Step by step instructions to apply the matrix-based method using R

The analyses are carried out using the R software. The MCMC is performed using the `Stan` library (http://mc-stan.org) via the `rstan` package. The integrated likelihood is obtained using the `bridge_sampler` function of the `bridgesampling` package. This approach is now detailed.

Obtaining and installing R

R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. Sources binaries, documentation and additional packages can be obtained via the CRAN http://cran.R-project.org. It is available for a wide range of operating systems (Windows, Linux, Mac OS, . . .).

Installing `rstan` and `bridgesampling` packages

To carry out our Bayesian analysis it is necessary to install the `rstan` and `bridgesampling` packages, it can be performed by running the following command lines:

```
install.packages("rstan")
install.packages("bridgesampling")
```

Some particular attention may be taken for the `rstan` package installation, since it requires an operational C++ compiler for running the underlying Stan program. See https://mc-stan.org/users/interfaces/rstan for more details.

Reading the discovery matrix

R can read a variety of format see for instance R Data Import/Export manual. In particular it can read csv (comma separated values) files which can be obtained from MS Excel Save menu. Let consider the `d.csv` file, it can be imported in R through the following command line:

```
d = read.csv("d.csv")
class(d)
```

```
## [1] "data.frame"
```

```
d[1:5,1:21]
```

```
##   V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21
## 1  0  0  1  0  0  0  1  0  0   0   1   1   1   0   0   0   0   0   0   0   0
## 2  0  1  0  0  0  0  0  1  0   0   1   1   0   0   0   0   0   1   0   0   0
## 3  1  0  0  0  0  0  0  0  0   0   0   1   0   0   1   0   0   1   1   0   1
## 4  0  0  0  0  0  0  0  0  0   0   0   0   0   1   0   0   0   0   0   0   0
## 5  1  0  0  0  0  0  0  0  1   1   1   0   0   0   0   0   0   0   0   0   0
```

The data-frame `d` can then be converted into matrix through the `as.matrix` command:

```r
d = as.matrix(d)
class(d)
```

```
## [1] "matrix" "array"
```

Performing the MCMC algorithm through `rstan` for a given m

Our aim is to draw values coming from $p(m, \mu, \sigma^2|\mathbf{d})$. To do this we will first draw μ et σ^2 for each possible value of m, $m \in \{1, \ldots, M\}$: i.e. $p(\mu, \sigma|m, \mathbf{d})$ and deduce a numerical approximation of $p(\mathbf{d}|m)$ from the Monte-Carlo sample. Finally it will be possible to deduce $p(m|\mathbf{d})$ from Bayes formula.

Sampling from $p(\mu, \sigma^2|m, \mathbf{d})$ with `rstan`

First:

$$p(\mu, \sigma^2|m, \mathbf{d}) \propto p(\mathbf{d}|\mu, \sigma^2, m)p(\mu)p(\sigma^2)p(m)$$

Second:

$$p(\mathbf{d}|\mu, \sigma^2, m) \propto A_m^j \times p(\hat{\mathbf{x}}^m|\mu, \sigma^2)$$

where $\hat{\mathbf{x}}^m$ is the $\mathbf{d}$ matrix padded with $m - j$ null columns.

Thus in practice for m fixed we will consider sampling from $p(\mu, \sigma^2|\hat{\mathbf{x}}^m)$, deduce the integrated likelihood $p(\hat{\mathbf{x}}^m)$ using bridge sampling and get $p(\mathbf{d}|m)$ up to a multiplicative constant not depending of m.

In order to perform the simulation of the parameters μ and σ^2 given $\hat{\mathbf{x}}^m$ we will consider the parameter space augmented by p_1, …, p_m, the discovery probabilities associated with each column of $\hat{\mathbf{x}}^m$. Thus we will now sample from $\mu, \sigma^2, p_1, \ldots, p_m|\hat{\mathbf{x}}^m$.

The resulting Stan model is described in the `draw_mu_s2.stan` file. This file basically specifies the prior distribution on the parameters and the likelihood of the model.

The first step is to load the `rstan` package through the command line:

```r
library(rstan)
```

Then to compile the Stan file for latter use:

```r
model <- stan_model('draw_mu_s2.stan')
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
```

The Stan model can now be used to sample from the posterior distribution given the data: $\mu, \sigma^2, p_1, \ldots, p_m | \hat{\mathbf{x}}^m$.

For instance, for $m = 30$, we compute the required usability summary data

```r
set.seed(1234) # to get reproducible results
m = 30
j = ncol(d)
n = nrow(d)
nl = colSums(d)
x <- c(nl, rep(0, m - j)) # exhaustive statistic according to the model
usability_dat <- list(n = n, m = m, x = x, mu_prior_mean = 0,
                      mu_prior_sd = 10000, s2_prior_a = 0.5,
                      s2_prior_b = 0.5)
```

Then the object `usability_dat`, is used as input for the Stan program through the use of the `sampling` function of `rstan`:

```r
fit <- sampling(object = model,  data = usability_dat, refresh = 0)
```

Sampled values for the given value m can eventually be extracted:

```r
mu = do.call("c", lapply(fit@sim$samples, function(x) x$mu[1001:2000]))
# Where 1000 first burn in iterations are removed
s2 = do.call("c", lapply(fit@sim$samples, function(x) x$s2[1001:2000]))
simu = cbind(mu = mu, s2 = s2, m = m)
head(simu)
```

```
##             mu        s2  m
## [1,] -1.673605 0.8568641 30
## [2,] -1.786976 0.6222704 30
## [3,] -1.767101 0.3632042 30
## [4,] -1.468858 0.4040611 30
## [5,] -1.376064 0.3740194 30
## [6,] -1.549735 1.3503627 30
```

