Week 6 Notes Part 2 Multiple Testing

Code ▾

Author: Brendan Gongol

Last update: 07 January, 2023

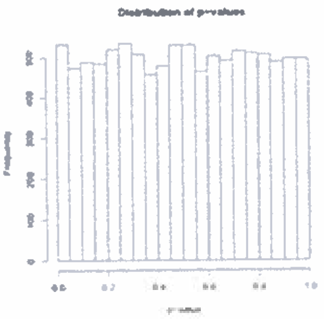
1 Overview

In this Note we discuss multiple testing and multiple testing corrections (with emphasis on Bonferroni). We will discuss different metrics that can be used to characterize performance of a test, and further consider the false discovery rate as one such metrics very important in exploratory data analysis. Finally, we will learn how to use the CRAN package qvalue in order to estimate FDR in a dataset.

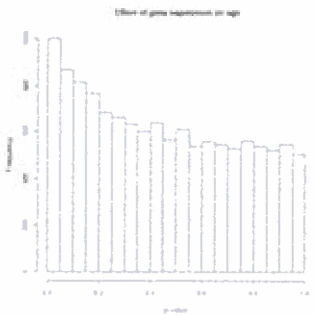
2 Multiple Testing

On quite a few occasions during the past few weeks we were performing multiple tests at once. This is a common scenario when we have a “large-scale” dataset, where the same type of measurement is performed in parallel for large number of subjects or entities. Since the type of data is the same, we also want to apply same type of analysis: for instance, quantify, for each gene, the significance of (mean) expression level differences between males and females or between patients with different subtypes of BCR-ABL1 fusion protein detected, or significance of association of expression level with age, remission status, etc. Since these tests are performed individually, per every gene, this situation is commonly referred to as “multiple testing” scenario.

In a few exercises we worked through so far, we were careful enough to examine the distribution of p-values resulting from individual tests. We know by now that under the null hypothesis, if we simply perform multiple tests (we simulated this in a homework by doing multiple resampling), the p-value distribution is uniform, as shown in the figure below; this means that even in the absence of any effect (i.e. when the null is true), 5% of tests will result is p-value below 0.05.



We have also observed that when we perform multiple tests for the effect of gene expression on age (our approach was to use a linear model), the distribution of p-values is skewed towards low p-values, as it is shown below:



This last figure can be interpreted (qualitatively, for now) as follows: there is a subpopulation of genes with no effect on age. As we know, the distribution of p-values for those genes is going to be flat (the null is true for them, so there will be 100*x% of p-values below any significance cutoff x we choose, just by random chance). There is also a subpopulation of genes with tru effect on age; those will have predominantly small p-values (note that accidentally we may observe by random chance a large, insignificant p-value in a situation when there is an effect, this is called a “false negative”, since we declare “no effect” and do not reject the null while the null is not true). When we draw the distribution of p-values for all genes, this distribution is a superposition of the distributions from the two subpopulations, and it must have the shape shown above. In other words, enrichment at low p-values indicates that there should be some genes among the thousands tested that do exhibit real effect.

The question is if (and how) we can identify those genes that truly exhibit the effect, against the background of genes that exhibit no effect but have small p-value nevertheless, just by random chance. Any approach aimed at this goal is what is known as “multiple testing correction” (there are many ways to do it!).

2.1 Bonferroni Correction

Let us start with the simplest and most straightforward approach. When we run a test and set the confidence threshold at 0.05, we are agreeing to accept the 5% chance for the test to give significant result when in reality the null hypothesis is true. Thus if we were to run ~~N replicas~~ of the test, we have the expected false positive rate, FPR (also known as Type I Error) set to 5%, where FPR=FP/N, FP is the number of false positive findings we are going to see. Note that there is a great deal of arguing you can stumble upon, where different sources, forums, tutorials and textbooks (correctly!) claim that the p-value is not the Type I error. This statement is definitely correct in general case; p-value of 0.05 only tells you that 5% of the repeated tests will be called significant (i.e. will represent FP) when the null is true. But this is precisely the simplified case we are considering so far. So in our specific example (null is true), the FPR is indeed simply given by the chosen p-value cutoff.

When N is large, the number of FPs becomes unacceptably large too: when testing 10000 genes at 5% confidence level we expect 500 false positives (if in reality there is no effect for any of them). The problem, as we can see, is that we set significance level for an individual test and then we run many parallel tests.

The idea behind Bonferroni correction is to look at the properly defined false positive rate of the findings. The logic goes as follows: if we run a single test, the “finding” is whether that test is significant or not, that's the “experimental unit” (that we can repeat if we wish). If we run a microarray and measure 10,000 genes, the “finding” is the collection of significant genes that we detected at any particular significance threshold that we chose. The “experimental unit” is the whole microarray. We can rerun it many times, of course, just like any experiment. But in these settings the “false positive rate” refers to the frequency of false positive findings as defined above, i.e. whole collections of genes. In this framework, we want to set the thresholds in such a way that the whole set of genes we found significant has only 5% chance to be incorrect (or whatever threshold α we prefer). “Incorrect” means here that at least onegene in the family of detected significant genes is a false positive. In other words, if we were to rerun the whole microarray 100 times, only in 5 cases we would observe the whole collection, or “family” of jointly significant genes where the null is actually true for at least one of the genes in the family.

The error rate defined in this way is called “family-wise error rate” or FWER. It is easy to see that if we are performing N individual tests and want to keep FWER at the threshold level p_0, we need to set the significance threshold for each individual test as p_0/N. This is known as Bonferroni correction.

The derivation is simple: if the probability to make a false positive call in a single test is α , then the probability to not call an FP is $1-\alpha$, and the probability to make no false positive calls in N tests is $(1-\alpha)^N$. The probability to make at least one false positive call in N tests (i.e. to have the whole family of significant calls deemed a false positive, according to our family-wise definition) is thus $1-(1-\alpha)^N \approx N\alpha$, where we use Tailor expansion and assume $\alpha << 1$. We want to keep the latter probability to make at least one false positive call in N tests (and thus to have the whole selected family of significant genes "wrong") below p_0. Hence, we need to set the significance threshold for each individual test such that , thus .

