BHTSpack: Bayesian Multi-Plate High-Throughput Screening of Compounds

Ivo D. Shterev * David B. Dunson Cliburn Chan
Gregory D. Sempowski

September 7, 2018

1 Introduction

This vignette describes the use of the R extension package BHTSpack for Bayesian multi-plate high-throughput screening of compounds. The package implements the non-parametric Bayes model described in [2]. The model infers potential active compounds (hits) from multiple plates, simultaneously. The plate geometry can be arbitrary and plates may contain different numbers of unique compounds.

This vignette mainly describes the package functionalities, along with examples. We also provide code that reproduces figures in the main body of [2]. For more detailed description of the statistical model, please see [2].

2 Data Format

The R package BHTSpack assumes that the compound data set is in the form of a list of vectors. Each vector represents a plate. The vector names are the plate names. The element names within a vector are the compound names.

3 Synthetic Data Generation

Synthetic compound data containing specified proportion of compound hits can be generated via the package function `data.create()`. The resulting data set can be in a list of vectors format or in a list of matrices format. Additionally, the user has the option to add a plate noise to each of the compound plates, thus simulating a plate noise effect. The function `data.create()` has the following input parameters:

- `N`: Number of compounds per plate.
- {`nr`, `nc`}: Number of rows and columns per plate, respectively.
- `M`: Number of plates.
- `p`: Probability of a compound being a hit.

*Correspondence: i.shterev@duke.edu

- `s`: Random seed (for reproducibility purposes). Default is `NULL`.
- {`covrow`, `covcol`}: Noise plate row-covariance and column-covariance matrices, respectively. Defaults are `NULL`.
- `c`: Constant for scaling plate noise. Default is `0.0001`.
- `mat`: Specifies a matrix (`TRUE`) or a vector (`FALSE`) plate format. Default is `FALSE`.

The output of `create.data()` is in the form of a list with the following elements:

- `Z`: A list of matrices (`mat=TRUE`) or vectors (`mat=FALSE`) of compounds.
- `I`: A list of compound indicators specifying the mixture component (from `1` to `K`).
- `B`: A list of compound indicators specifying a hit (`1`) or a non-hit (`0`).

4 Input

The package main function is `bhts()`. The following input data and parameters need to be specified:

- `Z`: A list of vectors containing unique compound readouts. The vectors should have names, as well as the elements within each vector.
- `iters`: Total number of iterations to be performed by the MCMC sampler. The first half of the iterations will be discarded as burn in iterations and only the second half will be used in the result.
- {`H`, `K`}: Number of local and global DP components, respectively.
- {`mu00, mu10`}: Activity levels (means) of non-hit and hit compounds, respectively. If `NULL` (default), `mu00` is set as half the mean of all compounds and `mu10=3*mu00`.
- {`a.alpha, b.alpha`}: Gamma parameters specifying local DP concentration prior.
- {`a.tau, b.tau`}: Gamma parameters specifying global DP concentration prior.
- `pnorm`: Plate normalization. If `TRUE`, each plate is normalized to zero mean and unit variance, prior to analysis. Default is `FALSE`.
- `s`: Used to specify a random seed (for reproducibility purposes). Default is `NULL`.
- `store`: If `TRUE`, all samples of certain latent variables are stored in the output object. Default is `FALSE`.

5 Output

The output of `bhts()` is in the form of a list with the following elements:

- `hatpai`: A list of vectors of posterior probabilities, estimating the probability of a compound being a hit.

- `dat.store`: If `store=TRUE` (default is `FALSE`), the output contains a list of `iters`×`K` matrices of samples. Each matrix contains the samples of a separate latent variable. At each iteration, the following six variables are stored in a different row of their corresponding matrix, $(\lambda_1^{(0)}, \ldots, \lambda_K^{(0)})$, $(\lambda_1^{(1)}, \ldots, \lambda_K^{(1)})$, $(\mu_{01}, \ldots, \mu_{0K})$, $(\mu_{11}, \ldots, \mu_{1K})$, $(\sigma_{01}^2, \ldots, \sigma_{0K}^2)$ and $(\sigma_{11}^2, \ldots, \sigma_{1K}^2)$.

6 Determining Significant Hits

The output from `bhts()` can be further processed by the use of the function `r.fdr()`. For a given FDR estimate [1], the function computes a significant hit probability threshold `r`, based on which a list of all significant compound hits is determined. The compounds are sorted in decreasing order of `hatpai`.

The output of `r.fdr()` is in the form of a list with the following elements:

- `res`: A data frame containing significant hits and their probabilities.
- `r`: The computed significant hit probability threshold.

7 Convergence Diagnostics

The package has the capability to visualize trace and autocorrelation function (ACF) plots of specific latent variables, via the function `ptrace()`. These plots can be useful in assessing convergence of the algorithm. Given a specific variable, the function produces a composite plot consisting of `K` trace plots, one for each component. It has to be noted that for each iteration, the `K` components of a variable are sorted in increasing order to avoid label switching. The `pval()` function input is as follows:

- `res`: An output object from `bhts()`.
- `var`: Variable for which to display trace plots. Current options are `mu0` (displaying $\mu_{01}, \ldots, \mu_{0K}$), `mu1` (displaying $\mu_{11}, \ldots, \mu_{1K}$), `sigma0` (displaying $\sigma_{01}^2, \ldots, \sigma_{0K}^2$), `sigma1` (displaying $\sigma_{11}^2, \ldots, \sigma_{1K}^2$), `pk0` (displaying $\lambda_1^{(0)}, \ldots, \lambda_K^{(0)}$) and `pk1` (displaying $\lambda_1^{(1)}, \ldots, \lambda_K^{(1)}$).
- `ndisc`: Number of iterations for which to discard samples.
- `nr`: Number of rows in the resulting composite plot.
- `nc`: Number of columns in the resulting composite plot.
- `type`: Type of convergence diagnostic. Currently implemented are trace plots (default `type="trace"`) and ACF plots (`type="acf"`).

