Courtship\_Code\_Summary\_Compilation

Nathan Phipps

Loading all packages

library(tidyverse)  
library(lmerTest)  
library(plyr)  
library(dplyr)  
library(car)  
require(MASS)  
library(mlmRev)  
library(agridat)  
library(MCMCglmm)  
library(ggplot2)  
library(emmeans)

### INITIAL IMPORT, INSPECTION, AND CLEANUP OF DATA

Importing and checking structure of data frame. CourtMCMC is created because I will alter the dataframe differently in order to run MCMC, which would disrupt further non-MCMC analysis.

Court <- read.csv("C:/Users/phipp/OneDrive/Documents/Grad School/Courtship Analysis/Data/FINAL\_court.csv")  
CourtMCMC <- read.csv("C:/Users/phipp/OneDrive/Documents/Grad School/Courtship Analysis/Data/FINAL\_court.csv")  
str(Court)

## 'data.frame': 398 obs. of 22 variables:  
## $ ID : int 6 9 11 11 14 14 17 17 18 18 ...  
## $ Pop : chr "QC" "QC" "CQ" "CQ" ...  
## $ Family : int 401 419 418 418 419 419 418 418 419 419 ...  
## $ DateBorn : chr "4/16/2016" "4/16/2016" "4/16/2016" "4/16/2016" ...  
## $ Sex : chr "M" "M" "M" "M" ...  
## $ Length : num 1.96 1.72 1.61 1.61 1.62 ...  
## $ Stage : chr "Before" "Before" "Before" "After" ...  
## $ Date : chr "7/20/2016" "7/21/2016" "7/21/2016" "7/22/2016" ...  
## $ Time : int 1355 1700 1725 1635 1700 1600 1725 1635 1725 1635 ...  
## $ Coder : chr "NMP" "NMP" "NMP" "NMP" ...  
## $ Chase : int 0 0 1 3 2 1 2 0 0 1 ...  
## $ Forced\_Cop\_Attempt: int 0 0 4 0 1 0 0 3 22 0 ...  
## $ G\_Swing : int 1 14 11 6 2 4 20 9 39 11 ...  
## $ Ignore : int 1 2 1 1 2 2 1 2 0 1 ...  
## $ Lunge : int 1 0 0 2 3 0 2 1 1 0 ...  
## $ Nip : int 0 0 1 1 0 0 0 0 5 5 ...  
## $ Sigmoid : int 1 1 1 1 0 0 0 0 7 0 ...  
## $ Suc\_Forced\_Cop : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ Suc\_Sigmoid\_Cop : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ TTSigmoid : num 20.266 2.658 1.685 0.979 0 ...  
## $ TTAvgSigmoid : num 20.266 2.658 1.685 0.979 0 ...  
## $ Copulated : chr "N" "N" "N" "N" ...

Specifying data types and variable types for each column

cols(ID = col\_integer(),  
 Pop = col\_character(),  
 Family = col\_integer(),  
 DateBorn = col\_character(),  
 Sex = col\_character(),  
 Length = col\_double(),  
 Stage = col\_character(),  
 Date = col\_character(),  
 Coder = col\_character(),  
 Chase = col\_integer(),  
 Forced\_Cop\_Attempt = col\_integer(),  
 G\_Swing = col\_integer(),  
 Sigmoid = col\_integer(),  
 Suc\_Sigmoid\_Cop = col\_integer(),  
 Ignore = col\_integer(),  
 Lunge = col\_integer(),  
 Nip = col\_integer(),  
 Suc\_Forced\_Cop = col\_integer(),  
 Copulated = col\_character(),  
 TTSigmoid = col\_integer(),  
 TTAvgSigmoid = col\_integer())

## cols(  
## ID = col\_integer(),  
## Pop = col\_character(),  
## Family = col\_integer(),  
## DateBorn = col\_character(),  
## Sex = col\_character(),  
## Length = col\_double(),  
## Stage = col\_character(),  
## Date = col\_character(),  
## Coder = col\_character(),  
## Chase = col\_integer(),  
## Forced\_Cop\_Attempt = col\_integer(),  
## G\_Swing = col\_integer(),  
## Sigmoid = col\_integer(),  
## Suc\_Sigmoid\_Cop = col\_integer(),  
## Ignore = col\_integer(),  
## Lunge = col\_integer(),  
## Nip = col\_integer(),  
## Suc\_Forced\_Cop = col\_integer(),  
## Copulated = col\_character(),  
## TTSigmoid = col\_integer(),  
## TTAvgSigmoid = col\_integer()  
## )

#Making four variables into factors  
Court %>%  
 mutate(Pop = as.factor(Pop)) %>%  
 mutate(Stage = as.factor(Stage)) %>%  
 mutate(Family = as.factor(Family)) %>%  
 mutate(ID = as.factor(ID)) -> Court

Checking the sample size for each encounter with female and each genetic line

#How many "first encounter" trials?  
nrow(Court[Court$Stage == "Before",])

## [1] 198

#How many "second encounter" trials?  
nrow(Court[Court$Stage == "After",])

## [1] 200

#Create dataframe object of ONLY "second encounter" trials  
Court.tmp <- Court [Court$Stage == "After",]  
  
#EXAMPLE: How many "second encounter" trials for the Campo line? (Can be edited for any genetic line/encounter combo)  
length (unique(Court.tmp$Pop == "CM"))