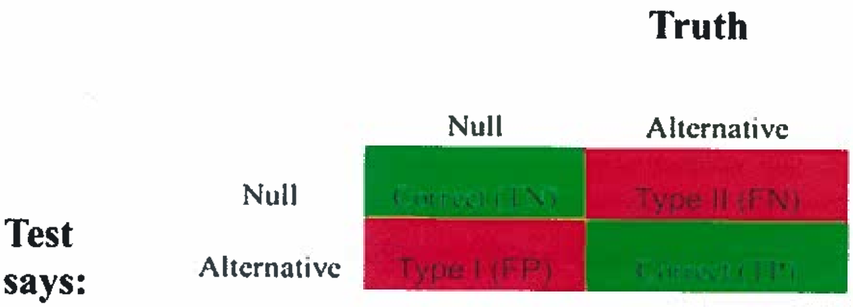
Bonferroni correction can be used in some applications, but as we can see it is extremely conservative: it requires the whole family of the detected significance genes to have a pretty low chance to be a false positive, jointly. Let us return to the example used few weeks earlier: significance of difference between mean expression levels in males and females in ALL dataset (running `t.test(expr.levels[sex==”M”],expr.levels[sex=”F”] )` on each gene in a dataset is certainly an under-three-minutes exercise for you by now!). If we apply Bonferroni correction (you can try it!), and require that the whole set of significant genes we find had pvalue p0=0.05, we find only 11 out of 12625 individual p-values that are less than Bonferroni□corrected threshold of 0.05/12625 (10 of which are less than 10-14). Most of those genes are associated with Y chromosome (that's why they are so tremendously significant), and thus they are actually a trivial and not particularly interesting finding as far as male/female differences in expression are concerned. Since both males and females in this dataset are patients, we are much more interested, biologically, in genes that exhibit more subtle differences (potentially reflecting differences in disease progression in males and females), rather than in homing in on the "interesting" fact that females lack Y chromosome.

Furthermore, in the example of gene expression – age association, none of the p-values are under 0.05/12625, so we cannot find anything significant enough to pass the family-wise threshold at all. However, as the distribution of p-values show (the last of the two figures we started this Note with), there is a very clear overrepresentation of low p-values, so there are genes with true effect. We just need a way to mine them out of the false positives.

There is of course no way to extract from the limited data more information than they contain, so the fact that there is no family of genes that are sufficiently significant jointly is the truth of life. However, we may want to be less conservative and control not for FWER but for something different. So let us take a little break from the significance and consider different metrics that can be used to characterize performance of a test.

3 Measures of Prediction Quality

So far, in conjunction with hypothesis testing we have been looking only at the false positive rate of a test (also known as Type I error): the probability to obtain significant test result while the null hypothesis is correct (i.e. we erroneously reject the null and accept the alternative). Another type of error that we can make is the opposite: the null hypothesis is incorrect, but due to random sampling the statistic that actually comes from the alternative distribution falls within the range of what we could expect under the null. In this case we erroneously keep the null and reject the alternative, which is a false negative result. The probability to make a mistake of this type is known as false negative rate or Type II error. The table below illustrates the two types of error:



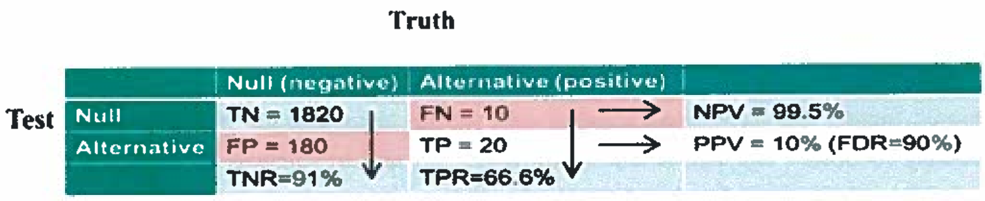
		Truth	
		Null	Alternative
Test says:	Null	Correct (TN)	Type II (FN)
	Alternative	Type I (FP)	Correct (TP)

When we run multiple tests, there are a few metrics we can look into in addition to false positive and false negative rates. Depending on the breakdown between true negatives (TN), true positives (TP), false negatives (FN) and false positives (FP), a particular test can be more or less useful depending on the applications. The following metrics can be considered (where TN,TP,FN,FP are calls (i.e. what the test says), and P,N (Positive, Negative) is the truth -actual, albeit unknown, number of true events and non-events, respectively:

- Negative Predictive Value, NPV =TN/(TN+FN); low NPV means that we are biased towards the null (lots of false negatives) and tend to miss true events, i.e. we are very conservative.
- Positive Predictive Value, PPV = TP/(TP+FP); low PPV means that we are too willing to reject the null even when we should not and thus make too many false positive calls
- False Discovery Rate, FDR=1-PPV=FP/(TP+FP); obviously, quantifies the same effect as PPV but defined as fraction of false positive calls out of all positive ("significant") calls.
- Sensitivity or True Positive Rate, TPR = TP/P = TP/(TP+FN); low TPR means that we are missing many true events (so the test is not very sensitive)
- Specificity, or True Negative Rate, TNR = TN/N = TN/(TN+FP)=1-FPR; low TNR means that we are calling too many no-events as significant (i.e. false positives), so the test is not very specific (picks up lots of cases that it should not).

Note that all these metrics capture different aspects of the test performance. For instance, low NPV and low TPR both mean that we are miscalling true alternative events as nulls (no-event), however, depending on the relative amounts of positive and negative cases in the dataset, they can have very different values. For instance, if the number of positive cases is very small (P<<N), then it is possible for a test that has very low TPR (TP/P << 1, i.e. TP<<P, we are detecting very small fraction of positive cases) to have large NPV. Indeed, we can have almost all positive cases miscalled as (false) negatives, so FN ~ P, but since P<<N, these false negatives are also a small fraction of all (true) negatives. If the false positive rate is also very low (so that the test is conservative), then TN~N and thus NPV=TN/(TN+FN)~1.

