Power Analysis

Part 2 Statistical Power Analysis

Step 1 - build the models

```
# Null model
nullmod<- function(p,x,y){
  B0=p[1]
  sigma=exp(p[2])

  expected=B0

  null=-sum(dnorm(x=y,mean=expected,sd=sigma, log = TRUE))
  return(null)
}

# Regression model
lin<-function(p,x,y){
  B0=p[1]
  B1=p[2]
  sigma=exp(p[3])

  expected=B0+B1*x

  lmmod=-sum(dnorm(x=y,mean=expected,sd=sigma, log = TRUE))
  return(lmmod)
}

# T-test model
ttest<-function(p,x,y){
  B0=p[1]
  B1=p[2]
  sigma=exp(p[3])
  x1=x
  expected=B0+B1*x1

  tmod=-sum(dnorm(x=y,mean=expected,sd=sigma, log = TRUE))
  return(tmod)
}

# ANOVA - four groups
anovamod<-function(p,x,y){
B0=p[1]
B1=p[2]
B2=p[3]
B3=p[4]
sigma=exp(p[5])

expected=B0+B1*x[,1]+B2*x[,2]+B3*x[,3]

anova=-sum(dnorm(x=y,mean=expected,sd=sigma,log = TRUE))
return(anova)
}

# Now an ANOVA with 8 groups
eightanovamod<-function(p,x,y){
  B0=p[1]
  B1=p[2]
```

```
  B2=p[3]
  B3=p[4]
  B4=p[5]
  B5=p[6]
  B6=p[7]
  B7=p[8]
  sigma=exp(p[9])

  expected=B0+B1*x[,1]+B2*x[,2]+B3*x[,3]+B4*x[,4]+B5*x[,5]+B6*x[,6]+B7*x[,7]

  eightanovamod=-sum(dnorm(x=y,mean=expected,sd=sigma, log = TRUE))
  return(eightanovamod)
}
```

