Non-Normality and Heteroscedasticity in Regression and ANOVA

Harry Wu

Statistics Department

California Polytechnic State University, San Luis Obispo

June 2017

1. Introduction

In this project, we will investigate the effects of non-constant variance and non-normality on the results of simple linear regression and one-way ANOVA as well as develop interactive SHINY applets in R as a convenient tool to examine these effects. To test the effects of non-constant variance, we will simulate data sets from the normal distribution under different variances and compare them by their Type I error rates and their power. To test the effects of non-normality, we will simulate data sets from a generalization of the Tukey-Lambda distribution with parameters that allow us to control the kurtosis and skewness of the distribution, and compare them by their Type I and Type II error rates.

A hypothesis test consists of a null hypothesis (H_0) and an alternative hypothesis (H_a). We either reject or fail to reject the null hypothesis depending on whether or not the p-value is less than the significance level (α). A Type I error rate is the probability of rejecting the null hypothesis given that it is true. It is equal to the significance level α. Type II error rate is the probability of failing to reject the null hypothesis given that it is false. The power is the probability of correctly rejecting the null hypothesis when the alternative hypothesis is true, his is simply 1 minus the probability of a Type II error.

In our simulation, we will compare the error rates given non-normality or non-constant variance and compare them to their normal or equal variance counterparts to determine the robustness of the regression t-test or the one-way ANOVA F-test when certain assumptions are violated.

Through the use of a SHINY applet, we can have a simple interface where the user can input values for the parameters for the distribution to sample from, the sample sizes, the significance level, and the number of iterations. The simulated Type I error rate or simulated power is then returned to the user as the output, along with a 95% confidence interval. The expected Type I error rate and power are also computed based on the model assumptions being true.

2. Simple Linear Regression and One-way ANOVA

The general linear model used in simple regression analysis is given by:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

where y is a quantitative response variable and x is a quantitative explanatory variable. The subscript *i* differentiates the different cases. The parameters β_0 and β_1 quantify the linear relationship, if any, between x and y. The parameter β_1 is called the "slope" of the regression equation, it determines how y changes with changes in x. The parameter β_0 is called the "intercept" of the regression equation. β_0 is the value of y that you predict when x equals zero. ε_i is a random error term that is independent and normally distributed with mean 0 and constant variance σ^2.

The assumptions of the model are as follows:

- The population regression function is **linear**. The response y_i is a function of linear trend ($\beta_0+\beta_1 x_i$) plus some error ε_i.
- The error terms are **independent**.
- The error terms are **normally distributed**.
- The error terms have **equal variance**. The variance of errors is the same across all levels of the explanatory variable.

In our simulation, the Type I and Type II error rates will be calculated with significance level α=.05.

We are testing the following hypotheses:

$$H_0: \beta_1 = 0$$

$$H_a: \beta_1 \neq 0$$

When β_1 is equal to 0, the proportion of rejections will be the estimated Type I error rate. When β_1 is not

equal to zero, the proportion of non-rejections will be the estimated Type II error rate.

The one-way analysis of variance (ANOVA) is a procedure used to determine if there are any statistically significant differences between the means of two or more (typically at least three) groups. In a one-way ANOVA, there is one categorical explanatory variable (or factor) and one quantitative response variable. The hypotheses of this procedure are formulated about the means of the dependent variable for each level of the independent variable.

The ANOVA model is given by:

$$y_{ij} = \mu_j + \varepsilon_{ij}$$

where y_{ij} is a quantitative response variable for the i^{th} observation of population j. The parameter μ_j is the mean of population j. ε_{ij} is a random error term that is independent and normally distributed with mean 0 and constant variance σ^2.

The assumptions of the ANOVA model are as follows:

- The error terms (ε_{ij}) are **independent**.
- The error terms (ε_{ij}) are **normally distributed**.
- The error terms (ε_{ij}) have **equal variance**. The variance of errors is the same across all levels of the explanatory variable.

In our simulation, the Type I error rate and power will be calculated with significance level $\alpha = .05$.

When there are three populations to compare, we are testing the following hypotheses:

$$H_0: \mu_1 = \mu_2 = \mu_3$$

H_a: Not all means are equal.

When H_0 is true, the proportion of rejections will be the estimated Type I error rate. When H_a is true, the proportion of rejections will be the estimated power. We will compare these estimated values when sampling under equal variances to the estimated values when sampling under unequal variances.

3. Non-Normality in Simple Linear Regression

As a continuation of Hongyan Wang's Senior Project from Fall 2008, a Shiny application was written using her R code that uses the Tukey-Lambda distributions with varying parameters *a* and *b* as the error distribution in a simple linear regression model.

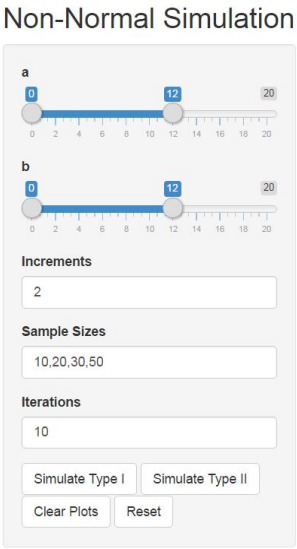

Figure 1: Input for Shiny Application: Non-Normal Simulation

Figure 1 shows the input for the Shiny application, for this particular setting, the simulation covers a grid of values for a = 0, 2, 4, 6, 8, 10, and 12 and b = 0, 2, 4, 6, 8, 10, and 12 for sample size n = 10, 20, 30, and 50 with significance level α =0.05. A total of 196 simulations with different combinations of (a,b,n)

will be run for both Type I and Type II errors and the results will be displayed in graphs as the output.

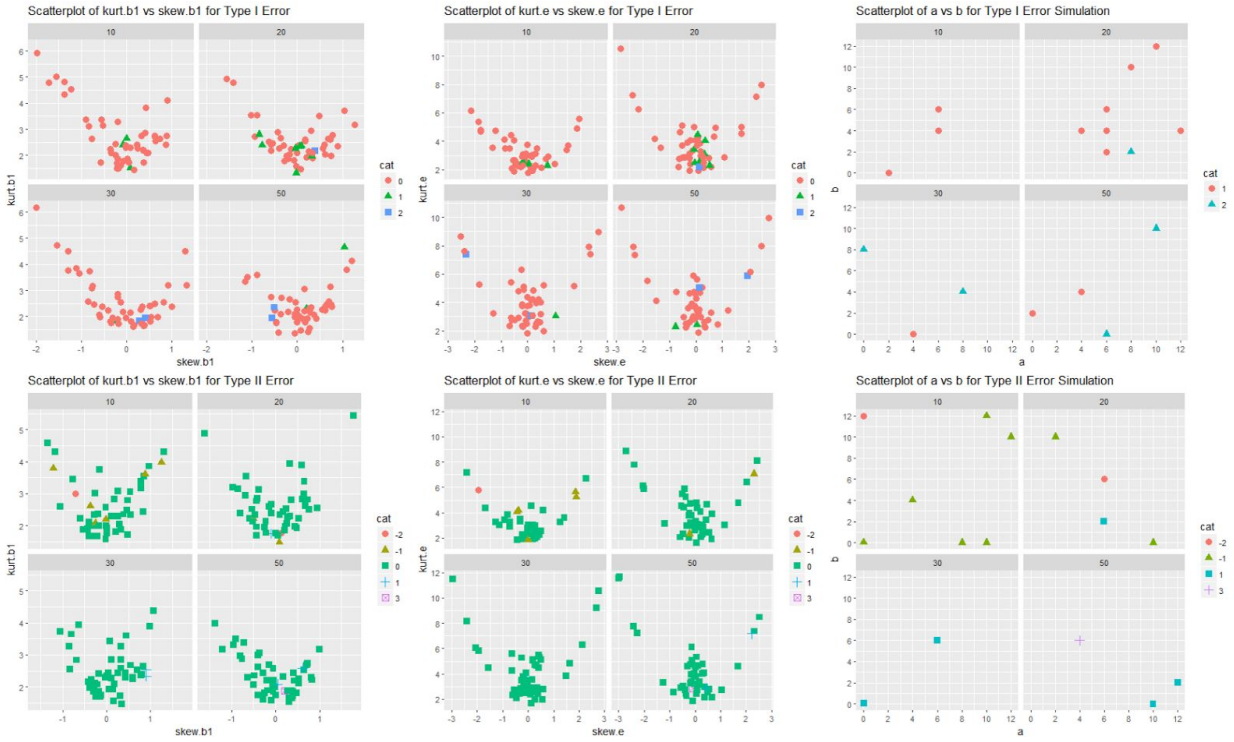

Figure 2: Output for Shiny Application: Non-Normal Simulation

Figure 2 shows the output, graphs of the Skewness vs. Kurtosis for the simulated error distributions and the estimated Slopes, as well as a graph of the error parameters *a* and *b* that produced extreme Type I or Type II error rates.