8 Examples

In this section we demonstrate the use of `BHTSpack` on synthetically generated data.

```
# loading library
library(BHTSpack)


## Loading required package:  R2HTML
## Loading required package:  xtable
```

```r
# Generating a data set of 100 8x10 plates, each plate containing 80 compounds.
# A total of 8000 compounds. 10% of the compounds are hits.
Z = data.create(N=80, nr=8, nc=10, M=100, p=0.4, s=1234)

# Generating the data set as before, but this time adding plate noise to all compounds
Z = data.create(N=80, nr=8, nc=10, M=100, p=0.4, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))

# Running the model with 200 iterations
system.time(b.est <- bhts(Z[["Z"]], iters=200, H=10, K=10, a.alpha=10, b.alpha=5, a.tau=10, b.tau=5, s=1234, store=TRUE))
```

```
## iter=10, iter=20, iter=30, iter=40, iter=50, iter=60, iter=70, iter=80, iter=90, iter=100,
## iter=110, iter=120, iter=130, iter=140, iter=150, iter=160, iter=170, iter=180, iter=190, iter=200,
##    user  system elapsed
##   9.033   0.080   9.113
```

```r
# Compute threshold (r) for significant hit probabilities at FDR=0.05
res = r.fdr(b.est, fdr=0.05)
names(res)
```

```
## [1] "res" "r"
```

```r
res[["r"]]
```

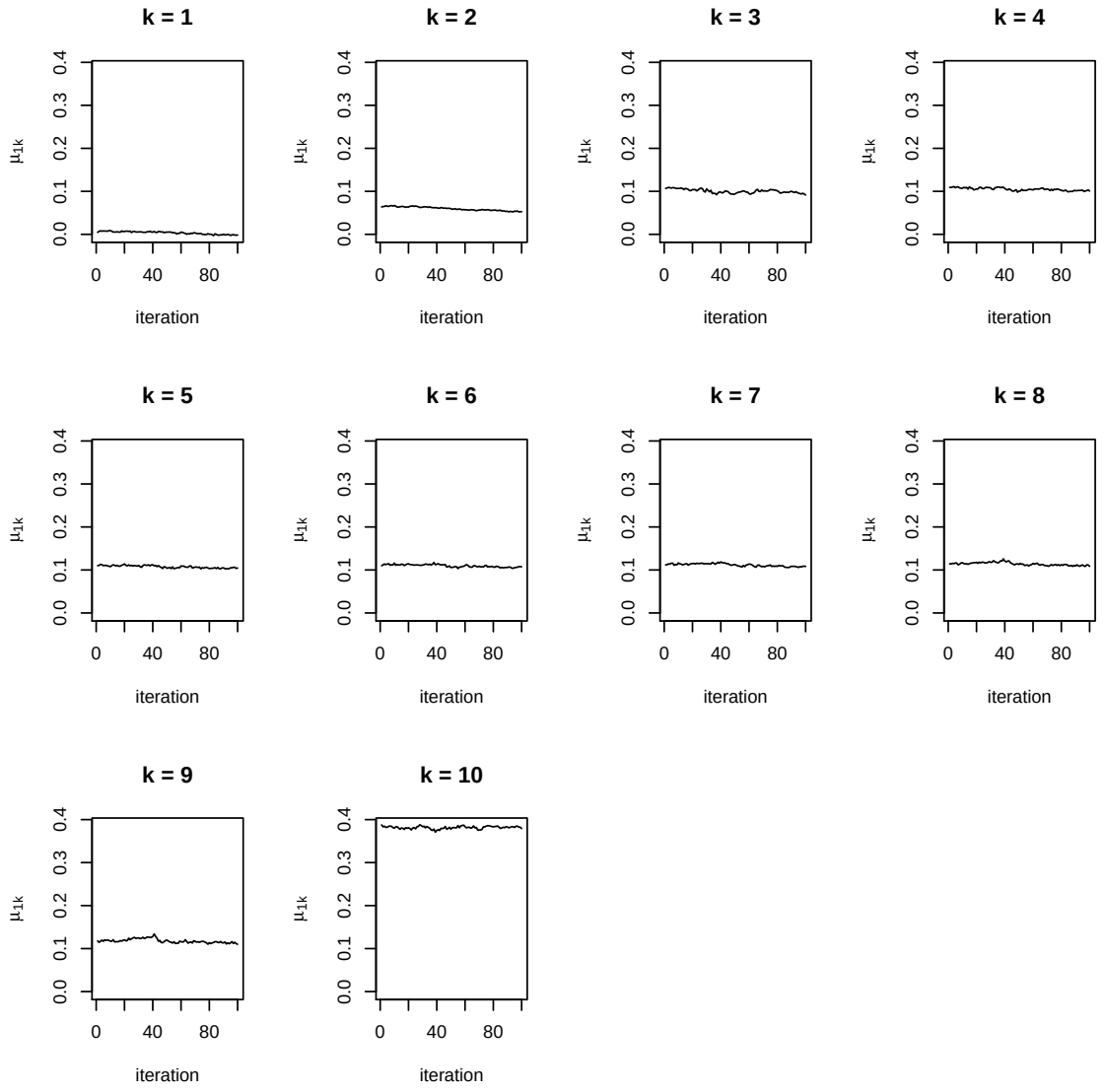
```
## [1] 0.4976846
```

```r
# Significant compound hit list
head(res[["res"]])
```

```
##          ID hatpai
## 1 Plate1.E8      1
## 2 Plate1.F8      1
## 3 Plate1.G8      1
## 4 Plate3.B3      1
## 5 Plate3.C4      1
## 6 Plate4.C3      1
```