## [1] 2

#How many total trials for the Campo line? (Includes both trials for each ind.)  
nrow(Court[Court$Pop == "CM",])

## [1] 24

#Create dataframe object of all trials for ONLY Campo individuals  
Court.tmp1 <- Court [Court$Pop == "CM",]  
#How many Campo individuals were observed?  
length (unique(Court.tmp1$ID))

## [1] 13

nrow(Court [Court$Pop == "CQ",])

## [1] 224

Court.tmp2 <- Court [Court$Pop == "CQ",]  
#How many CQ intercross individuals were observed?  
length (unique(Court.tmp2$ID))

## [1] 122

nrow(Court[Court$Pop == "QC",])

## [1] 120

Court.tmp3 <- Court [Court$Pop == "QC",]  
#How many QC intercross individuals were observed?  
length (unique(Court.tmp3$ID))

## [1] 65

nrow(Court[Court$Pop == "QH",])

## [1] 30

Court.tmp4 <- Court [Court$Pop == "QH",]  
#How many Quare individuals were observed?  
length (unique(Court.tmp4$ID))

## [1] 16

Checking family size for a single given family, and then a for loop to do the same for all families.

#How many families are there?  
length(unique(Court$Family))

## [1] 20

#How many individuals are in family 401?  
nrow(Court[Court$Family == "401",])

## [1] 68

#How many "second encounter" trials?  
nrow(Court[Court$Stage == "After",])

## [1] 200

#Create dataframe object of ONLY "second encounter" trials  
Court.tmp <- Court [Court$Stage == "After",]  
  
#Creating an object called FamilySize for the for loop to use. I am unsure why this is necessary, but it's the only way I've found to make the loop run.  
FamilySize <- 0  
   
#For each family from 401 to 438, find the number of individuals in that family. If there are zero members of the family, skip it and move to the next family. Print the output next to the family number.  
  
for(i in 401:438) {  
 FamilySize[i] <- nrow(Court[Court$Family == i,])  
 {if (FamilySize[i]==0)  
 next  
 }   
 FamilySize[i] <- nrow(Court[Court$Family == i,])  
 print(c(i, FamilySize[i]))  
}

## [1] 401 68  
## [1] 404 69  
## [1] 405 11  
## [1] 409 12  
## [1] 411 8  
## [1] 414 15  
## [1] 416 21  
## [1] 417 6  
## [1] 418 12  
## [1] 419 19  
## [1] 420 11  
## [1] 424 4  
## [1] 426 36  
## [1] 430 24  
## [1] 431 27  
## [1] 432 8  
## [1] 434 22  
## [1] 435 4  
## [1] 436 6  
## [1] 438 15

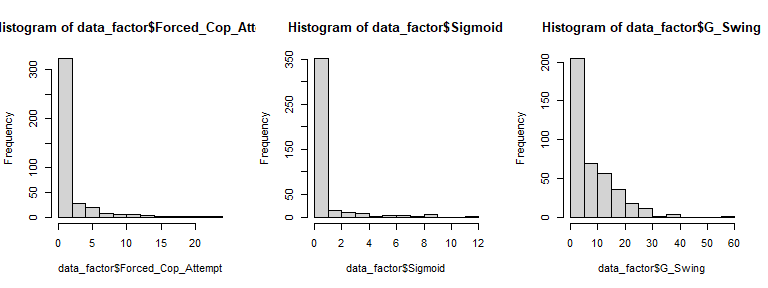
### FIRST ANALYSIS STAGE: DATA DIAGNOSTICS AND CORRELATION TESTING

# Dataset prep

#Apply factor mutation to characters  
data\_factor <- Court %>%  
 mutate\_if(sapply(Court, is.character), as.factor)  
 #Check that it worked  
sapply(data\_factor, class)

## ID Pop Family DateBorn   
## "factor" "factor" "factor" "factor"   
## Sex Length Stage Date   
## "factor" "numeric" "factor" "factor"   
## Time Coder Chase Forced\_Cop\_Attempt   
## "integer" "factor" "integer" "integer"   
## G\_Swing Ignore Lunge Nip   
## "integer" "integer" "integer" "integer"   
## Sigmoid Suc\_Forced\_Cop Suc\_Sigmoid\_Cop TTSigmoid   
## "integer" "integer" "integer" "numeric"   
## TTAvgSigmoid Copulated   
## "numeric" "factor"

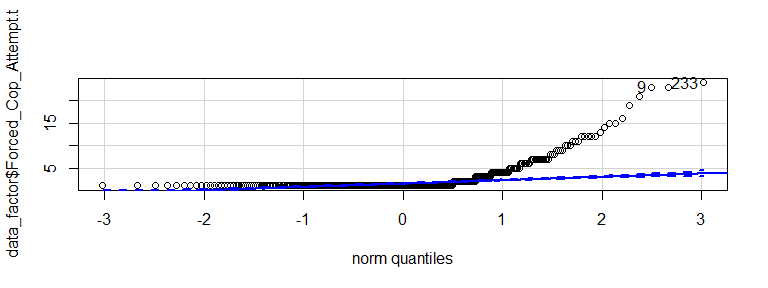
#Visualize data in histogram  
par(mfrow = c(1,3))  
hist(data\_factor$Forced\_Cop\_Attempt)  
hist(data\_factor$Sigmoid)  
hist(data\_factor$G\_Swing)



