@MPRA package vignette

Dandi Qiao

2019-01-30

Contents

1 Introduction

The analysis toolset for MPRA data (@MPRA) includes functions for simulating and analyzing MPRA data, and for power calculations of MPRA experiments. This tutorial briefly introduces the functions provided by the @MPRA package, using the example data included in the package.

We can load the library using:

```
library(atMPRA)
```

```
## Warning: package 'DESeq2' was built under R version 3.5.2
```

```
## Warning: package 'BiocParallel' was built under R version 3.5.2
```

We can do a quick power calculation:

```
nsim = 10
ntag = 10

result = getPower(nsim = nsim, ntag = ntag, nrepIn = 3, nrepOut = 3,
    slope = c(rep(1, ntag * nsim), rep(2, ntag * nsim)), method = c("MW",
        "mpra_lm"), scenario = "fixTotalDepth")
```

```
## Warning in getPower(nsim = nsim, ntag = ntag, nrepIn = 3, nrepOut = 3, slope = c(rep(1, : The input
```

```
## [1] 0.7
```

```
result$Power
```

```
## [1] 0.7
```

2 Data available in the package

The estimated distributional parameters of the MPRA data (GSE70531 in GEO database) was obtained using the `estimateMPRA` function in this package. The basic parameters include \ `inputProp`: The proportion of counts per tag among all tags in the library\ `transEff`: The distribution of transfection efficiencies (normalized RNA/DNA ratio) across tags\ `dispFunc_input`: The dispersion function of the input tag counts across replicates as a function of the mean \ `dispFunc_output`: The dispersion function of the output tag counts across replicates as a function of the mean\ \

We assume that the tag counts across replicates follow a Negative Binomial distribution with mean μ and dispersion σ^2. Then the variance is $\mu + \sigma^2\mu^2$.

However, it is observed that the dispersion parameter is not constant in RNA-Seq data. In DESeq2, it was assumed that:

$$log(\sigma^2) \sim N(log(a + b/\mu), \sigma_d^2)$$

We will use the dispersion function estimated by DESeq2 to generate count data here.

The estimation parameters for GSE70531 was done using the `estimateMPRA` function which used DESeq2 package. This distribution will be the default distribution for simulating MPRA data in this package if not specified otherwise. The data is loaded with the package automatically.

