Lab 8: Multiple testing in R

This lab is about doing multiple testing in R, using the package multtest. We will follow the manuals found on the Bioconductor site.

The questions on the OHMS page are in the middle of this webpage marked as **Question X**. Make sure you answer them on the OHMS page.

Getting started

Load the package with

```
require(multtest)
```

We'll illustrate some of the functionality of multtest with gene expression data from the leukemia ALL/AML study of Golub et al. (1999). Load the leukemia dataset:

```
data(golub)
class(golub)
```

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

```
dim(golub)
```

```
## [1] 3051   38
```

```
head(golub)
```

```
##         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[,7]    [,8]
## [1,] -1.4577 -1.39420 -1.42779 -1.40715 -1.42668 -1.21719
-1.3739 -1.3683
## [2,] -0.7516 -1.26278 -0.09052 -0.99596 -1.24245 -0.69242
-1.3739 -0.5080
## [3,]  0.4570 -0.09654  0.90325 -0.07194  0.03232  0.09713
-0.1198  0.2338
## [4,]  3.1353  0.21415  2.08754  2.23467  0.93811  2.24089
3.3658  1.9786
## [5,]  2.7657 -1.27045  1.60433  1.53182  1.63728  1.85697
3.0185  1.1285
## [6,]  2.6434  1.01416  1.70477  1.63845 -0.36075  1.73451
3.3658  0.9687
##         [,9]   [,10]   [,11]   [,12]   [,13]   [,14]  [,15]
[,16]   [,17]
## [1,] -1.4765 -1.2158 -1.2814 -1.0321 -1.3615 -1.3998 0.1763
-1.401 -1.5678
## [2,] -1.0453 -0.8126 -1.2814 -1.0321 -0.7400 -0.8316 0.4120
-1.277 -0.7437
## [3,]  0.2399  0.4420 -0.3956 -0.6253  0.4518  1.0952 1.0932
0.343  0.2001
## [4,]  2.6647 -1.2158  0.5911  3.2605 -1.3615  0.6418 2.3262
-1.401 -1.5678
## [5,]  2.1702 -1.2158 -1.1013  2.5998 -1.3615  0.2285 2.3449
-1.401 -1.5678
## [6,]  2.7237 -1.2158  1.2019  2.8342 -1.3615  1.3274 1.5246
-1.401 -1.5678
##        [,18]    [,19]   [,20]   [,21]   [,22]   [,23]
[,24]   [,25]
## [1,] -1.2047 -1.24482 -1.6077 -1.0622 -1.1266 -1.2096
-1.4833 -1.2527
## [2,] -1.2047 -1.02380 -0.3878 -1.0622 -1.1266 -1.2096
-1.1219 -0.6526
## [3,]  0.3899  0.00641  1.1093  0.2195 -0.7227  0.5169
0.2858  0.6194
## [4,]  0.8350 -1.24482 -1.6077 -1.0622  3.6944  3.7084
-1.4833  2.3670
## [5,]  0.9453 -1.24482 -1.6077 -1.0622  3.5246  3.7084
-1.4833  1.7917
## [6,] -0.2378 -1.24482 -1.6077 -1.0622  3.2547  2.7392
-1.4833  2.2343
##        [,26]   [,27]    [,28]   [,29]   [,30]   [,31]
[,32]   [,33]
## [1,] -1.2762 -1.2305 -1.43337 -1.0890 -1.2987 -1.2618
-1.44434  1.1015
## [2,] -1.2762 -1.2305 -1.18065 -1.0890 -1.0509 -1.2618
-1.25918  0.9781
## [3,]  0.2009  0.2928  0.26624 -0.4338 -0.1082 -0.2939
0.05067  1.6943
## [4,] -1.2762  2.8960  0.71990  0.2960 -1.2987  2.7687
2.08960  0.7000
## [5,] -1.2762  2.2489  0.02799 -1.0890 -1.2987  2.0052
1.17454 -1.4722
## [6,] -1.2762  1.8359  1.31110 -1.0890 -1.2987  1.7378
0.89347 -0.5288
##        [,34]    [,35]    [,36]  [,37]   [,38]
## [1,] -1.3416 -1.22961 -0.75919 0.8490 -0.6646
## [2,] -0.7936 -1.22961 -0.71792 0.4513 -0.4580
## [3,] -0.1247  0.04609  0.24347 0.9077  0.4651
## [4,]  0.1385  1.75908  0.06151 1.3030  0.5819
```

```
## [5,] -1.3416  1.55086 -1.18107 1.0160  0.1579
## [6,] -1.2217  0.90832 -1.39906 0.5127  1.3625
```

Note that each column is a sample, and golub[j, i] is the expression level for gene j in tumor mRNA sample i.