# Statistical tests for best distribution

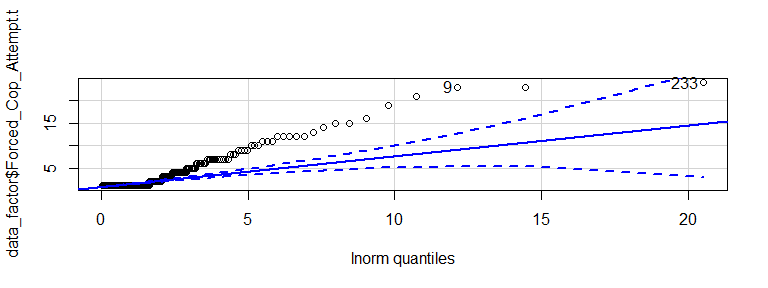
Forced copulation attempts distribution

data\_factor$Forced\_Cop\_Attempt.t <- data\_factor$Forced\_Cop\_Attempt + 1   
  
 #normal test  
 qqp(data\_factor$Forced\_Cop\_Attempt.t, "norm")



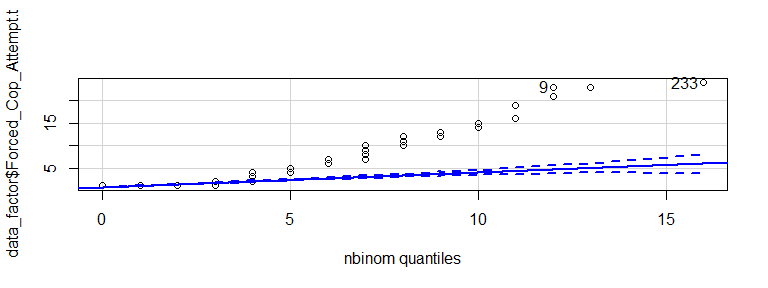
## [1] 233 9

#lognormal  
 qqp(data\_factor$Forced\_Cop\_Attempt.t, "lnorm")



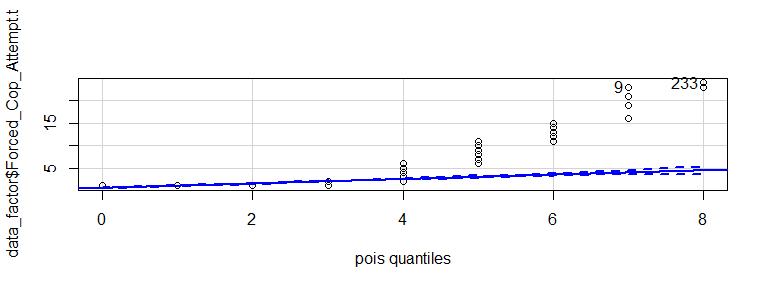
## [1] 233 9

#Negative binomial  
nbinom <- fitdistr(data\_factor$Forced\_Cop\_Attempt.t, "Negative Binomial")  
qqp(data\_factor$Forced\_Cop\_Attempt.t, "nbinom", size = nbinom$estimate[[1]], mu = nbinom$estimate[[2]])



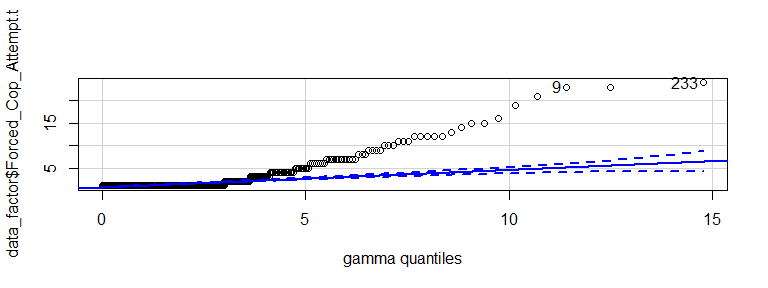
## [1] 233 9

#poisson  
 poisson <- fitdistr(data\_factor$Forced\_Cop\_Attempt.t, "Poisson")  
qqp(data\_factor$Forced\_Cop\_Attempt.t, "pois", lambda = poisson$estimate)



## [1] 233 9

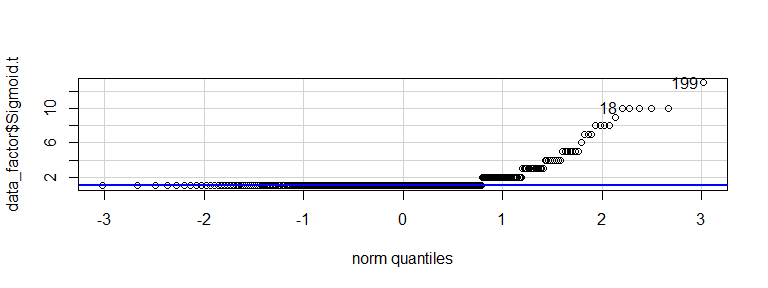
#gamma  
gamma <- fitdistr(data\_factor$Forced\_Cop\_Attempt.t, "gamma")  
qqp(data\_factor$Forced\_Cop\_Attempt.t, "gamma", shape = gamma$estimate[[1]], rate = gamma$estimate[[2]])