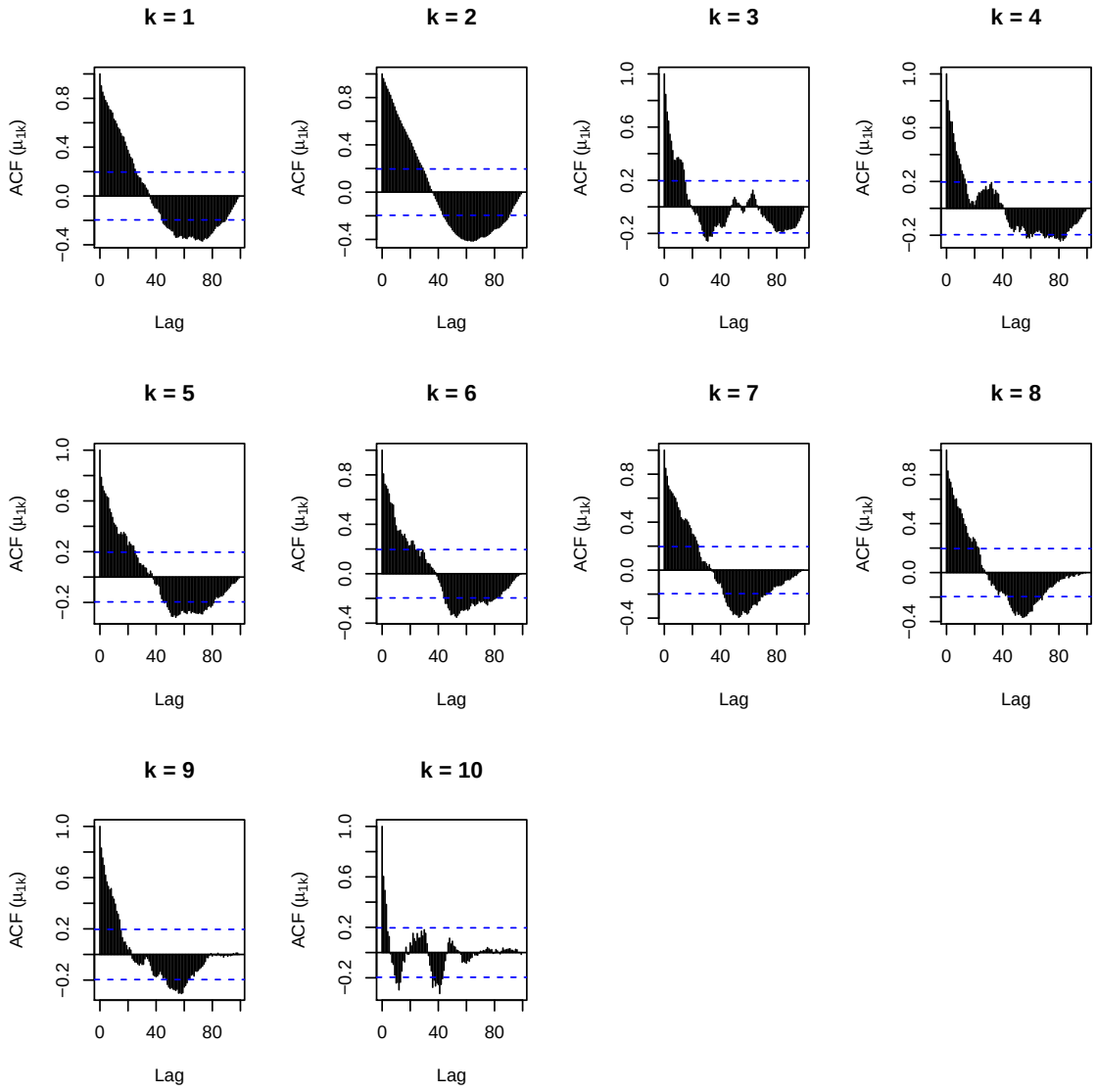
```r
# Trace plots of hit compound activity
ptrace(b.est, "mu1", ndisc=100, nr=3, nc=4)

# ACF plots of hit compound activity
ptrace(b.est, "mu1", ndisc=100, nr=3, nc=4, type="acf")
```




```
sessionInfo()
```

```
## R version 3.4.4 (2018-03-15)
## Platform: i686-pc-linux-gnu (32-bit)
## Running under: Ubuntu 18.04.1 LTS
##
## Matrix products: default
## BLAS: /usr/lib/i386-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/i386-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] BHTSpack_0.5 xtable_1.8-3 R2HTML_2.3.2
##
## loaded via a namespace (and not attached):
## [1] compiler_3.4.4 magrittr_1.5   tools_3.4.4    stringi_1.2.4
## [5] knitr_1.20     stringr_1.3.1  evaluate_0.11
```



9 Exporting Results to an HTML File

The user has the option to export the results of `bhtsPACK` to an HTML file via the function `bhts2HTML`. This option allows for improved readability of the results.

```
# loading library
library(BHTSpack)

# Generating a data set of 100 8x10 plates, each plate containing 80 compounds.
# A total of 8000 compounds. 40% of the compounds are hits.
Z = data.create(N=80, nr=8, nc=10, M=100, p=0.4, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))

# Running the model with 200 iterations
b.est = bhts(Z[["Z"]], iters=200, H=10, K=10, a.alpha=10, b.alpha=5, a.tau=10, b.tau=5, s=1234, store=TRUE)


## iter=10, iter=20, iter=30, iter=40, iter=50, iter=60, iter=70, iter=80, iter=90, iter=100,
## iter=110, iter=120, iter=130, iter=140, iter=150, iter=160, iter=170, iter=180, iter=190, iter=200,


# create an html file
#bhts2HTML(res, dir="/dir/", fname="tophits")
```

10 Reproducing Results

In this section we provide code that reproduces figures in the main body of [2].