From the simulations we can expect to obtain more extreme Type I or Type II error rates when errors are very non-normal with large skewness and kurtosis values, and small sample sizes. Kurtosis appears to have more of an effect than the skewness on both the Type I and Type II error rates. The effect of non-normality on the Type I and Type II error rates is reduced as the sample size increases.

For more information on the experimental design and results of this study, please refer to Hongyan

Wang's Senior Project paper, "Robustness to Non-Normality of the Regression T-test".

4. Non-Constant Variance in One-way ANOVA

To test for the effects of non-constant variance in one-way ANOVA, a Shiny application was written that takes in values for n, mean, sd, iterations, and alpha level as inputs and returns the Type I error rate or power as output.

We are comparing three populations in this simulation. The inputs for this simulation are as follows:

n1, n2, n3: sample sizes for each sample

Mean 1, Mean 2, Mean 3: means for each population

sd1, sd2, sd3: standard deviations for each population

Iterations: iterations for calculating Type I error rate or power

Alpha level: significance level

The output shows the simulated Type I error rate, if all the means are equal, or the simulated power, if some of the means are not equal. It also outputs a 95% confidence interval for that simulated value as well as the expected value under equal variance. If the 95% confidence interval does not capture the expected value, then there is significant evidence that these settings of unequal variance affect the Type I error rate or power.

Figure 3: Shiny Application: Non-Constant Variance Simulation

To compare different settings at once, a script was run using R that takes in various settings from a .csv file and returns the output as a .txt file.

```
   n1 n2 n3 sigma1 sigma2 sigma3 PooledSD  mu1 mu2  mu3 TargetPower Power Conf.Lower Conf.Upper
1   5  5  5      1      2     10 5.916080 50.0  50 50.0         0.05  0.10       0.05       0.18
2  50 50 50      1      2     10 5.916080 50.0  50 50.0         0.05  0.07       0.03       0.14
3   2  5  8      1      2     10 7.729812 50.0  50 50.0         0.05  0.04       0.01       0.10
4  20 50 80      1      2     10 7.429945 50.0  50 50.0         0.05  0.02       0.00       0.07
5   8  5  2      1      2     10 3.201562 50.0  50 50.0         0.05  0.40       0.30       0.50
6  80 50 20      1      2     10 3.846546 50.0  50 50.0         0.05  0.34       0.25       0.44
7   5  5  5      1      2     10 5.916080 55.9  50 50.0         0.29  0.25       0.17       0.35
8  50 50 50      1      2     10 5.916080 55.9  50 50.0         1.00  1.00       0.96       1.00
9   2  5  8      1      2     10 7.729812 57.7  50 50.0         0.17  0.04       0.01       0.10
10 20 50 80      1      2     10 7.429945 57.4  50 50.0         0.97  1.00       0.96       1.00
11  8  5  2      1      2     10 3.201562 53.2  50 50.0         0.32  0.64       0.54       0.73
12 80 50 20      1      2     10 3.846546 53.8  50 50.0         1.00  1.00       0.96       1.00
13  5  5  5      1      2     10 5.916080 50.0  50 55.9         0.29  0.35       0.26       0.45
14 50 50 50      1      2     10 5.916080 50.0  50 55.9         1.00  0.99       0.95       1.00
15  2  5  8      1      2     10 7.729812 50.0  50 57.7         0.32  0.30       0.21       0.40
16 20 50 80      1      2     10 7.429945 50.0  50 57.4         1.00  1.00       0.96       1.00
17  8  5  2      1      2     10 3.201562 50.0  50 53.2         0.17  0.50       0.40       0.60
18 80 50 20      1      2     10 3.846546 50.0  50 53.8         0.96  0.81       0.72       0.88
```

Figure 4: Text Output for Multiple Simulations

From the output in Figure 4, we can see that certain settings affect the Type I error rate or power more than others. Table 1 shows the settings that significantly affected the Type I error rate or power.

Table 1: Settings of interest for ANOVA simulations

n1	n2	n3	sd1	sd2	sd3	mu1	mu2	mu3	Type I/Power	Expected Power	95% Conf. Interval
8	5	2	1	2	10	50	50	50	0.40	0.05	(0.30, 0.50)
80	50	20	1	2	10	50	50	50	0.34	0.05	(0.25, 0.44)
2	5	8	1	2	10	57.7	50	50	0.04	0.17	(0.01, 0.10)
8	5	2	1	2	10	53.2	50	50	0.64	0.32	(0.54, 0.73)
8	5	2	1	2	10	50	50	53.2	0.50	0.17	(0.40, 0.60)
80	50	20	1	2	10	50	50	53.8	0.81	0.96	(0.72, 0.88)

As we can see from Table 1, the Type I error rate drastically increases when the lowest variance is in the largest sample size group and greatest variance in the smallest sample size group. The effect on Type I error rate under those settings appears to decrease as the total sample size increases. Power appears to significantly decrease when the sample with the different mean has low variance and a smaller sample size. Power appears to significantly increase when the sample with the different mean has large variance and a smaller sample size.

5. Non-Constant Variance in Simple Linear Regression

To test for the effects of non-constant variance in simple linear regression, R was used to run simulations. Recall the simple linear regression model:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

In this simulation, we have x_i from 1 to 10, and we simulate values of epsilon that are normally distributed but with different variances across the x-values. Using those values, we fit the model over many iterations and calculate the Type I error rate or the power for the t-test of the regression slope depending on the value of β_1.

The inputs for the simulation are as follows:

beta1: Set as 0 to simulate the Type I error rate , or 1 to simulate the power.

n: the base sample size for each x.

n.state: a sample size multiplier: n + n.state*c(-2, -2, -1, -1, 0, 0, 1, 1, 2, 2)

when n.state < 0: the sample size decreases as x increases,

when n.state = 0: the sample size is balanced,

when n.state > 0: the sample size increases as x increases

sigma: the standard deviation of regression errors.

sigma.state: the exponent of unequal variance. $SD(\varepsilon_i) = sigma*x_i\text{^}sigma.state$

iter: the number of iterations.

n.state is used to model unbalanced or balanced data, and sigma.state is the degree of unequal variance. For example, with a base sample size of n= 5, n.state= -2, and sigma.state =1, the sample sizes would be (9, 9, 7, 7, 5, 5, 3, 3, 1, 1), and the sigmas would be (10, 20, 30, 40, 50, 60, 70, 80, 90, 100) for x from 1 to 10.

The outputs of the simulation are as follows:

power: the simulated Type I error rate or Power

epower: the expected Type I error rate or Power.

CI.power (L.power, U.power): a 95% confidence interval for the simulated Type I error rate or power.

power.diff: the difference in simulated and expected Type I error rate or power.

CI.power.diff (L.power.diff, U.power.diff): a 95% confidence interval for the difference in simulated and expected Type I error rate or power.