There are also gene identifiers and tumor class labels (0 for ALL, 1 for AML).

```
dim(golub.gnames)
```

```
## [1] 3051    3
```

```
golub.gnames[1:4, ]
```

```
##      [,1] [,2]
## [1,] "36" "AFFX-HUMISGF3A/M97935_MA_at (endogenous control)"
## [2,] "37" "AFFX-HUMISGF3A/M97935_MB_at (endogenous control)"
## [3,] "38" "AFFX-HUMISGF3A/M97935_3_at (endogenous control)"
## [4,] "39" "AFFX-HUMRGE/M10098_5_at (endogenous control)"
##      [,3]
## [1,] "AFFX-HUMISGF3A/M97935_MA_at"
## [2,] "AFFX-HUMISGF3A/M97935_MB_at"
## [3,] "AFFX-HUMISGF3A/M97935_3_at"
## [4,] "AFFX-HUMRGE/M10098_5_at"
```

```
golub.cl
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1
## [36] 1 1 1
```

Now we are ready to dive in.

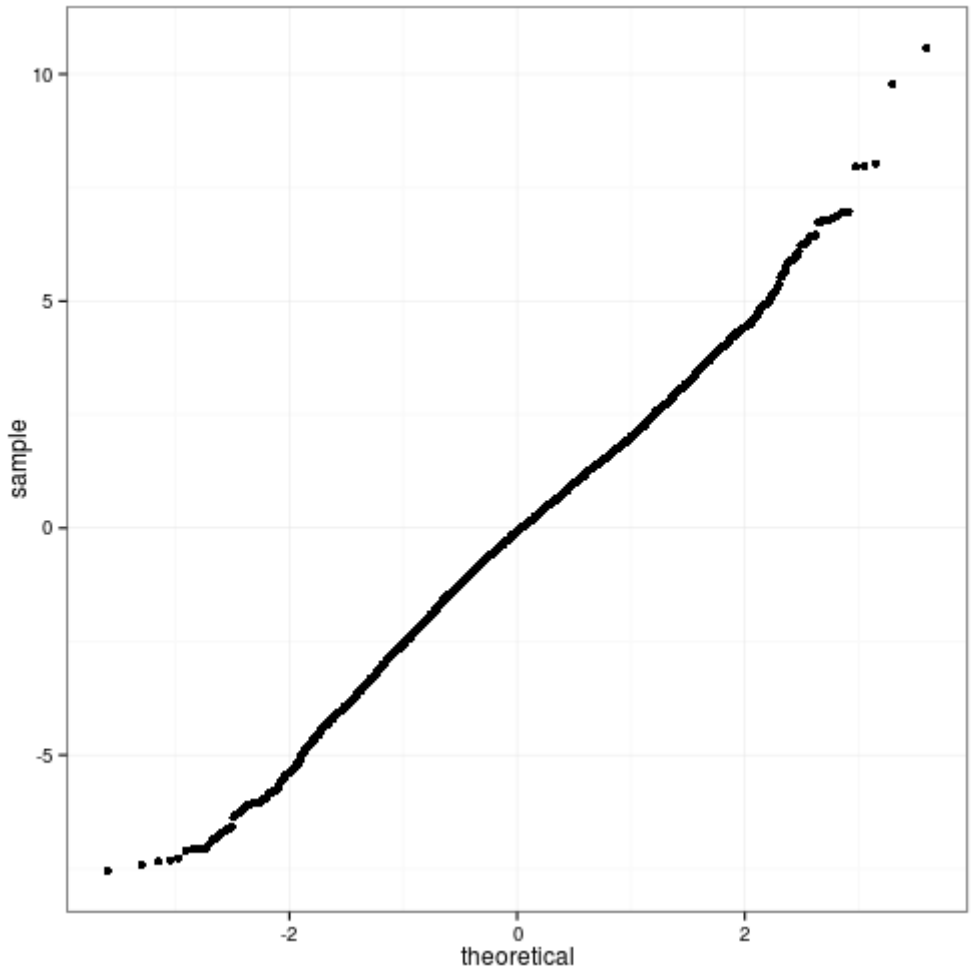
Computing simple test statistics

The mt.teststat and mt.teststat.num.denum functions provide a convenient way to compute test statistics for each row of a data frame, e.g., two-sample Welch t-statistics, Wilcoxon statistics, F-statistics, paired t-statistics, and block F-statistics.

