# Bayesian Composition Estimator: manual

Karel Van den Meersche, Karline Soetaert

April 24, 2008

## 1 R

The Bayesian Composition Estimator (BCE) works as a package in the statistical software R. R can be downloaded from the R-website http://cran.R-project.org.
The package 'bce' can be installed in R like any regular R package. For now (22/11/2007), the package is not available from the CRAN website but can be installed from the local zip file. It is available as windows binary (bce_x.x.zip) and from source (bce_x.x.tar.gz).

### 1.1 installation from windows binary

Either use the menu tools in R-gui or at the command line:

```
> install.packages("BCE_x.x.zip", repos=NULL)
```

### 1.2 compile from source (linux)

in R:

```
> install.packages("BCE_x.x.tar.gz", repos=NULL, type="source")
```

or in console (as superuser):

```
R CMD INSTALL BCE_x.x.tar.gz
```

You can now load the package into R:

```
> library(BCE)
```

## 2 Demo: biomarkerComposition

An example script with two accompanying datasets is included in the package. *ratiomatrix* is a matrix containing the biomarker concentrations per species in function of biomass, *datamatrix* is a matrix containing the biomarker concentrations in function of biomass, measured in the samples. The example file now performs sequentially 5 runs of the algorithm, using different parameter settings. After each step we investigate the results and change settings to improve these results. An overview of the different parameter settings is given in the next sections.

```
> demo(biomarkerComposition)
```

In the first analysis, we assume a relative standard deviation of 0.2 on ratio matrix and data matrix. Other settings are chosen arbitrarily, with a low number of iterations to prevent long run time.

```
> X <- BCE(ratiomatrix,datamatrix,Relsdrat=.2,Relsddat=.2,
+             iter=1000,outputlength=5000,
+             jmpx=.01,jmprat=.01,export=FALSE)
```

The number of accepted runs is too low; we play around with the jump lengths jmpx and jmprat:

```
> X <- BCE(ratiomatrix,datamatrix,Relsdrat=.2,Relsddat=.2,
+             iter=1000,outputlength=5000,
+             jmpx=.02,jmprat=.002,export=FALSE)
```

We want to inspect the output:

```
> plot(X)
```

Mixing is still a little poor. To optimize mixing in the ratio matrix, it is often a good idea to make the jump length linear to the ratio matrix standard deviation (sdrat=.2*ratiomatrix) :

```
> X <- BCE(ratiomatrix,datamatrix,Relsdrat=.2,Relsddat=.2,
+             iter=1000,outputlength=5000,
+             jmpx=.02,jmprat=.2*(.2*ratiomatrix))
```

Mixing improved a lot; we repeat the run with more iterations to improve the reliability of the results. The following run can take a few minutes.

```
> X <- BCE(ratiomatrix,datamatrix,Relsdrat=.2,Relsddat=.2,
+             iter=100000,outputlength=5000,
+             jmpx=.02,jmprat=.2*(.2*ratiomatrix))
```

# 3 How to get your data into R?

Usually data are stored in spreadsheets or databases. There are numerous ways to get this data into R, one already more cumbersome than the other. One of the mostly recommended ways is to save your data in a comma-delimited text file and subsequently read this file into R:

```
> ratiomatrix <- read.csv("ratiomatrix.csv",row.names=1)
> datamatrix <- read.csv("datamatrix.csv",row.names=1)
```

An often useful alternative is to use an open database connection (ODBC) to access the data stored in a database. This has the advantage that if data is changed afterwards, one doesn't need to change any settings in the script to redo statistical analyses. Rerunning the script is sufficient.

```
> library(RODBC)
> channel1 <- odbcConnectExcel("data.xls")
> sqlTables(channel1)
> dataset1 <- sqlFetch(channel1, "sheet1",...)
> close(channel)
```

```
> library(RODBC)
> channel2 <- odbcConnectAccess("data.mdb")
> sqlTables(channel2)
> dataset2 = sqlQuery(channel2, "select * from query1")
> close(channel)
```

"query1" in this last line is the name of a query available in an Access database.