## [1] 233 9

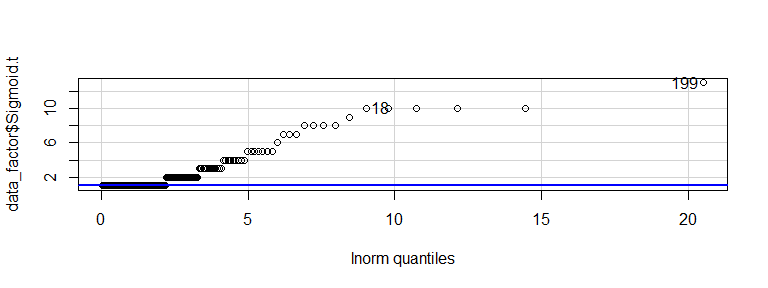
Sigmoids distribution

data\_factor$Sigmoid.t <- data\_factor$Sigmoid + 1  
qqp(data\_factor$Sigmoid.t, "norm")



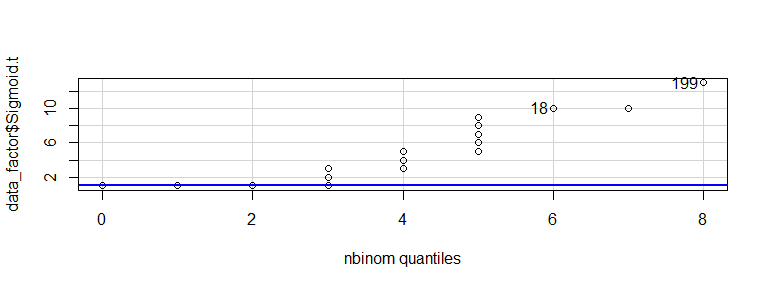
## [1] 199 18

#lognormal  
qqp(data\_factor$Sigmoid.t, "lnorm")



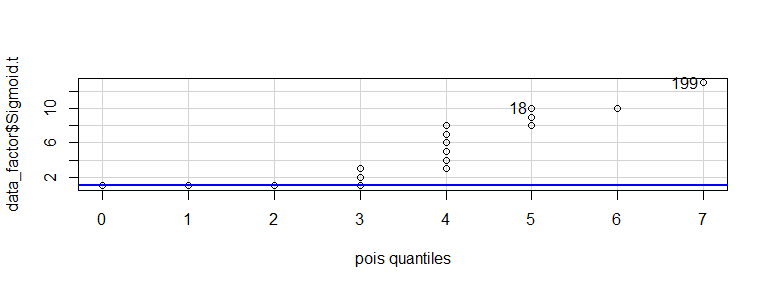
## [1] 199 18

#negative binomial  
nbinom <- fitdistr(data\_factor$Sigmoid.t, "Negative Binomial")  
qqp(data\_factor$Sigmoid.t, "nbinom", size = nbinom$estimate[[1]], mu = nbinom$estimate[[2]])



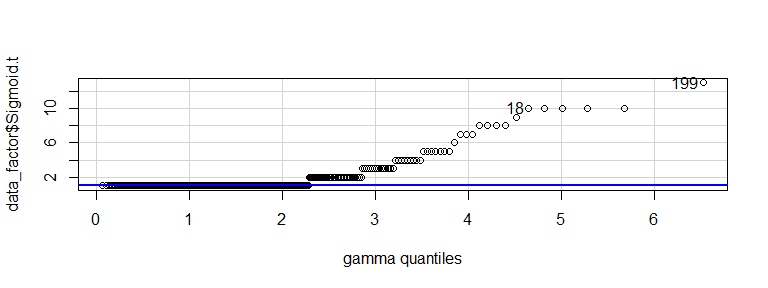
## [1] 199 18

#poisson  
poisson <- fitdistr(data\_factor$Sigmoid.t, "Poisson")  
qqp(data\_factor$Sigmoid.t, "pois", lambda = poisson$estimate)



## [1] 199 18

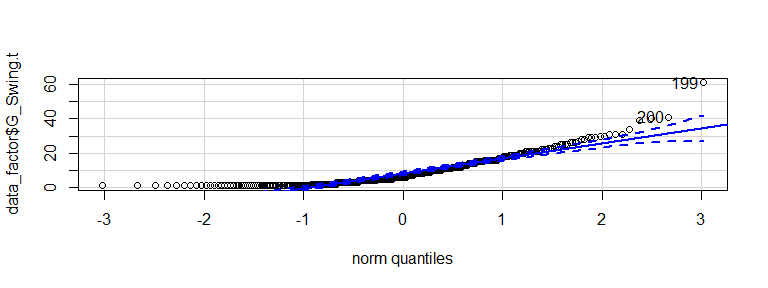
#gamma  
gamma <- fitdistr(data\_factor$Sigmoid.t, "gamma")  
qqp(data\_factor$Sigmoid.t, "gamma", shape = gamma$estimate[[1]], rate = gamma$estimate[[2]])



## [1] 199 18

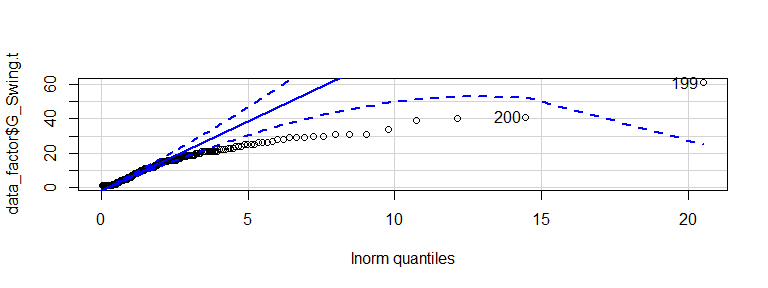
Gonopoodium Swings distribution

data\_factor$G\_Swing.t <- data\_factor$G\_Swing + 1   
  
 #normal  
qqp(data\_factor$G\_Swing.t, "norm")