Step 2: Make your for loops!

```
# Set up an empty data frame in which to store your p-values for each sigma and model.
dfp <- setNames(data.frame(matrix(ncol = 5, nrow = 8)), c("Sigmas", "Regression", "T.tes
t", "Anova",  "Eight.Anova"))

# Set up your vector of sigmas to loop through
sigmas <- c(1,2,4,6,8,12,16,24)

# Write your nested for loop:
for (i in 1:length(sigmas)){ # sigma runs
  # set up your vectors for your p-values from each monte-carlo run.
  reg.p <- numeric(10) # changes each time you use a new sigma
  t.p <- numeric(10)
  anova.p <- numeric(10)
  eightanova.p <- numeric(10)
  for (j in 1:10){ # monte carlo runs
    x <- runif(24, 0, 50) # build set of x's
    e <- rnorm(24, 0, sd = sigmas[i]) # put in your error
    y <- 10 + 0.4*x + e # write your linear equation
    df <- cbind(x, y) # bind x and y into a data frame
    df <- as.data.frame(df)
    df <- df[order(df$x),] # order for later sorting.

    # dummy coding:

    # For t-test
    tdummy <- matrix(0, 24, 1)
    tdummy[13:24,1] <- 1

    # For anova
    dummy <- matrix(0, 24, 3)
    dummy[7:12,1] <- 1
    dummy[13:18,2] <-1
    dummy[19:24,3] <-1

    #for eight-level anova
    edummy <- matrix(0, 24, 7)
    edummy[4:6,1]<-1
    edummy[7:9,2]<-1
    edummy[10:12,3]<-1
    edummy[13:15,4]<-1
    edummy[16:18,5]<-1
    edummy[19:21,6]<-1
    edummy[22:24,7]<-1

    # guesses
    nullmodguess <- c(1,2) # 2
    lmmodguess <- c(1, 2, 3) # 3
    tmodguess<-c(1,1,1,1,1) # 5
    aovguess<- c(9,10,-6,8,2) # 5
    eightaovguess<-c(1, 12.5,15,17.5,20,22.5,25,27.5,30) # 9

    #optims
    nullfitmod=optim(par=nullmodguess,fn=nullmod,x=df$x,y=df$y) # converges
```

```
    lmfitmod=optim(par=lmmodguess,fn=lin,x=df$x,y=df$y) # converges
    tfitmod=optim(par=tmodguess,fn=ttest,x=tdummy,y=df$y) # converges
    anovafitmod=optim(par=aovguess,fn=anovamod,x=dummy,y=df$y, control=list(maxit=1e5))
# converges
    eightanovafitmod=optim(par=eightaovguess,fn=eightanovamod,x=edummy,y=df$y, control=l
ist(maxit=1e5)) # converges

    # degrees of freedom
     degfreg <- length(lmfitmod$par) - length(nullfitmod$par)
     degft <- length(tfitmod$par) - length(nullfitmod$par)
     degfanova <- length(anovafitmod$par) - length(nullfitmod$par)
     degfeightanova <- length(eightanovafitmod$par) - length(nullfitmod$par)

     # test stats
    t.statreg <- 2*(nullfitmod$value - lmfitmod$value)
    t.statt <- 2*(nullfitmod$value - tfitmod$value)
    t.statanova <- 2*(nullfitmod$value - anovafitmod$value)
    t.stateightanova <- 2*(nullfitmod$value - eightanovafitmod$value)

    # pvalues
    reg.p[j] <- 1-pchisq(t.statreg, df=degfreg)
    t.p[j]<- 1-pchisq(t.statt, df=degft)
    anova.p[j]<- 1-pchisq(t.statanova, df=degfanova)
    eightanova.p[j]<- 1-pchisq(t.stateightanova, df=degfeightanova)
  }

  # the mean p values for each of ten monte carlo runs, and each of 8 sigmas into a data
 frame.
  dfp[,1]<- c(1, 2, 4, 6, 8, 12, 16, 24)
  dfp[i,2] <- mean(reg.p)
  dfp[i,3] <- mean(t.p)
  dfp[i,4] <- mean(anova.p)
  dfp[i,5] <- mean(eightanova.p)
}
```

Now reshape your dataframe of p-values into something you can plot!