In order to better illustrate these different metrics consider the following example: we have a population of 2,030, with P=30 positive cases (e.g. disease) and N=2,000 negative cases (healthy subjects), and we have developed a diagnostic test that attempts to detect disease cases. The table below shows one possible set of results of our test applied to all 2,030 subjects:



		Truth		
		Null (negative)	Alternative (positive)	
Test	Null	TN = 1820 ↓	FN = 10 ↓ →	NPV = 99.5%
	Alternative	FP = 180	TP = 20 →	PPV = 10% (FDR=90%)
		TNR=91%	TPR=66.6%	

Note that the first three metrics described above (NPV, PPV, and FDR=1-PPV) are computed along the rows, i.e. with respect to the breakdown of one type of calls into true and false ones (Null, i.e. negative call into TN vs FN, Alt, i.e. positive call, into TP vs FP); the last two metrics (sensitivity and specificity) are computed along the columns and assess the breakdown of truth into calls (true Null into TN vs FP and true Alt into FN vs TP). It is not realistic to get all the metrics perfect for a given test. For instance, for the specific example shown above, due to the very small fraction of true disease cases the characteristics of our imaginary test are very unfavorable. Indeed, while we have true negative rate of 91% (i.e. we call most negative cases negative indeed, but most subjects are negative anyway!), the true positive rate of 66% -i.e. we do detect larger fraction of (the small number of) subjects who do have the disease - we still have the FDR at the whopping 90%, i.e. 90% of those who test positive for the disease are actually healthy! This table helps better understand the tradeoffs we are making when optimizing different characteristics. Namely,

- with low specificity (TNR), we are diagnosing large fraction of healthy subjects (truth is Null) with a disease: does this mean costly, invasive and/or unnecessary treatment?
- with low sensitivity (TPR), we are failing to diagnose many subjects with a disease (when truth is Alt): but is the disease life-threatening? Does it progress fast? Or we can institute an annual testing program and catch originally undiagnosed cases three years down the road (which might be beneficial overall if we can get very high specificity at the price of lower sensitivity, and the treatment is costly/invasive)?

- With low NPV, many negative test results are wrong (false negatives); this does not tell us much about sensitivity though; even with high NPV we can still have low sensitivity (as shown above) if true positive and negative cases have very different representations in the population
- With low PPV (high FDR), many positive test results are wrong (false positives); this does not tell us much about specificity or sensitivity – note how in our example FDR is 90%, i.e. 9 out of 10 positive diagnoses are wrong, however true positive rate (TPR) is relatively OK (meaning that when the subject does have a disease, we do not miss that too often).

The choice of the "optimal" test often involves choosing the metric(s) that are more important than others in a specific context the test will be applied in. There might be a situation when type II error (FN) is absolutely unacceptable while extra cost associated with many false positives (type I) can be tolerated, or it can be the other way around.

In the exploratory analysis that we are often performing (such as: "are there significant gender-related differences in gene expression"), sensitivity is very important of course, but a sensitive test that returns too many false positives is not very useful, since we might have to follow-up on too many potential findings. Hence, we usually want to know first how many of our predictions (test positives) are false positives – this means we should control for FDR.

4 FDR Evaluation

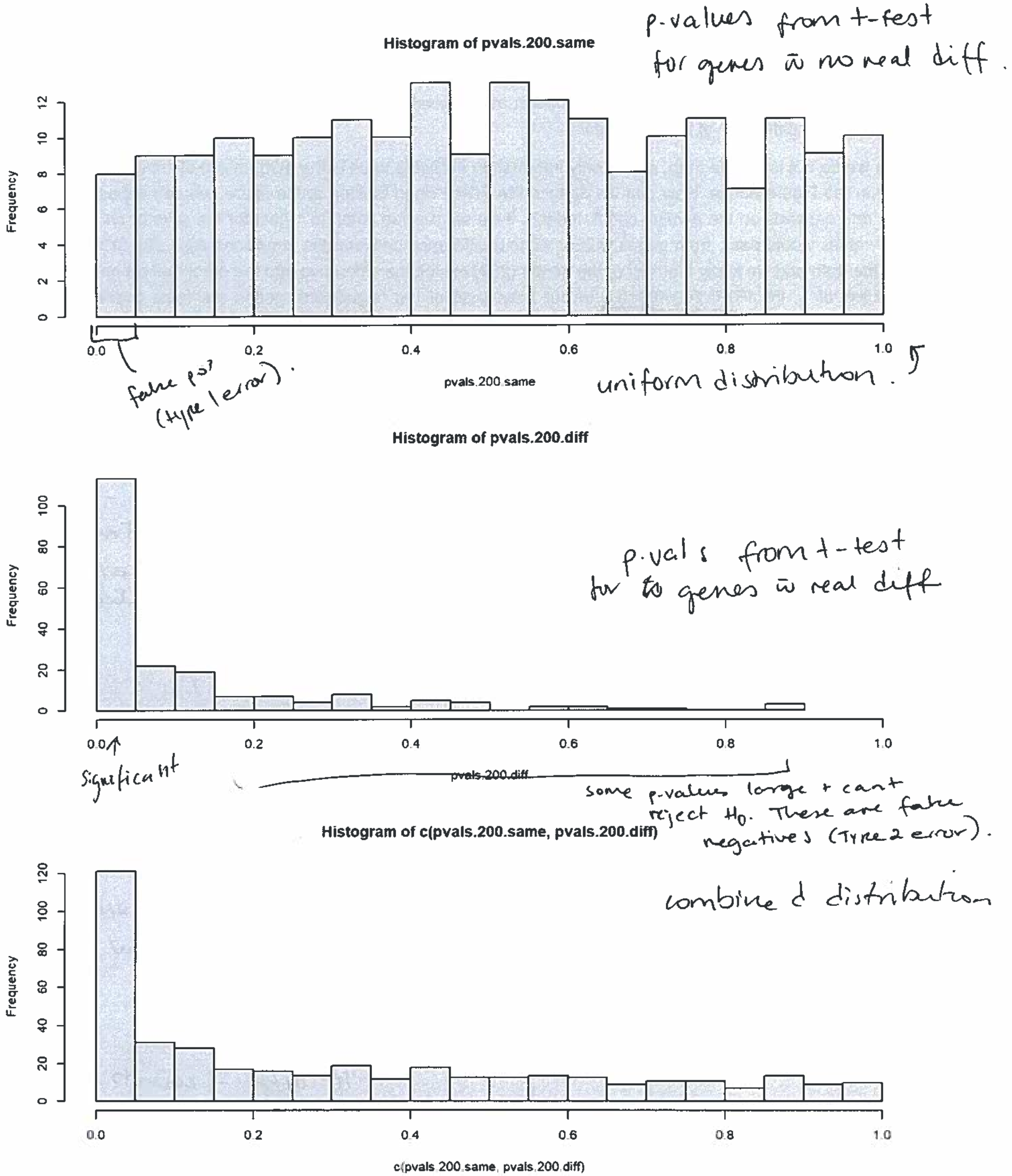
In order to better understand FDR and how it works let us consider a simple example. In line with our established tradition, we will simulate the data with known truth and see how they would present themselves if we measured such data and how our methods and approaches would work. Let us generate a dataset that consists of:

