Senior Thesis in Mathematics

Simulation and Application Study of Online False Discovery Control Methods

Author:
Nick Waalkes

Advisor:
Dr. Jo Hardin

Submitted to Pomona College in Partial Fulfillment
of the Degree of Bachelor of Arts

April 18, 2022

Abstract

In modern multiple hypothesis testing problems, often traditional false discovery rate control algorithms require structure that does not align with the structure of experimental data. This has led to the development of online false discovery rate control algorithms since 2015. These methods allow for a potentially infinite number of hypotheses to be tested in sequence. Online algorithms, including LORD, LOND [4], and SAFFRON [7], have theoretical guarantees but limited simulation studies. The goal of this paper is to elaborate on these algorithms and compare the performance of online false discovery rate control algorithms against the Benjamini-Hochberg algorithm.

Contents

Chapter 1

Introduction

An important feature of modern statistics is hypothesis testing. Hypothesis testing is a scheme that attempts to determine whether differences in observed data from the hypothesized value are due to random chance or due to the data originating from an actual effect. A hypothesis test is divided into two components: the null hypothesis, H_0, and the alternate hypothesis, H_A. While there are several types of hypothesis tests, the type of interest to this paper is constructed as:

$$H_i = \begin{cases} H_0 : \theta = \theta_0 \\ H_A : \theta \neq \theta_0 \end{cases}$$

Where θ is the parameter of interest and θ_0 is the hypothesized value of θ imposing that there is no effect.

In order to make decisions for hypothesis tests, either determining that there is an effect (rejecting the null hypothesis) or determining that the difference in data is not large enough to declare that there is an effect (failing to reject the null hypothesis), each hypothesis test has an associated p-value. This p-value is the probability that, given that the null hypothesis is true, data this strange or stranger would be generated. The hypothesis tests in this paper are two-sided. This means that we are concerned with the statistics being larger or smaller than the observed statistic. When deciding whether to reject the null hypothesis or not, the p-value is compared to a predetermined cutoff. If the p-value is smaller than that cutoff, the null hypothesis is rejected.

When conducting hypothesis tests, two critical ideas are type-one and

type-two errors. A type-one error occurs when the null hypothesis is rejected but it is actually true. In essence, this is a false positive. A type-two error occurs when a statistician fails to reject the null hypothesis, but the alternate is actually true. In essence, a false negative.

Given these two separate error types, which is more relevant to control? Controlling the type-one error rate would allow scientists and statisticians to be confident that only a set proportion of their discoveries over time will be false, but this may cause a lower proportion of true hypotheses to be discovered. Alternatively, controlling the type-two error rate would allow statisticians to make certain that if an alternate hypothesis was true, it would be rejected at some predetermined level. Controlling type-two errors, however, would result in not controlling the proportion of nulls that are falsely rejected. In most instances of scientific research, hypothesis tests are only published if the null hypothesis is rejected. Since hypothesis testing was constructed for science, the decision was made to control type-one errors in order to ensure the veracity of the scientific literature. A type-one error can result in incorrect literature and understanding, whereas a type-two error likely will obscure the truth, but not result in incorrect application.

In modern biomedical research, an important inquiry is the multiple hypothesis testing problem, essentially posing many research questions simultaneously. When testing multiple hypotheses, the probability of a type-one error, which is essentially a false positive, increases. If researchers are interested in controlling the rate of type-one errors to some constant, generally 0.05, additional controls must be placed on the decision rules for rejecting hypotheses.

Let us consider a concrete situation in which multiple hypothesis testing is employed in biomedical research:

A scientist is interested in understanding which genes, if any, play an important role in the rate of breast cancer in the population. A common technique is a genome-wide association study (GWAS). Such a procedure involves collecting a large sample size of individuals, both breast cancer survivors and individuals without breast cancer. Then, the scientist will sequence the entire genome of all participants in the study to attempt to draw connections between the presence of genes and breast cancer risk. GWASs can offer incredible insight into disease risk, but there are two important considerations for an example like this:

1. The number of hypotheses are unknown and potentially extremely nu-

merous. Due to the vast number of potential genes involved with a given disease, combined with the continued introduction of new participants in a GWAS, having all hypotheses known before conducting the experiment would limit the research potential.

2. Research is being conducted throughout the world simultaneously. With a disease as prevalent as breast cancer, research labs around the country are constantly incorporating new data and testing new hypotheses. Being able to incorporate the research of other labs into new research generates an enormous advantage in advancing scientific knowledge.

When trying to control the type-one error rate, there are two metrics that are primarily considered: family-wise error rate (FWER) and false discovery rate (FDR). The FWER is a measure of the probability of having at least one type-one error among the tested hypotheses. When controlling the FWER, the most common method is the Bonferoni Inequality:

For n hypothesis tests $H_1, H_2, ..., H_n$, adjust the p-value cutoff as follows:

$$\alpha_j = \min\{\frac{\alpha}{n}, 1\}$$

Essentially, the original p-value cutoff gets divided by the number of hypotheses tested to generate a new adjusted cutoff which is used as the decision rule.

Chapter 2

Online FDR Control Algorithms

2.1 Online and Offline FDR

To control false positives, the primary metric that this paper focuses on is FDR. Generally, the FDR is defined as:

$$\text{FDR} = E[\text{FDP}] = E[\frac{V}{R}] \tag{2.1}$$

Where V is a random variable that represents the number of type-one errors and R is a random variable that represents the total number of rejections.

In order to understand online FDR control, we first must understand the difference between offline and online FDR.

Offline FDR is what is normally considered when thinking about FDR. The FDR is associated with a group of hypotheses tested simultaneously and each multiple hypothesis test is considered independently of all other multiple hypothesis tests. Additionally, because the hypotheses are decided before the algorithm is run, the order in which the hypotheses are tested can be changed to increase power.

In contrast, online FDR considers hypotheses across many research questions. It controls FDR over a sequence of hypotheses or batches of hypotheses. Just as offline FDR expanded the scope of control from individual hypotheses to a single batch of hypotheses, online FDR extends the scope one step further to control across multiple research questions and projects.

In essence, offline FDR is considered within a single batch of hypotheses, whereas online considers a sequence of hypotheses or batches of hypotheses.

2.2 Online FDR Control

This section will go into detail regarding the existing online false discovery rate control methods. This includes descriptions of LORD, LOND [4], and SAFFRON[7] and their technical conditions. This section will also include some proofs regarding the effectiveness of these methods, but will likely only feature a few of these algorithms.

The advancements in online false discovery rate control can be reflected generally in the various algorithms that have been described in the literature. Understanding the function and evolution of these algorithms can give insight into the effectiveness and reasoning behind this online control.