A Shiny application was written to perform a single replication of this simulation. The inputs and outputs are as described above. The sample sizes, standard deviations, and the pooled standard deviation associated with the inputs are also included for reference.

Figure 5: Shiny Application: Regression Simulation

To compare different settings at once, a script was run using R that takes in various settings from a .csv file and returns the output as a .csv file. For this script, there is also an input for replications, to perform multiple replications of each simulation. The output is visualized in the following graphs.

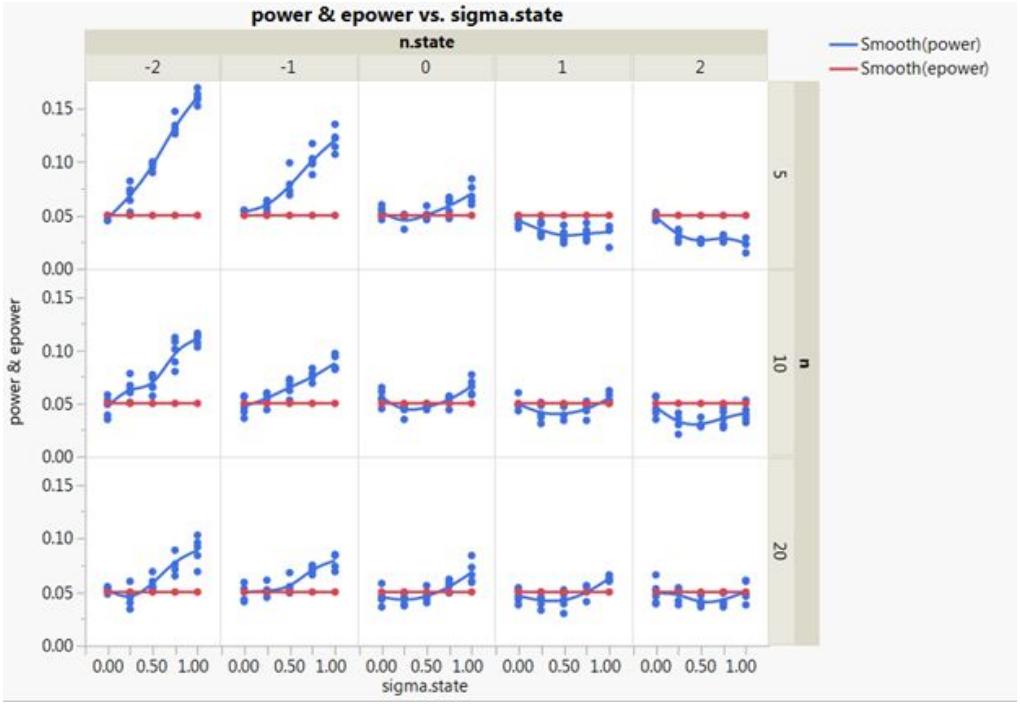

Figure 6: Simulated vs. Expected Type I Error

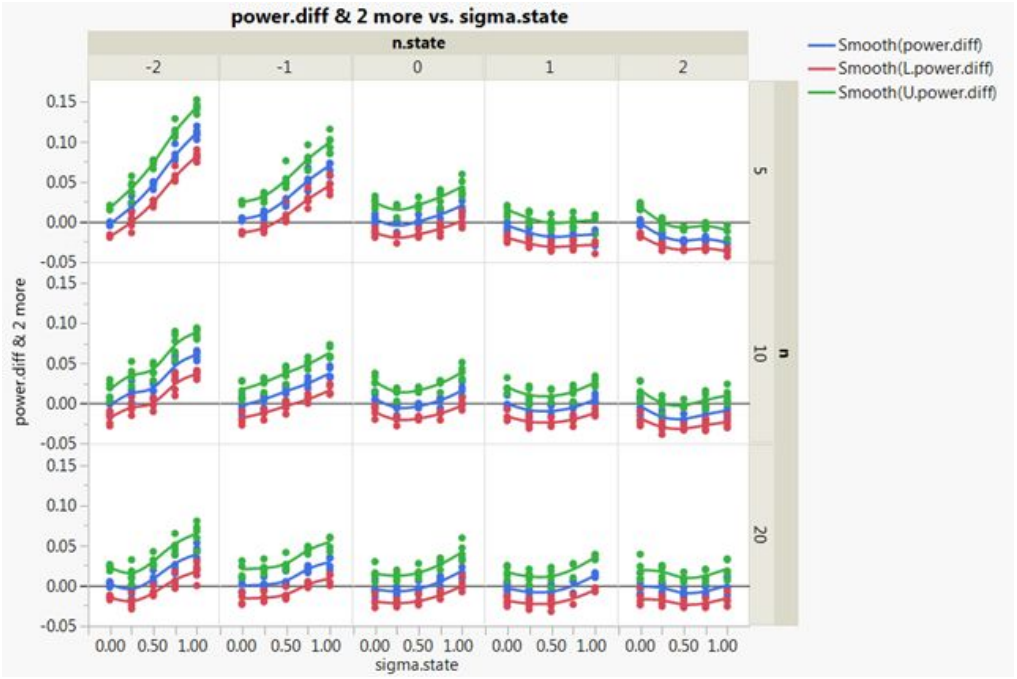

Figure 7: Difference in Simulated and Expected Type I Error

Figure 6 displays the simulated Type I error rates vs. the expected Type I error rates. The simulated Type I error rate is plotted in blue and the expected Type I error rate is plotted in red. Figure 7 shows the same information in terms of the differences. The difference in simulated and expected Type I error rates is plotted in blue and the 99% confidence bounds are plotted in red and green. Unequal variance becomes worse as the points go from left to right in each individual plot. Different plots in the grid represent different degrees of balance or imbalance in the sample sizes. There are three replicates at each combination of inputs. When n = 5 (or a total sample size of 50), Type I error rate increases when there is larger variance in the smaller sample size group. Type I error rate decreases when there is larger variance in the larger sample size groups. When n = 5 with balanced data from x = 1 to x = 10, unequal variance does not have a significant effect on the Type I error rate. When n > 10 (or a total sample size of 100), Type I error rate increases when there is larger variance in the smaller sample size groups, but does not seem to change much when there is larger variance in the larger sample size groups.