- 400 cases/entities repeatedly measured in two conditions (for the sake of certainty, let us assume that we measure expression of 400 genes in healthy controls and in subjects with disease)
- In each condition we have 10 measurements (i.e. for each gene we have 10 expression values in controls and 10 expression values in subjects with disease).
- 200 profiles exhibit no difference between disease and control (but we still have measurement error and biological variation, so we model this by drawing all 10+10 expression values from the same normal with mean=0 and sd=1; some of those pairs of samples may end up being "different enough" simply by random chance).
- Remaining 200 genes do exhibit true underlying (moderate) difference between the (mean) levels of expression. We model this by drawing 10 control measurements from the same normal with mean=0 and sd=1, but the disease values are sampled from a different normal, with mean=1 and same sd=1.

For each gene we then compare control and disease states using t-test (this is what we would be doing in real life with a dataset like this). Since we have the luxury of knowing a priori which genes not change between the two states, and which do, we will be keeping track of the t-test p-values from the no-difference and true-difference cases. The code used to set up this experiment is shown below:

Hide

```
pvals.200.same <- numeric()
pvals.200.diff <- numeric()
for (i in 1:200) {
x <- rnorm(10)
y <- rnorm(10) # x/y: control/disease measurements for no-change genes
w <- rnorm(10) # w/z: control/disease measurements for changing genes
z <- rnorm(10,mean=1)
pvals.200.same[i] <- t.test(x,y)$p.value
pvals.200.diff[i] <- t.test(w,z)$p.value
}
oldpar <- par(mfrow=c(3,1))
hist(pvals.200.same,breaks=20,xlim=c(0,1))
hist(pvals.200.diff,breaks=20,xlim=c(0,1))
hist(c(pvals.200.same,pvals.200.diff),breaks=20,xlim=c(0,1))
```



p-values from t-test for genes w̄ no real diff.

false pos? (type I error).

uniform distribution.

p-vals from t-test for to genes w̄ real diff

significant

some p-values large + can't reject H_0. These are false negatives (Type 2 error).

combined distribution

Hide

```
par(oldpar)
```

The plot generated by this code is shown below. As we can see (and definitely could anticipate), the p-values from genes with no difference follow uniform distribution (so we have some Type I error, or false positives, at low p-values); the p-values from genes with built in difference are heavily concentrated at low end, but despite the underlying difference some samples are "unlucky draws", so that the p-values are large and we wouldn't be able to reject the null for those genes (Type II error, or false negatives) if we were analyzing this dataset in real life, without knowing the truth. Finally, the union of the two distributions, obviously represents some mixture of false positives and genes with the true difference between the conditions at low p-values, and a mixture of false negatives and genes that truly exhibit no difference at high p-values.

In real life we do not know the truth, so the only information available to us is the third, mixed distribution of p-values as shown in the Figure above. How can we deduce the FDR? Upon looking at the figure, we can expect that false discovery rate depends on the p-value cutoff. Indeed, if we set p-value cutoff to 1 (just for the sake of an argument), then all p-values would pass, from genes with or without difference between the conditions alike. But in this dataset we know the truth and we know that half of the genes (200) exhibit the difference and the other half do not. Hence at p-value cutoff of 1, the FDR=200/400=0.5 in our case (half of the "significant" genes are false positives). If we decrease the p-value cutoff, we are enriching the selected "significant" genes for those coming from the cusp in the figure above due to genes with true difference.

Let us see how it works. Note that by analogy with p-values, FDR is often called a q-value.

$$FDR = \frac{\text{False Pos}}{\text{True Pos + False pos.}}$$

Hide

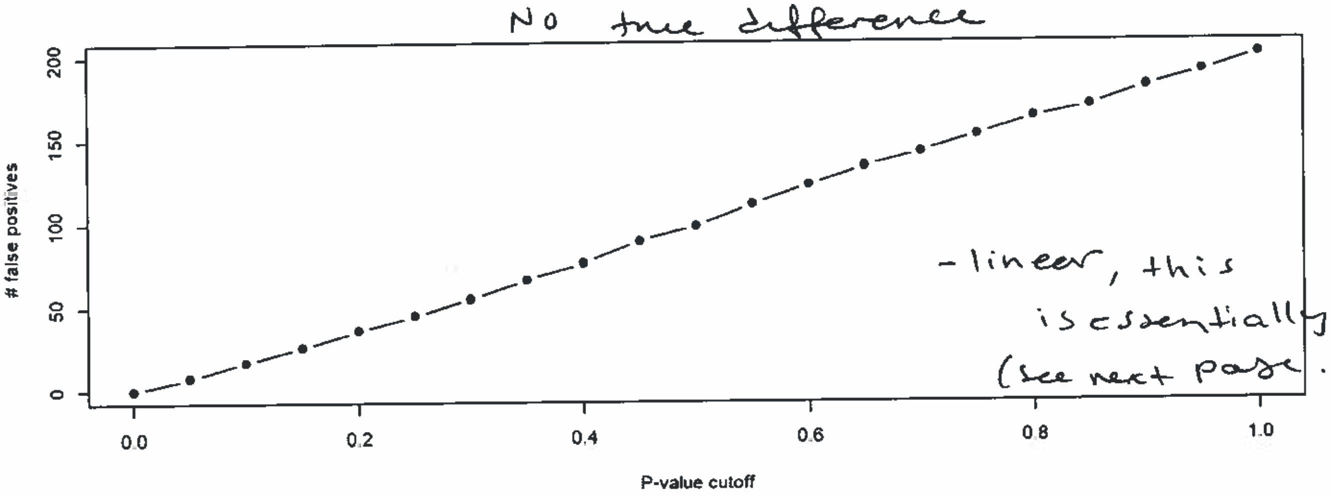
```
oldpar <- par(mfrow=c(3,1))
pval.cutoff <- seq(0,1,by=0.05)
n.same <- sapply(pval.cutoff,function(x) sum(pvals.200.same<x))
n.diff <- sapply(pval.cutoff,function(x) sum(pvals.200.diff<x))
qval <- n.same/(n.same+n.diff)
plot(pval.cutoff,n.same,pch=19,type='b',xlab="P-value cutoff",
 ylab="# false positives",main="")
plot(pval.cutoff,n.diff,pch=19,type='b',xlab="P-value cutoff",
 ylab="# true positives",main="")
plot(pval.cutoff,qval,pch=19,type='b',xlab="P-value cutoff",
 ylab="Q-value",main="")
```

logical expression - results in True/False. Then Sum() counts the true values.