Obtaining the integrated likelihood through the `bridgesampling` package

The posterior sample can then be used as an input of the `bridge_sampler` function of the `bridgesampling` package in order to compute the log of the integrated likelihood $\log p(\mathbf{x}_m)$:

```r
library(bridgesampling)
lp <- bridge_sampler(fit,silent = TRUE)$logml
```

Then the log of the $p(\mathbf{d}|m)$ can be obtained from

$$\log p(\mathbf{d}|m) = \log A_m^j + \log p(\hat{\mathbf{x}}^m) + C$$

where C is a constant not depending of m.

```r
lp <- lp + lchoose(m,j)
```

where `lchoose(m,j)`$= \log A_m^j - \log j!$.

Making such computation for each possible value of m and by using the Bayes formula we get $p(m|\mathbf{d})$.

Performing the whole approche through the `heterogeneous_bayes` function

The whole approach is implemented in the file `functions.R`. This file can simply be sourced:

```r
source("functions.R")
```

Then it is possible to use the `heterogenous_bayes` function in the following way:

```r
mbest = heterogeneous_bayes(d, M = 50)
mbest
```

```
## [1] 31
```

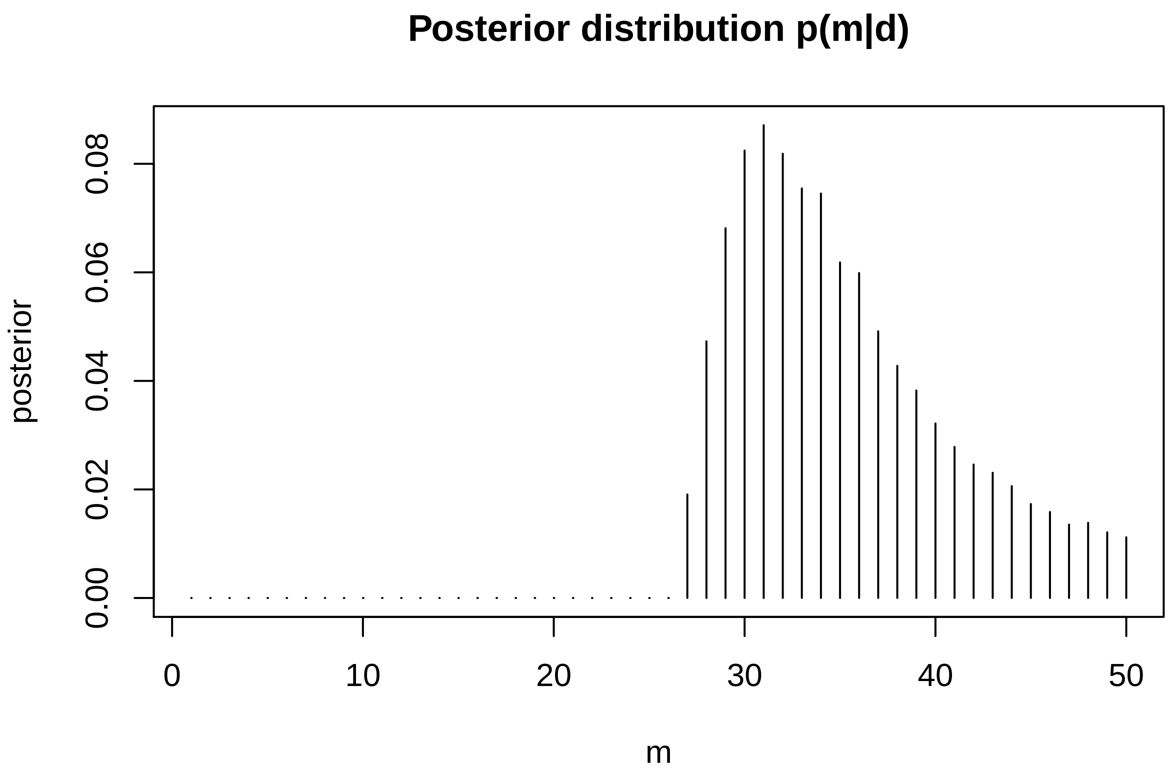
where M is the maximum number of problem. Here by default we only return the value maximizing $p(m|\mathbf{d})$. However we can also return more information with the option `full_output = TRUE`:

```r
full_output = heterogeneous_bayes(d, M = 50,full_output = TRUE)
str(full_output)
```

```
## List of 2
##  $ posterior_m: num [1:50, 1:2] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "m" "posterior"
##  $ simu_mu_s2 : num [1:96000, 1:3] -1.32 -1.55 -1.28 -1.28 -1.41 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "mu" "s2" "m"
```

It is possible to look at the posterior distribution of m:

```r
plot(full_output$posterior_m,type = "h", main = "Posterior distribution p(m|d)")
```



Or to look at the sampled values of μ and σ^2 given m:

```r
head(full_output$simu_mu_s2)
```

```
##             mu        s2  m
## [1,] -1.319039 0.1993071 27
## [2,] -1.548304 0.2899753 27
## [3,] -1.277625 0.2479372 27
## [4,] -1.282017 0.2484649 27
## [5,] -1.409928 0.1549367 27
## [6,] -0.973836 0.9351232 27
```