters are specified as list elements of the `adapt.control` argument to `micore`.

- `init`: (default 0.1) The initial stepsize for the adaptive Metropolis parameters.
- `a`: (default 0.5) The adaptation rate. A higher value means the adaptations will vanish more quickly with the number of MCMC steps.
- `sigma.zero`: (default 1) The initial value of the adaptive Metropolis variance scaling parameter.

Sometimes the `a` parameter must be changed to get convergence for `eta`.

```
# Running micore with a different value for "a" parameter in adaptive Metropolis.
# Setting a = 0.3.
mc.fit <- micore(counts, X, n.burn = n.burn, n.samp = n.samp,
                 n.chain=4, n.cores=4, verbose=TRUE,
                 adapt.control = list(a=0.3))
```