One can also insert data directly from the clipboard:

```
> dataset3 <- read.table("clipboard",header=TRUE)
```

Data stored in spreadsheets should be in the following format:

| | marker1 | marker2 | marker3 | marker4 |
|---|---|---|---|---|
| sample1 | 0.14717883 | 0.005304267 | 0.355269294 | 0.033341105 |
| sample2 | 0.151750423 | 0.004587804 | 0.368153586 | 0.034655562 |
| sample3 | 0.137554688 | 0.004045726 | 0.31392012 | 0.030437033 |
| sample4 | 0.136817915 | 0.005077227 | 0.337021004 | 0.031906365 |
| sample5 | 0.148508168 | 0.008032942 | 0.33873363 | 0.036519509 |
| sample6 | 0.118413262 | 0.082297217 | 0.348727057 | 0.04499704 |

Also the ratio matrix should be in the same format:

| | marker1 | marker2 | marker3 | marker4 |
|---|---|---|---|---|
| species1 | 0.27 | 0.13 | 0.35 | 0.076 |
| species2 | 0.084 | 0 | 0.5 | 0.24 |
| species3 | 0.195 | 0.3 | 0 | 0.1 |
| species4 | 0.06 | 0 | 0 | 0 |
| species5 | 0 | 0 | 0 | 0 |
| species6 | 0 | 0 | 0 | 0 |

You can then read your data into R using one of the aforementioned methods. If you also have standard deviation data, read also these into the program in a similar way:

```
> sdrat <- read.csv("sdrat.csv", row.names = 1)
> sddat <- read.csv("sddat.csv", row.names = 1)
```

# 4 Functions

What follows is an overview of the main functions included in the BCE package.

## 4.1 BCE()

```
> result <- BCE(ratiomatrix, datamatrix,...)
```

### 4.1.1 parameter settings for BCE()

Parameter settings for the function *BCE*, with default values: parameters *rat* (ratio matrix) and *dat* (data matrix) are the only obligatory parameters, but specification of some other parameters is recommended.

| | |
|---|---|
| Rat | Initial ratio matrix |
| Dat | Initial data matrix |
| relsdRat=0 | relative standard deviation for ratio matrix; can be a single value, a vector with length the number of biomarkers (1 value per biomarker) or a matrix with the same dimensions as the ratio matrix. |
| abssdRat=0 | absolute standard deviation for ratio matrix; use similar to relsdRat. |
| minRat=-Inf | minimum value of ratio matrix; use similar to relsdRat |
| maxRat=+Inf | maximum value of ratio matrix; use similar to relsdRat |
| relsdDat=0 | relative standard deviation of data matrix; can be a single value, a vector with length the number of biomarkers (1 value per biomarker) or a matrix with the same dimensions as the data matrix. |
| abssdDat=0 | absolute standard deviation of data matrix; use similar to relsdDat |
| tol = 1e-4 | minimum standard deviation for data matrix |
| tolX = 1e-4 | minimum X values for MCMC initiation (prevents numerical problems) |

| | |
|---|---|
| positive=1:ncol(Rat) | which columns contain strictly positive data? Other columns can become negative |
| iter = 100 | number of iterations for MCMC |
| outputlength=1000 | number of iterations kept in the output |
| burninlength=0 | number of initial iterations to be removed from output |
| jmpRat = 0.01 | jump length of Rat (also a vector with a value for each column, or a matrix with dimensions like Rat is accepted) |
| jmpX = 0.01 | jump length of X |
| unif = FALSE | do we take uniform distributions for ratio matrix? (as in chemtax) |
| verbose=TRUE | if FALSE, no feedback is provided during the run. |
| initRat=NULL | here you can optionally give a starting ratio matrix for the MCMC simulation. |
| initX=NULL | here you can optionally give a starting composition matrix for the MCMC simulation. |
| userProb = NULL | posterior probability for a given ratio matrix and composition matrix: should be a function with 2 arguments RAT and X, and as returned value a number giving the log posterior probability of ratio matrix RAT and composition matrix X. Dependence of the probability on the data should be incorporated in the function. If not specified, the default probability distribution is the gamma function |
| confidenceInterval = 2/3 | confidence interval in output; because the distributions are not symmetrical, standard deviations are not a useful measure; instead, upper and lower boundaries of the given confidence interval are given. Default is 2/3 (equivalent to standard deviation), but a more or less stringent criterion can be used. |
| export = FALSE | logical; if TRUE, a list of variables and plots are exported to the specified filename. |
| filename = "BCE" | Only if export is TRUE. If not NULL, a character string specifying the filename for saved objects. |