∴ n.same - counts the # of false positives genes below each cutoff.

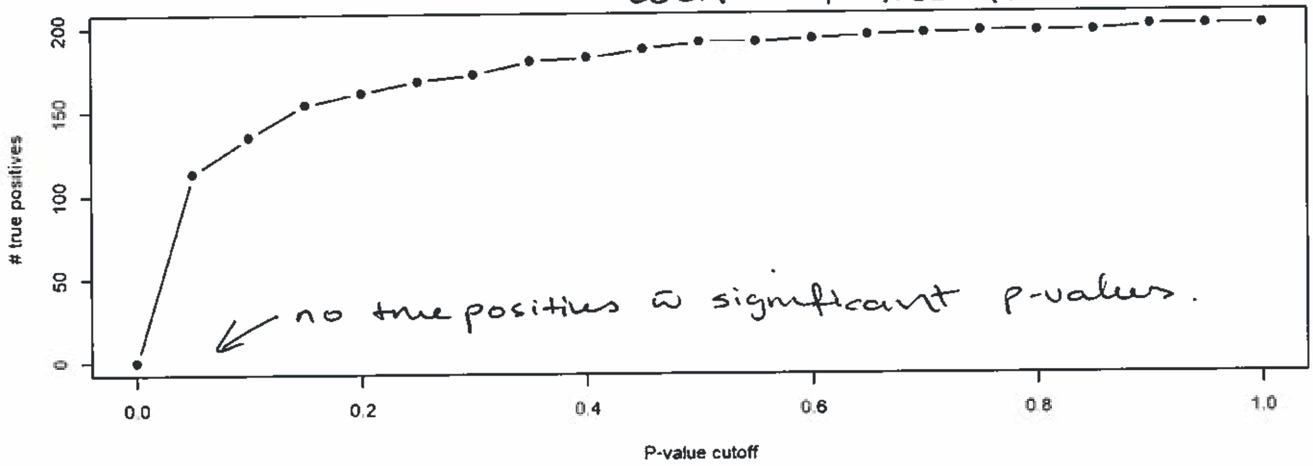
n.diff - counts # of true pos genes below each cutoff.

No true difference



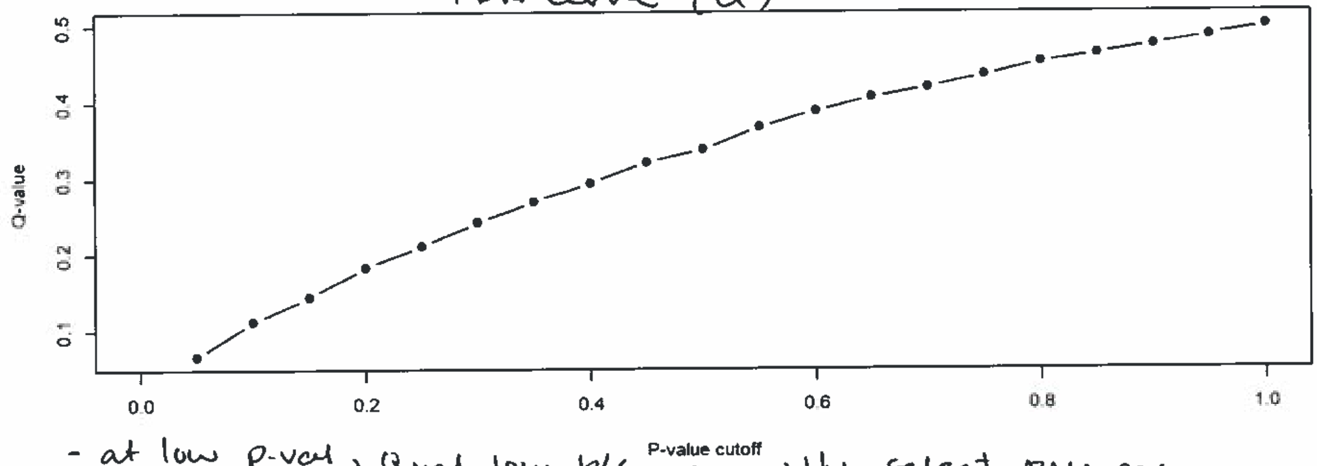
- linear, this is essentially a CDF. (see next page.

counts of true positives.



no true positives @ significant p-values.

FDR curve (Q)



- at low p-val, Q val low b/c we mostly select true pos.
- as cutoff increases FDR ↑ b/c we start including more false pos.
- at p=1, FDR is 50%. (as everything is selected + we know 1/2 of the data set is false positives.

Hide

```
par(oldpar)
```

We again use the fact that in our simulated dataset we know exactly what the truth is. So we can study how the FDR would depend on the p-value cutoff. To this end, in the code above we first generate the sequence of p-value cutoffs we might want to consider (or not – we probably would not be that interested in cutoff of 0.8; but we still run the whole range solely for the purpose of seeing the complete picture). Next, for each value of the p-value cutoff, we count how many genes with no true difference have p-values below that cutoff (false positives, vector `n.same`) ; we also count how many genes that exhibit true difference have p-values below each of the cutoffs (true positives, vector `n.diff`). Those counts are drawn in the top two panels of the figure shown below. Note that the count of false positives grows linearly (which makes perfect sense: since the distribution of p-value in that case is uniform, then the count below a cutoff, which is essentially a CDF, grows linearly with the cutoff value). The count of true positives also reflects the shape of the corresponding p-value distribution: there are no true positives with arbitrarily small p-values, so the curve also always starts at 0. But since large(r) fraction of true positives have small p-values (the cusp in the distribution shown in the previous figure), the curve (which is again a CDF) moves up fast: by modest increase in the cutoff, we catch many of those true positives in the cusp.