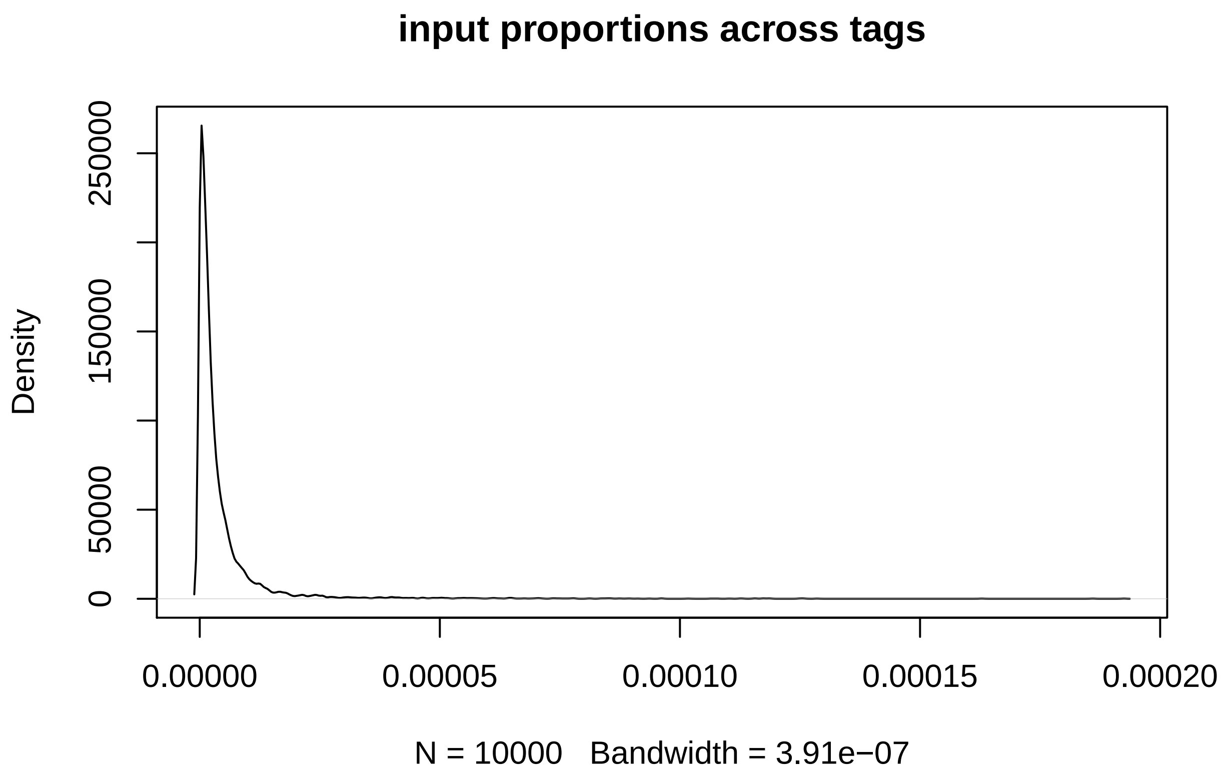
```
GSE70531_params
```

```
## $dispFunc_input
## function (means)
## exp(predict(fit, data.frame(logMeans = log(means))))
## <bytecode: 0x7fc6ddb796b8>
## <environment: 0x7fc6f0a17040>
## attr(,"fitType")
## [1] "local"
## attr(,"varLogDispEsts")
## [1] 0.4812829
## attr(,"dispPriorVar")
## [1] 0.25
##
## $dispFunc_output
## function (q)
## coefs[1] + coefs[2]/q
## <bytecode: 0x7fc6dd66b828>
## <environment: 0x7fc6ddb7a090>
## attr(,"coefficients")
## asymptDisp  extraPois
##  0.5183263 12.7233435
## attr(,"fitType")
## [1] "parametric"
## attr(,"varLogDispEsts")
## [1] 0.8172103
## attr(,"dispPriorVar")
## [1] 0.3268525
##
## $inputProp
## function (v)
## .approxfun(x, y, v, method, yleft, yright, f)
## <bytecode: 0x7fc6dd661eb0>
## <environment: 0x7fc6dd66c190>
##
## $transEff
## function (v)
## .approxfun(x, y, v, method, yleft, yright, f)
## <bytecode: 0x7fc6dd65f878>
## <environment: 0x7fc6dd662968>
```

```
##
## $sizeFactor_input
## K562_minP_DNA1 K562_minP_DNA2
##      1.2060454      0.8291562
##
## $sizeFactor_output
## K562_CTRL_minP_RNA1 K562_CTRL_minP_RNA2 K562_CTRL_minP_RNA3
##           0.6613516           0.4404059           1.4904598
## K562_CTRL_minP_RNA4 K562_CTRL_minP_RNA5 K562_CTRL_minP_RNA6
##           1.2337356           1.8488591           1.3372992
```