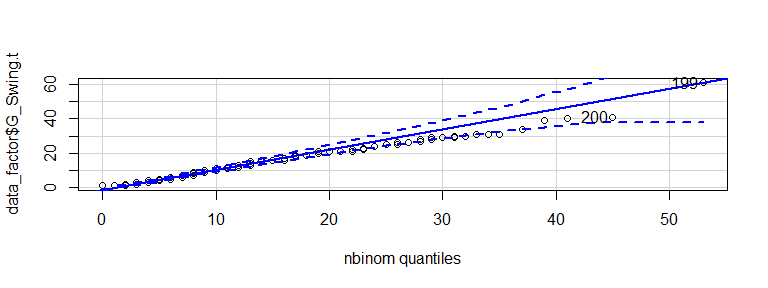
## [1] 199 200

#lognorm  
qqp(data\_factor$G\_Swing.t, "lnorm")



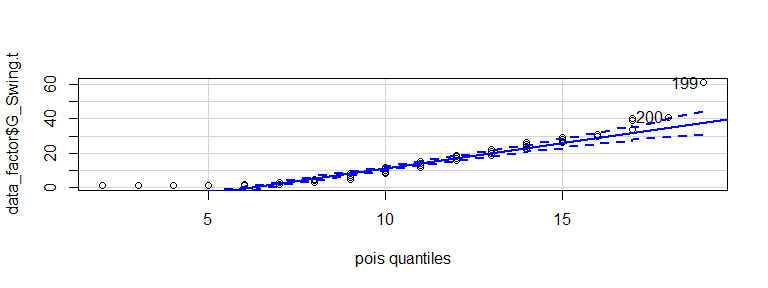
## [1] 199 200

#Negative Binomial  
nbinom <- fitdistr(data\_factor$G\_Swing.t, "Negative Binomial")  
qqp(data\_factor$G\_Swing.t, "nbinom", size = nbinom$estimate[[1]], mu = nbinom$estimate[[2]])



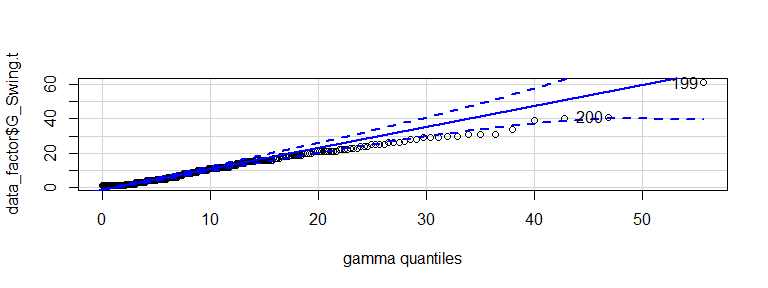
## [1] 199 200

#poisson  
poisson <- fitdistr(data\_factor$G\_Swing.t, "Poisson")  
qqp(data\_factor$G\_Swing.t, "pois", lambda = poisson$estimate)



## [1] 199 200

#gamma  
gamma <- fitdistr(data\_factor$G\_Swing.t, "gamma")  
qqp(data\_factor$G\_Swing.t, "gamma", shape = gamma$estimate[[1]], rate = gamma$estimate[[2]])



## [1] 199 200

Model comparisons, residual tests, and other fit analyses were performed using the “try” function, but they’re too janky and interfere with the knit function. See the file “Relevent code for Laura v. 2. R” for this code.

### SECOND ANALYSIS STAGE: LINEAR MODELS AND TRYING DIFFERENT DISTRIBUTION FITS

#Creating linear mixed models for sigmoids, forced copulations, and gonopodium swings using genetic line and experience as predictors, accounting for body length

Sigmoids

lmm.Sig <- lmer(Sigmoid ~ Pop\*Stage + Length   
 + (1|Family/ID),  
 data=Court)  
  
anova(lmm.Sig)

## Type III Analysis of Variance Table with Satterthwaite's method  
## Sum Sq Mean Sq NumDF DenDF F value Pr(>F)   
## Pop 26.8300 8.9433 3 94.72 3.8305 0.01226 \*  
## Stage 13.5202 13.5202 1 202.61 5.7908 0.01701 \*  
## Length 0.0951 0.0951 1 189.62 0.0407 0.84031   
## Pop:Stage 15.7263 5.2421 3 199.64 2.2452 0.08426 .  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

ranova(lmm.Sig)

## ANOVA-like table for random-effects: Single term deletions  
##   
## Model:  
## Sigmoid ~ Pop + Stage + Length + (1 | ID:Family) + (1 | Family) + Pop:Stage  
## npar logLik AIC LRT Df Pr(>Chisq)   
## <none> 12 -752.79 1529.6   
## (1 | ID:Family) 11 -753.19 1528.4 0.8068 1 0.36908   
## (1 | Family) 11 -755.86 1533.7 6.1456 1 0.01317 \*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Forced Copulations

lmm.Force <- lmer(Forced\_Cop\_Attempt ~ Pop\*Stage + Length  
 + (1|Family/ID),  
 data=Court)  
  
anova(lmm.Force)