Let's compute two-sample t-statistics that compares the gene expressions for each gene in the ALL and AML cases. This can be done with the mt.teststat function. The default test is the two-sample Welch t-test.

```
teststat = mt.teststat(golub, golub.cl)
```

```
require(ggplot2)
plt = ggplot(data.frame(teststat), aes(sample = teststat)) +
stat_qq() + theme_bw()
plt
```



Question 1 (Multiple Choice) What can we say about those points on the plot that look like outliers?

a. They *could* correspond to genes whose expression levels differ between the ALL and AML groups

b. They *definitely* correspond to genes whose expression levels differ between the ALL and AML groups

c. We cannot say anything

We might want to compute the numerators and denominators of the test statistics. These can be done together with the `mt.teststat.num.denum()` function.

```
parts = mt.teststat.num.denum(golub, golub.cl)
names(parts)
```

```
## [1] "teststat.num"   "teststat.denum"
```

This returns a 2-column dataframe, and the numerator and denominator can be extracted with

```
head(parts$teststat.num)
```

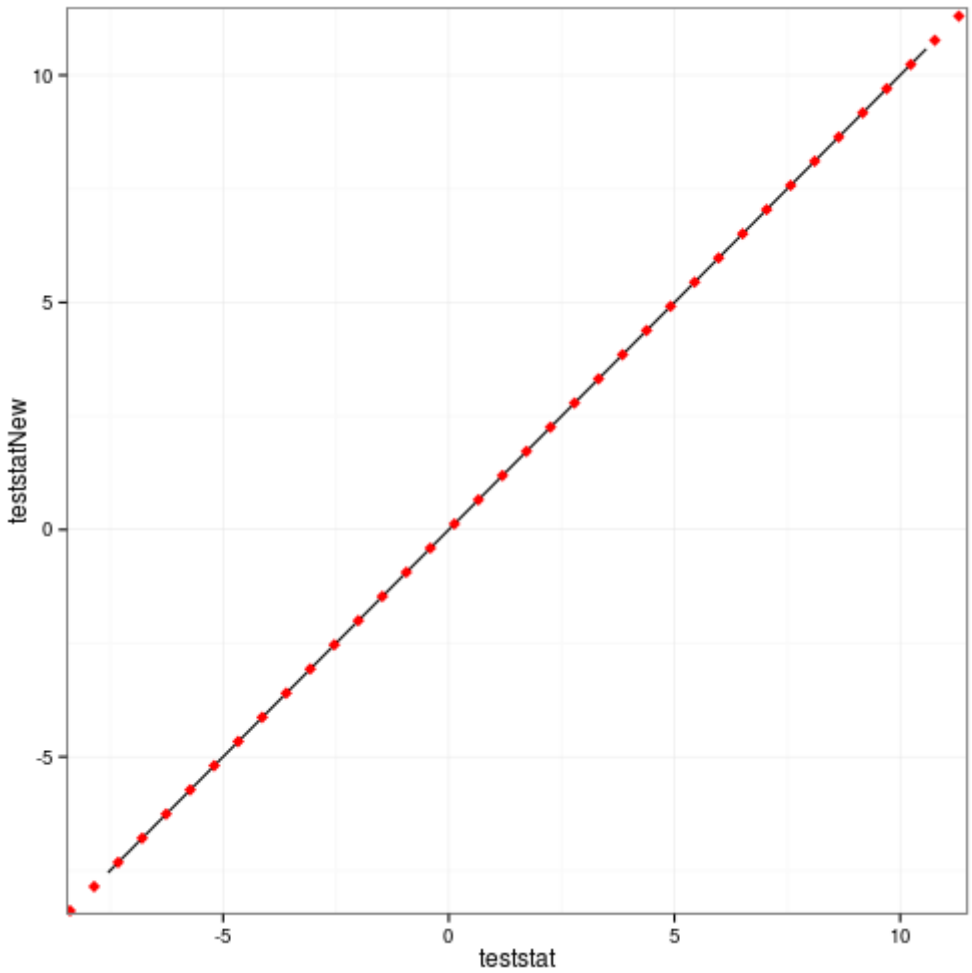
```
## [1]  0.49226  0.21787 -0.01994 -0.16947 -0.72660 -0.62998
```

```
head(parts$teststat.denum)
```

```
## [1] 0.2798 0.2395 0.2034 0.5000 0.5303 0.4980
```

