MF-2dFDR: A model free two dimensional false discovery rate control procedure for powerful confounder adjustment in omics association studies

Describing the arguments of tdfdr.np()

- X: an $n \times q$ matrix for the covariates, where n is the sample size. q may be 1 or >1
- Y: an $m \times n \times l$ array for the responses. m denotes the number of hypotheses, or the number of omics features. l denotes the dimension of the response for each sample. Currently only supporting $l = 1$
- Z: $n \times p$ matrix of confounders. p may be greater than 1.
- ngrid: The size of the grid over which the search for the cutoffs must be conducted. Default value is 50.
- level: level of significance, default value is 0.05
- method: The method employed for computing the statistics. Can be any of "RV", "HSIC", "MS", or "OTHER". If method is specified as "OTHER", please define a function for computing the statistics (the statistics must be returned by the form c(t1, t2)) and pass it as an argument in stat.fun
- response: The type of data the response is. Currently supporting "continuous", "poisson", "nb", "binary". "nb" stands for negative binomial response.
- covariate: the type of data the covariate is. Currently supporting "continuous", "poisson", "nb", "binary", "multinomial." Currently multinomial covariate can only be used in conjunction with continuous response.
- link: Link function between Y and X, Z. Ignore if using any methods other than "MS". If method = "MS" this can be either of "log", "identity", "logit"
- etype: can be "FDR" or "FWER"
- stat.fun: user defined function if method = "OTHER". Should take an $X(n \times q)$, $Y(m \times l)$ and $Z(n \times p)$ and return the conditional(t1) and marginal(t2) statistics in the form c(t1, t2)

Additional note: ensure that you specify ncores = 1 in the tdfdr.np function if you are using a Windows operating system.

Scenario A: Linear/nonlinear models with continuous X and Z.

Here, we call the tdfdr.np() function for a toy example in Scenario A. Let us consider the DGP in 1, S3 of Supplement.

```r
delta = 0.1
m = 1000
n = 100
B = 50
dof = 3
rho = 0.5
l = 0.4
p = 0.1
r = runif(m)
alpha = if_else(r < (1- p), 0,
                 if_else(r < (1- p/2), runif(m, -l-delta, -l), runif(m, l,l+delta))
)
r = runif(m)
beta = if_else(r < (1- p), 0,
                if_else(r < (1- p/2), runif(m, -l-delta, -l), runif(m, l,l+delta))
)
tp = abs(alpha) > 0
tn = alpha == 0

## Generate X,Z


Z = rnorm(n)
X = rnorm(n, mean = rho*(Z ) )

##Generate Y

Y = matrix(nrow = m, ncol = n)
for(j in 1:m)
{
  Y[j,] =  alpha[j] * scale(X) +  beta[j]*Z + rnorm(n)

}

obj = tdfdr.np(X, Y, Z, method = "MS", response = "continuous",
               covariate = "continuous")
```

```
## Computing Statistics...
## Searching for thresholds in each dimension...
## 2D Gridsearch...
## Done!
```

```r
pos = obj$rejections
fdr = sum(tn * pos)/max(1, sum(pos))
pow = sum(tp * pos)/sum(tp)
print(paste("FDR = ", fdr, "; power = ", pow))
```

```
## [1] "FDR =  0.038961038961039 ; power =  0.795698924731183"
```