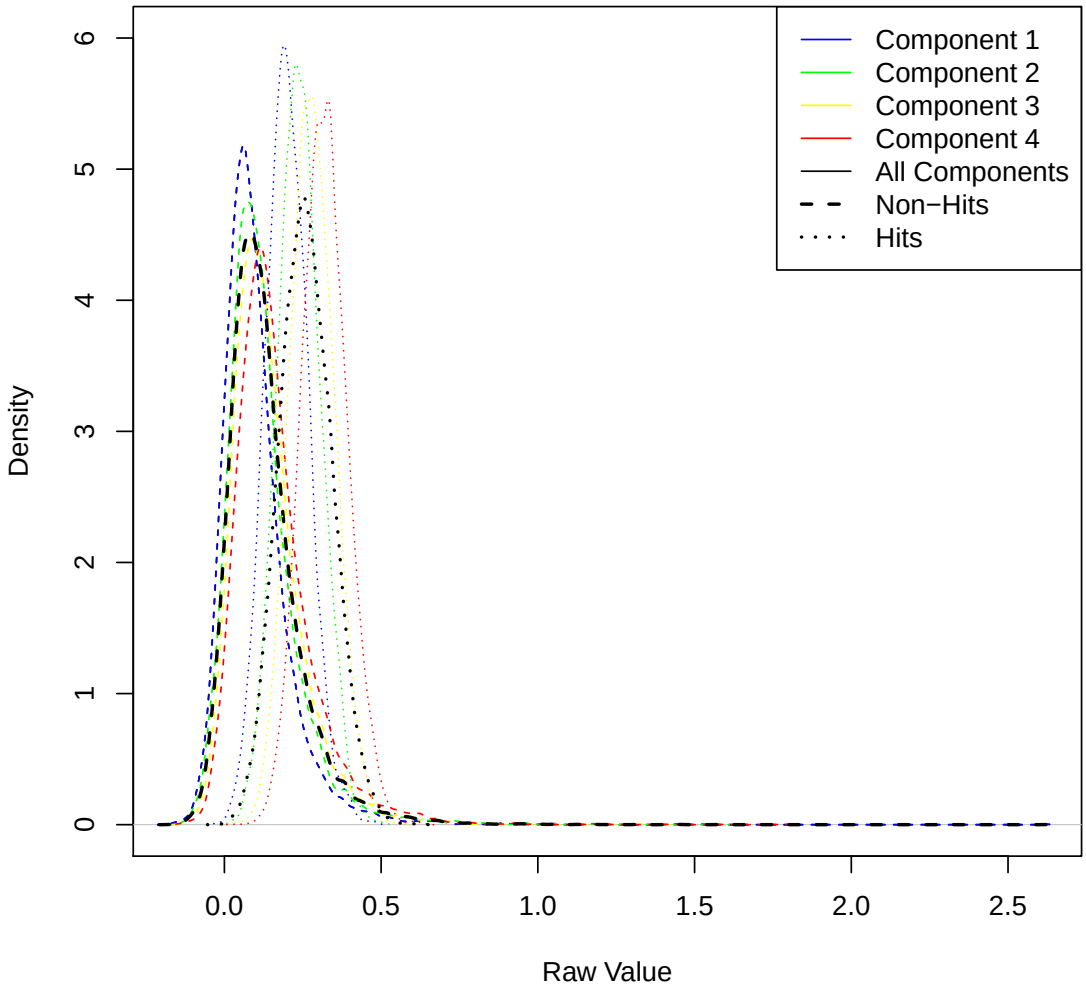
10.1 Reproducing Figure 5 (Synthetic Data)

```
library(BHTSpack)
Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.4, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))
I = unlist(Z[["I"]])
B = unlist(Z[["B"]])
Z = unlist(Z[["Z"]])

plot(density(Z[I==1 & B==0]), xlim=range(Z), ylim=c(0,6), col="black", lty=2, ylab="Density", main="", xlab="Raw Value")
lines(density(Z[I==1 & B==0]), col="blue", lty=2)
lines(density(Z[I==2 & B==0]), col="green", lty=2)
lines(density(Z[I==3 & B==0]), col="yellow", lty=2)
lines(density(Z[I==4 & B==0]), col="red", lty=2)
lines(density(Z[B==0]), col="black", lty=2, lwd=2)

lines(density(Z[I==1 & B==1]), col="blue", lty=3)
lines(density(Z[I==2 & B==1]), col="green", lty=3)
lines(density(Z[I==3 & B==1]), col="yellow", lty=3)
lines(density(Z[I==4 & B==1]), col="red", lty=3)
lines(density(Z[B==1]), col="black", lty=3, lwd=2)

legend("topright", legend=c("Component 1", "Component 2", "Component 3", "Component 4", "All Components", "Non-Hits", "Hits"),
col=c("blue", "green", "yellow", "red", "black", "black", "black"), lty=c(1, 1, 1, 1, 1, 2, 3), lwd=c(1, 1, 1, 1, 1, 2, 2))
```