### 4.1.2 Output of BCE()

The output of the function BCE() is a list with 4 elements:

| | |
|---|---|
| Rat | array with dimension c(nrow(Rat),ncol(Rat),iter) containing the random walk values of the ratio matrix |
| X | array with dimension c(nrow(X),ncol(X),iter) containing the random walk values of the composition matrix |
| logp | vector with length iter containing the random walk values of the log posterior probability |
| naccepted | integer indicating the number of runs that were accepted |

The elements of this list can be used for further analyses, and for plotting. Three convenience functions are implemented for accessing the results of BCE(): summary.bce() and export.bce().

## 4.2 summary.bce()

The output of summary.bce() is provided as a list. All elements in the list *bceSummary* can be addressed by typing *result$<name>* or by attaching the result *attach(bceSummary)* and then typing the <*name*>. The following objects are available in this list:

| | |
|---|---|
| firstX | X determined through least squares regression from the initial ratio matrix and the data matrix |
| bestRat | Ratio matrix for which the posterior probability is maximal |
| bestX | Composition matrix for which the posterior probability is maximal |
| bestp | Maximal posterior probability |
| bestDat | Product of bestRat and bestX |
| meanRat | Means of the elements of the ratio matrix |
| sdRat | Standard deviation of the elements of the ratio matrix |
| lbRat | Lower boundary of the confidence interval of the elements of the ratio matrix |
| ubRat | Upper boundary of the confidence interval of the elements of the ratio matrix |
| covRat | Covariance matrix of the elements of the ratio matrix |
| meanX | Means of the elements of the composition matrix |
| sdX | Standard deviation of the elements of the composition matrix |
| lbX | Lower boundary of the confidence interval of the elements of the composition matrix |
| ubX | Upper boundary of the confidence interval of the elements of the composition matrix |
| covX | Covariance matrix of the elements of the composition matrix |

## 4.3 plot.bce()

Calling the plot-function with a bce-object as argument, will produce a series of plots with the random walks of all variables. The layout of these plots is kept very sober, as they are primarily intended for inspection of the random walk (see section 5). The user is free to write her/his own publication quality plots. Click or hit Enter to see the next plot, hit Esc to stop seeing new plots.

```
> result <- BCE(...)
> plot(result)
```

## 4.4 export.bce()

For people not familiar to R, it can be more "user-friendly" to set the parameter *export* in the function BCE() to *export=TRUE* or *export=<path/to/outputdirectory>*. The same result is obtained if you use the function *export.bce()* on a BCE object.
All summary output will be written to the specified folder or to a new folder *out*, created in the working directory. An R object containing all the results is saved and can be called using the function *load()*. Summary results are written to separate .csv files that can be read into a spreadsheet program. Also will the MCMC output be plotted into .png files. Take a good look at these plots before accepting your results (see next section).

# 5 Producing sensible output

## 5.1 mcmc

Markov Chain Monte Carlo simulations are not as straightforward as one might wish; several preliminary runs might be necessary to determine the desired number of iterations, burn-in length and jump length. Figure 1 shows what a good random walk should look like (a) and should certainly not look like (b).