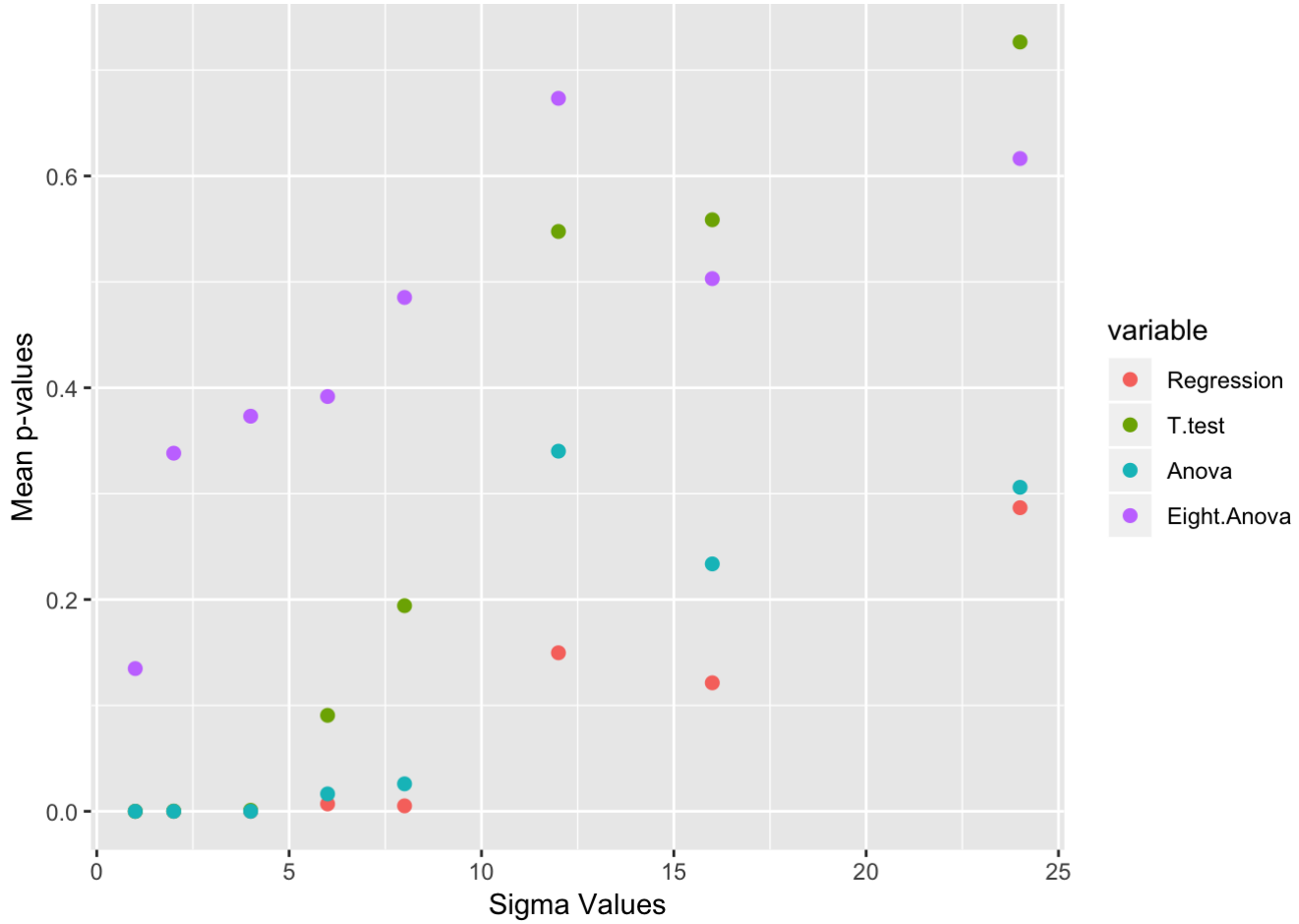
```
mframe <- melt(dfp)
```

```
## No id variables; using all as measure variables
```

```
mframe <- mframe[9:40,]
allsigmas <- c(1,  2,  4,  6,  8, 12, 16, 24, 1,  2,  4,  6,  8,12, 16, 24, 1,  2,  4,
6,  8, 12, 16, 24, 1,  2,  4,  6,  8, 12, 16, 24)
mframe$sigma <- allsigmas
```

Step 3: Visualize!!

```
mframe <- as.data.frame(mframe)
p <- ggplot(mframe, aes(x=mframe$sigma, y=mframe$value, colour=variable)) + geom_point
(size=2) + xlab("Sigma Values") + ylab("Mean p-values")
p
```



Step 4: Interpret

The regression model performs best across the sigmas, as evidenced by it's low p-value scores (see figure above). We originally believed that the eight level anova (Eight.Anova) should perform close to the regression, when in fact it performed the worst out of all four models. The four level anova (Anova) performs in between the t-test (T.test) and the regression. Finally the t-test, as expected, performed worse than either the regression or the four level ANOVA. This all makes sense, as our data is linearly correlated, so a linear regression should fit best.

Upon talking to Stuart, with the eight level ANOVA, we are starting to make finer and finer comparisons. When there's a linear relationship with noise and you split the x finer, the p-values should be high. With fewer groups, as in the four-way ANOVA, the groups shouldn't overlap as much, and the p-values should be lower.

It's clearly best to try to fit linear data to a linear model, but when resources are constrained, it seems the best strategy is to fit as many non-overlapping groups as possible. In this case, you need at least four experimental units (four way anova), to detect a significant signal within the noise.