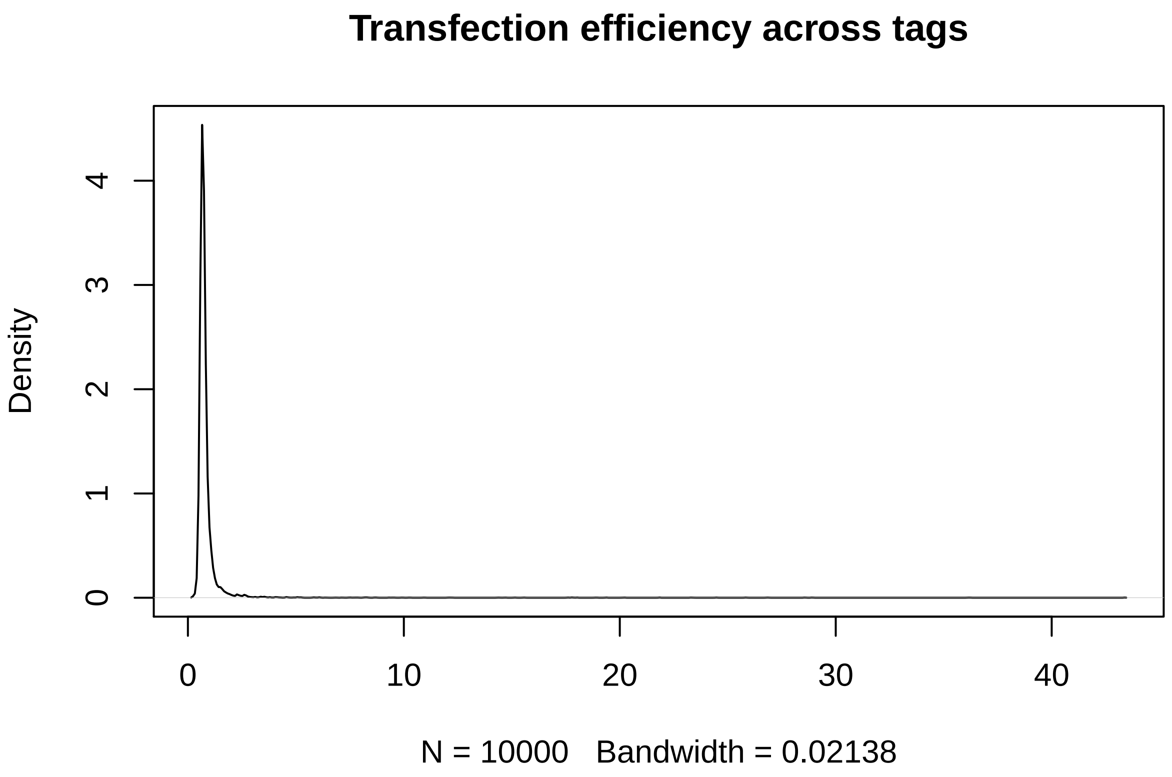
The input proportions are usually very skewed due to cloning and PCR.

```
plot(density(GSE70531_params[[3]](runif(10000))), main="input proportions across tags")
```



The transfection efficiency distribution for GSE70531 looks like this:

```
plot(density(GSE70531_params[[4]](runif(10000))), main="Transfection efficiency across tags")
```



3 Functions

3.1 Simulating MPRA data

There are multiple ways to simulate MPRA data in this package:

1. Simulating MPRA data using default distribution.
2. Simulating MPRA data using estimated distributions from observed data
3. Simulating MPRA data by specifying parameters in the model.

The parameters for simulating a MPRA dataset includes:

1. number of SNPs in the data (`nsim`)
2. number of tags per SNP (`ntag`)
3. number of replicates in the input and output (`nrepIn`, `nrepOut`)
4. RNA/DNA ratio for all tags (`slope`)
5. Total depth for one replicate (`fixTotalD`) or mean depth per tag (`fixMeanD`)