2.2.1 Why is Online FDR control Useful?

Successful offline control methods, like the Benjamini-Hochberg procedure [1], exist and can effectively control false discovery rate in single sets of multiple hypothesis studies. This poses the question of why online methods are being explored at all. There are several changes to modern research techniques that cause online methods to be necessary. First, modern research has become increasingly decentralized. With labs across the world researching similar topics, being able to incorporate different data and hypotheses into a study without compromising the effectiveness of FDR control methods would allow for more effective incorporation of work by other researchers. Second, the increased computing power available to researchers has allowed the study of high-throughput (HT) data. HT data can be defined as data in which large amounts of information is gathered simultaneously. The more instances that data is collected, the more difficult it is to consider offline FDR across its entirety. Biomedical data are often expansive, high-dimensional, and very fast to collect. Researchers are attempting to interpret a fire hose of data. Attempting to dissect and categorize HT data is extremely difficult. With offline control methods, a testing period needs to be rigorously defined because all hypotheses must be tested at once and no more can be added. At some point, a scientist must decide when they have enough data and ignore future data for the hypotheses in question. Online control methods can be an

alternative to having to draw distinct beginnings and ends to testing periods.

2.2.2 LORD and LOND

LORD (Levels based On Recent Discoveries) and LOND (Levels based On Number of Discoveries) are the first attempts by statisticians to control online false FDR. They appeared in Adel Javanmard and Andrea Montanari's 2015 paper [4]. These two algorithms show the fundamental methodology that is used in many subsequent attempts to control FDR. Because of its impact, let's breakdown the LOND and LORD algorithms in detail to understand the approach:

Given a set of hypotheses $H_1, H_2, ...$, choose a desired significance level α and a non-negative sequence $(\beta_i)_{i=1}^{\infty}$ such that $\sum_{i=1}^{\infty} \beta_i = \alpha$ The significance threshold for p_i, the ith p-value is:

$$\alpha_i = \beta i(D(i-1)+1)$$

Where $D(n)$ is defined as the number of significant results from the first n hypotheses tested.

When employing this algorithm, we get the following:

Theorem 2.1. *If the p-values for the hypotheses are independent, the LOND procedure controls FDR at a level less than or equal to* α

The goal of the proof for the LOND algorithm is to show that, if the procedure is followed, $\text{FDR}(n) \leq \alpha$ for all $n \geq 1$ [4]. That is to say, the FDR over the first n hypothesis tests will be controlled at the nominal cutoff. Before elaborating on the proof, a few functions must be defined. $V(n)$ is a function that outputs the number of false rejections through n hypothesis tests. $R(n)$ is a function that outputs the total number of rejections through n hypothesis tests. FDR is defined as:

$$\text{FDR}(n) = E\left[\frac{V(n)}{\text{Max}\{R(n), 1\}}\right]$$

$$\text{FDR}(n) = E\left[\sum_{j=1}^{n} \frac{F_j}{\text{Max}\{R(n), 1\}}\right]$$

We can multiply by $\frac{\alpha_j}{\alpha_j}$ in order to incorporate the adjusted cutoff:

$$\text{FDR}(n) = E\left[\sum_{j=1}^{n} \frac{F_j}{\alpha_j} \cdot \frac{\alpha_j}{\text{Max}\{R(n), 1\}}\right]$$

We now need to show that each of these components of the product multiply to be less than β_j. First let's look at $\frac{\alpha_j}{\text{Max}\{R(n),1\}}$:

We know that

$$\text{Max}\{R(n), 1\} \leq \text{Max}\{R(j), 1\}$$

Since the number of total rejections, $R(x)$ is monotonically increasing. By construction of α_j:

$$\frac{\alpha_j}{\text{Max}\{R(n), 1\}} = \frac{\beta_j(R(j-1)+1)}{\text{Max}\{R(n), 1\}} \leq \beta_j$$

For the second half of the product, $\frac{F_j}{\alpha_j}$, consider the condition placed upon the hypotheses:

$$P_{\theta_j=0}(T_j = 1|R(j-1)) \leq E[\alpha_j|R(j-1)]$$

This condition is satisfied by the independence of p-values. Consider:

$$E\left[\frac{F_j}{\alpha_j}\right] = E\left[E\left[\frac{F_j}{\alpha_j}|R(j-1)\right]\right] \leq E[1] = 1$$

Thus,

$$\text{FDR}(n) \leq \sum_{j=1}^{\infty} \beta_j = \alpha$$

[4]

Because FDR is an unobservable value, proofs regarding the long-run behavior of the false discovery proportion. By bounding the infinite sequence β so that its sum equals α, the construction of α_i allows for guarantees that the FDR is controlled for that hypothesis together with each previous hypothesis at the nominal cutoff.

In practice, the way that LOND and LORD work is to leverage a bounded infinite series. The control method needs to hold for a potentially infinite

number of hypotheses. In order to achieve FDR control with potentially infinite hypotheses, it balances a monotonically decreasing sequence β_i with an increasing factor based on the number of rejections thus far in the testing sequence. As the number of hypotheses tested increases, β_i decreases and $D(i-1)$ increases. This prevents the sequence of critical values from approaching zero or increasing to infinity except in very specific cases (for instance, if all of the hypotheses tested fail to reject the null).

Next, let's examine LORD. Rather than focusing on the number of previous rejections like LOND, LORD only considers how recently a hypothesis has been rejected[4]. The cutoff for LORD is determined as:

Given a set of hypotheses $H_1, H_2, ...$, choose a desired significance level α and a non-negative sequence $(\beta_i)_{i=1}^{\infty}$ such that $\sum_{i=1}^{\infty} \beta_i = \alpha$
The significance threshold for p_i, the ith p-value is equal to:

$$\alpha_i = \beta i$$

Until the first discovery occurs. After this first discovery, the new adjusted cutoff becomes:

$$\alpha_i = \beta_{i-\tau_i}$$

Where τ_i is defined as:

$$\tau_i := \max\{l < i | H_l \text{ Is Rejected}\}$$

Theorem 2.2. *If the p-values for the hypotheses are independent, the LORD procedure controls FDR at a level less than or equal to α[4]*

The intuition from this proof is a bit more complex than LOND because LORD relies on random indicator functions. We have this string of inequalities that result in the theoretical result. Rather than using all of the past rejections to inform the cutoff, LORD only uses how recently a hypothesis is rejected.

The cutoff is adjusted down the infinite sequence β as hypotheses are failed to be rejected. Each time a hypothesis is rejected, the cutoff is reset to β_1. The process continues as the sequence of hypotheses are inputted into the control algorithm.