## Type III Analysis of Variance Table with Satterthwaite's method  
## Sum Sq Mean Sq NumDF DenDF F value Pr(>F)   
## Pop 43.543 14.514 3 76.27 1.9402 0.13019   
## Stage 0.000 0.000 1 195.49 0.0000 0.99524   
## Length 36.488 36.488 1 178.55 4.8777 0.02848 \*  
## Pop:Stage 7.549 2.516 3 193.87 0.3364 0.79905   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

ranova(lmm.Force)

## ANOVA-like table for random-effects: Single term deletions  
##   
## Model:  
## Forced\_Cop\_Attempt ~ Pop + Stage + Length + (1 | ID:Family) + (1 | Family) + Pop:Stage  
## npar logLik AIC LRT Df Pr(>Chisq)   
## <none> 12 -1035.8 2095.6   
## (1 | ID:Family) 11 -1046.9 2115.9 22.2501 1 2.393e-06 \*\*\*  
## (1 | Family) 11 -1036.5 2094.9 1.2882 1 0.2564   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Gonopodium Swings

lmm.Swing <- lmer(G\_Swing ~ Pop\*Stage + Length  
 + (1|Family/ID),  
 data=Court)  
  
anova(lmm.Swing)

## Type III Analysis of Variance Table with Satterthwaite's method  
## Sum Sq Mean Sq NumDF DenDF F value Pr(>F)   
## Pop 394.75 131.583 3 111.00 3.4557 0.01895 \*  
## Stage 40.83 40.827 1 201.19 1.0722 0.30168   
## Length 85.64 85.643 1 208.13 2.2492 0.13520   
## Pop:Stage 54.97 18.324 3 199.70 0.4813 0.69568   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

ranova(lmm.Swing)

## ANOVA-like table for random-effects: Single term deletions  
##   
## Model:  
## G\_Swing ~ Pop + Stage + Length + (1 | ID:Family) + (1 | Family) + Pop:Stage  
## npar logLik AIC LRT Df Pr(>Chisq)   
## <none> 12 -1366.8 2757.6   
## (1 | ID:Family) 11 -1381.7 2785.3 29.746 1 4.926e-08 \*\*\*  
## (1 | Family) 11 -1371.9 2765.8 10.230 1 0.001381 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Turn ANOVAS into objects

anova.Sig<-anova(lmm.Sig)  
anova.Force<-anova(lmm.Force)  
anova.Swing<-anova(lmm.Swing)

Attempt pairwise comparisons using emmeans and Tukey’s HSD (These aren’t running. Notes from STAT 511 should show how to make these models properly)

#emmeans(lmm.Sig)  
  
#aov.Swing<-aov(G\_Swing ~ Pop\*Stage + Length, data=Court)  
#TukeyHSD(aov.Swing, which, ordered = FALSE, conf.level = 0.95)

### THIRD ANALYSIS STAGE: MCMCglmm (Markov Chain Monte Carlo)

Changing column name from “Family” to “famil” to avoid confusion with MCMC function “family”, and checking that it worked

colnames(CourtMCMC)[colnames(CourtMCMC) == "Family"] <- "famil"  
str(CourtMCMC)

## 'data.frame': 398 obs. of 22 variables:  
## $ ID : int 6 9 11 11 14 14 17 17 18 18 ...  
## $ Pop : chr "QC" "QC" "CQ" "CQ" ...  
## $ famil : int 401 419 418 418 419 419 418 418 419 419 ...  
## $ DateBorn : chr "4/16/2016" "4/16/2016" "4/16/2016" "4/16/2016" ...  
## $ Sex : chr "M" "M" "M" "M" ...  
## $ Length : num 1.96 1.72 1.61 1.61 1.62 ...  
## $ Stage : chr "Before" "Before" "Before" "After" ...  
## $ Date : chr "7/20/2016" "7/21/2016" "7/21/2016" "7/22/2016" ...  
## $ Time : int 1355 1700 1725 1635 1700 1600 1725 1635 1725 1635 ...  
## $ Coder : chr "NMP" "NMP" "NMP" "NMP" ...  
## $ Chase : int 0 0 1 3 2 1 2 0 0 1 ...  
## $ Forced\_Cop\_Attempt: int 0 0 4 0 1 0 0 3 22 0 ...  
## $ G\_Swing : int 1 14 11 6 2 4 20 9 39 11 ...  
## $ Ignore : int 1 2 1 1 2 2 1 2 0 1 ...  
## $ Lunge : int 1 0 0 2 3 0 2 1 1 0 ...  
## $ Nip : int 0 0 1 1 0 0 0 0 5 5 ...  
## $ Sigmoid : int 1 1 1 1 0 0 0 0 7 0 ...  
## $ Suc\_Forced\_Cop : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ Suc\_Sigmoid\_Cop : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ TTSigmoid : num 20.266 2.658 1.685 0.979 0 ...  
## $ TTAvgSigmoid : num 20.266 2.658 1.685 0.979 0 ...  
## $ Copulated : chr "N" "N" "N" "N" ...