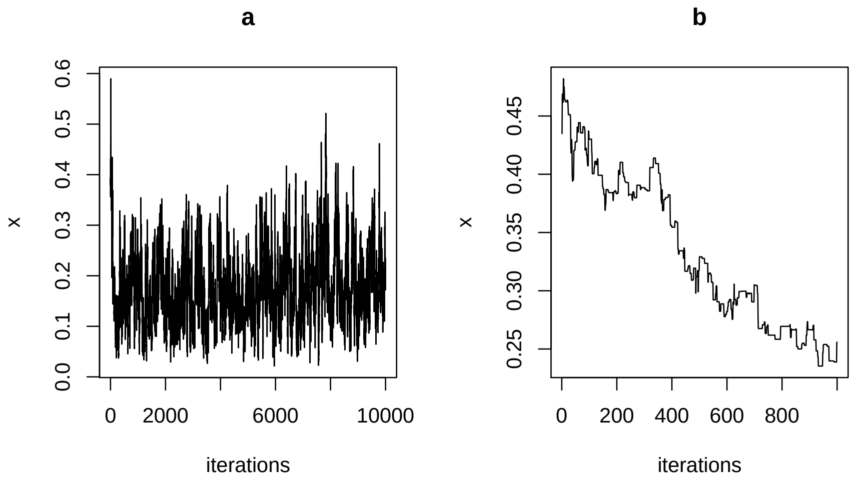


Figure 1:

- **jump length** The jump length determines how big the jumps are for each step in the random walk. A longer jump length will make you jump around faster in the parameter space, but acceptance of new points can get very low. Smaller jump lengths increase the acceptance rate, but the algorithm will move too slowly, and a lot more runs will be needed to scan the whole parameter space. A good way to find a good jump length, is look at the number of points accepted. If the output is saved under the name *MCMC*, you can find the number of accepted points under *MCMC$naccepted*. It is also given if you run the model with *verbose=TRUE* (default). This value should be somewhere between 5% and 40%. For long runs, 5 % can be acceptable, for short runs, you will prefer a higher acceptance in order to have enough different points. 20% accepted is usually a good number. Do some preliminary runs with *iter=1000-10000* and tune the jump lengths. You can set different jump lengths for each column of the ratio matrix, or 1 jump length for the whole ratio matrix, and 1 jump length for the composition matrix. Decreasing the jump lengths will generally increase the acceptance rate and vice versa. Also the mixing rate (the speed with which accepted points change their values) will be influenced. You want this mixing rate to be as high as possible.

- **burninlength** The program uses the solution of lsei using the original ratio matrix as starting values for the MCMC. This might in some cases be far from the optimal solution, and the MCMC algorithm will start with moving towards this optimal solution. This is called a burn-in. When there is a slow mixing rate, this can take a considerable number of cycles. As it can influence the averages and standard deviations, you might want to remove it from the mcmc objects. By defining a burnin length, the first <burninlength> cycles will not be written to the output. Look at some plots to determine if you need to specify a burnin length (fig 2)

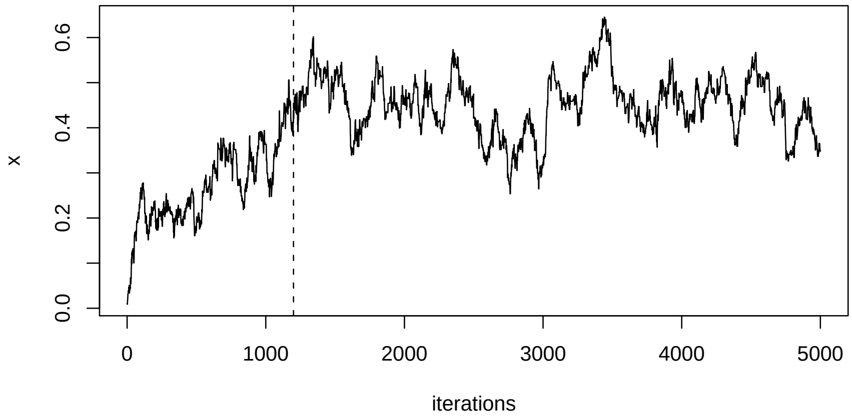


Figure 2: parameter values show a clear trend in the first 1200 cycles, left from the dashed line; we call this a burn-in. You can remove it from the output by setting the parameter *burninlength*.

- **iter** the number of iterations: start with 10000 runs; check the mcmc output and estimate how many runs you will need to get a random pattern in the output (fig 1a).