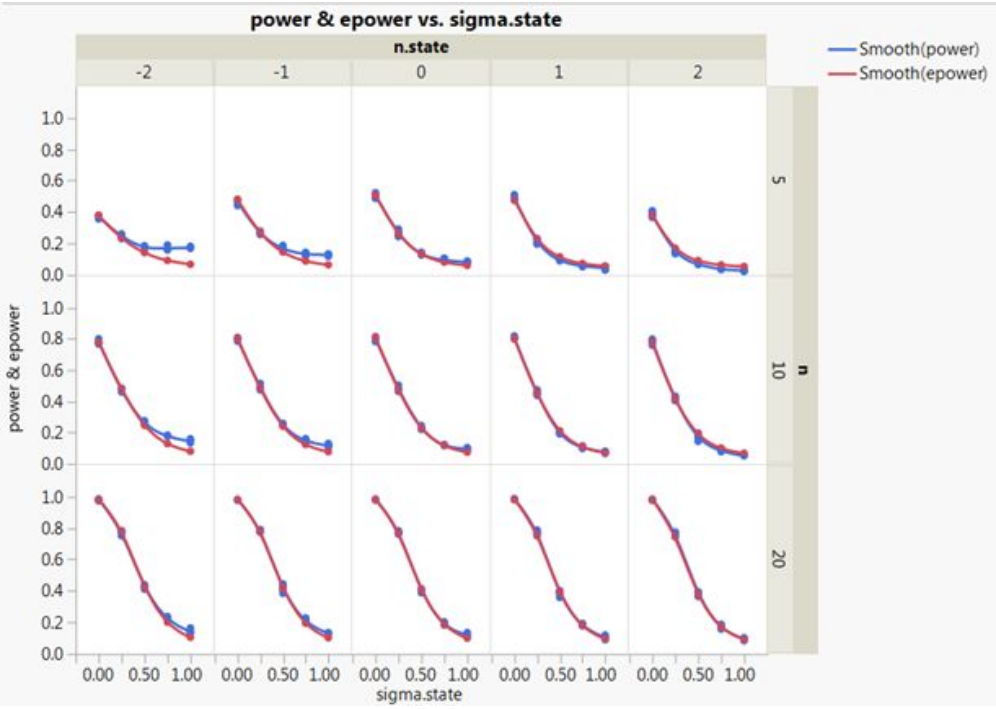

Figure 8: Simulated vs. Expected Power

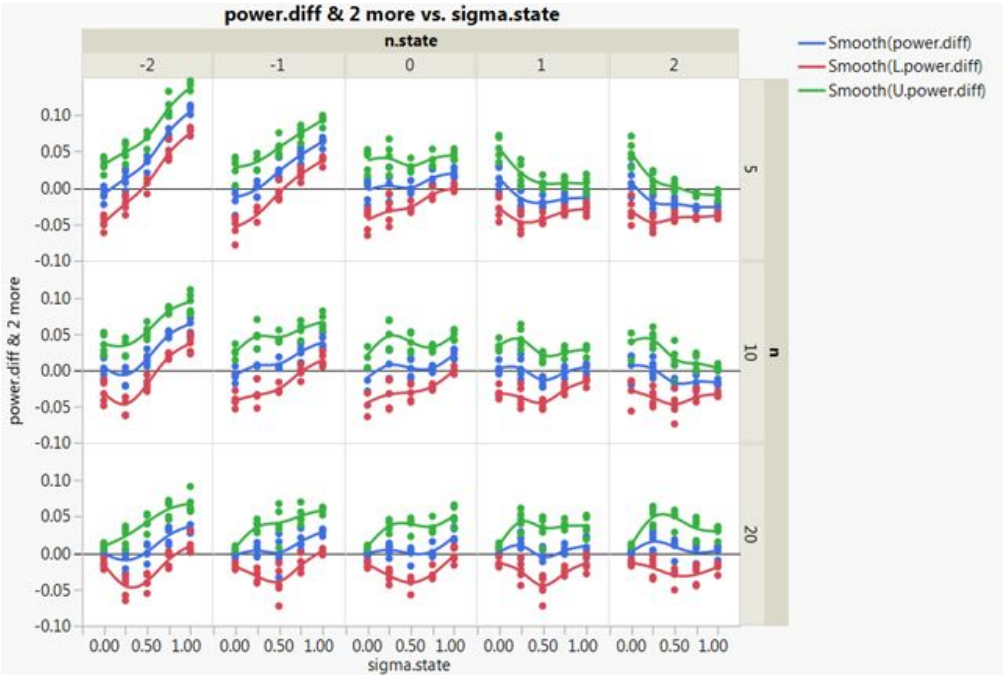

Figure 9: Difference in Simulated and Expected Power

Figure 8 displays the simulated power vs. the expected power. The simulated power is plotted in blue and the expected power is plotted in red. Figure 9 shows the same information in terms of the differences. The difference in simulated and expected power is plotted in blue and the 99% confidence bounds are plotted in red and green. As expected, the trend in the power is similar to the trend in the Type I error rate.

A setting of interest would be n.state = -2, sigma.state = 1. With these settings, the sample sizes are: (9, 9, 7, 7, 5, 5, 3, 3, 1, 1), and the sigmas are: (10, 20, 30, 40, 50, 60, 70, 80, 90, 100) for x from 1 to 10. The sample size is decreasing as x increases with extreme unequal variance. These settings are referring to the first plot on the top left and we can see that there is a significant increase in power when there is larger variance in the smaller sample size groups. As we move down each column of plots, however, when the total sample size increases, the effects of unequal variance on the power diminishes.

6. Conclusion

In Statistics courses, we learn about the assumptions of various statistical methods but we do not learn much about them other than the fact that they need to be checked and fulfilled. This paper analyzes the effects of non-normality on simple linear regression t-tests, and non-constant variance on simple linear regression t-tests and one-way ANOVA F-tests.

The results suggest that for simple linear regression:

- Type I and Type II error rates decrease for highly skewed error distributions when the sample size is small.
- Unequal variance is an issue when there are small sample sizes and an unbalanced distribution of data across the range of x. Type I error rates and power increase when there are larger variances in the smaller sample size groups. Type I error rates and power decrease when there are larger variances in the larger sample size groups.

For one-way ANOVA:

- The Type I error rate increases when there is lower variance in the larger sample size groups and greater variance in the smaller sample size groups. Power decreases when the sample with the different mean has lower variance and a smaller sample size. Power increases when the sample with the different mean has larger variance and a smaller sample size.

The effects of non-normality and non-constant variance diminish as the total sample size increases.

7.

8. References

Jeff Miller and Patricia Harden. “Statistical Analysis with The General Linear Model”. Feb. 2006.

Jason W. Osborne and Elaine Waters. “Four Assumptions Of Multiple Regression That Researchers Should Always Test”. Jan. 2002. Web. <http://pareonline.net/getvn.asp?n=2&v=8>

Hongyan Wang. “Robustness to Non-Normality of the Regression T-test”. Senior project, California Polytechnic State University San Luis Obispo, 2001. Microform.

9. Appendix

Shiny Application for Non-Normality Simulation in Simple Linear Regression:

ui:

```
library(shiny)
library(gridExtra)
library(ggplot2)
library(dplyr)
library(moments)
library(DT)


shinyUI(fluidPage(
 titlePanel("Non-Normal Simulation", windowTitle="Non-Normal Simulation"),
 sidebarLayout(
                sidebarPanel(
                sliderInput("aInput", "a", min=0, max=20, value=c(0,12)),
                sliderInput("bInput", "b", min=0, max=20, value=c(0,12)),
                numericInput("incInput", "Increments", value=2),
                textInput("nInput", "Sample Sizes", "10,20,30,50"),
                numericInput("itersInput", "Iterations", value = 10),
                actionButton("t1", "Simulate Type I"),
                actionButton("t2", "Simulate Type II"),
                actionButton("erase", "Clear Plots"),
                actionButton("reset", "Reset"),
                width=2
                ),
                mainPanel(
                plotOutput("plot"),
                plotOutput("plot2")
                )
 )
)
)
```