Scenario B: Linear/Nonlinear models with discrete X and continuous Z

We apply this for the DGP 5 as described in the Supplement.

```r
delta = 0.2
m = 1000
n = 100
B = 50
dof = 5
rho = 1
l = 0.4
p = 0.2
r = runif(m)
    alpha = if_else(r < (1- p), 0,
                     if_else(r < (1- p/2), runif(m, -l-delta, -l), runif(m, l,l+delta))
    )
    r = runif(m)
    beta = if_else(r < (1- p), 0,
                    if_else(r < (1- p/2), runif(m, -l-delta, -l), runif(m, l,l+delta))
    )
    tp = abs(alpha) > 0
    tn = alpha == 0

    ## Generate X,Z


    Z = rnorm(n)
    prob = exp(rho * Z)/(1 + exp(rho * Z))
    X = rbinom(n, 1, prob)

    ##Generate Y

    Y = matrix(nrow = m, ncol = n)
    for(j in 1:m)
    {
      Y[j,] = alpha[j]*exp(as.numeric(X)) + beta[j]*Z + rnorm(n)

    }

obj = tdfdr.np(as.factor(X), Y, Z, method = "MS", response = "continuous",
               covariate = "binary", link = "identity")
```

```
## Computing Statistics...
## Searching for thresholds in each dimension...
## 2D Gridsearch...
## Done!
```

```r
pos = obj$rejections
fdr = sum(tn * pos)/max(1, sum(pos))
pow = sum(tp * pos)/sum(tp)
 print(paste("FDR = ", fdr, "; power = ", pow))
```

```
## [1] "FDR =  0.0469798657718121 ; power =  0.743455497382199"
```

Scenario C: Linear models with discrete X and Z
We apply this for DGP 8 in Section 3 of the supplement.

```
delta = 0.2
m = 1000
n = 100
B = 50
dof = 5
rho = 2
l = 0.4
p = 0.2
r = runif(m)
    alpha = if_else(r < (1- p), 0,
                     if_else(r < (1- p/2), runif(m, -l-delta, -l), runif(m, l,l+delta))
    )
    r = runif(m)
    beta = if_else(r < (1- p), 0,
                    if_else(r < (1- p/2), runif(m, -l-delta, -l), runif(m, l,l+delta))
    )
    tp = abs(alpha) > 0
    tn = alpha == 0
    Z = rbinom(n, 1, prob = 0.7)
    prob = exp(rho * Z)/(1 + exp(rho * Z))
    X = rbinom(n, 1, prob)

    ##Generate Y

    Y = matrix(nrow = m, ncol = n)
    for(j in 1:m)
    {
      Y[j,] = alpha[j]*exp(as.numeric(X)) + beta[j]*as.numeric(Z) + rnorm(n)

    }

obj = tdfdr.np(as.factor(X), Y, as.factor(Z), method = "RV", response = "continuous",
               covariate = "binary", link = "identity")
```

```
## Computing Statistics...
## Searching for thresholds in each dimension...
## 2D Gridsearch...
## Done!
```

```
pos = obj$rejections
fdr = sum(tn * pos)/max(1, sum(pos))
pow = sum(tp * pos)/sum(tp)
print(paste("FDR = ", fdr, "; power = ", pow))
```

```
## [1] "FDR =  0.0691489361702128 ; power =  0.870646766169154"
```

Scenario D/E: Next we have shown the performance of the method for a poisson response. The generating function would look as follows:

We apply this to DGP 10 described in Section 3 of the Supplementary material.

```r
delta = 0.2
m = 1000
n = 100
B = 50
dof = 1
rho = 0.7
l = 0.35
p = 0.2
r = runif(m)
    alpha = if_else(r < (1- p), 0,
                     if_else(r < (1- p/2), runif(m, -l-delta, -l), runif(m, l,l+delta))
    )
    r = runif(m)
    beta = if_else(r < (1- p), 0,
                    if_else(r < (1- p/2), runif(m, -l-delta, -l), runif(m, l,l+delta))
    )
    tp = abs(alpha) > 0
    tn = alpha == 0

    ## Generate X,Z


    Z = rnorm(n)
    X = rnorm(n, rho*Z, 1)
    Y = matrix(nrow = m, ncol = n)
    for(j in 1:m)
    {
      #prob = sapply(alpha[j]* scale(X)+ beta[j]*Z+ rnorm(n), logit)
      Y[j,] = rpois(n, lambda = exp(1 + alpha[j]* scale(X)+ beta[j]*Z))
    }


obj = tdfdr.np(X, Y, Z, ngrid = 50, method = "MS", response = "poisson",
               covariate = "continuous", link = "log")
```

```
## Computing Statistics...
## Searching for thresholds in each dimension...
## 2D Gridsearch...
## Done!
```

```r
pos = obj$rejections
fdr = sum(tn * pos)/max(1, sum(pos))
pow = sum(tp * pos)/sum(tp)
print(paste("FDR = ", fdr, "; power = ", pow))
```

```
## [1] "FDR =  0.00869565217391304 ; power =  0.995633187772926"
```