2.2.3 SAFFRON

SAFFRON is an adaptation based on the original LOND and LORD algorithms [7]. The algorithm expand upon the original structure of the LORD and LOND algorithms, while adding adaptive components. The following is the SAFFRON algorithm:

SAFFRON

1. Given a target FDR level α, the user first picks a constant $\lambda \in (0, 1)$, an initial wealth $W_0 < (1-\lambda)\alpha$, and a positive non-increasing sequence $\{\gamma_j\}_1^\infty$ summing to one.

2. We then need to determine which p-values are candidates. Define the indicator function $C_t := 1(P_t \leq \lambda)$, and τ_j be the time of the j-th rejection. We also must define the number of candidates after the j-th rejection as $C_{j+}(t) := \sum_{i=\tau_j+1}^{t-1} C_i$

3. SAFFRON first allocates $\alpha_1 = \min\{\gamma_1 W_0, \lambda\}$. Then at time $t = 2, 3, ...$ it allocates as follows:

$$\alpha_t := \min(\lambda, \bar{\alpha}_t)$$

where

$$\bar{\alpha}_t := W_0\gamma_{t-C_{0+}} + ((1-\lambda)\alpha - W_0)\gamma_{t-\tau_1-C_{1+}} + \sum_{j\geq 2}(1-\lambda)\alpha\gamma_{t-\tau_j-C_{j+}}$$

This algorithm uses an infinite sequence that forms a bounded series with a few additions. λ is a maximum value to test p-values. This means that regardless of what the α_t value for a hypothesis is, SAFFRON will not even consider it if the p-value is less than λ. Since SAFFRON is able to discard certain hypotheses using λ, the algorithm sees fewer null hypotheses. This means that the cutoff does not suffer as much from failures to reject and can reject more true hypotheses. This results in higher potential power [7].

Chapter 3

Offline Control Methods

3.1 Prominent Offline FDR Control Algorithms

In order to control false discovery rate, there are several accepted offline methods, however far-and-away the most popular is the Benjamini-Hochberg[1] (BH) class of algorithms. BH employs the offline assumptions when controlling false discovery rate, which require finite and known hypotheses and p-values.

As a reminder, the quantity that we want to control is the false discovery rate, which is defined as:

$$\text{FDR} = E[\text{FDP}] = E\left[\frac{V}{R}\right] \tag{3.1}$$

Where V is a random variable that represents the number of type-one errors and R is a random variable that represents the total number of rejections.

An important concept when understanding the effectiveness of FDR control algorithms is the difference between nominal FDR control rate and effective FDR control rate. While each algorithm has the theoretical result of controlling FDR at or below the chosen cutoff α, the reality can be different. The nominal control rate is equal to the chosen cutoff α. The effective control rate is the actual FDR calculated from simulation studies. Large differences between these two values can indicate that technical conditions are not being held or the algorithm is being employed in an inappropriate situation.

3.1.1 Benjamini-Hochberg Algorithm

To gain an understanding of the most commonly used FDR control method, let's highlight the procedure of the Benjamini-Hochberg algorithm. The Benjamini-Hochberg algorithm is as follows[1]:

Benjamini-Hochberg Algorithm

1. Given a finite set of hypotheses $H_1, H_2, ..., H_n$, generate $p_1, p_2, ..., p_n$, the corresponding independent p-values.
2. Reorder the p-values from smallest to largest, creating the new set $\{p_i'\}_1^i$.
3. Generate α_i, the adjusted p-value cut-offs for each p_i' using $\alpha_i = \frac{l_i \alpha m}{}$. Where l_i is the rank of the ith p-value, m is the total number of p-values, and α is the nominal false discovery rate level.
4. Find the largest p-value that is less than it's adjusted cut-off. Reject all hypotheses with p-values smaller than this maximal value and fail to reject all p-values larger than it.

Using the BH procedure, the false discovery rate can be controlled at a specified level [1]. It is important to remember that the Benjamini-Hochberg procedure and the online control methods assume the independence of p-values, which is often violated in RNASeq data.

3.1.2 Why Not Extend Offline Algorithms Directly

The existence and prominence of the BH algorithm begs the question: why can we not just apply the BH algorithm multiple times as additional hypotheses are put forward. In order to understand why this method does not maintain the control of normal BH, let's dig a bit deeper into what the false discovery rate is.

Consider a situation where we are testing $2n$ hypotheses. The false discovery rate, in the context of actual experiments, is defined as:

$$\text{FDR} = E[Q] = E\left[\frac{V}{R}\right]$$

Where Q is the false discovery proportion, V is a random variable representing the number of false rejections from the $2n$ hypotheses, and R is a random variable representing the total number of rejections from the $2n$ hypotheses.

Using the BH procedure ensures that $E[Q] \leq \alpha$ where α is the nominal FDR control rate. Let's now consider a slightly altered situation in which we now consider two separate batches with n hypotheses. Let Q' be the false discovery proportion for both batches counted together defined as

$$Q' = \frac{V_1 + V_2}{R_1 + R_2}$$

Where V_i is the number of false rejections for batch i and R_i is the total number of rejections for batch i.

Let's examine the FDR of both batches together.

$$\text{FDR} = E[Q'] = E[\frac{V_1 + V_2}{R_1 + R_2}]$$

$$\text{FDR} = E[Q'] = E[\frac{V_1}{R_1 + R_2}] + E[\frac{V_2}{R_1 + R_2}] \leq E[\frac{V_1}{R_1}] + E[\frac{V_2}{R_2}] \leq 2\alpha$$

Once both batches are considered, Q' is only controlled beneath 2α. This does not mean that the naively combined BH algorithms will always result in the higher FDR control rate. The more replicates that are present when conducting the experiment, the more that this effect can be diluted. However, the theoretical control afforded by the BH procedure are no longer guaranteed [9].

Considering long-term research by individuals and labs, this problem also applies. When labs or individuals ask multiple research questions over time, they are applying the BH algorithm in multiple batches. If we consider the FDR over all experiments over time, the FDR control offered by BH is lost.

Chapter 4

Simulation Study

4.1 Simulation Purpose

In actual experiments, FDR and power are immeasurable metrics. If we possessed knowledge of which hypotheses were truly null or non-null, there would be no need for any statistical tests. Thus, in order to understand the performance of online and offline FDR control algorithms, simulation studies are necessary. Using simulated data based off of estimated parameters from real data allows for the knowledge of which hypotheses are null and which are non-null. This allows for the direct calculation of both FDR and power.