As a sanity check, let's visually check that their ratio agrees with the previously computed statistics.

```
teststatNew = parts$teststat.num/parts$teststat.denum
plt = ggplot(data.frame(teststat, teststatNew), aes(x =
teststat, y = teststatNew)) +
    geom_line() + geom_abline(colour = "red", linetype =
"dotted", size = 2) +
    theme_bw()
plt
```



Question 2 (Multiple Response) Which of the following commands can be used to check that two numeric vectors or matrices are identical?

a) `identical(A, B)`

b) `max(abs(A-B)) == 0`

Question 3 (Multiple Response) The commands in Question 2 work for the vast majority of cases. But there are exceptions. Run the following code:

```
identical(2^100, 2^100 - 1)
identical(2^20, 2^20 - 1)
```

What should we be mindful of when comparing two numbers?

a) Their relative sizes

b) Their absolute sizes

The reason that this happens is because your computer rounds off numbers after some number of digits. This is called machine epsilon. Try the following code below:

```
identical(2^23, 2^23 - 1)
identical(2^24, 2^24 - 1)
identical(2^53, 2^53 - 1)
identical(2^54, 2^54 - 1)
identical(2^100, 2^100 - 1)
```

Adjusting p-values

The `mt.rawp2adjp` function computes adjusted p-values for simple multiple testing procedures from a vector of raw (unadjusted) p-values. The procedures include the Bonferroni, Holm (1979), Hochberg (1988), and Sidak procedures for strong control of the family-wise Type I error rate (FWER), and the Benjamini and Hochberg (1995) and Benjamini and Yekutieli (2001) procedures for (strong) control of the false discovery rate (FDR). First we will compute raw nominal two-sided p-values. For this data, we'll assume that it's safe to use a standard normal distribution for the 3,051 test statistics.

```
rawp = 2 * (1 - pnorm(abs(teststat)))
```

We can adjust these using the specified procedure as follows:

```
procedures = c("Bonferroni", "Holm", "Hochberg", "SidakSS",
"SidakSD", "BH",
    "BY")
adjusted = mt.rawp2adjp(rawp, procedures)
adjusted$adjp[1:10, ]
```

```
##            rawp Bonferroni      Holm  Hochberg   SidakSS
SidakSD
##  [1,] 0.000e+00  0.000e+00 0.000e+00 0.000e+00 0.000e+00
0.000e+00
##  [2,] 0.000e+00  0.000e+00 0.000e+00 0.000e+00 0.000e+00
0.000e+00
##  [3,] 8.882e-16  2.710e-12 2.708e-12 2.708e-12 2.710e-12
2.708e-12
##  [4,] 1.332e-15  4.065e-12 4.061e-12 4.061e-12 4.065e-12
4.061e-12
##  [5,] 1.554e-15  4.742e-12 4.736e-12 4.736e-12 4.742e-12
4.736e-12
##  [6,] 4.419e-14  1.348e-10 1.346e-10 1.346e-10 1.348e-10
1.346e-10
##  [7,] 1.208e-13  3.685e-10 3.678e-10 3.678e-10 3.685e-10
3.678e-10
##  [8,] 2.010e-13  6.131e-10 6.117e-10 6.117e-10 6.131e-10
6.117e-10
##  [9,] 2.607e-13  7.953e-10 7.933e-10 7.933e-10 7.953e-10
7.933e-10
## [10,] 3.419e-13  1.043e-09 1.040e-09 1.040e-09 1.043e-09
1.040e-09
##              BH         BY
##  [1,] 0.000e+00 0.000e+00
##  [2,] 0.000e+00 0.000e+00
##  [3,] 9.033e-13 7.769e-12
##  [4,] 9.484e-13 8.157e-12
##  [5,] 9.484e-13 8.157e-12
##  [6,] 2.247e-11 1.932e-10
##  [7,] 5.265e-11 4.528e-10
##  [8,] 7.664e-11 6.591e-10
##  [9,] 8.837e-11 7.600e-10
## [10,] 1.043e-10 8.973e-10
```