Scenario- Smoking Data: Next we demonstrate the application of tdfdr.np on the smoking dataset. For the purpose of this demonstration we have taken the case with the continuous response.

```r
X = throat.meta$SmokingStatus
Z = throat.meta$Sex

##Filter out the OTUs with a large number of zeros
index_mat = rowSums(throat.otu.tab == 0) < 54
Y_new = throat.otu.tab[index_mat,]
Y = as.matrix(clr(Y_new + 0.5))
require(sgof)
```

```
## Loading required package: sgof

## Loading required package: poibin
```

```r
obj = tdfdr.np(as.factor(X), Y, Z,ngrid = 500,level = 0.08, B = 500, method = "RV",
               response = "continuous", covariate = "binary", link = "logit")
```

```
## Computing Statistics...
## Searching for thresholds in each dimension...
## 2D Gridsearch...
## Done!
```

```r
print(paste("There were ", sum(obj$rejections), "rejections"))
```

```
## [1] "There were  8 rejections"
```

In conclusion, the tdfdr.np function can be applied to a wide array of data types, and for each data type, it is possible to come up with a function stat.fun that captures the conditional and marginal independence respectively. The available methods consider a wide array of data types. The functions generate.discrete() and generate.cont() which has been used in the above demonstrations relies on residual permutation, and may not be optimal if the underlying assumptions of a linear model fit of X on Z, (eg homoscedasticity, uncorrelated errors) is not satisfied.

Additionally, we provide simulations for DGPs where X and Z is multivariate but Y is not. In the following toy data, Y is binary. Note that the case where X is multivariate and Y is Negative Binomial is still not supported by this method.

```r
logit = function(x) exp(x)/ (1 + exp(x))
delta = 0.2
m = 1000
n = 100
B = 50
dof = 5

l = 0.4
p = 0.2
r = runif(m)
    alpha = if_else(r < (1- p), 0,
                    if_else(r < (1- p/2), runif(m, -l-delta, -l), runif(m, l,l+delta))
    )
    r = runif(m)
    beta = if_else(r < (1- p), 0,
                    if_else(r < (1- p/2), runif(m, -l-delta, -l), runif(m, l,l+delta))
    )
    tp = abs(alpha) > 0
    tn = alpha == 0

    ## Generate X,Z


    Z = matrix(rnorm(n*3), nrow = n)
    rho = 0.5
    X = matrix(rnorm(n*3, rho*Z), nrow = n)
    Y = matrix(nrow = m, ncol = n)
    for(j in 1:m)
    {
      #rpois(n, lambda = exp(1 + alpha[j]*X_dummy%*%c(2,1,1) + beta[j]*(Z%*%c(1,0.5,0.2))))

      Y[j,] = rbinom(n, 1, prob = logit(alpha[j]*(X%*%c(2,1,1)) + beta[j]*(Z%*%c(1,0.5,0.2))))

    }

obj = tdfdr.np(X, Y, Z, method = "MS", response = "binary",
               covariate = "continuous", link = "logit")
```

```
## Computing Statistics...
## Searching for thresholds in each dimension...
## 2D Gridsearch...
## Done!
```

```r
pos = obj$rejections
fdr = sum(tn * pos)/max(1, sum(pos))
pow = sum(tp * pos)/sum(tp)
 print(paste("FDR = ", fdr, "; power = ", pow))
```

```
## [1] "FDR =  0.0208333333333333 ; power =  0.974093264248705"
```