One modern biomedical research technique that online FDR control algorithms are particularly well-suited for is the collection of RNASeq data. The collection of RNASeq data involves exposing a cell to control and sample conditions (such as a change in temperature, the presence of a drug, a lack of nutrients, etc.) and measuring how much mRNA is expressed from each gene of interest in the various conditions. mRNA is the mechanism by which the gene is expressed, so the more mRNA is present in the cell, the more the gene is being activated. Understanding how different genes are expressed under different conditions can give insights into gene function and potential drug efficacy.

RNASeq data is often modeled as negative binomial data, parameterized with μ and σ^2. The simulations carried out for this paper simulated RNASeq data using mean and dispersion estimates generated from Pickrell et al [6]. Using these parameter estimates of the negative binomial model, we gain the ability to generate data that comes from the same population that Pickrell

was examining. In the paper, Pickrell et al sought to map of the human transcriptional landscape. Essentially, which of the approximately 20,000 human genes are being activated in different situations. Thus, the simulation for this paper will specifically apply to human RNASeq data, although it is likely to be generalizable.

For all comparisons made in this paper, the offline FDR control method employed will be Benjamini-Hochberg and the online FDR control method will be SAFFRON. The simulation procedure will be as follows:

Simulation Procedure

1. Generate batch B_i of 1000 hypothesis tests, with a proportion of true hypotheses equal to π
2. Use differential expression analysis via DESeq2 to associate each hypothesis in B_i with a p-value
3. Generate offline adjusted p-values for B_i using BH on the p-values of B_i
4. Generate online adjusted p-values for B_i using SAFFRON on all tests in batches B_j such that $j \leq i$
5. Generate a list of rejections for both the offline and online methods
6. Compute the FDR and power that result from each FDR control method
7. Repeat this procedure for $1 \leq i \leq r$

This algorithm seeks to duplicate how offline and online FDR control methods would be employed in scientific inquiry. Each batch B_i is analogous to a current research question that is being explored. For example, B_1 could be examining the effect of higher temperatures on gene expression and B_2 could be examining the effects of a more acidic environment. Additionally, having 1000 hypotheses for each batch attempts to replicate the high-throughput data that is characteristic to RNASeq data. In practice, BH is applied to individual batches with no consideration for past results. This is why the offline FDR control method only considers the p-values of B_i,

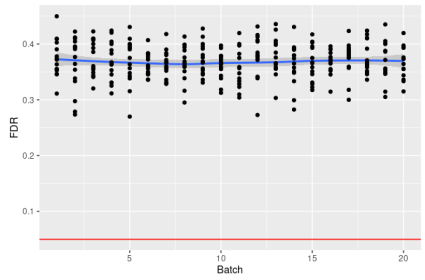

Figure 4.1: FDR over 15 Runs of 20 Batches with No Control Algorithm

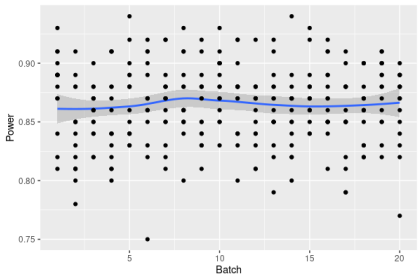

Figure 4.2: Power over 15 Runs of 20 Batches with No Control Algorithm

whereas online FDR control methods, by their sequential nature, consider all previously tested hypotheses in order. SAFFRON is run on all batches before the current batch before attempting to control B_i.

4.2 Baseline Comparison

The first simulation of interest is a direct comparison of of the offline and online control methods given typical RNASeq conditions. The proportion of non-null hypotheses, π, will equal 0.1. This low proportion of non-null hypotheses is typical for exploratory studies that RNASeq data is usually employed in. The total number of runs, r, is set to 30 to try and understand the characteristics of BH and SAFFRON over many experiments, as the benefits of online FDR control are seen over the course of many research questions.

4.2.1 No FDR Control

Before exploring the results of the two FDR control methods, let's quickly examine the results of the simulation when no FDR controls are put into place. We can see in figure 4.1 that the FDR over thirty batches has a mean of just over 0.35. This means that thirty five percent of the rejections were false. Such a result is terrible for science. We are seeing seven times more type-one errors than the nominal cutoff. This demonstrates the need to employ FDR control algorithms, especially when testing this many hypotheses. A lack of FDR control does result in high power, as seen in figure 4.2. The

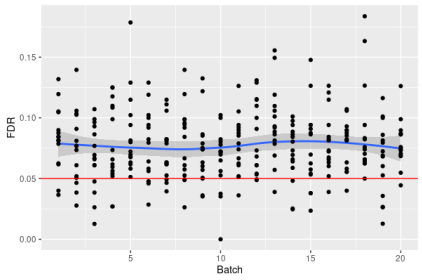

Figure 4.3: FDR over 15 Runs of 20 Batches with BH Control Algorithm

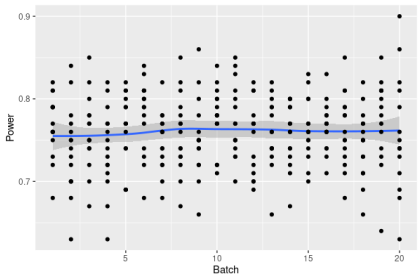

Figure 4.4: Power over 15 Runs of 20 Batches with BH Control Algorithm

mean power across the thirty batches is just over 0.85. So we are rejecting eighty five percent of true hypotheses. This is to be expected. Failing to put in FDR controls means that more hypotheses will be rejected, both null and non-null, which implies that not controlling FDR will result in a higher power than controlling FDR.

4.2.2 Offline FDR Control

Let's now apply the BH algorithm separately to each of the thirty batches and examine how the FDR and power behaves.

The BH algorithm works well in controlling FDR across the thirty batches. As seen in figure 4.3, the mean FDR is about 0.08. Thus, only about eight percent of rejections are false. However, the variance between batches is large. A reasonable portion of batches have an FDR of at least twelve percent, with one above sixteen percent. The power is slightly lower than not employing an FDR control method, with a mean of approximately 0.75, but this is still extremely good for such a low proportion of non-nulls.