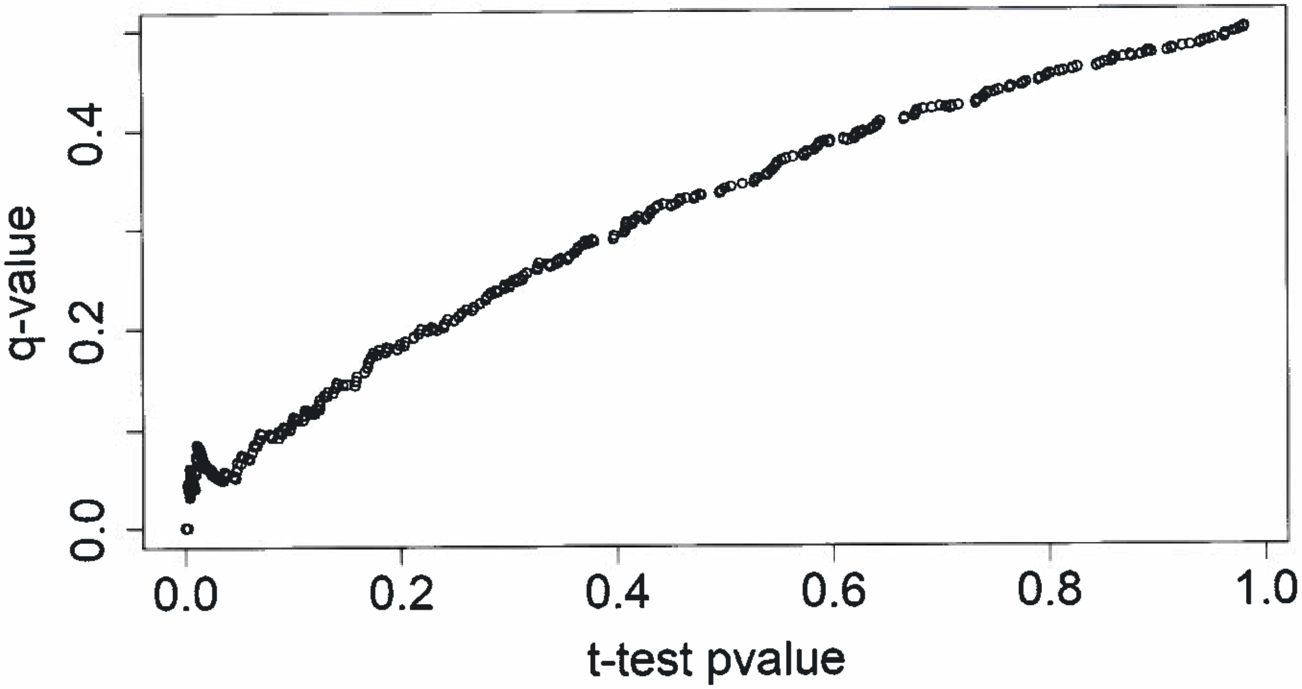
The plot in the bottom panel shows the ratio of false positives (n.same) and the total number of genes below any given p-value cutoff (`n.same + n.diff`), which is FDR, or q-value. Note how q-values start at low values at small p-value cutoffs, and gradually increase with the increase in the cutoff. The reason is of course that by increasing the cutoff, we are letting in more and more false positives, while adding fewer true positives. This can be also interpreted as increasing sensitivity (at p-value cutoff of 1, we select all the genes, including all true cases, of course), at the price of decreasing FDR. By looking at the FDR curve, it becomes clear that we can, at least in principle, adjust p-value cutoff in such a way that we get an optimal (at least for particular application and given the test we are running) balance between the number of cases we can detect and the fraction of false positives among the selected cases. Note that the FDR curve as in the figure above depends on the dataset (how much noise we have, how strong the effect is, etc), and the test itself. Indeed, for a given dataset, a more powerful test might exist, in principle, so that it would allow us to detect more true cases and have lower FDR at the same time.

Another important point to make a note of is that the q-value is the property of a selected set of cases. Only when we apply some cutoff, we select a set of cases, and only for that set the question "what is the fraction of false positives" is meaningful. Note that we used a p-value cutoff to select genes, but we could instead select all genes that exhibit at least two-fold change between mean expression levels in the two conditions, which is a less principled and likely suboptimal way of analyzing the data, but perfectly legal one, of course. We could similarly ask what the FDR is for that set of genes, it's a legitimate question (not necessarily with an easy-to-obtain answer).

In practice, q-value is often reported for each given p-value in the dataset (i.e. it looks like it is reported for each particular feature, gene in our case). The interpretation is as follows: if we have a gene with associated p-value p, the corresponding q-value is the FDR that we would obtain if we selected all the genes with p-values up to p (i.e. if we used p as a cutoff). Let us illustrate this with some code:

Hide

```
pval.cutoff <- c(pvals.200.same,pvals.200.diff)
n.same <- sapply(pval.cutoff,function(x) sum(pvals.200.same<x))
n.diff <- sapply(pval.cutoff,function(x) sum(pvals.200.diff<x))
qval <- n.same/(n.same+n.diff)
plot(pval.cutoff,qval,xlab="t-test pvalue",
 ylab="q-value",cex.axis=1.7,cex.lab=1.7,cex=0.8)
```



Note that the code is the same as above, just instead of specifying the vector of possible cutoffs externally, we use the very p-values from t-test, for each gene, as the "cutoff". Hence our calculated q-values correspond to each such "cutoff", and the resulting plot (see below) illustrates this relation. The benefit of this representation is that we are dealing with actual p-values as they come from the t-test (or any other test we are running), so every point represents an actual gene. For any value of the FDR we might want to set, we can select all genes with p-values up to the p-value of the specific gene that exhibits the desired q-value. For instance, it follows from the curve shown, that for our particular dataset we can select all the genes with individual p-values up to ~0.2 (!) if we want to keep an FDR (q-value) at respectable 20%.