server:

```
library(shiny)
numextractall <- function(string){ # http://stackoverflow.com/questions/19252663/extracting-decimal-numbers-from-a-string
 unlist(regmatches(string,gregexpr("[[:digit:]]+\\.*[[:digit:]]*",string)), use.names=FALSE)
}
typeIerror <- function(beta0,beta1,iters,n,alpha,a,b,c){
 t.stats <- rep(NA, iters)
 skewin <- rep(NA, iters)
 skewout <- rep(NA, iters)
 kurtin <- rep(NA, iters)
 kurtout <- rep(NA, iters)
 b1.list <- rep(NA, iters)
 se1.list <- rep(NA, iters)
 for(i in 1:iters){
        x <- rep(c(1:10),n/10)
        u <- runif(n,0,1)
        errors <- u^a - (1-u)^b
        A <- 1/(1+a)-1/(1+b)
        mu <- A/c
        B <- 1/(1+2*a)+1/(1+2*b)-2*(gamma(1+a)*gamma(1+b)/gamma(2+a+b))
        sigma <- sqrt((B-A^2)/c^2)
        e <- (errors -mu)/sigma
        y <- beta0 + beta1*x + e
        skewin[i] <- skewness(e)
        kurtin[i] <- kurtosis(e)
        sim.ls <- lsfit(x,y)
        b1 <- sim.ls$coef[2]
        resid <- sim.ls$residuals
        se1 <- ls.diag(sim.ls)$std.err[2]
        skewout[i] <- skewness(resid)
        kurtout[i] <- kurtosis(resid)
        b1.list[i] <- b1
        se1.list[i] <- se1
        t.stats[i] <- (b1-0)/se1
 }
 reject <- (abs(t.stats) > qt(1-alpha/2,n-2))
```

```
  typeIerror <- mean(reject)
  skew.e <- median(skewin)
  kurt.e <- median(kurtin)
  skew.r <- median(skewout)
  kurt.r <- median(kurtout)
  skew.b1 <- skewness(b1.list)
  kurt.b1 <- kurtosis(b1.list)
  mean.b1 <- mean(b1.list)
  med.b1 <- median(b1.list)
  med.se1 <- median(se1.list)
  std <- sqrt(alpha*(1-alpha)/iters)
  z <- (typeIerror-alpha)/std
  p <- (1-pnorm(abs(z),0,1))*2

  if(p > .05) cat <- 0
  else if(.01 <= p & p < .05) cat <- 1*sign(z)
  else if(.05/196 <= p & p < .01) cat <- 2*sign(z)
  else if (p < .05/196) cat <- 3*sign(z)
  data <- c(typeIerror,std,alpha,z,p,cat,skew.e,kurt.e,skew.r,kurt.r,skew.b1,kurt.b1,mean.b1,med.b1,med.se1)
  return (data)
}
typeIIerror <- function(beta0,beta1,iters,n,alpha,a,b,c){
  t.stats <- rep(NA, iters)
  skewin <- rep(NA, iters)
  skewout <- rep(NA, iters)
  kurtin <- rep(NA, iters)
  kurtout <- rep(NA, iters)
  b1.list <- rep(NA, iters)
  se1.list <- rep(NA, iters)
  for(i in 1:iters){
                x <- rep(c(1:10),n/10)
                u <- runif(n,0,1)
                errors <- u^a - (1-u)^b
                A <- 1/(1+a)-1/(1+b)
                mu <- A/c
```

```
        B <- 1/(1+2*a)+1/(1+2*b)-2*(gamma(1+a)*gamma(1+b)/gamma(2+a+b))
        sigma <- sqrt((B-A^2)/c^2)
        e <- (errors -mu)/sigma
        y <- beta0 + beta1*x + e
        skewin[i] <- skewness(e)
        kurtin[i] <- kurtosis(e)
        sim.ls <- lsfit(x,y)
        b1 <- sim.ls$coef[2]
        resid <- sim.ls$residuals
        se1 <- ls.diag(sim.ls)$std.err[2]
        skewout[i] <- skewness(resid)
        kurtout[i] <- kurtosis(resid)
        b1.list[i] <- b1
        se1.list[i] <- se1
        t.stats[i] <- (b1-0)/se1
}
reject <- (abs(t.stats) > qt(1-alpha/2,n-2))
power <- mean(reject)
typeIIerror <- (1-power)
if(n==10) beta <- .77472
if(n==20) beta <- .55406
if(n==30) beta <- .37508
if(n==50) beta <- .15267
std <- sqrt(beta*(1-beta)/iters)
z <- (typeIIerror-beta)/std
p <- (1-pnorm(abs(z),0,1))*2
skew.e <- median(skewin)
kurt.e <- median(kurtin)
skew.r <- median(skewout)
kurt.r <- median(kurtout)
skew.b1 <- skewness(b1.list)
kurt.b1 <- kurtosis(b1.list)
mean.b1 <- mean(b1.list)
med.b1 <- median(b1.list)
med.se1 <- median(se1.list)
```

```
  if(p > .05) cat <- 0
  else if(.01 <= p & p < .05) cat <- 1*sign(z)
  else if(.05/196 <= p & p < .01) cat <- 2*sign(z)
  else if (p < .05/196) cat <- 3*sign(z)
  data <- c(typeIIerror,std,beta,z,p,cat,skew.e,kurt.e,skew.r,kurt.r,skew.b1,kurt.b1,mean.b1,med.b1,med.se1)
  return (data)
  #return(list(mean= mean(e),std=stdev(e)))
  #test=ks.gof(t.stats, distribution="normal",mean=0,sd=1)$p))
}
shinyServer(function(input, output, session) {
  observeEvent(input$t1, {
  output$plot=renderPlot({
                a <- seq(input$aInput[1],input$aInput[2],input$incInput)
                b <- seq(input$bInput[1],input$bInput[2],input$incInput)
                n <- as.numeric(numextractall(input$nInput))
                iters <- input$itersInput
                alpha <- .05
                nrows <- length(a)*length(b)*length(n)*length(alpha)
                data <- matrix(,ncol=19,nrow=nrows)
                dimnames(data) <-
list(NULL,c("a","b","n","iters","typeI","se","alpha","zscore","pvalue","cat","skew.e","kurt.e","skew.r","kurt.r","skew.b1","kurt.b1","mean.b1","med.b1","med.se1"))
                row <- 1
                for(g in 1:length(alpha))
                for(k in 1:length(n))
                for(i in 1:length(a))
                for(j in 1:length(b)){
                if(a[i] != 0 || b[j] != 0){
                data[row,1] <- a[i]
                data[row,2] <- b[j]
                data[row,3] <- n[k]
                data[row,4] <- iters
                data[row,5:19] <- typeIerror(0,0,iters,n[k],alpha[g],a[i],b[j],1)
                row <- row +1
                }
                else if(a[i] == 0 && b[j] == 0){
                data[row,1] <- .01
```

```
                data[row,2] <- .01
                data[row,3] <- n[k]
                data[row,4] <- iters
                data[row,5:19] <- typeIerror(0,0,iters,n[k],alpha[g],.01,.01,1)
                row <- row +1
                }
                }
                dataf=as.data.frame(data)
                dataf$cat=as.factor(dataf$cat)
                dataf$n=as.factor(dataf$n)
                plot1=ggplot(data=dataf, aes(x=skew.b1, y=kurt.b1))+
                ggtitle("Scatterplot of kurt.b1 vs skew.b1 for Type I Error")+
                geom_point(size=3, aes(color=cat, shape=cat))+
                facet_wrap(~n)
                plot2=ggplot(data=dataf, aes(x=skew.e, y=kurt.e))+
  scale_y_continuous(breaks=c(0,2,4,6,8,10,12)) +
                ggtitle("Scatterplot of kurt.e vs skew.e for Type I Error")+
                geom_point(size=3, aes(color=cat, shape=cat))+
                facet_wrap(~n)
 plot3=ggplot(data=subset(dataf,cat==1|cat==-1|cat==2|cat==-2|cat==-3|cat==3), aes(x=a, y=b))+
  scale_x_continuous(breaks=c(0,2,4,6,8,10,12)) +
  scale_y_continuous(breaks=c(0,2,4,6,8,10,12)) +
                ggtitle("Scatterplot of a vs b for Type I Error Simulation")+
                geom_point(size=3, aes(color=cat, shape=cat))+
                facet_wrap(~n)
                grid.arrange(plot1, plot2, plot3, ncol=3)
})
})
observeEvent(input$t2, {
                output$plot2=renderPlot({
                a <- seq(input$aInput[1],input$aInput[2],input$incInput)
                b <- seq(input$bInput[1],input$bInput[2],input$incInput)
                n <- as.numeric(numextractall(input$nInput))
                iters <- input$itersInput
                alpha <- .05
```

```
nrows <- length(a)*length(b)*length(n)*length(alpha)

data <- matrix(,ncol=19,nrow=nrows)

dimnames(data) <-
list(NULL,c("a","b","n","iters","typeII","se","beta","zscore","pvalue","cat","skew.e","kurt.e","skew.r","kurt.r","skew.b1","kurt.b1","mean.b1","med.b1","med.se1"))

row <- 1

for(k in 1:length(n))

for(i in 1:length(a))

for(j in 1:length(b)){

if(a[i] != 0 || b[j] != 0){

data[row,1] <- a[i]

data[row,2] <- b[j]

data[row,3] <- n[k]

data[row,4] <- iters

data[row,5:19] <- typeIIerror(0,.15,iters,n[k],alpha,a[i],b[j],1)

row <- row +1

}

else if(a[i] == 0 &&b[j] == 0){

data[row,1] <- .01

data[row,2] <- .01

      data[row,3] <- n[k]

data[row,4] <- iters

data[row,5:19] <- typeIIerror(0,.15,iters,n[k],alpha,.01,.01,1)

row <- row +1

}

}

data

dataf=as.data.frame(data)

dataf$cat=as.factor(dataf$cat)

dataf$n=as.factor(dataf$n)

  plot3=ggplot(data=subset(dataf,cat==1|cat==-1|cat==2|cat==-2|cat==-3|cat==3), aes(x=a, y=b))+

   scale_x_continuous(breaks=c(0,2,4,6,8,10,12)) +

scale_y_continuous(breaks=c(0,2,4,6,8,10,12)) +

ggtitle("Scatterplot of a vs b for Type II Error Simulation")+

geom_point(size=3, aes(color=cat, shape=cat))+

facet_wrap(~n)

plot2=ggplot(data=dataf, aes(x=skew.e, y=kurt.e))+
```

```
    scale_y_continuous(breaks=c(0,2,4,6,8,10,12)) +
            ggtitle("Scatterplot of kurt.e vs skew.e for Type II Error")+
            geom_point(size=3, aes(color=cat, shape=cat))+
            facet_wrap(~n)
            plot1=ggplot(data=dataf, aes(x=skew.b1, y=kurt.b1))+
            ggtitle("Scatterplot of kurt.b1 vs skew.b1 for Type II Error")+
            geom_point(size=3, aes(color=cat, shape=cat))+
            facet_wrap(~n)
            grid.arrange(plot1, plot2, plot3, ncol=3)
            }
            )
}
)
observeEvent(input$erase, {
            output$plot=renderPlot({})
            output$plot2=renderPlot({})
})
observeEvent(input$reset, {
            output$plot=renderPlot({})
            output$plot2=renderPlot({})
            updateNumericInput(session, "incInput", value=2)
            updateSliderInput(session, "aInput", value=c(0,12))
            updateSliderInput(session, "bInput", value=c(0,12))
            updateTextInput(session, "nInput", "Sample Sizes", "10,20,30,50")
            updateNumericInput(session, "itersInput", value = 10)
})
})
```