Creating Gaussian regression models using genetic line and experience as interactive predictors

fit\_gaussian\_Sig <- MCMCglmm(Sigmoid ~ Pop\*Stage,  
 random = ~idv(1):famil + idv(1):ID,  
 data = CourtMCMC,  
 nitt=20000)

##   
## MCMC iteration = 0  
##   
## MCMC iteration = 1000  
##   
## MCMC iteration = 2000  
##   
## MCMC iteration = 3000  
##   
## MCMC iteration = 4000  
##   
## MCMC iteration = 5000  
##   
## MCMC iteration = 6000  
##   
## MCMC iteration = 7000  
##   
## MCMC iteration = 8000  
##   
## MCMC iteration = 9000  
##   
## MCMC iteration = 10000  
##   
## MCMC iteration = 11000  
##   
## MCMC iteration = 12000  
##   
## MCMC iteration = 13000  
##   
## MCMC iteration = 14000  
##   
## MCMC iteration = 15000  
##   
## MCMC iteration = 16000  
##   
## MCMC iteration = 17000  
##   
## MCMC iteration = 18000  
##   
## MCMC iteration = 19000  
##   
## MCMC iteration = 20000

fit\_gaussian\_Cop <- MCMCglmm(Forced\_Cop\_Attempt ~ Pop\*Stage,  
 random = ~idv(1):famil + idv(1):ID,  
 data = CourtMCMC,  
 nitt=20000)

##   
## MCMC iteration = 0  
##   
## MCMC iteration = 1000  
##   
## MCMC iteration = 2000  
##   
## MCMC iteration = 3000  
##   
## MCMC iteration = 4000  
##   
## MCMC iteration = 5000  
##   
## MCMC iteration = 6000  
##   
## MCMC iteration = 7000  
##   
## MCMC iteration = 8000  
##   
## MCMC iteration = 9000  
##   
## MCMC iteration = 10000  
##   
## MCMC iteration = 11000  
##   
## MCMC iteration = 12000  
##   
## MCMC iteration = 13000  
##   
## MCMC iteration = 14000  
##   
## MCMC iteration = 15000  
##   
## MCMC iteration = 16000  
##   
## MCMC iteration = 17000  
##   
## MCMC iteration = 18000  
##   
## MCMC iteration = 19000  
##   
## MCMC iteration = 20000

fit\_gaussian\_Swing <- MCMCglmm(G\_Swing ~ Pop\*Stage,  
 random = ~idv(1):famil + idv(1):ID,  
 data = CourtMCMC,  
 nitt=20000)

##   
## MCMC iteration = 0  
##   
## MCMC iteration = 1000  
##   
## MCMC iteration = 2000  
##   
## MCMC iteration = 3000  
##   
## MCMC iteration = 4000  
##   
## MCMC iteration = 5000  
##   
## MCMC iteration = 6000  
##   
## MCMC iteration = 7000  
##   
## MCMC iteration = 8000  
##   
## MCMC iteration = 9000  
##   
## MCMC iteration = 10000  
##   
## MCMC iteration = 11000  
##   
## MCMC iteration = 12000  
##   
## MCMC iteration = 13000  
##   
## MCMC iteration = 14000  
##   
## MCMC iteration = 15000  
##   
## MCMC iteration = 16000  
##   
## MCMC iteration = 17000  
##   
## MCMC iteration = 18000  
##   
## MCMC iteration = 19000  
##   
## MCMC iteration = 20000

Creating Poisson regression models using genetic line and experience as interactive predictors

fit\_poisson\_Sig <- MCMCglmm(Sigmoid ~ Pop\*Stage,  
 random = ~idv(1):famil + idv(1):ID,  
 family = "poisson", data = CourtMCMC,  
 nitt=20000)

##   
## MCMC iteration = 0  
##   
## Acceptance ratio for liability set 1 = 0.000701  
##   
## MCMC iteration = 1000  
##   
## Acceptance ratio for liability set 1 = 0.395151  
##   
## MCMC iteration = 2000  
##   
## Acceptance ratio for liability set 1 = 0.399204  
##   
## MCMC iteration = 3000  
##   
## Acceptance ratio for liability set 1 = 0.396505  
##   
## MCMC iteration = 4000  
##   
## Acceptance ratio for liability set 1 = 0.367673  
##   
## MCMC iteration = 5000  
##   
## Acceptance ratio for liability set 1 = 0.392269  
##   
## MCMC iteration = 6000  
##   
## Acceptance ratio for liability set 1 = 0.395276  
##   
## MCMC iteration = 7000  
##   
## Acceptance ratio for liability set 1 = 0.381211  
##   
## MCMC iteration = 8000  
##   
## Acceptance ratio for liability set 1 = 0.367095  
##   
## MCMC iteration = 9000  
##   
## Acceptance ratio for liability set 1 = 0.377565  
##   
## MCMC iteration = 10000  
##   
## Acceptance ratio for liability set 1 = 0.370666  
##   
## MCMC iteration = 11000  
##   
## Acceptance ratio for liability set 1 = 0.382158  
##   
## MCMC iteration = 12000  
##   
## Acceptance ratio for liability set 1 = 0.371103  
##   
## MCMC iteration = 13000  
##   
## Acceptance ratio for liability set 1 = 0.396992  
##   
## MCMC iteration = 14000  
##   
## Acceptance ratio for liability set 1 = 0.396158  
##   
## MCMC iteration = 15000  
##   
## Acceptance ratio for liability set 1 = 0.384111  
##   
## MCMC iteration = 16000  
##   
## Acceptance ratio for liability set 1 = 0.388769  
##   
## MCMC iteration = 17000  
##   
## Acceptance ratio for liability set 1 = 0.364950  
##   
## MCMC iteration = 18000  
##   
## Acceptance ratio for liability set 1 = 0.381296  
##   
## MCMC iteration = 19000  
##   
## Acceptance ratio for liability set 1 = 0.383578  
##   
## MCMC iteration = 20000  
##   
## Acceptance ratio for liability set 1 = 0.380256

fit\_poisson\_Cop <- MCMCglmm(Forced\_Cop\_Attempt ~ Pop\*Stage,  
 random = ~idv(1):famil + idv(1):ID,  
 family = "poisson", data = CourtMCMC,  
 nitt=20000)