The results are stored in increasing order of the raw p-values. To display them based on the original data order, use

```
adjusted$adj[order(adjusted$index)[1:10], ]
```

```
##          rawp Bonferroni Holm Hochberg SidakSS SidakSD
BH BY
##  [1,] 0.07854          1    1   0.9998       1       1
0.1820  1
##  [2,] 0.36290          1    1   0.9998       1       1
0.5355  1
##  [3,] 0.92191          1    1   0.9998       1       1
0.9590  1
##  [4,] 0.73464          1    1   0.9998       1       1
0.8385  1
##  [5,] 0.17064          1    1   0.9998       1       1
0.3188  1
##  [6,] 0.20585          1    1   0.9998       1       1
0.3618  1
##  [7,] 0.47020          1    1   0.9998       1       1
0.6379  1
##  [8,] 0.59365          1    1   0.9998       1       1
0.7375  1
##  [9,] 0.98667          1    1   0.9998       1       1
0.9932  1
## [10,] 0.89268          1    1   0.9998       1       1
0.9429  1
```

We can also adjust the p-values with permutations using the `mt.maxT()` and `mt.minP()` functions. From the vignette: compute permutation adjusted p-values for the maxT and minP step-down multiple testing procedure described in Westfall and Young (1993). These procedure provide strong control of the FWER and also incorporate the joint dependence structure between the test statistics. There are thus in general less conservative than the standard Bonferroni procedure. The permutation algorithm for the maxT and minP procedures is described in Ge et al. (2003).

```
resT = mt.maxT(golub, golub.cl, B = 10000)
```

```
## b=100    b=200   b=300   b=400   b=500   b=600   b=700
b=800   b=900   b=1000
## b=1100   b=1200  b=1300  b=1400  b=1500  b=1600  b=1700
b=1800  b=1900  b=2000
## b=2100   b=2200  b=2300  b=2400  b=2500  b=2600  b=2700
b=2800  b=2900  b=3000
## b=3100   b=3200  b=3300  b=3400  b=3500  b=3600  b=3700
b=3800  b=3900  b=4000
## b=4100   b=4200  b=4300  b=4400  b=4500  b=4600  b=4700
b=4800  b=4900  b=5000
## b=5100   b=5200  b=5300  b=5400  b=5500  b=5600  b=5700
b=5800  b=5900  b=6000
## b=6100   b=6200  b=6300  b=6400  b=6500  b=6600  b=6700
b=6800  b=6900  b=7000
## b=7100   b=7200  b=7300  b=7400  b=7500  b=7600  b=7700
b=7800  b=7900  b=8000
## b=8100   b=8200  b=8300  b=8400  b=8500  b=8600  b=8700
b=8800  b=8900  b=9000
## b=9100   b=9200  b=9300  b=9400  b=9500  b=9600  b=9700
b=9800  b=9900  b=10000
```

```
names(resT)
```

```
## [1] "index"    "teststat" "rawp"     "adjp"
```

```
head(resT$teststat)
```

```
## [1] 10.578  9.776  8.033  7.983  7.966 -7.548
```

The p-values are sorted in decreasing order of the absolute values of the test statistics.

Question 4 (Multiple Response) Which of the following commands get the p-values in the original data order?