10.2 Reproducing Figure 7

The code in this subsection takes longer to execute and is therefore disabled. The user can copy, paste and run the code.

```
#library(BHTSpack)
#library(pROC)
#library(sights)

#score = function(t, sdat, B){
#  res = unlist(lapply(sdat, as.vector))
#  ind = rep(0, length(res))
#  ind[res>t] = 1

#  a = auc(B, ind)
#  return(a)
#}

### Left Column
#Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.1, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))
#system.time(b.est <- bhts(Z[["Z"]], iters=7000, H=10, K=10, a.alpha=10, b.alpha=5, a.tau=10, b.tau=5, s=1234, store=TRUE))
#hatpai = unlist(b.est[["hatpai"]])
#res = data.frame(IDmatch=names(hatpai), hatpai)
#Btab = data.frame(IDmatch=names(unlist(Z[["B"]])), hitind=unlist(Z[["B"]]))
#res = merge(res, Btab, by="IDmatch")
```

```
#Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.1, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"), mat=TRUE)

## Top plot
#bs = unlist(lapply(Z[["Z"]], function(x){medpolish(x)[["residuals"]]/mad(x)}))
#summary(bs)

#rs = unlist(lapply(Z[["Z"]], function(x){matrix(normR(as.vector(t(x)), 8, 10), 8, 10, byrow=TRUE)}))
#summary(rs)

#r = seq(-4, 21, 0.5)
#AUC = unlist(lapply(r, function(x){score(x, bs, unlist(Z[["B"]]))}))
#summary(AUC)
#btmax = r[which.max(AUC)]
#plot(r, AUC, type="l", xlab="Threshold", ylab="AUC", lwd=2, xaxt="n", col="red", ylim=c(0.5, 0.75))
#abline(v=btmax, col="red", lty=2)
#axis(1, at=c(-5, 5, 10, 15))
#axis(1, at=btmax)

#r = seq(-4, 21, 0.5)
#AUC = unlist(lapply(r, function(x){score(x, rs, unlist(Z[["B"]]))}))
#summary(AUC)
#rtmax = r[which.max(AUC)]
#lines(r, AUC, type="l", xlab="Threshold", ylab="AUC", lwd=2, xaxt="n", col="green")
#axis(1, at=c(-5, 5, 10, 15))
#axis(1, at=rtmax)
#legend("topright", legend=c("R-score", "B-score"), col=c("green", "red"), lty=c(1,1))

## Bottom plot
#rhitind = rep(0, length(rs))
#rhitind[rs>rtmax] = 1

#bhitind = rep(0, length(bs))
#bhitind[bs>btmax] = 1

#plot.roc(res[["hitind"]], res[["hatpai"]], col="blue")
#lines.roc(unlist(Z[["B"]]), bhitind, col="red")
#lines.roc(unlist(Z[["B"]]), rhitind, col="green")
#legend("bottomright", legend=c(paste("BHTS", " (AUC=", round(auc(res[["hitind"]], res[["hatpai"]]), 3), ")", sep=""), paste("R-score",
#" (AUC=", round(auc(unlist(Z[["B"]]), rhitind), 3), ")", sep=""), paste("B-score", " (AUC=", round(auc(unlist(Z[["B"]]), bhitind), 3),
#")", sep="")), col=c("blue", "green", "red"), lty=c(1,1,1))

### Middle Column
#Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.05, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))
#system.time(b.est <- bhts(Z[["Z"]], iters=7000, H=10, K=10, a.alpha=10, b.alpha=5, a.tau=10, b.tau=5, s=1234, store=TRUE))
#hatpai = unlist(b.est[["hatpai"]])
#res = data.frame(IDmatch=names(hatpai), hatpai)
#Btab = data.frame(IDmatch=names(unlist(Z[["B"]])), hitind=unlist(Z[["B"]]))
#res = merge(res, Btab, by="IDmatch")

#Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.05, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"), mat=TRUE)

## Top plot
#bs = unlist(lapply(Z[["Z"]], function(x){medpolish(x)[["residuals"]]/mad(x)}))
#summary(bs)

#rs = unlist(lapply(Z[["Z"]], function(x){matrix(normR(as.vector(t(x)), 8, 10), 8, 10, byrow=TRUE)}))
#summary(rs)

#r = seq(-4, 21, 0.5)
#AUC = unlist(lapply(r, function(x){score(x, bs, unlist(Z[["B"]]))}))
#summary(AUC)
#btmax = r[which.max(AUC)]
#plot(r, AUC, type="l", xlab="Threshold", ylab="AUC", lwd=2, xaxt="n", col="red", ylim=c(0.5, 0.75))
#abline(v=btmax, col="red", lty=2)
#axis(1, at=c(-5, 5, 10, 15))
#axis(1, at=btmax)

#r = seq(-5, 26, 0.5)
#AUC = unlist(lapply(r, function(x){score(x, rs, unlist(Z[["B"]]))}))
#summary(AUC)
#rtmax = r[which.max(AUC)]
#lines(r, AUC, type="l", xlab="Threshold", ylab="AUC", lwd=2, xaxt="n", col="green")
#axis(1, at=c(-5, 5, 10, 15))
#axis(1, at=rtmax)
#legend("topright", legend=c("R-score", "B-score"), col=c("green", "red"), lty=c(1,1))

## Bottom plot
#rhitind = rep(0, length(rs))
#rhitind[rs>rtmax] = 1

#bhitind = rep(0, length(bs))
#bhitind[bs>btmax] = 1

#plot.roc(res[["hitind"]], res[["hatpai"]], col="blue")
#lines.roc(unlist(Z[["B"]]), bhitind, col="red")
#lines.roc(unlist(Z[["B"]]), rhitind, col="green")
#legend("bottomright", legend=c(paste("BHTS", " (AUC=", round(auc(res[["hitind"]], res[["hatpai"]]), 3), ")", sep=""), paste("R-score",
#" (AUC=", round(auc(unlist(Z[["B"]]), rhitind), 3), ")", sep=""), paste("B-score", " (AUC=", round(auc(unlist(Z[["B"]]), bhitind), 3),
#")", sep="")), col=c("blue", "green", "red"), lty=c(1,1,1))

### Right Column
```

```
#Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.01, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))
#system.time(b.est <- bhts(Z[["Z"]], iters=7000, H=10, K=10, a.alpha=10, b.alpha=5, a.tau=10, b.tau=5, s=1234, store=TRUE))
#hatpai = unlist(b.est[["hatpai"]])
#res = data.frame(IDmatch=names(hatpai), hatpai)
#Btab = data.frame(IDmatch=names(unlist(Z[["B"]])), hitind=unlist(Z[["B"]]))
#res = merge(res, Btab, by="IDmatch")

#Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.01, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"), mat=TRUE)

## Top plot
#bs = unlist(lapply(Z[["Z"]], function(x){medpolish(x)[["residuals"]]/mad(x)}))
#summary(bs)

#rs = unlist(lapply(Z[["Z"]], function(x){matrix(normR(as.vector(t(x)), 8, 10), 8, 10, byrow=TRUE)}))
#summary(rs)

#r = seq(-4, 23, 0.5)
#AUC = unlist(lapply(r, function(x){score(x, bs, unlist(Z[["B"]]))}))
#summary(AUC)
#btmax = r[which.max(AUC)]
#plot(r, AUC, type="l", xlab="Threshold", ylab="AUC", lwd=2, xaxt="n", col="red", ylim=c(0.5, 0.75))
#abline(v=btmax, col="red", lty=2)
#axis(1, at=c(-5, 5, 10, 15))
#axis(1, at=btmax)

#r = seq(-5, 28, 0.5)
#AUC = unlist(lapply(r, function(x){score(x, rs, unlist(Z[["B"]]))}))
#summary(AUC)
#rtmax = r[which.max(AUC)]
#lines(r, AUC, type="l", xlab="Threshold", ylab="AUC", lwd=2, xaxt="n", col="green")
#axis(1, at=c(-5, 5, 10, 15))
#axis(1, at=rtmax)
#legend("topright", legend=c("R-score", "B-score"), col=c("green", "red"), lty=c(1,1))

## Bottom plot
#rhitind = rep(0, length(rs))
#rhitind[rs>rtmax] = 1

#bhitind = rep(0, length(bs))
#bhitind[bs>btmax] = 1

#plot.roc(res[["hitind"]], res[["hatpai"]], col="blue")
#lines.roc(unlist(Z[["B"]]), bhitind, col="red")
#lines.roc(unlist(Z[["B"]]), rhitind, col="green")
#legend("bottomright", legend=c(paste("BHTS", " (AUC=", round(auc(res[["hitind"]], res[["hatpai"]]), 3), ")", sep=""), paste("R-score",
#" (AUC=", round(auc(unlist(Z[["B"]]), rhitind), 3), ")", sep=""), paste("B-score", " (AUC=", round(auc(unlist(Z[["B"]]), bhitind), 3),
#")", sep="")), col=c("blue", "green", "red"), lty=c(1,1,1))
```