5 qvalue package

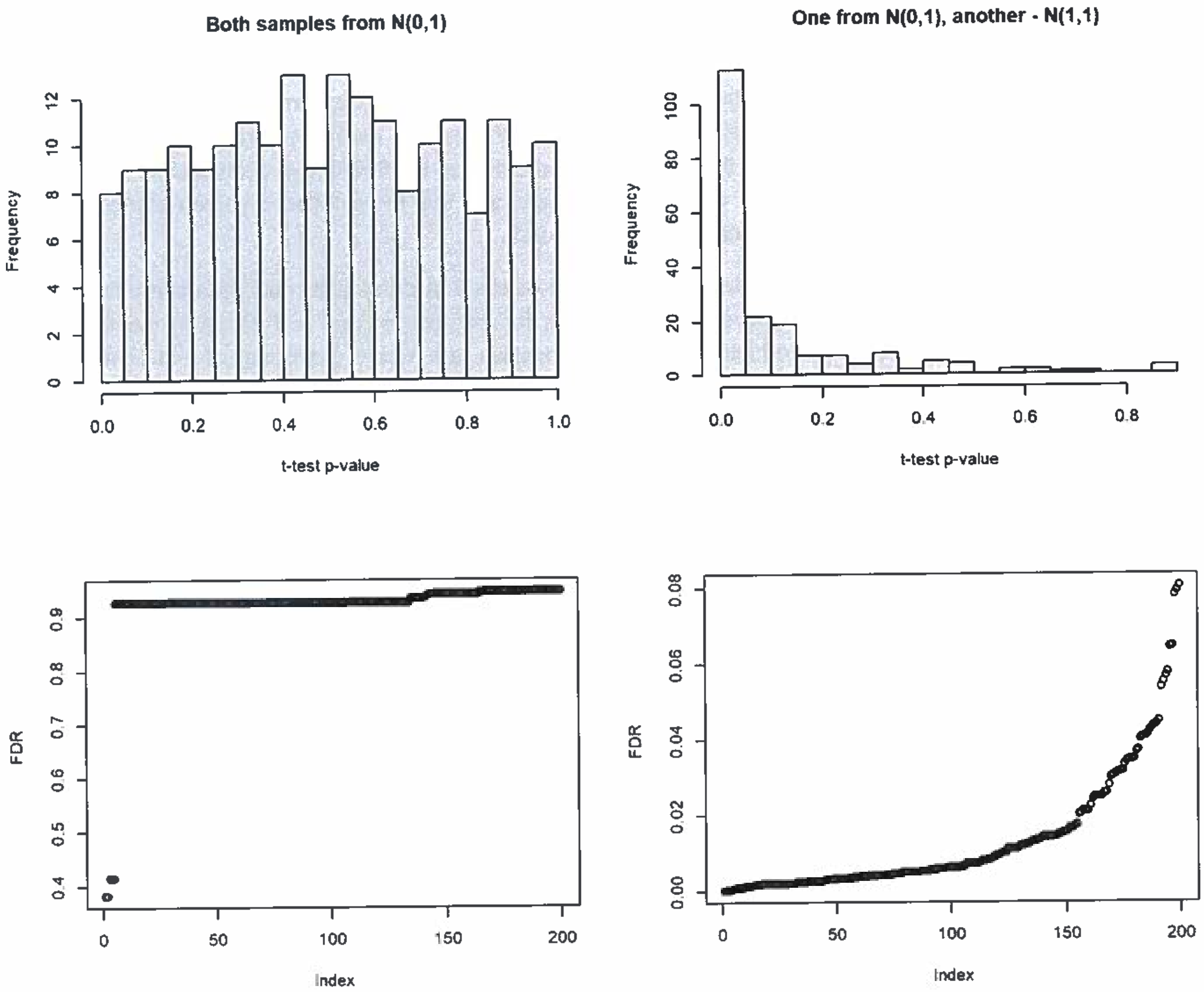
In the previous section we were taking a closer look at what q-value represents. In real life, we only have the whole dataset and no knowledge of the underlying truth, so we have to estimate the FDR from the data. There are many approaches aimed at this goal. Here we will consider just one. This approach is implemented in qvalue package, which is currently available for download from Bioconductor (with the usual method, `if (!require("BiocManager", quietly = TRUE)); install.packages("BiocManager"); BiocManager::install("qvalue")` . The method employed in qvalue library is described in detail in the following publication: Storey & Tibshirani (2003) PNAS 100:9440 (NOT a required reading).

The qualitative idea behind this approach is very straightforward: if we look at our earlier exercise (the mix of genes that change and genes that do no change between control and disease), we can try fitting the combined distribution of p-values (bottom panel in the figure, and that's the only information we would have in real life) as a sum of two distributions, one of which is uniform (that's what we expect from false positives). The far right tail of the combined distribution (where the contribution is predominantly from genes with no effect) should give us a decent idea of the height of the whole (uniform!) distribution of p-values of the genes with no effect, so that we can extrapolate to low p-values and estimate the false positive rate there.

The use of the qvalue package is really simple. Let's look at the code:

Hide

```
library(qvalue)
oldpar <- par(mfrow=c(3,2))
plot(hist(pvals.200.same,plot=F,breaks=20),xlab="t-test p-value",
main="Both samples from N(0,1)")
plot(hist(pvals.200.diff,plot=F,breaks=20),
xlab="t-test p-value",main="One from N(0,1), another - N(1,1)")
plot(sort(qvalue(pvals.200.same)$qvalues),ylab="FDR")
# qvalue(pvals.200.diff)
# [1] "ERROR: The estimated pi0 <= 0. Check that you have valid p-values
# or use another lambda method."
# [1] 0
plot(sort(qvalue(pvals.200.diff,lambda=0.5)$qvalues),ylab="FDR")
```



After loading the qvalue library, the evaluation is really simple: `qvalue(pvals)` will return an object with the field qvalues filled with q-values calculated for each pvalue in the vector pval. In the code fragment above, we peruse the numerical example we set up earlier, and we apply this function separately to the collections of t-test p-values computed for the genes that do not exhibit expression changes between the conditions (left column in the plot) and genes that do change (right column). In the top row we reproduce histograms of the corresponding p-value distributions. The next row shows calculated q-values (sorted), against their rank in the list.