a) `resT$rawp[order(resT$index)]`

b) `resT$adjp[order(resT$index)]`

c) `resT$rawp[sort(resT$index)]`

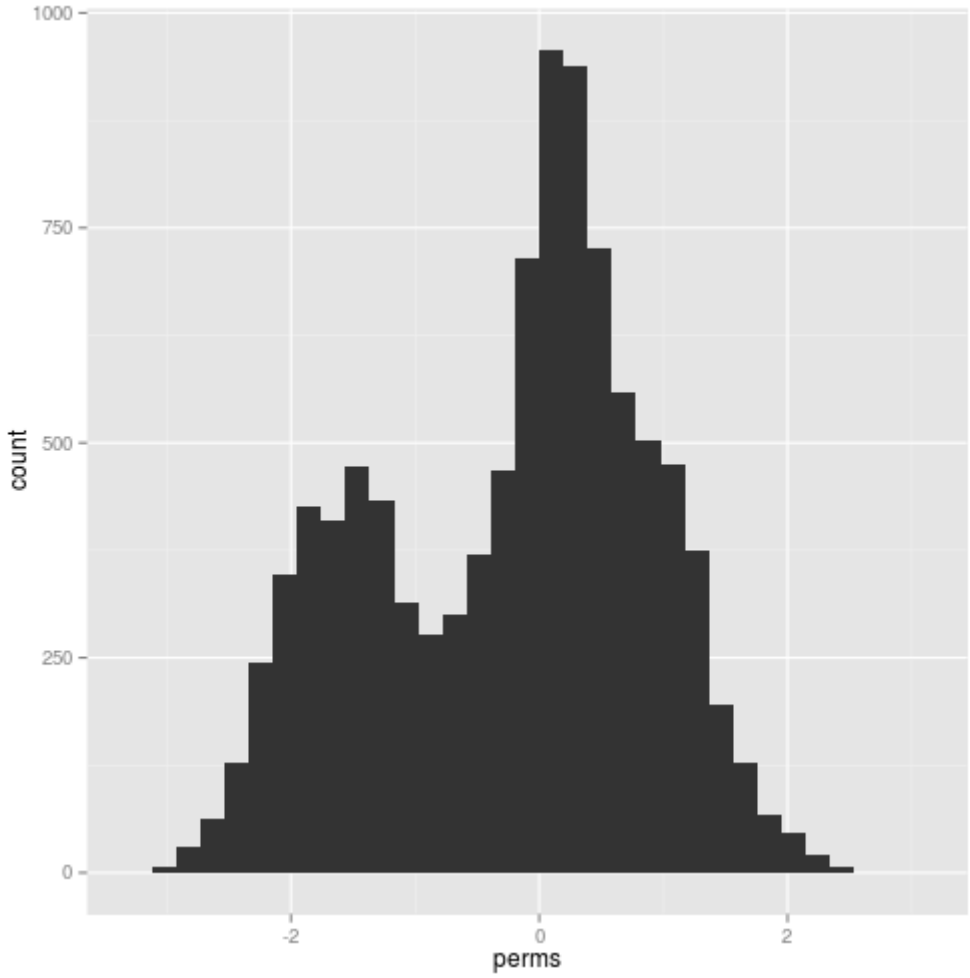
The functions `mt.sample.teststat` and `mt.sample.rawp` can be used to investigate the permutation distribution of test statistics and raw p-values. For example,

```
perms = as.numeric(mt.sample.teststat(golub[1, ], golub.cl, B =
10000))
```

gives a vector of 10,000 permutation test statistics for gene 1. Look at the histogram:

```
plt = ggplot(data.frame(perms), aes(x = perms)) +
stat_bin(position = "identity")
plt
```

```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x'
to adjust
## this.
```



Question 5 (Multiple Choice) Run the following commands:

```
resT = mt.maxT(matrix(golub[1, ], nrow = 1), golub.cl, B =
10000)
resT$rawp
```

This gives the raw (unadjusted) p-value from the permutation test on gene 1. Referring to `perms` above, which of the following commands is the right way to compute this p-value?

a) `sum(abs(perms)>=resT$teststat) / length(perms)`

b) `sum(max(perms)>=resT$teststat) / length(perms)`

The `mt.reject` function

The function mt.reject returns the identity and number of rejected hypotheses for several multiple testing procedures and different nominal Type I error rates.