Shiny Application for Non-Constant Variance in ANOVA Simulation:

```
library(shiny)
shinyServer(function(input, output, session) {
 observeEvent(input$st1, {
 output$t1= renderText({
 iters=input$iters
```

```
alpha=input$a
n1=input$n1; n2=input$n2; n3=input$n3
mu1=input$mu1; mu2=input$mu2; mu3=input$mu3
sd1=input$sd1; sd2=input$sd2; sd3=input$sd3
sum1=0
for(i in 1:iters){
        a=rnorm(n1,mu1,sd1)
        b=rnorm(n2,mu2,sd2)
        c=rnorm(n3,mu3,sd3)
        data=c(a,b,c)
        sample = factor(rep(letters[1:3], c(n1,n2,n3)))
        fit = lm(formula = data ~ sample)
        anova=anova(fit)
  if(anova$"Pr(>F)"[1]<alpha){
        sum1=sum1+1
        }
        power1=sum1/iters
}
mu=((mu1*n1)+(mu2*n2)+(mu3*n3))/sum(n1+n2+n3)
psd= sqrt(((n1-1)*sd1^2+(n2-1)*sd2^2+(n3-1)*sd3^2)/((n1-1)+(n2-1)+(n3-1)))
ncp= (n1*((mu1-mu)^2)/psd^2)+(n2*((mu2-mu)^2)/psd^2)+(n3*((mu3-mu)^2)/psd^2)
df1=2
df2=sum(n1+n2+n3)-1
crit.value = qf(1-alpha,df1,df2)
epower= round((1-pf(crit.value,df1,df2,ncp)),2)
moe=round(sqrt(power1*(1-power1)/sum(n1+n2+n3)),2)
if(mu1==mu2 & mu1==mu3){
paste("Type 1 Error Rate:", power1, (","), "Pooled SD:", psd)
}else{
paste("Power:", power1, (","), "95% CI:", "(",power1-moe,",",power1+moe,")", "," , "Expected Power:", epower, (","), "Pooled SD:", psd)
}
})
output$t2= renderText({
        iters=input$iters
        alpha=input$a
```

```
        n1=input$n1; n2=input$n2; n3=input$n3
        mu1=input$mu1; mu2=input$mu2; mu3=input$mu3
        sd4=input$sd4; sd5=input$sd5; sd6=input$sd6
        sum2=0
 if(is.numeric(sd4)==TRUE&is.numeric(sd5)==TRUE&is.numeric(sd6)==TRUE){
        for(i in 1:iters){
        a=rnorm(n1,mu1,sd4)
        b=rnorm(n2,mu2,sd5)
        c=rnorm(n3,mu3,sd6)
        data=c(a,b,c)
        sample = factor(rep(letters[1:3], c(n1,n2,n3)))
        fit = lm(formula = data ~ sample)
        anova=anova(fit)
    if(anova$"Pr(>F)"[1]<alpha){
        sum2=sum2+1
        }
         power2=sum2/iters
        }
   mu=((mu1*n1)+(mu2*n2)+(mu3*n3))/sum(n1+n2+n3)
        psd= sqrt(((n1-1)*sd4^2+(n2-1)*sd5^2+(n3-1)*sd6^2)/((n1-1)+(n2-1)+(n3-1)))
        ncp= (n1*((mu1-mu)^2)/psd^2)+(n2*((mu2-mu)^2)/psd^2)+(n3*((mu3-mu)^2)/psd^2)
        df1=2
        df2=sum(n1+n2+n3)-1
        crit.value = qf(1-alpha,df1,df2)
        epower= round((1-pf(crit.value,df1,df2,ncp)),2)
   moe=round(sqrt(power2*(1-power2)/sum(n1+n2+n3)),2)
        if(mu1==mu2 & mu1==mu3){
        paste("Type 1 Error Rate:", power2, (","), "Pooled SD:", psd)
        }else{
        paste("Power:", power2, (","), "95% CI:", "(",power2-moe,",",power2+moe,")", "," , "Expected Power:", epower, (","), "Pooled SD:", psd)
        }
        }

})
output$t3= renderText({
```

```
iters=input$iters
alpha=input$a
n1=input$n1; n2=input$n2; n3=input$n3
mu1=input$mu1; mu2=input$mu2; mu3=input$mu3
sd7=input$sd7; sd8=input$sd8; sd9=input$sd9
                sum3=0
                if(is.numeric(sd7)==TRUE&is.numeric(sd8)==TRUE&is.numeric(sd9)==TRUE){
                for(i in 1:iters){
                a=rnorm(n1,mu1,sd7)
                b=rnorm(n2,mu2,sd8)
                c=rnorm(n3,mu3,sd9)
                data=c(a,b,c)
                sample = factor(rep(letters[1:3], c(n1,n2,n3)))
                fit = lm(formula = data ~ sample)
                anova=anova(fit)
     if(anova$"Pr(>F)"[1]<alpha){
                sum3=sum3+1
                }
                power3=sum3/iters
                }
   mu=((mu1*n1)+(mu2*n2)+(mu3*n3))/sum(n1+n2+n3)
                psd= sqrt(((n1-1)*sd7^2+(n2-1)*sd8^2+(n3-1)*sd9^2)/((n1-1)+(n2-1)+(n3-1)))
                ncp= (n1*((mu1-mu)^2)/psd^2)+(n2*((mu2-mu)^2)/psd^2)+(n3*((mu3-mu)^2)/psd^2)
                df1=2
                df2=sum(n1+n2+n3)-1
                crit.value = qf(1-alpha,df1,df2)
                epower= round((1-pf(crit.value,df1,df2,ncp)),2)
   moe=round(sqrt(power3*(1-power3)/sum(n1+n2+n3)),2)
                if(mu1==mu2 & mu1==mu3){
                paste("Type 1 Error Rate:", power3, (","), "Pooled SD:", psd)
                }else{
                paste("Power:", power3, (","), "95% CI:", "(",power3-moe,",",power3+moe,")", "," , "Expected Power:", epower, (","), "Pooled SD:", psd)
                }
                }
})
```

```
output$t4= renderText({
        iters=input$iters
        alpha=input$a
        n1=input$n1; n2=input$n2; n3=input$n3
        mu1=input$mu1; mu2=input$mu2; mu3=input$mu3
        sd10=input$sd10; sd11=input$sd11; sd12=input$sd12
        sum4=0
 if(is.numeric(sd10)==TRUE&is.numeric(sd11)==TRUE&is.numeric(sd12)==TRUE){
        for(i in 1:iters){
        a=rnorm(n1,mu1,sd10)
        b=rnorm(n2,mu2,sd11)
        c=rnorm(n3,mu3,sd12)
        data=c(a,b,c)
        sample = factor(rep(letters[1:3], c(n1,n2,n3)))
        fit = lm(formula = data ~ sample)
        anova=anova(fit)
    if(anova$"Pr(>F)"[1]<alpha){
        sum4=sum4+1
        }
        power4=sum4/iters
        }

  mu=((mu1*n1)+(mu2*n2)+(mu3*n3))/sum(n1+n2+n3)
        psd= sqrt(((n1-1)*sd10^2+(n2-1)*sd11^2+(n3-1)*sd12^2)/((n1-1)+(n2-1)+(n3-1)))
        ncp= (n1*((mu1-mu)^2)/psd^2)+(n2*((mu2-mu)^2)/psd^2)+(n3*((mu3-mu)^2)/psd^2)
        df1=2
        df2=sum(n1+n2+n3)-1
        crit.value = qf(1-alpha,df1,df2)
        epower= round((1-pf(crit.value,df1,df2,ncp)),2)
  moe=round(sqrt(power4*(1-power4)/sum(n1+n2+n3)),2)
        if(mu1==mu2 & mu1==mu3){
        paste("Type 1 Error Rate:", power4, (","), "Pooled SD:", psd)

        }else{
        paste("Power:", power4, (","), "95% CI:", "(",power4-moe,",",power4+moe,")", "," , "Expected Power:", epower, (","), "Pooled SD:", psd)
```

```
                }
                }
})
output$t5= renderText({
                iters=input$iters
                alpha=input$a
                n1=input$n1; n2=input$n2; n3=input$n3
                mu1=input$mu1; mu2=input$mu2; mu3=input$mu3
                sd13=input$sd13; sd14=input$sd14; sd15=input$sd15
                sum5=0
                if(is.numeric(sd13)==TRUE&is.numeric(sd14)==TRUE&is.numeric(sd15)==TRUE){
                for(i in 1:iters){
                a=rnorm(n1,mu1,sd13)
                b=rnorm(n2,mu2,sd14)
                c=rnorm(n3,mu3,sd15)
                data=c(a,b,c)
                sample = factor(rep(letters[1:3], c(n1,n2,n3)))
                fit = lm(formula = data ~ sample)
                anova=anova(fit)
    if(anova$"Pr(>F)"[1]<alpha){
                sum5=sum5+1
                }
                power5=sum5/iters
                }

   mu=((mu1*n1)+(mu2*n2)+(mu3*n3))/sum(n1+n2+n3)
                psd= sqrt(((n1-1)*sd13^2+(n2-1)*sd14^2+(n3-1)*sd15^2)/((n1-1)+(n2-1)+(n3-1)))
                ncp= (n1*((mu1-mu)^2)/psd^2)+(n2*((mu2-mu)^2)/psd^2)+(n3*((mu3-mu)^2)/psd^2)
                df1=2
                df2=sum(n1+n2+n3)-1
                crit.value = qf(1-alpha,df1,df2)
                epower= round((1-pf(crit.value,df1,df2,ncp)),2)
   moe=round(sqrt(power5*(1-power5)/sum(n1+n2+n3)),2)
                if(mu1==mu2 & mu1==mu3){
                paste("Type 1 Error Rate:", power5, (","), "Pooled SD:", psd)
```

```
            }else{
            paste("Power:", power5, (","), "95% CI:", "(",power5-moe,",",power5+moe,")", "," , "Expected Power:", epower, (","), "Pooled SD:", psd)
            }
            }
})
})
observeEvent(input$clear, {
output$t1=renderText({})
output$t2=renderText({})
output$t3=renderText({})
output$t4=renderText({})
output$t5=renderText({})
})
observeEvent(input$reset, {
            updateNumericInput(session, "n1", value=20)
            updateNumericInput(session, "n2", value=20)
            updateNumericInput(session, "n3", value=20)
            updateNumericInput(session, "mu1", value=5)
            updateNumericInput(session, "mu2", value=5)
            updateNumericInput(session, "mu3", value=5)
            updateNumericInput(session, "sd1", value=1)
            updateNumericInput(session, "sd2", value=1)
            updateNumericInput(session, "sd3", value=1)
            updateNumericInput(session, "sd4", value="a")
            updateNumericInput(session, "sd5", value="a")
            updateNumericInput(session, "sd6", value="a")
            updateNumericInput(session, "sd7", value="a")
            updateNumericInput(session, "sd8", value="a")
            updateNumericInput(session, "sd9", value="a")
            updateNumericInput(session, "sd10", value="a")
            updateNumericInput(session, "sd11", value="a")
            updateNumericInput(session, "sd12", value="a")
            updateNumericInput(session, "sd13", value="a")
            updateNumericInput(session, "sd14", value="a")
            updateNumericInput(session, "sd15", value="a")
```

```
            updateNumericInput(session, "iters", value=100)
            updateNumericInput(session, "a", value=.05)
            output$t1=renderText({})
            output$t2=renderText({})
            output$t3=renderText({})
            output$t4=renderText({})
            output$t5=renderText({})
 })
})
```