4.2.3 Online FDR Control

Finally, let's examine the FDR and power when running the SAFFRON algorithm. The SAFFRON algorithm seems to perform well in these conditions. The mean FDR across the thirty batches is about 0.075 as seen in figure 4.5. The variance of the FDR, however, is much lower as more batches are tested. The power is a bit lower than the offline control methods, with a mean of

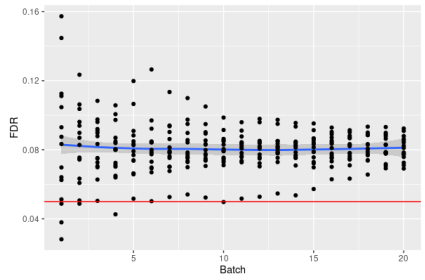

Figure 4.5: FDR over 15 Runs of 20 Batches with SAFFRON Control Algorithm

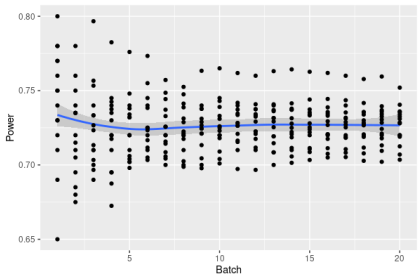

Figure 4.6: Power over 15 Runs of 20 Batches with SAFFRON Control Algorithm

0.72, but this difference is not major compared to the reduction in variance of the FDR.

4.3 Domain Expertise Comparison

One of the properties of online FDR control methods that makes them so attractive is the ability to leverage domain expertise more effectively. When employing BH, domain expertise can only be leveraged in the selection of hypotheses. The online methods though can reorder the way that hypotheses are tested. In the next simulation, the proportion of non-null hypotheses, π, will equal 0.5 for the first nine batches, after which π will switch to 0.1. This is designed to simulate a scientist's ability to first test hypotheses that they are more confident in. This allows the online control algorithm to leverage this higher initial true rejection rate to perform better under the later, worse conditions. The total number of runs, r, is set to 100 to try and understand the characteristics of BH and SAFFRON over many experiments and how the implementation of domain expertise can affect FDR and power.

4.3.1 No FDR Control

As before, let's understand the performance of FDR and power when no control algorithms are implemented under these conditions. We can see in figure 4.7 that the proportion of non-null hypotheses has a massive impact on the uncontrolled FDR. The FDR for the four batches with $\pi = 0.5$ sits at just above 0.05, the cutoff. However, once the proportion of non-nulls

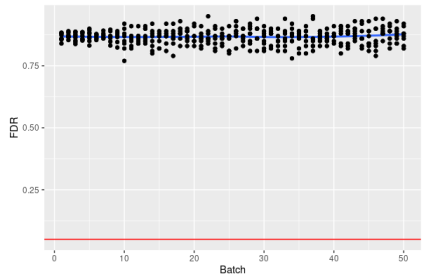

Figure 4.7: FDR over 8 Runs with 50 Batches with No Control Algorithm

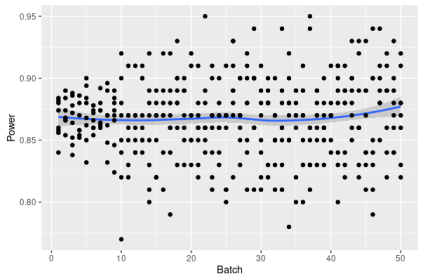

Figure 4.8: Power over 8 Runs with 50 with No Control Algorithm

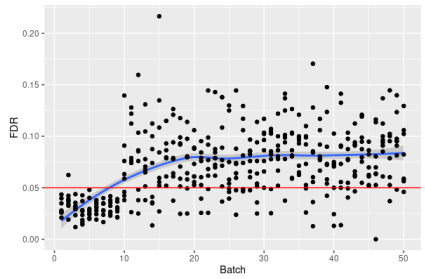

Figure 4.9: FDR over 8 Runs with 50 Batches with BH Control Algorithm

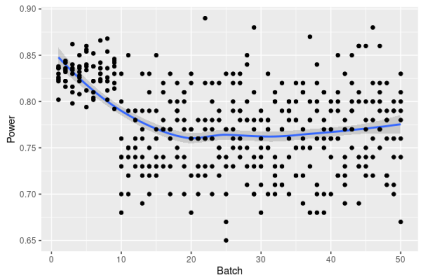

Figure 4.10: Power over 8 Runs with 50 Batches with BH Control Algorithm

decreases back to 0.1, we see similarly poor performance as we did when $\pi = 0.1$ for all batches. FDR control methods are still necessary when some domain expertise is put into place. A lack of FDR control once again results in high power, as seen in figure 4.8. The mean power across the thirty batches is about 0.875.

4.3.2 Offline FDR Control

Let's now apply the BH algorithm to each of the thirty batches individually and examine how the FDR and power behaves.

As seen in figure 4.9, the BH algorithm has similar structure to the uncontrolled case. The first nine batches have low variance and low FDR. After π decreases back to 0.1, the performance looks similar to the original condi-

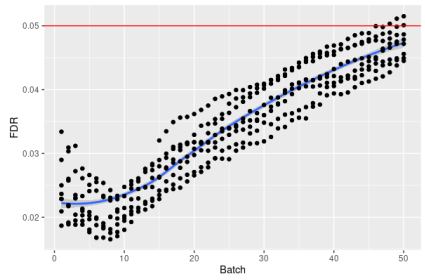

Figure 4.11: FDR over 8 Runs with 50 Batches with SAFFRON Control Algorithm

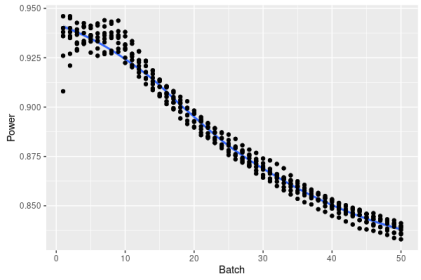

Figure 4.12: Power over 8 Runs with 50 Batches with SAFFRON Control Algorithm

tions. This is exactly the behavior we would expect given that each of the batches are considered independently of previous batches. The variance of the FDR is still high once the proportion of non-nulls increases back to the original values. The power is similar to the non-domain expertise example.

4.3.3 Online FDR Control

Finally, let's examine the FDR and power when running the SAFFRON algorithm. SAFFRON displays fascinating structure when domain expertise is implemented. Rather than a stark gap between the low and high proportion of non-nulls as seen in the offline and no control methods, the FDR behavior appears to be monotonic as seen in figure 4.11. SAFFRON allows scientists to leverage the domain knowledge beyond the experiments that the domain expertise applies to. Even though the FDR is increasing throughout the 100 batches, the FDR performs better than offline mean in every instance. Similarly in figure 4.12, the power begins much higher than BH, and although it decreases, the power of each batch remains above the maximum power when using offline FDR control methods. This domain expertise shows the true potential of the online FDR control algorithms. Domain knowledge in a few experiments improves the performance of all other experiments.

Chapter 5

Conclusion

In the conclusion, I want to focus on what this really means. How worried should we be about the false discovery rate of multiple hypothesis tests? Additionally, should online control methods be explored further, and if so, how would it look to implement them into modern biomedical research? This subject is far-reaching and important, where should we go from here?