It is instructive to see how the algorithm performs in the two extreme cases we are studying here. As we have noted above, it tries to fit the empirical distribution of the p-values as a mixture of two distributions, one of which is uniform (and thus accounts for cases where null is not rejected). In the left column, we pass a collection of p-values computed on the null cases only (no change between conditions). Due to finite experiment size (200 cases), we observe noticeable fluctuations in the p-value histogram, and it even throws the `qvalue()` function off: since it does not know that there are no positive examples in this case at all and enforces the fit with two distributions, it overfits to the noise. But the good news is that it does not overfit by too much: note that the smallest q-value reported is 75%, and by the time we select half of the genes, the calculated FDR is 95%.

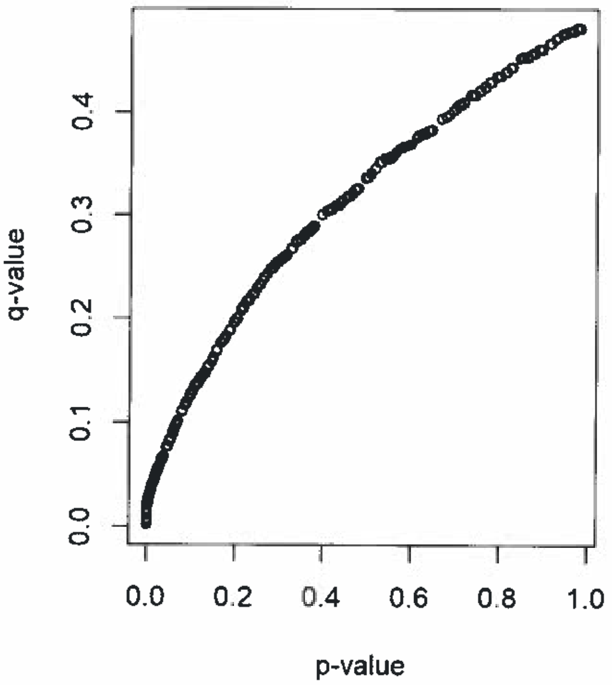
In contrast, if we use a set of p-values calculated for only those genes that do exhibit real difference between the conditions, the error comes from false negatives (type II error): for the lack of real null cases in the tail of the distribution at large p, the algorithm is tricked into deriving false positive rate from the existing tail that contains only real cases (false negatives). Still the FDR reported by the algorithm is very low at small p-value cutoffs and never exceeds ~5%, so the function is doing a pretty good job at detecting that all the cases are actually real.

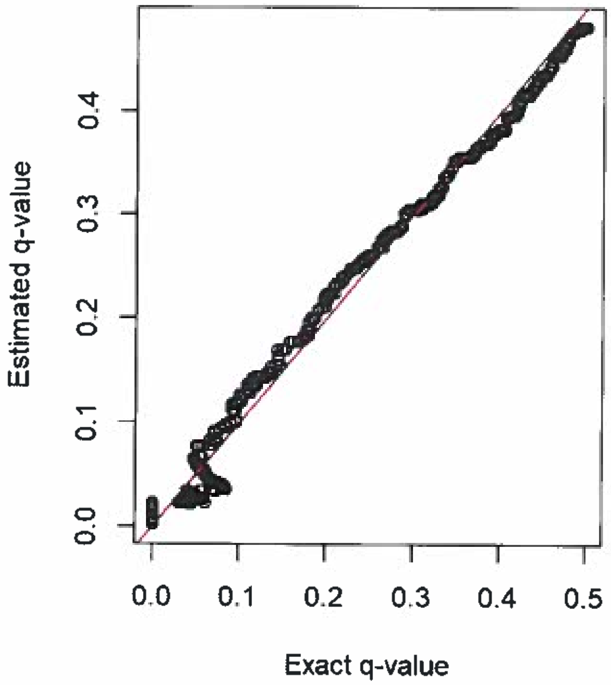
Note that in default mode the `qvalue()` function tries to autodetect which part of the distribution is due to null cases (no change in expression). This is a preferred way to run it, but in some cases the algorithm breaks: this was the case in the code above for the distribution of p-values from genes with actual changes only. The distribution is apparently too extreme (too much of a cusp, too little of a tail) for the `qvalue()` function to be able to detect the null cases (which in reality are absent in this case!!). It is possible to nudge the algorithm in such situations (or in general when you have a really good reason to do that), by manually specifying a value for parameter “lambda” (as we did in our example above). This parameter defines which part ($p>\lambda$) of the p-value distribution is due to the null. Note that if you set smaller lambda, the resulting FDR will be higher.

To conclude, let us apply `qvalue()` in more “realistic” scenario, using our full simulated dataset (200 genes with no change, and 200 genes with change). In the previous code fragment we already generated a vector pval.cutoff which is a concatenation of the 200+200 p-values from both sets of genes. We also calculated the exact q-values (based on the fact that we know which genes came from null and thus are false positives). Let us now apply `qvalue()` and see how the algorithm performs and compares to the truth, which is known in this case:

Hide

```
# just reproducing old code:
pval.cutoff <- c(pvals.200.same,pvals.200.diff)
n.same <- sapply(pval.cutoff,function(x) sum(pvals.200.same<x))
n.diff <- sapply(pval.cutoff,function(x) sum(pvals.200.diff<x))
qval <- n.same/(n.same+n.diff) # exact q-values, we know the truth!
# here new part starts: estimation; suppose we do not know the
# truth and just ran qvalue on the whole dataset:
qval1 <- qvalue(pval.cutoff)$qvalues
oldpar=par(mfrow=c(1,2))
plot(pval.cutoff,qval1,xlab="p-value",ylab="q-value",cex=0.8)
plot(qval,qval1,xlab="Exact q-value",ylab="Estimated q-value")
abline(a=0,b=1,col='red')
```





In the code, we apply `qvalues()` to our combined set of p-values. Then we plot estimated q-values vs corresponding p-values. Finally we plot q-values estimated from the data by `qvalues()` function vs exact q-values. It is seen that the latter dependence is close to linear (red straight line is the diagonal), and while `qvalues()` overestimates FDR in our particular example (nobody is perfect!), the algorithm still performs reasonably well.