Script for Multiple ANOVA Simulations:

```
library(xlsx)
a=read.xlsx("Senior Project Settings.xlsx",1)
a
n1=a$n1; n2=a$n2; n3=a$n3
mu1=a$mu1; mu2=a$mu2; mu3=a$mu3
sd1=a$sigma1; sd2=a$sigma2; sd3=a$sigma3
alpha=.05
a$TargetPower=rep(NA,length(a$Setting))
a$Power=rep(NA,length(a$Setting))
for (i in 1:length(a$Setting)){
mu=((mu1[i]*n1[i])+(mu2[i]*n2[i])+(mu3[i]*n3[i]))/sum(n1[i]+n2[i]+n3[i])
psd= sqrt(((n1[i]-1)*sd1[i]^2+(n2[i]-1)*sd2[i]^2+(n3[i]-1)*sd3[i]^2)/((n1[i]-1)+(n2[i]-1)+(n3[i]-1)))
ncp= (n1[i]*((mu1[i]-mu)^2)/psd^2)+(n2[i]*((mu2[i]-mu)^2)/psd^2)+(n3[i]*((mu3[i]-mu)^2)/psd^2)
df1=2
df2=sum(n1[i]+n2[i]+n3[i])-1
crit.value = qf(1-alpha,df1,df2)
epower= 1-pf(crit.value,df1,df2,ncp)
epower
iters=100
sum1=0
  for(j in 1:iters){
            x=rnorm(n1[i],mu1[i],sd1[i])
            y=rnorm(n2[i],mu2[i],sd2[i])
            z=rnorm(n3[i],mu3[i],sd3[i])
            data=c(x,y,z)
            sample = factor(rep(letters[24:26], c(n1[i],n2[i],n3[i])))
            fit = lm(formula = data ~ sample)
```

```
                anova=anova(fit)
    if(anova$"Pr(>F)"[1]<alpha){
                sum1=sum1+1
                }
                power1=sum1/iters
 }
a$TargetPower[i]=round(epower,2)
a$Power[i]=power1
}
## write.table(a, "output.txt")
max.print <- getOption('max.print')
options(max.print=nrow(a) * ncol(a))
sink('output.txt')
a
sink()
options(max.print=max.prin)
```