It is important to remember that FDR is an expected value of the false discovery proportion. This means that for each individual research question, the FDR is not guaranteed to be below the nominal control rate. This means that one of the primary statistics of interest should be the variance of the false discovery proportion. Offline control algorithms have been a staple of research since Benjamini and Hochberg released their algorithm in 1996. This was a landmark paper in statistics and certainly greatly contributed to the scientific process. And while the BH procedure works well, understanding the current status of data analysis can lead to opportunities to develop more specialized algorithms. What makes FDR control algorithms powerful is the ability to leverage the structure and generation of the data. Online FDR control is much more specific than Benjamini-Hochberg. In situations where there is a large amount of domain expertise, there is a large amount of HT data, or there is research being done by many labs throughout the world, online FDR control can be implemented to take advantage of this structure.

While online control algorithms are still being developed, they pose a new paradigm that could aid in new, HT research that is being conducted in many fields around the world. If the goal of statistics is to clarify truths from data, the avoidance of type-one errors should be a top priority. Online FDR control offers new tools to scientists and statisticians to take advantage of

existing structure within data and have more flexibility in hypothesis choice. More effective and specialized tools can result in better science which will result in a better world and I believe that online hypothesis testing can be a small part of that better world.

Appendix A

Simulation Code

```
knitr::opts_chunk$set(echo = TRUE)

# Load libraries
library(DESeq2)
library(onlineFDR)
library(seqgendiff)
library(compcodeR)
library(tidyverse)

my_alpha <- 0.05
n_samples <- 15
n_batches <- 10
n_in_batch <- 10000
pi_de <- 0.1


generateData <- function(numHyp, diffRat, repID, nSamples = n_samples){


  # numHyp: Number of hypotheses tested for this dataset
  # diffExp: Number of hypotheses that are truly differentially expressed in the
  # nSamples: Number of each experimental condition (control and treatment)

  #Create a vector that describes experimental conditions. In this case, control
  condVec <- c(rep("control", nSamples), rep("treatment", nSamples))
```

```
    # Set seed for simulated data reproducability. Only uncomment if running fin
    #set.seed(i)

    # Create synthObject, an object that contains the count matrix, differential
    # Important notes about the inputs of this function: fraction.upregulated re

  synthObject <- generateSyntheticData(dataset = paste0("sim", repID, "-", diffRat
                                       n.vars = numHyp,
                                       samples.per.cond = nSamples,
                                       n.diffexp = numHyp*diffRat,
                                       repl.id = repID,
                                       seqdepth = 1e7,
                                       fraction.upregulated = 0.5,
                                       between.group.diffdisp = FALSE,
                                       filter.threshold.total = 1,
                                       filter.threshold.mediancpm = 0,
                                       fraction.non.overdispersed = 0,
                                       output.file = "OutputTestSynth.rds")

    # Add column headers with the experimental conditions so that the differenti
  colnames(synthObject@count.matrix) <- condVec

  # Create a vector that contains which genes are differentially expressed by in
#  diffExpressionDf <- synthObject@variable.annotations %>%
#    mutate(gene_id = paste0("g", row_number()))
  # Save the generated count matrix as synthData
#  synthData <- synthObject@count.matrix
#  rownames(synthData) <- paste0("g",seq(numHyp))
  # Return a list that contains the count matrix, sample conditions, and differe
  synthObject@variable.annotations['Row.names'] <- paste0('g',seq(from=1,to=length
  return(synthObject)
}

DESeqPipeline <- function(countMatrix, conditions) {

  colData <- data.frame(condition = conditions) %>%
```

```
    mutate(condition = as.factor(condition))

  ddsFromMatrix <- DESeqDataSetFromMatrix(countData = countMatrix,
                                          colData = colData,
                                          design = ~ condition)

  dds <- DESeq(ddsFromMatrix)
  res <- results(dds)

  #left_join
  resData <- merge(as.data.frame(res), as.data.frame(counts(dds, normalized = TRUE
                   by = 'row.names',
                   sort = FALSE)
  resData
}

uncontrolledFDR <- function(unconDf, cutoff=0.05){
  n_positives <- sum(unconDf$pvalue < cutoff, na.rm = TRUE)
  n_true_positives <- sum((unconDf$pvalue < cutoff) & (unconDf$differential.expres
  n_false_positives <- sum((unconDf$pvalue < cutoff) & (unconDf$differential.expre
  fdr <- n_false_positives/n_positives
  power <- n_true_positives/sum(unconDf$differential.expression == 1)
  tibble(n_positives = n_positives,
         n_true_positives = n_true_positives,
         n_false_positives = n_false_positives,
         fdr = fdr,
         power = power)
}

control_offlineFDR <- function(unconDf, cutoff = 0.05, .method = "BH"){
  #pValueVec is the vector of raw p-values
  #diffExpVec is the differential expression vector for each corresponding p-val
  #cutoff is the nominal cutoff

  adjP <- p.adjust(unconDf$pvalue, method = .method)
  n_positives <- sum(adjP < cutoff, na.rm = TRUE)
  n_true_positives <- sum((adjP < cutoff) & (unconDf$differential.expression==1),
  n_false_positives <- sum((adjP < cutoff) & (unconDf$differential.expression==0),
```

```
  fdr <- n_false_positives/n_positives
  power <- n_true_positives/sum(unconDf$differential.expression == 1)
  tibble(n_positives = n_positives,
         n_true_positives = n_true_positives,
         n_false_positives = n_false_positives,
         fdr = fdr,
         power = power)
}


control_onlineFDR <- function(unconDf, cutoff=0.05){
  #This is set up very similarly the offline control function. But SAFFRON gener
  unconDf <- drop_na(unconDf)
  saffron_res <- SAFFRON(unconDf, alpha=cutoff)
  n_positives <- sum(saffron_res$R)
  n_true_positives <- sum(saffron_res$R & unconDf$differential.expression==1, na.r
  n_false_positives <- sum(saffron_res$R & unconDf$differential.expression==0, na.
  fdr <- n_false_positives/n_positives
  power <- n_true_positives/sum(unconDf$differential.expression==1)
  tibble(n_positives = n_positives,
         n_true_positives = n_true_positives,
         n_false_positives = n_false_positives,
         fdr = fdr,
         power = power)
}
```

```{r generate_results}
generate_results <- function(.numHyp, .diffRat, .repID){
mainDf <- data.frame()


#P-Value Generation
dataDf <- data.frame()

numHyp <- .numHyp
diffRat <- .diffRat
repID <- .repID
```

```
dataObj <- generateData(numHyp, diffRat, repID)
deRes <- DESeqPipeline(dataObj@count.matrix, dataObj@sample.annotations$condition)

mainDf <- merge(dataObj@variable.annotations, deRes, by='Row.names')

return(mainDf)
}

#This is the block of code that needs to be run to actually conduct the analysis

repID <- 14286126841
aggRes <- list()
for (j in 1:15){
de_res <- list()
noFDR <- c()
noPower <- c()
onlineFDR <- c()
onlinePower <- c()
offlineFDR <- c()
offlinePower <- c()
finalRes <- data.frame()
#These parameters are extremely important. runs is the number of times we simula
runs <- 20
numHyp <- 1000
diffRat <- 0.1

#Each time we run generate_results(), we get a data frame with all of the releva



for(i in 1:runs){
  de_res[[i]] <- generate_results(numHyp, diffRat, repID) %>%
    mutate(batch = i)
  repID <- repID + i
}

finalRes <- dplyr::bind_rows(de_res) %>%
```

```
  select(pvalue, differential.expression, batch, Row.names)