We demonstrate with the golub data by first computing the p-values for all the genes:

```
resT = mt.maxT(golub, golub.cl, B = 10000)
```

```
## b=100    b=200   b=300   b=400   b=500   b=600   b=700
b=800   b=900   b=1000
## b=1100   b=1200  b=1300  b=1400  b=1500  b=1600  b=1700
b=1800  b=1900  b=2000
## b=2100   b=2200  b=2300  b=2400  b=2500  b=2600  b=2700
b=2800  b=2900  b=3000
## b=3100   b=3200  b=3300  b=3400  b=3500  b=3600  b=3700
b=3800  b=3900  b=4000
## b=4100   b=4200  b=4300  b=4400  b=4500  b=4600  b=4700
b=4800  b=4900  b=5000
## b=5100   b=5200  b=5300  b=5400  b=5500  b=5600  b=5700
b=5800  b=5900  b=6000
## b=6100   b=6200  b=6300  b=6400  b=6500  b=6600  b=6700
b=6800  b=6900  b=7000
## b=7100   b=7200  b=7300  b=7400  b=7500  b=7600  b=7700
b=7800  b=7900  b=8000
## b=8100   b=8200  b=8300  b=8400  b=8500  b=8600  b=8700
b=8800  b=8900  b=9000
## b=9100   b=9200  b=9300  b=9400  b=9500  b=9600  b=9700
b=9800  b=9900  b=10000
```

```
ord = order(resT$index)
rawp = resT$rawp[ord]
maxT = resT$adjp[ord]
teststat = resT$teststat
```

The number of hypotheses rejected using unadjusted p-values and maxT p-values for different Type I error rates ($\alpha = 0,0.1,0.2, \ldots ,1$) can be obtained by

```
mt.reject(cbind(rawp, maxT), seq(0, 1, 0.1))$r
```

```
##     rawp maxT
## 0      0    0
## 0.1 1324  119
## 0.2 1661  150
## 0.3 1900  178
## 0.4 2090  202
## 0.5 2263  237
## 0.6 2441  265
## 0.7 2611  311
## 0.8 2755  354
## 0.9 2893  409
## 1   3051 3051
```

Question 6 Part A (Multiple Response) What does this code do?

```
which = mt.reject(cbind(rawp, maxT), 0.01)$which[, 2]
golub.gnames[which, 2]
```