Shiny App for Non-Constant Variance in Regression Simulation:

ui:

```
library(shiny)

shinyUI(fluidPage(
 titlePanel("Regression Simulation"),
 sidebarLayout(
                sidebarPanel(
                numericInput("beta1", "beta1", value=1),
                numericInput("n", "n", value=10),
                numericInput("n.state", "n.state", value=0),
                numericInput("sigma", "Sigma", value=10),
                numericInput("sigma.state", "sigma.state", value=0, step=0.25),
                numericInput("alpha", "Alpha level", value=.05, step=.01, min=0, max=1),
                numericInput("iter", "Iterations", value=1000),
                actionButton("st1", "Simulate"),
                br(),
                br(),
                actionButton("clear", "Clear Output"),
                 actionButton("reset", "Reset"),
                width=2
```

```
                ),
                mainPanel(
                tableOutput("t1"),
                br(),br(),br(),br(),br(),
                textOutput("t3"),
                tableOutput("t2")
                )
 )
))
```

server:

```
library(shiny)
shinyServer(function(input, output, session) {
  observeEvent(input$st1, {
                output$t1=renderTable({
                data.out=matrix(data=NA,ncol=5,nrow=1)
     dimnames(data.out)=list(NULL,c("power","epower","CI.power","power.diff","CI.power.diff"))
                data.out=as.data.frame(data.out)
                ex=1:10
                beta0=0;beta1=input$beta1
                # m = number of intervals for Bonferroni-adjusted binominal confidence intervals
                m = 1
                alpha=input$alpha
                # Changed dx formatting so that now n.state>0 means increasing and n.state<0 means decreasing
                dx=c(-2,-2,-1,-1,0,0,1,1,2,2)

                n2=input$n+input$n.state*dx
                x=rep(ex, n2)
        sigma2=input$sigma*(ex^input$sigma.state)
                sigma.v=rep(sigma2, n2)
        psigma=sqrt(sum(((n2-1)*(sigma2^2)))/(sum(n2-1)))
                sd.x=sd(x)
                slopes=NA
                p.value=NA
                for(j in 1:input$iter){
                epsilon=rnorm(length(x),0,sigma.v)
```

```
        y=beta0+beta1*x+epsilon
        fit=summary(lm(y~x))
        slopes[j]=fit$coefficients[2]
        p.value[j]=fit$coefficients[8]
        }
        reject=p.value<alpha
        power=sum(reject)/input$iter
        df=sum(n2)-2
        crit.value= qt(1-alpha/2,df)
        sd.b1=psigma/(sd.x*sqrt((sum(n2)-1)))
        ncp=(beta1-0)/sd.b1
 epower=pt(-crit.value,df,ncp)+(1-pt(crit.value,df,ncp))
        binom=binom.test(sum(reject),input$iter,epower,conf.level=1-alpha/m)
        power.diff=power-epower
        data.out$power=power
        data.out$epower=epower
 data.out$CI.power=paste("(",round(binom$conf.int[1],2),",",round(binom$conf.int[2],2),")")
        data.out$power.diff=power.diff
 data.out$CI.power.diff=paste("(",round(binom$conf.int[1]-epower,2),",",round(binom$conf.int[2]-epower,2),")")
        data.out
        })
        output$t3=renderText({
        paste("For x = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)")
        })
        output$t2=renderTable({
        data.list=matrix(data=NA,ncol=3,nrow=1)
dimnames(data.list)=list(NULL,c("SampleSizes", "Sigmas", "PooledSigma"))
        data.list=as.data.frame(data.list)
        ex=1:10
        dx=c(-2,-2,-1,-1,0,0,1,1,2,2)
        n2=input$n+input$n.state*dx
        x=rep(ex, n2)
sigma2=input$sigma*(ex^input$sigma.state)
psigma=sqrt(sum(((n2-1)*(sigma2^2)))/(sum(n2-1)))
          data.list$SampleSizes=paste(n2,collapse=", ")
```

```
    data.list$Sigmas=paste(round(sigma2,2),collapse=", ")
        data.list$PooledSigma=psigma
        data.list
        })
})
observeEvent(input$clear, {
        output$t1=renderTable({})
        output$t2=renderTable({})
        output$t3=renderText({})
})
observeEvent(input$reset, {
        output$t1=renderTable({})
        output$t3=renderText({})
        output$t2=renderTable({})
        updateNumericInput(session, "beta1", value=1)
        updateNumericInput(session, "n", value=10)
        updateNumericInput(session, "n.state", value=0)
        updateNumericInput(session, "sigma", value=10)
        updateNumericInput(session, "sigma.state", value=0)
        updateNumericInput(session, "alpha", value=.05)
        updateNumericInput(session, "iter", value=1000)

})
```

Script for Multiple Regression Simulations:

```
data=read.csv("input.csv")

data.out=matrix(data=NA,ncol=18,nrow=1)

dimnames(data.out)=list(NULL,c("n","n.state","sigma","sigma.state","alpha","iter","rep","psigma","sigma.x10","ncp","df","power","epower","L.power","U.power","
power.diff","L.power.diff","U.power.diff"))

data.out=as.data.frame(data.out)

data.out2=NULL

ex=1:10

beta0=0;beta1=1

# m = number of intervals for Bonferroni-adjusted binominal confidence intervals

m = 5
```

```
for(h in 1:nrow(data)){
        for (i in 1:data$rep[h]){
        alpha=data$alpha[h]
        # Changed dx formatting so that now n.state>0 means increasing and n.state<0 means decreasing
        dx=c(-2,-2,-1,-1,0,0,1,1,2,2)
        n2=data$n[h]+data$n.state[h]*dx
        x=rep(ex, n2)
 sigma2=data$sigma[h]*(ex^data$sigma.state[h])
        sigma.v=rep(sigma2, n2)
 psigma=sqrt(sum(((n2-1)*(sigma2^2)))/(sum(n2-1)))
        sd.x=sd(x)
        slopes=NA
        p.value=NA
        for(j in 1:data$iter[h]){
        epsilon=rnorm(length(x),0,sigma.v)
        y=beta0+beta1*x+epsilon
        fit=summary(lm(y~x))
        slopes[j]=fit$coefficients[2]
        p.value[j]=fit$coefficients[8]
        }
        reject=p.value<alpha
        power=sum(reject)/data$iter[h]
        df=sum(n2)-2
        crit.value= qt(1-alpha/2,df)
        sd.b1=psigma/(sd.x*sqrt((sum(n2)-1)))
        ncp=(beta1-0)/sd.b1
 epower=pt(-crit.value,df,ncp)+(1-pt(crit.value,df,ncp))
 binom=binom.test(sum(reject),data$iter[h],epower,conf.level=1-alpha/m)
        power.diff=power-epower
        data.out$n=data$n[h]
        data.out$n.state=data$n.state[h]
        data.out$sigma=data$sigma[h]
        data.out$sigma.state=data$sigma.state[h]
        data.out$alpha=data$alpha[h]
        data.out$iter=data$iter[h]
```

```
                data.out$rep=i
                data.out$psigma=psigma
                data.out$ncp=ncp
                data.out$df=df
                data.out$power=power
                data.out$epower=epower
                data.out$L.power=binom$conf.int[1]
                data.out$U.power=binom$conf.int[2]
                data.out$sigma.x10=sigma2[10]
                data.out$power.diff=power.diff
   data.out$L.power.diff=binom$conf.int[1]-epower
                data.out$U.power.diff=binom$conf.int[2]-epower
                data.out2=rbind(data.out2,data.out)
 }
}
write.csv(data.out2, "output.csv")
```