for (i in 1:runs){

  noRes <- uncontrolledFDR(filter(finalRes, batch==i))
  offRes <- control_offlineFDR(filter(finalRes, batch==i))
  onRes <- control_onlineFDR(filter(finalRes, batch<=i))

  noFDR <- c(noFDR, noRes$fdr)
  noPower <- c(noPower, noRes$power)
  offlineFDR <- c(offlineFDR, offRes$fdr)
  offlinePower <- c(offlinePower, offRes$power)
  onlineFDR <- c(onlineFDR, onRes$fdr)
  onlinePower <- c(onlinePower, onRes$power)
}
aggRes[j] <- list(data.frame(noFDR, noPower, offlineFDR, offlinePower, onlineFDR,
}
save(finalRes, file=paste0('finalRes1agg')

data.frame(noFDR) %>%
  ggplot(aes(x=seq(from=1,to=length(noFDR),by=1), y = noFDR)) +
  geom_point() +
  geom_hline(yintercept=mean(noFDR))

data.frame(noPower) %>%
  ggplot(aes(x=seq(from=1,to=length(noPower),by=1), y = noPower)) +
  geom_point() +
  geom_hline(yintercept=mean(noPower))


data.frame(offlineFDR) %>%
  ggplot(aes(x=seq(from=1,to=length(offlineFDR),by=1), y = offlineFDR)) +
  geom_point() +
  geom_hline(yintercept=mean(offlineFDR))

data.frame(offlinePower) %>%
  ggplot(aes(x=seq(from=1,to=length(offlinePower),by=1), y = offlinePower)) +
  geom_point() +
```

```
  geom_hline(yintercept=mean(offlinePower))


data.frame(onlineFDR) %>%
  ggplot(aes(x=seq(from=1,to=length(onlineFDR),by=1), y = onlineFDR)) +
  geom_point() +
  geom_hline(yintercept=mean(onlineFDR))

data.frame(onlinePower) %>%
  ggplot(aes(x=seq(from=1,to=length(onlinePower),by=1), y = onlinePower)) +
  geom_point() +
  geom_hline(yintercept=mean(onlinePower))



#This is the block of code that needs to be run to actually conduct the analysis

de_res <- list()
noFDR <- c()
noPower <- c()
onlineFDR <- c()
onlinePower <- c()
offlineFDR <- c()
offlinePower <- c()
#These parameters are extremely important. runs is the number of times we simula
runs <- 30
numHyp <- 1000
diffRat <- 0.5
repID <- 032020221139
#Each time we run generate_results(), we get a data frame with all of the releva
for(i in 1:runs){
  de_res[[i]] <- generate_results(numHyp, diffRat, repID) %>%
    mutate(batch = i)
  repID <- repID + i
}

finalRes <- dplyr::bind_rows(de_res) %>%
  select(pvalue, differential.expression, batch, Row.names)
```

```
for (i in 1:runs){

  noRes <- uncontrolledFDR(filter(finalRes, batch==i))
  offRes <- control_offlineFDR(filter(finalRes, batch==i))
  onRes <- control_onlineFDR(filter(finalRes, batch<=i))

  noFDR <- c(noFDR, noRes$fdr)
  noPower <- c(noPower, noRes$power)
  offlineFDR <- c(offlineFDR, offRes$fdr)
  offlinePower <- c(offlinePower, offRes$power)
  onlineFDR <- c(onlineFDR, onRes$fdr)
  onlinePower <- c(onlinePower, onRes$power)
}

save(finalRes, file=paste0('finalRes2')

data.frame(noFDR) %>%
  ggplot(aes(x=seq(from=1,to=length(noFDR),by=1), y = noFDR)) +
  geom_point() +
  geom_hline(yintercept=mean(noFDR))

data.frame(noPower) %>%
  ggplot(aes(x=seq(from=1,to=length(noPower),by=1), y = noPower)) +
  geom_point() +
  geom_hline(yintercept=mean(noPower))


data.frame(offlineFDR) %>%
  ggplot(aes(x=seq(from=1,to=length(offlineFDR),by=1), y = offlineFDR)) +
  geom_point() +
  geom_hline(yintercept=mean(offlineFDR))

data.frame(offlinePower) %>%
  ggplot(aes(x=seq(from=1,to=length(offlinePower),by=1), y = offlinePower)) +
  geom_point() +
  geom_hline(yintercept=mean(offlinePower))
```

```r
data.frame(onlineFDR) %>%
  ggplot(aes(x=seq(from=1,to=length(onlineFDR),by=1), y = onlineFDR)) +
  geom_point() +
  geom_hline(yintercept=mean(onlineFDR))

data.frame(onlinePower) %>%
  ggplot(aes(x=seq(from=1,to=length(onlinePower),by=1), y = onlinePower)) +
  geom_point() +
  geom_hline(yintercept=mean(onlinePower))



#This is the block of code that needs to be run to actually conduct the analysis
aggRes3 <- list()
repID <- 032020221144
for(j in 1:8){
de_res <- list()
noFDR <- c()
noPower <- c()
onlineFDR <- c()
onlinePower <- c()
offlineFDR <- c()
offlinePower <- c()
#These parameters are extremely important. runs is the number of times we simula
runs <- 50
numHyp <- 1000
diffRat <- 0.5

#Each time we run generate_results(), we get a data frame with all of the releva
for(i in 1:runs){
  if(i < 10){
  de_res[[i]] <- generate_results(numHyp, diffRat, repID) %>%
    mutate(batch = i)
  }
  else{
    de_res[[i]] <- generate_results(numHyp, 0.1, repID) %>%
    mutate(batch = i)
```

```
  }
  repID <- repID + i
}

finalRes <- dplyr::bind_rows(de_res) %>%
  select(pvalue, differential.expression, batch, Row.names)

for (i in 1:runs){

  noRes <- uncontrolledFDR(filter(finalRes, batch==i))
  offRes <- control_offlineFDR(filter(finalRes, batch==i))
  onRes <- control_onlineFDR(filter(finalRes, batch<=i))

  noFDR <- c(noFDR, noRes$fdr)
  noPower <- c(noPower, noRes$power)
  offlineFDR <- c(offlineFDR, offRes$fdr)
  offlinePower <- c(offlinePower, offRes$power)
  onlineFDR <- c(onlineFDR, onRes$fdr)
  onlinePower <- c(onlinePower, onRes$power)
}
aggRes3[j] <- list(data.frame(noFDR, noPower, offlineFDR, offlinePower, onlineFDR,
}
save(aggRes3, file='aggfinalRes3')

data.frame(noFDR) %>%
  ggplot(aes(x=seq(from=1,to=length(noFDR),by=1), y = noFDR)) +
  geom_point() +
  geom_hline(yintercept=mean(noFDR))

data.frame(noPower) %>%
  ggplot(aes(x=seq(from=1,to=length(noPower),by=1), y = noPower)) +
  geom_point() +
  geom_hline(yintercept=mean(noPower))


data.frame(offlineFDR) %>%
  ggplot(aes(x=seq(from=1,to=length(offlineFDR),by=1), y = offlineFDR)) +
  geom_point() +
```

```
  geom_hline(yintercept=mean(offlineFDR))

data.frame(offlinePower) %>%
  ggplot(aes(x=seq(from=1,to=length(offlinePower),by=1), y = offlinePower)) +
  geom_point() +
  geom_hline(yintercept=mean(offlinePower))


data.frame(onlineFDR) %>%
  ggplot(aes(x=seq(from=1,to=length(onlineFDR),by=1), y = onlineFDR)) +
  geom_point() +
  geom_hline(yintercept=mean(onlineFDR))

data.frame(onlinePower) %>%
  ggplot(aes(x=seq(from=1,to=length(onlinePower),by=1), y = onlinePower)) +
  geom_point() +
  geom_hline(yintercept=mean(onlinePower))



totalDf <- data.frame()
for (i in 1:15){
aggRes[[i]]$batch <- 1:20
}

for (i in 1:15){
totalDf <- rbind(totalDf, aggRes[[i]])
}


data.frame(totalDf) %>%
  ggplot(aes(x=batch, y = noPower)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='FDR') +
  geom_hline(yintercept=0.05, col='red')

data.frame(totalDf) %>%
```

```
  ggplot(aes(x=batch, y = noPower)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='Power')

data.frame(totalDf) %>%
  ggplot(aes(x=batch, y = offlineFDR)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='FDR') +
  geom_hline(yintercept=0.05, col='red')

data.frame(totalDf) %>%
  ggplot(aes(x=batch, y = offlinePower)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='Power')

data.frame(totalDf) %>%
  ggplot(aes(x=batch, y = onlineFDR)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='FDR') +
  geom_hline(yintercept=0.05, col='red')

data.frame(totalDf) %>%
  ggplot(aes(x=batch, y = onlinePower)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='Power')



totalDf3 <- data.frame()
for (i in 1:8){
aggRes3[[i]]$batch <- 1:50
}
```

```
for (i in 1:8){
totalDf3 <- rbind(totalDf3, aggRes3[[i]])
}
```

```
data.frame(totalDf3) %>%
  ggplot(aes(x=batch, y = noPower)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='FDR') +
  geom_hline(yintercept=0.05, col='red')

data.frame(totalDf3) %>%
  ggplot(aes(x=batch, y = noPower)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='Power')

data.frame(totalDf3) %>%
  ggplot(aes(x=batch, y = offlineFDR)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='FDR') +
  geom_hline(yintercept=0.05, col='red')

data.frame(totalDf3) %>%
  ggplot(aes(x=batch, y = offlinePower)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='Power')

data.frame(totalDf3) %>%
  ggplot(aes(x=batch, y = onlineFDR)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='FDR') +
  geom_hline(yintercept=0.05, col='red')
```

```
data.frame(totalDf3) %>%
  ggplot(aes(x=batch, y = onlinePower)) +
  geom_smooth() +
  geom_point() +
  labs(x = 'Batch', y='Power') +
  theme_gray(base_size=20)
```