10.3 Reproducing Figure 8

Parts of the code in this subsection take longer to execute and are therefore disabled. The user can copy, paste and run the code.

```
library(BHTSpack)
#library(pROC)

aucfunc = function(dat, B){
  Btab = data.frame(hitind=unlist(B))
  Btab = data.frame(IDmatch=rownames(Btab), Btab)

  Res = merge(dat, Btab, by="IDmatch")

  return(auc(Res[["hitind"]], Res[["hatpai"]]))
}


## Left plot
Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.1, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))

mu = mean(unlist(Z[["Z"]]))
mu00 = seq(mu, 0, -mu/25)
mu10 = seq(mu, 2*mu, mu/25)

#res = lapply(1:25, function(x){print(x); res=bhts(Z[["Z"]], iters=7000, H=10, K=10, mu00[x], mu10[x], a.alpha=10,
#b.alpha=5, a.tau=10, b.tau=5, s=1234); return(res);})

#hatpai = lapply(res, function(x){unlist(x[["hatpai"]])})
#hatpai = lapply(hatpai, function(x){data.frame(IDmatch=names(x), hatpai=x)})
#AUC = unlist(lapply(hatpai, aucfunc, Z[["B"]]))

#plot((mu10-mu00)[1:25], AUC, pch=16, xlab=expression(paste(mu[1][0]-mu[0][0])), cex=1.5, cex.lab=1.5, ylim=c(0.8, 0.9))
#abline(v=mu, col="red", lty=2, lwd=2)


## Middle plot
Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.05, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))
```

```
mu = mean(unlist(Z[["Z"]]))
mu00 = seq(mu, 0, -mu/25)
mu10 = seq(mu, 2*mu, mu/25)

#res = lapply(1:25, function(x){print(x); res=bhts(Z[["Z"]], iters=7000, H=10, K=10, mu00[x], mu10[x], a.alpha=10,
#b.alpha=5, a.tau=10, b.tau=5, s=1234); return(res);})

#hatpai = lapply(res, function(x){unlist(x[["hatpai"]])})
#hatpai = lapply(hatpai, function(x){data.frame(IDmatch=names(x), hatpai=x)})
#AUC = unlist(lapply(hatpai, aucfunc, Z[["B"]]))

#plot((mu10-mu00)[1:25], AUC, pch=16, xlab=expression(paste(mu[1][0]-mu[0][0])), cex=1.5, cex.lab=1.5, ylim=c(0.8, 0.9))
#abline(v=mu, col="red", lty=2, lwd=2)


## Right plot
Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.01, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))

mu = mean(unlist(Z[["Z"]]))
mu00 = seq(mu, 0, -mu/25)
mu10 = seq(mu, 2*mu, mu/25)

#res = lapply(1:25, function(x){print(x); res=bhts(Z[["Z"]], iters=7000, H=10, K=10, mu00[x], mu10[x], a.alpha=10,
#b.alpha=5, a.tau=10, b.tau=5, s=1234); return(res);})

#hatpai = lapply(res, function(x){unlist(x[["hatpai"]])})
#hatpai = lapply(hatpai, function(x){data.frame(IDmatch=names(x), hatpai=x)})
#AUC = unlist(lapply(hatpai, aucfunc, Z[["B"]]))

#plot((mu10-mu00)[1:25], AUC, pch=16, xlab=expression(paste(mu[1][0]-mu[0][0])), cex=1.5, cex.lab=1.5, ylim=c(0.8, 0.9))
#abline(v=mu, col="red", lty=2, lwd=2)

## Right plot
Z = data.create(N=80, nr=8, nc=10, M=1000, p=0.01, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))

mu = mean(unlist(Z[["Z"]]))
mu00 = seq(mu, 0, -mu/25)
mu10 = seq(mu, 2*mu, mu/25)

#res = lapply(1:25, function(x){print(x); res=bhts(Z[["Z"]], iters=7000, H=10, K=10, mu00[x], mu10[x], a.alpha=10,
#b.alpha=5, a.tau=10, b.tau=5, s=1234); return(res);})

#hatpai = lapply(res, function(x){unlist(x[["hatpai"]])})
#hatpai = lapply(hatpai, function(x){data.frame(IDmatch=names(x), hatpai=x)})
#AUC = unlist(lapply(hatpai, aucfunc, Z[["B"]]))

#plot((mu10-mu00)[1:25], AUC, pch=16, xlab=expression(paste(mu[1][0]-mu[0][0])), cex=1.5, cex.lab=1.5, ylim=c(0.8, 0.9))
#abline(v=mu, col="red", lty=2, lwd=2)
```