3.1.1 Simulating MPRA data using estimated distributions from observed data

We have simulated the MPRA using defulat settings above. Now we want to demonstrate how to simulate MPRA using estimated distribution from observed data. Here we will use the parameters we estimated for GSE70531.

```r
totalDepth = 2e+05

ntag = 10

nsim = 10

nrepIn = 5

nrepOut = 5

inputProp = GSE70531_params[[3]](runif(ntag * nsim * 2))

slopel = GSE70531_params[[4]](runif(nsim * 2))

inputDispFunc = GSE70531_params[[1]]

outputDispFunc = GSE70531_params[[2]]

slope = rep(slopel, each = ntag)

datt = sim_fixDepth(inputProp, ntag, nsim, nrepIn, nrepOut, slope,
    inputDispFunc = inputDispFunc, outputDispFunc = outputDispFunc,
    sampleDepth = totalDepth)

datt[1:10, ]
```

```
##    allele simN input_rep1 input_rep2 input_rep3 input_rep4 input_rep5
## 1     Ref    1        123        146        103        116        110
## 2     Ref    1        188        148        152        118        139
## 3     Ref    1       1138       1171       1094        976        915
## 4     Ref    1        703        712        706        694        625
## 5     Ref    1       1004        967        895        994        869
## 6     Ref    1        394        358        385        382        324
## 7     Ref    1       1908       2086       2037       1861       2040
## 8     Ref    1        157        114        112        117        130
## 9     Ref    1          9         28          4          1         23
## 10    Ref    1         27         10         20         13         20
##    output_rep1 output_rep2 output_rep3 output_rep4 output_rep5
## 1           18          43         119          49          99
## 2           21          40          95         160         144
## 3          481        1152         176           8         812
## 4          852         403          75         242         596
## 5          691         988         520         412         931
## 6          492         150         184         102         391
## 7         1377        1709         436         337         772
## 8           54          71          63         115           0
## 9            4           1           1           0           4
## 10          11           6           6           4          28
```

If we would like to simulate data based on an observed MPRA dataset, we can estimate the parameters using the function `estimateMPRA`.

```r
rnaCol=8

new_params=estimateMPRA(datt, nrepIn, rnaCol, nrepOut, nsim, ntag)
```

```
new_params
```

```
## $dispFunc_input
## function (q)
## coefs[1] + coefs[2]/q
## <bytecode: 0x7fc6ef2e8200>
## <environment: 0x7fc6ef2e6270>
## attr(,"coefficients")
##   asymptDisp    extraPois
## 0.0006601409 3.2067962718
## attr(,"fitType")
## [1] "parametric"
## attr(,"varLogDispEsts")
## [1] 0.7241005
## attr(,"dispPriorVar")
## [1] 0.25
##
## $dispFunc_output
## function (q)
## coefs[1] + coefs[2]/q
## <bytecode: 0x7fc6ef2e8200>
## <environment: 0x7fc6f0169c28>
## attr(,"coefficients")
## asymptDisp  extraPois
##  0.6141227  6.7553871
## attr(,"fitType")
## [1] "parametric"
## attr(,"varLogDispEsts")
## [1] 0.932977
## attr(,"dispPriorVar")
## [1] 0.2880429
##
## $inputProp
## function (v)
## .approxfun(x, y, v, method, yleft, yright, f)
## <bytecode: 0x7fc6de7d8788>
## <environment: 0x7fc6fae2fd30>
##
## $transEff
## function (v)
## .approxfun(x, y, v, method, yleft, yright, f)
## <bytecode: 0x7fc6de7d8788>
## <environment: 0x7fc6df121038>
##
## $sizeFactor_input
## input_rep1 input_rep2 input_rep3 input_rep4 input_rep5
##  1.0123682  1.0061564  0.9984757  1.0044977  0.9929253
##
## $sizeFactor_output
## output_rep1 output_rep2 output_rep3 output_rep4 output_rep5
##   1.0591985   1.2020193   1.0237531   0.9293456   1.0822782
```