Acknowledgements

This thesis would not be possible without the amazing support from my thesis advisor Professor Jo Hardin. She helped me constantly with both the theoretical and simulation aspects of the paper. Lathan Liou also helped me tirelessly with the code and theory. I would not have been able to complete the work without both Professor Hardin and Lathan. My friend and colleague Ciara Chow was also constantly push me to improve and gave amazing criticism and support. Finally, my parents offered support throughout the year.

Bibliography

[1] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300, 1995.

[2] Marine Gauthier, Denis Agniel, Rodolphe Thiébaut, and Boris P Hejblum. dearseq: a variance component score test for RNA-seq differential analysis that effectively controls the false discovery rate. *NAR Genomics and Bioinformatics*, 2(4), 11 2020. lqaa093.

[3] Debashis Ghosh. Shrunken p-values for assessing differential expression with applications to genomic data analysis. *Biometrics*, 62(4):1099–1106, 2006.

[4] Adel Javanmard and Andrea Montanari. On online control of false discovery rate, 2015.

[5] Xiaohong Li, Shesh N. Rai, Eric C. Rouchka, Timothy E. O'Toole, and Nigel G. F. Cooper. Adjusted sample size calculation for rna-seq data in the presence of confounding covariates. *BioMedInformatics*, 1(2):47–63, 2021.

[6] Joseph K Pickrell, John C Marioni, Athma A Pai, Jacob F Degner, Barbara E Engelhardt, Everlyne Nkadori, Jean-Baptiste Veyrieras, Matthew Stephens, Yoav Gilad, and Jonathan K Pritchard. Understanding mechanisms underlying human gene expression variation with rna sequencing. *Nature*, 464(7289):768–772, 2010.

[7] Aaditya Ramdas, Tijana Zrnic, Martin Wainwright, and Michael Jordan. Saffron: an adaptive algorithm for online control of the false discovery rate, 2019.

[8] Jinjin Tian and Aaditya Ramdas. Addis: an adaptive discarding algorithm for online fdr control with conservative nulls, 2019.

[9] Tijana Zrnic, Daniel L. Jiang, Aaditya Ramdas, and Michael I. Jordan. The power of batching in multiple hypothesis testing, 2021.