(a) Gives genes with maxT p-values less than or equal to 0.01

(b) Gives genes with maxT p-values strictly less than 0.01

© Gives genes with maxT p-values greater than or equal to 0.01

(d) Gives genes with maxT p-values strictly greater than 0.01

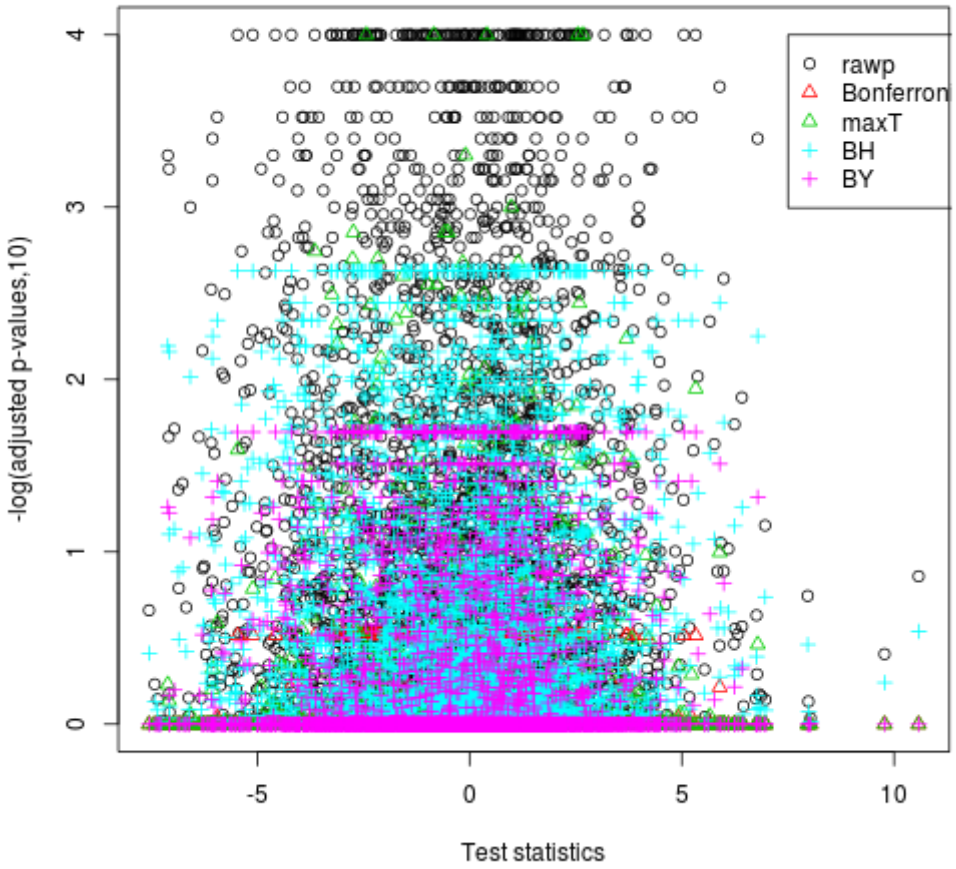
The `mt.plot()` function

The mt.plot function produces a number of graphical summaries for the results of multiple testing procedures and their corresponding adjusted p-values. To produce plots of sorted permutation unadjusted p-values and adjusted p-values for the Bonferroni, maxT, Benjamini and Hochberg, and Benjamini and Yekutieli procedures use

```
res = mt.rawp2adjp(rawp, c("Bonferroni", "BH", "BY"))
adjp = res$adjp[order(res$index), ]
allp = cbind(adjp, maxT)
dimnames(allp)[[2]] = c(dimnames(adjp)[[2]], "maxT")
procs = dimnames(allp)[[2]]
procs = procs[c(1, 2, 5, 3, 4)]
cols = c(1, 2, 3, 5, 6)
ltypes = c(1, 2, 2, 3, 3)
```

For plotting adjusted p-values vs test statistics:

```
mt.plot(allp[, procs], teststat, plottype = "pvst", logscale =
TRUE, proc = procs,
    leg = c(7.5, 4), pch = ltypes, col = cols)
```



Question 6 Part B True of False: If we set `plottype=pvsr`, we are plotting the sorted adjusted p-values.

Resampling-based multiple testing procedures: MTP function

The main user-level function for resampling-based multiple testing is MTP. The default setting controls the family-wise error rate, defined as the probability of any false positives. Let Vn denote the (unobserved) number of false positives. Then, control of FWER at level alpha means that Pr(Vn>0)<=alpha. The set of rejected hypotheses under a FWER controlling procedure can be augmented to increase the number of rejections, while controlling other error rates. The generalized family-wise error rate is defined as Pr(Vn>k)<=alpha, and it is clear that one can simply take an FWER controlling procedure, reject k more hypotheses and have control of gFWER at level alpha. The tail probability of the proportion of false positives depends on both the number of false postives (Vn) and the number of rejections (Rn). Control of TPPFP at level alpha means Pr(Vn/Rn>q)<=alpha, for some proportion q. Control of the false discovery rate refers to the expected proportion of false positives (rather than a tail probability). Control of FDR at level alpha means E(Vn/Rn)<=alpha.

Let's run MTP on the first gene in the golub dataset.

```
fwer = MTP(matrix(golub[1, ], nrow = 1), Y = golub.cl, typeone
= "fwer")
```

```
## running bootstrap...
## iteration = 100 200 300 400 500 600 700 800 900 1000
```

```
summary(fwer)
```

```
## MTP:  ss.maxT
## Type I error rate:  fwer
##
##   Level Rejections
## 1  0.05          1
##
##            Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
## adjp      0.039   0.039  0.039 0.039   0.039 0.039    0
## rawp      0.039   0.039  0.039 0.039   0.039 0.039    0
## statistic 1.760   1.760  1.760 1.760   1.760 1.760    0
## estimate  0.492   0.492  0.492 0.492   0.492 0.492    0
```

Now try controlling the FDr instead:

```
fdr = MTP(matrix(golub[1, ], nrow = 1), Y = golub.cl, typeone =
"fdr")
```

```
## running bootstrap...
## iteration = 100 200 300 400 500 600 700 800 900 1000
```

```
summary(fdr)
```

```
## MTP:  ss.maxT
## Type I error rate:  fdr (conservative)
##
##   Level Rejections
## 1  0.05          0
##
##            Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
## adjp      0.068   0.068  0.068 0.068   0.068 0.068    0
## rawp      0.034   0.034  0.034 0.034   0.034 0.034    0
## statistic 1.760   1.760  1.760 1.760   1.760 1.760    0
## estimate  0.492   0.492  0.492 0.492   0.492 0.492    0
```

Note that we needed to force the result of `golub[1,]` to be a matrix. You should try the commands without, and see what happens.

Question 6 Part C Does controlling from FDR or FWER result in higher power?