10.4 Reproducing Figure 9

The code in this subsection takes longer to execute and is therefore disabled. The user can copy, paste and run the code.

```
#library(BHTSpack)
#library(pROC)
#library(sights)

#score = function(t, sdat, B){
#  res = unlist(lapply(sdat, as.vector))
#  ind = rep(0, length(res))
#  ind[res>t] = 1

#  a = auc(B, ind)
#  return(a)
#}

#Z = data.create(N=80, nr=8, nc=10, M=5000, p=0.00021, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"))
#system.time(b.est <- bhts(Z[["Z"]], iters=7000, H=10, K=10, a.alpha=10, b.alpha=5, a.tau=10, b.tau=5, s=1234, store=TRUE))
#hatpai = unlist(b.est[["hatpai"]])
#res = data.frame(IDmatch=names(hatpai), hatpai)
#Btab = data.frame(IDmatch=names(unlist(Z[["B"]])), hitind=unlist(Z[["B"]]))
#res = merge(res, Btab, by="IDmatch")

#Z = data.create(N=80, nr=8, nc=10, M=5000, p=0.00021, s=1234, covrow=read.csv("covrow.csv"), covcol=read.csv("covcol.csv"), mat=TRUE)

## Top plot
#bs = unlist(lapply(Z[["Z"]], function(x){medpolish(x)[["residuals"]]/mad(x)}))
#summary(bs)

#rs = unlist(lapply(Z[["Z"]], function(x){matrix(normR(as.vector(t(x)), 8, 10), 8, 10, byrow=TRUE)}))
#summary(rs)

#r = seq(-4, 30, 0.5)
#AUC = unlist(lapply(r, function(x){score(x, bs, unlist(Z[["B"]]))}))
```

```
#summary(AUC)
#btmax = r[which.max(AUC)]

#r = seq(-5, 29, 0.5)
#AUC = unlist(lapply(r, function(x){score(x, rs, unlist(Z[["B"]]))}))
#summary(AUC)
#rtmax = r[which.max(AUC)]

#rhitind = rep(0, length(rs))
#rhitind[rs>rtmax] = 1

#bhitind = rep(0, length(bs))
#bhitind[bs>btmax] = 1

#plot.roc(res[["hitind"]], res[["hatpai"]], col="blue")
#lines.roc(unlist(Z[["B"]]), bhitind, col="red")
#lines.roc(unlist(Z[["B"]]), rhitind, col="green")
#legend("bottomright", legend=c(paste("BHTS", " (AUC=", round(auc(res[["hitind"]], res[["hatpai"]]), 3), ")", sep=""), paste("R-score",
#" (AUC=", round(auc(unlist(Z[["B"]]), rhitind), 3), ")", sep=""), paste("B-score", " (AUC=", round(auc(unlist(Z[["B"]]), bhitind), 3),
#")", sep="")), col=c("blue", "green", "red"), lty=c(1,1,1))
```