Then we can simulate new MPRA data using these parameters:

```
datt = sim_fixDepth(inputProp = new_params[[3]](runif(ntag * nsim * 2)),
    ntag, nsim, nrepIn, nrepOut, slope, inputDispFunc = new_params[[1]],
    outputDispFunc = new_params[[2]], sampleDepth = totalDepth)

datt[1:10, ]
```

```
##    allele simN input_rep1 input_rep2 input_rep3 input_rep4 input_rep5
## 1     Ref    1       1265       1406       1288       1435       1075
## 2     Ref    1       2685       2893       3147       3110       3029
## 3     Ref    1         94        107         91        115         69
## 4     Ref    1       2525       2772       2803       2764       3059
## 5     Ref    1        207        219        187        201        208
## 6     Ref    1         39         33         20         16         12
## 7     Ref    1        170        156        219        222        179
## 8     Ref    1          2          1          1         17          0
## 9     Ref    1         16         19         21         13         15
## 10    Ref    1        278        283        247        226        266
##    output_rep1 output_rep2 output_rep3 output_rep4 output_rep5
## 1          377         296         274         465          11
## 2         2773        4755        1439         409          18
## 3           60         134          44         160          54
## 4         5703         332        1448        2706         680
## 5            0          77          76          20          58
## 6            5           1           1          14           6
## 7           94          56          39         264          96
## 8            4           0           7           1           0
## 9           30          10           9          74           6
## 10         106         139         187         214         124
```

3.1.2 Simulating MPRA data by specifying parameters in the model.

We can also simulate MPRA data by specifying parameters. For example, we may want to specify different mean input counts across tags for allele A and allele B. We can check if methods are biased by the allelic imbalance in the input distribution later using this simulated data.

```
inputDist = GSE70531_params[[3]](runif(nsim * ntag * 2))

datt = sim_fixInputMean(mean_A = 10, mean_B = 100, ntag = ntag, nsim = nsim,
    nrepIn = nrepIn, nrepOut = nrepOut, slope = slope, inputDist = inputDist,
    inputDispFunc = inputDispFunc, outputDispFunc = outputDispFunc)
```

```
## converting counts to integer mode
## converting counts to integer mode
```

```
datt[c(1:5, 101:105), ]
```

```
##     allele simN input_rep1 input_rep2 input_rep3 input_rep4 input_rep5
## 1      Ref    1          5          9          6         15          8
## 2      Ref    1          7         11          9          1          8
## 3      Ref    1         16          6         11         27          7
## 4      Ref    1          2          0         13          7          5
## 5      Ref    1          5          2         23          4          3
## 101    Mut    1         94         80         68         62         42
## 102    Mut    1          7          6         18          7         12
## 103    Mut    1         62        122         89         95         82
## 104    Mut    1          23         32         24         37         17
## 105    Mut    1         111         91        121         91         89
##     output_rep1 output_rep2 output_rep3 output_rep4 output_rep5
## 1             6           8           4           5           1
## 2             0           9          12           1           5
## 3             5          18           4           5          15
## 4             6           0           0           6           0
## 5             0           2           1          16           0
## 101          16          64          69           5           0
## 102           0           2           3           2          10
## 103          22          12          45          45          26
## 104          34          28          10          16          18
## 105         474         154         168          24           6
```

Note the distribution of counts are very different between the two alleles `Ref` and `Mut`.

Another way to specify the dispersion function is through `inputDispParam` and `outputDispParam`. These parameters are required if `inputDispFunc` and `outputDispFunc` are not provided. Each of them should give the three parameter estimates (a, b, σ_d^2) for the dispersion function of the DNA input or RNA output counts across replicates. The three parameters correspond to a, b, and σ_d^2, which specify that the dispersion parameter is a lognormal distribution with mean $log(a + b/\mu)$ and sd σ_d^2, where μ is the mean of RNA count across the replicates.

```
datt = sim_fixDepth(inputProp = new_params[[3]](runif(ntag * nsim * 2)),
    ntag, nsim, nrepIn, nrepOut, slope, inputDispParam = c(0, 4.37, 0.25),
    outputDispParam = c(0.54, 12, 0.25), sampleDepth = totalDepth)
### Thesea are default dispersion parameters, if none was specified.
datt[1:10, ]
```

```
##    allele simN input_rep1 input_rep2 input_rep3 input_rep4 input_rep5
## 1     Ref    1       4438       4412       4606       4693       4440
## 2     Ref    1         11         38         45         23          6
## 3     Ref    1        448        589        477        454        499
## 4     Ref    1        241        222        230        218        240
## 5     Ref    1        393        419        461        459        345
## 6     Ref    1        604        503        443        393        611
## 7     Ref    1         65         63         77         64         62
## 8     Ref    1        195        274        222        221        240
## 9     Ref    1       2447       2712       2310       2407       2508
## 10    Ref    1        578        793        774        779        662
##    output_rep1 output_rep2 output_rep3 output_rep4 output_rep5
## 1         6329        1096        1683         997        2715
## 2            0           5           2           1          11
## 3          115         300         306         444          99
## 4           98         125          15         189         116
## 5          604         288          80         574         396
## 6          527         437          55         446         344
## 7           51          10          52          12          47
## 8           20         267         133          39         141
## 9         1685         211         657        2559        1005
## 10         495         355         646         491         450
```