10.5 Reproducing Figure 10

The code in this subsection takes longer to execute and is therefore disabled. The user can copy, paste and run the code.

```
#library(BHTSpack)
#library(pROC)
#library(sights)
#library(gdata)

#score = function(t, s, B){
#  ind = rep(0, length(s))
#  ind[s>t] = 1

#  a = auc(B, ind)
#  return(a)
#}

## It is assumed that data files are in a folder "temp"
## read data
#dat = read.csv("temp/EColiFilamentation2006_screeningdata.csv", sep="\t")
#dim(dat)

## read hit indicators
#hits = read.csv("temp/CompoundSearchResults.csv", sep=",")
#dim(hits)
#hits = data.frame(hits, hits=rep(1,nrow(hits)))
#hits = data.frame(ChembankId=hits[["ChemBank.Id"]], hitind=rep(1,nrow(hits)))

## merge with hit indicator
#dat = merge(dat, hits, by="ChembankId", all.x=TRUE)
#dim(dat)
#dat[["hitind"]][is.na(dat[["hitind"]])] = 0

## merge with map
#map = read.xls("map.xlsx")
#dat = merge(dat, map, by="AssayName")

## Organism DRC39 at 24h
#dat = subset(dat, Organism=="DRC39" & ExpTime=="24h")

#plates = unique(as.character(dat[["Plate"]]))
#unique(as.character(dat[["WellType"]]))
#dat = subset(dat, WellType=="compound-treatment")

#dat = lapply(plates, function(x){d=subset(dat, Plate==x)})
#names(dat) = plates
#l = unlist(lapply(dat, nrow))
#table(l)

## include only 352-well plates
#dat = dat[l==352]

#unique(as.character(unlist(lapply(dat, function(x){x£AssayName}))))
#sum(is.na(unlist(lapply(dat, function(x){x£RawValueA}))))
#sum(unlist(lapply(dat, function(x){x£hitind})))
#sum(!is.na(unlist(lapply(dat, function(x){x£RawValueA}))))

## sorting wells row-wise
#dat = lapply(dat, function(x){ix=sort.int(as.character(x[["Well"]]), index.return=TRUE)[["ix"]]; return(x[ix,]);})

## extracting raw values, hit indicators and well names
#Z = lapply(dat, function(x){x[["RawValueA"]]})
#B = lapply(dat, function(x){x[["hitind"]]})
#W = lapply(dat, function(x){x[["Well"]]})
```

```
## constructing plates of raw values, row-wise
#Z = lapply(Z, function(x){matrix(x, 16, 22, byrow=TRUE)})

## naming rows and columns of plates
#Z = lapply(Z, function(x){rownames(x)=LETTERS[1:16]; colnames(x)=formatC(seq(1,22),flag=0,digits=1); return(x);})

## constructing plates of indicator variables (row-wise) and vectorizing (column-wise) each plate
#B = lapply(B, function(x){as.vector(matrix(x, 16, 22, byrow=TRUE))})

## constructing plates of well names (row-wise) and vectorizing (column-wise) each plate
#W = lapply(W, function(x){as.vector(matrix(x, 16, 22, byrow=TRUE))})

## Left plot
#plot(density(unlist(Z)[unlist(B)==0]), col="blue", ylab="Density", main="", xlim=range(unlist(Z)), xlab="Raw Value")
#lines(density(unlist(Z)[unlist(B)==1]), col="red")
#legend("topright", legend=c("Non-Hits", "Hits"), col=c("blue", "red"), lty=c(1,1))

## normalizing plates of raw values
#Z = lapply(Z, function(x){(x-mean(x))/sd(x)})

## naming indicator variables
#bn = names(B)
#B = lapply(1:length(B), function(x){names(B[[x]])=W[[x]]; return(B[[x]]);})
#names(B) = bn

## construct object for B-score and R-score methods
#Zmat = list(Z=Z, B=B)

## construct object for BHTS method
## vectorizing (column-wise) each plate of raw values and naming them with well names
#zn = names(Z)
#Z = lapply(1:length(Z), function(x){d=as.vector(Z[[x]]); names(d)=W[[x]]; return(d);})
#names(Z) = zn
#Z = list(Z=Z, B=B)

## Run BHTS
#system.time(b.est <- bhts(Z[["Z"]], iters=7000, H=10, K=10, a.alpha=10, b.alpha=5, a.tau=10, b.tau=5, s=1234, store=TRUE))
#hatpai = unlist(b.est[["hatpai"]])
#res = data.frame(IDmatch=names(hatpai), hatpai)
#Btab = data.frame(IDmatch=names(unlist(Z[["B"]])), hitind=unlist(Z[["B"]]))
#res = merge(res, Btab, by="IDmatch")

## Run B-score
#bs = unlist(lapply(Zmat[["Z"]], function(x){medpolish(x)[["residuals"]]/mad(x)}))
#summary(bs)

## Middle plot
#r = seq(-31, 9, 0.5)
#AUC = unlist(lapply(r, function(x){score(x, bs, unlist(Zmat[["B"]]))}))
#summary(AUC)
#btmax = r[which.max(AUC)]
#plot(r, AUC, type="l", xlab="Threshold", ylab="AUC", lwd=2, xaxt="n", col="red", ylim=c(0.44, 0.56))
#abline(v=btmax, col="red", lty=2)
#axis(1)
#axis(1, at=btmax)

## Run R-score
#rs = unlist(lapply(Zmat[["Z"]], function(x){matrix(normR(as.vector(t(x)), 16, 22), 16, 22, byrow=TRUE)}))
#summary(rs)

#r = seq(-45, 29, 0.5)
#AUC = unlist(lapply(r, function(x){score(x, rs, unlist(Zmat[["B"]]))}))
#summary(AUC)
#rtmax = r[which.max(AUC)]
#lines(r, AUC, type="l", xlab="Threshold", ylab="AUC", lwd=2, xaxt="n", col="green")
#abline(v=rtmax, col="green", lty=2)
#axis(1, at=rtmax)
#legend("topright", legend=c("R-score", "B-score"), col=c("green", "red"), lty=c(1,1))

## Right plot
#rhitind = rep(0, length(rs))
#rhitind[rs>rtmax] = 1

#bhitind = rep(0, length(bs))
#bhitind[bs>btmax] = 1

#plot.roc(res[["hitind"]], res[["hatpai"]], col="blue")
#lines.roc(unlist(Zmat[["B"]]), bhitind, col="red")
#lines.roc(unlist(Zmat[["B"]]), rhitind, col="green")
#legend("bottomright", legend=c(paste("BHTS", " (AUC=", round(auc(res[["hitind"]], res[["hatpai"]]), 3), ")", sep=""), paste("R-score",
#" (AUC=", round(auc(unlist(Zmat[["B"]]), rhitind), 3), ")", sep=""), paste("B-score", " (AUC=", round(auc(unlist(Zmat[["B"]]), bhitind),
#3), ")", sep="")), col=c("blue", "green", "red"), lty=c(1,1,1))
```

11 Acknowledgement

This project was funded by the Division of Allergy, Immunology, and Transplantation, National Institute of Allergy and Infectious Diseases, National Institutes of Health, Department of Health and Human Services, under contract No. HHSN272201400054C entitled "Adjuvant Discovery For Vaccines Against West Nile Virus and Influenza", awarded to Duke University and lead by Drs. Herman Staats and Soman Abraham.

References

[1] P. Müller, G. Parmigiani, and K. Rice. FDR and Bayesian Multiple Comparisons Rules. In *Proc. Valencia / ISBA 8th World Meeting on Bayesian Statistics*, Benidorm (Alicante, Spain), June, 2006.

[2] I. D. Shterev, D. B. Dunson, C. Chan, and G. D. Sempowski. Bayesian Multi-Plate High-Throughput Screening of Compounds. *Scientific Reports*, 8(1):9551, 2018.