3.2 Analyze MPRA data

We provide a list of methods to analyze MPRA data. The input MPRA data frame should have `nsim*ntag*2` rows and `2+nrepIn+nrepOut` columns. The first column should be named 'allele', and the second column should be named 'simN'. The 'allele' columns should contain only two possible values 'Ref' and 'Mut' to refer to the two versions of alleles for each SNP.

A list of the methods that are available is here:

Test	singleReplicate	OptionName
Mann-Whitney	YES	MW
Matching	YES	Matching
Adaptive	YES	Adaptive
QuASAR-MPRA	YES	QuASAR
Fisher's Exact Test	YES	Fisher
T-test	NO	T-test
mpralm using mean	NO	mpralm
mpralm using sum	NO	mpralm
edgeR	NO	edgeR
DESeq2	NO	DESeq2

To analyze a formmated MPRA data:

```
datt[1:10, ]
```

```
##    allele simN input_rep1 input_rep2 input_rep3 input_rep4 input_rep5
## 1     Ref    1       4438       4412       4606       4693       4440
## 2     Ref    1         11         38         45         23          6
## 3     Ref    1        448        589        477        454        499
## 4     Ref    1        241        222        230        218        240
## 5     Ref    1        393        419        461        459        345
## 6     Ref    1        604        503        443        393        611
## 7     Ref    1         65         63         77         64         62
## 8     Ref    1        195        274        222        221        240
## 9     Ref    1       2447       2712       2310       2407       2508
## 10    Ref    1        578        793        774        779        662
##    output_rep1 output_rep2 output_rep3 output_rep4 output_rep5
## 1         6329        1096        1683         997        2715
## 2            0           5           2           1          11
## 3          115         300         306         444          99
## 4           98         125          15         189         116
## 5          604         288          80         574         396
## 6          527         437          55         446         344
## 7           51          10          52          12          47
## 8           20         267         133          39         141
## 9         1685         211         657        2559        1005
## 10         495         355         646         491         450
```

```
results = analyzeMPRA(datt, nrepIn, rnaCol, nrepOut, nsim, ntag, method = c("MW",
    "Adaptive", "QuASAR", "T-test", "mpralm", "DESeq2"), cutoff = 0, cutoffo = 0)

results
```

```
##    simN   resInput       resMW resMatching resAdaptive     QuASAR
## 1     1 1.00000000 0.314999242      1.0000 0.314999242 0.98207509
## 2     2 0.79593626 0.011496244      0.0872 0.011496244 0.45243864
## 3     3 0.97051246 0.352681374      0.8640 0.352681374 0.07904815
## 4     4 0.07525601 0.217562623      0.2288 0.217562623 0.15410489
## 5     5 0.85342831 0.011496244      0.0224 0.011496244 0.05916230
## 6     6 0.48125095 0.105122432      0.1740 0.105122432 0.62916706
## 7     7 0.85342831 0.001504687      0.0852 0.001504687 0.10100448
## 8     8 0.10512243 0.008930698      0.1276 0.008930698 0.01502813
## 9     9 1.00000000 0.089209552      0.3056 0.089209552 0.98060268
## 10   10 0.07525601 0.217562623      0.7396 0.217562623 0.68886566
##    ttest_paired      ttest mpralm_mean mpralm_sum      DESeq2
## 1    0.44446861 0.37507595 0.269569916 0.37960752 0.286942191
## 2    0.22322698 0.29722178 0.211592819 0.18683574 0.248481378
## 3    0.35043976 0.21744865 0.086849846 0.24751171 0.212805687
## 4    0.03657604 0.03954354 0.310377141 0.06730874 0.023359987
## 5    0.07334237 0.07277812 0.028273498 0.06926101 0.005441125
## 6    0.72556730 0.66269258 0.065699894 0.79922746 0.709444129
## 7    0.06416291 0.05573769 0.005847421 0.07336168 0.007924411
## 8    0.01293264 0.02014110 0.287775912 0.08057229 0.015623126
## 9    0.69410256 0.67519587 0.400501158 0.96056083 0.618170723
## 10   0.58425273 0.45841039 0.102252972 0.32639199 0.483012719
```

You can remove tags with mean counts less than the cutoffs specified for the input and output.

3.3 Power calculation

We can compute power based on simulated MPRA data specified using the options described above. Addition parameters here include the correction method for multiple testing, and the significance level to be used.

```
nrepIn = 2
nrepOut = 2
slopel = GSE70531_params[[4]](runif(nsim))
slopel = c(slopel, slopel + 1)
slope = rep(slopel, each = ntag)
result2 = getPower(nsim, ntag, nrepIn, nrepOut, slope, scenario = "fixInputDist",
    method = c("MW", "T-test", "mpralm", "edgeR", "DESeq2"),
    fixInput = c(20, 100), inputDist = inputDist, inputDispFunc = inputDispFunc,
    outputDispFunc = outputDispFunc, cutoff = -1, cutoffo = -1)
```

```
##        resMW ttest_paired        ttest  mpralm_mean   mpralm_sum
##          0.5          0.0          0.0          0.4          0.0
##        edgeR       DESeq2
##          0.5          0.6
```

```
result2$Power
```

```
##        resMW ttest_paired        ttest  mpralm_mean   mpralm_sum
##          0.5          0.0          0.0          0.4          0.0
##        edgeR       DESeq2
##          0.5          0.6
```

```
result3 = getPower(nsim, ntag, nrepIn, nrepOut, slope = 1, scenario = "fixTotalDepth",
    method = c("MW", "Matching", "Adaptive", "Fisher", "QuASAR", "T-test",
        "mpralm", "edgeR", "DESeq2"), fixTotalD = 2e+05, inputDist = inputDist,
    inputDispFunc = inputDispFunc, outputDispFunc = outputDispFunc, cutoff = -1,
    cutoffo = -1)
```

```
##        resMW  resMatching  resAdaptive       QuASAR fisherPvalue
##          0.0          0.0          0.0          0.0          0.9
## ttest_paired        ttest  mpralm_mean   mpralm_sum        edgeR
##          0.0          0.0          0.0          0.0          0.0
##       DESeq2
##          0.1
```

```
result3$Power
```

```
##        resMW  resMatching  resAdaptive       QuASAR fisherPvalue
##          0.0          0.0          0.0          0.0          0.9
## ttest_paired        ttest  mpralm_mean   mpralm_sum        edgeR
##          0.0          0.0          0.0          0.0          0.0
##       DESeq2
##          0.1
```

3.4. Other methods included

`calEnrichment`: This function uses hypergeometric tests to compute enrichment in SNPs with significant allelic DNA imbalance among the SNPs with significant allele-specific effect defined by each method.

```
calEnrichment(results)
```

```
## Compute enrichment in allele-imbalanced SNPs among significant results...
```

```
##         method q m enrichP
## 1        resMW 0 4       1
## 2  resMatching 0 1       1
## 3  resAdaptive 0 4       1
## 4       QuASAR 0 1       1
## 5 ttest_paired 0 2       1
## 6        ttest 0 2       1
## 7  mpralm_mean 0 2       1
## 8   mpralm_sum 0 0       1
## 9       DESeq2 0 4       1
```