##   
## MCMC iteration = 0  
##   
## Acceptance ratio for liability set 1 = 0.000616  
##   
## MCMC iteration = 1000  
##   
## Acceptance ratio for liability set 1 = 0.416452  
##   
## MCMC iteration = 2000  
##   
## Acceptance ratio for liability set 1 = 0.410588  
##   
## MCMC iteration = 3000  
##   
## Acceptance ratio for liability set 1 = 0.410525  
##   
## MCMC iteration = 4000  
##   
## Acceptance ratio for liability set 1 = 0.434206  
##   
## MCMC iteration = 5000  
##   
## Acceptance ratio for liability set 1 = 0.432666  
##   
## MCMC iteration = 6000  
##   
## Acceptance ratio for liability set 1 = 0.440239  
##   
## MCMC iteration = 7000  
##   
## Acceptance ratio for liability set 1 = 0.432136  
##   
## MCMC iteration = 8000  
##   
## Acceptance ratio for liability set 1 = 0.447658  
##   
## MCMC iteration = 9000  
##   
## Acceptance ratio for liability set 1 = 0.433505  
##   
## MCMC iteration = 10000  
##   
## Acceptance ratio for liability set 1 = 0.440598  
##   
## MCMC iteration = 11000  
##   
## Acceptance ratio for liability set 1 = 0.437420  
##   
## MCMC iteration = 12000  
##   
## Acceptance ratio for liability set 1 = 0.426879  
##   
## MCMC iteration = 13000  
##   
## Acceptance ratio for liability set 1 = 0.433771  
##   
## MCMC iteration = 14000  
##   
## Acceptance ratio for liability set 1 = 0.428307  
##   
## MCMC iteration = 15000  
##   
## Acceptance ratio for liability set 1 = 0.449259  
##   
## MCMC iteration = 16000  
##   
## Acceptance ratio for liability set 1 = 0.442389  
##   
## MCMC iteration = 17000  
##   
## Acceptance ratio for liability set 1 = 0.435033  
##   
## MCMC iteration = 18000  
##   
## Acceptance ratio for liability set 1 = 0.419568  
##   
## MCMC iteration = 19000  
##   
## Acceptance ratio for liability set 1 = 0.438113  
##   
## MCMC iteration = 20000  
##   
## Acceptance ratio for liability set 1 = 0.431462

fit\_poisson\_Swing <- MCMCglmm(G\_Swing ~ Pop\*Stage,  
 random = ~idv(1):famil + idv(1):ID,  
 family = "poisson", data = CourtMCMC,  
 nitt=20000)

##   
## MCMC iteration = 0  
##   
## Acceptance ratio for liability set 1 = 0.000435  
##   
## MCMC iteration = 1000  
##   
## Acceptance ratio for liability set 1 = 0.403183  
##   
## MCMC iteration = 2000  
##   
## Acceptance ratio for liability set 1 = 0.407676  
##   
## MCMC iteration = 3000  
##   
## Acceptance ratio for liability set 1 = 0.404425  
##   
## MCMC iteration = 4000  
##   
## Acceptance ratio for liability set 1 = 0.417754  
##   
## MCMC iteration = 5000  
##   
## Acceptance ratio for liability set 1 = 0.416136  
##   
## MCMC iteration = 6000  
##   
## Acceptance ratio for liability set 1 = 0.417455  
##   
## MCMC iteration = 7000  
##   
## Acceptance ratio for liability set 1 = 0.416550  
##   
## MCMC iteration = 8000  
##   
## Acceptance ratio for liability set 1 = 0.418349  
##   
## MCMC iteration = 9000  
##   
## Acceptance ratio for liability set 1 = 0.422915  
##   
## MCMC iteration = 10000  
##   
## Acceptance ratio for liability set 1 = 0.418008  
##   
## MCMC iteration = 11000  
##   
## Acceptance ratio for liability set 1 = 0.419716  
##   
## MCMC iteration = 12000  
##   
## Acceptance ratio for liability set 1 = 0.417101  
##   
## MCMC iteration = 13000  
##   
## Acceptance ratio for liability set 1 = 0.417387  
##   
## MCMC iteration = 14000  
##   
## Acceptance ratio for liability set 1 = 0.418794  
##   
## MCMC iteration = 15000  
##   
## Acceptance ratio for liability set 1 = 0.420058  
##   
## MCMC iteration = 16000  
##   
## Acceptance ratio for liability set 1 = 0.420153  
##   
## MCMC iteration = 17000  
##   
## Acceptance ratio for liability set 1 = 0.416997  
##   
## MCMC iteration = 18000  
##   
## Acceptance ratio for liability set 1 = 0.417884  
##   
## MCMC iteration = 19000  
##   
## Acceptance ratio for liability set 1 = 0.416580  
##   
## MCMC iteration = 20000  
##   
## Acceptance ratio for liability set 1 = 0.417744

#Using glmer for regression as comparison  
fit\_glmer <- glmer(Sigmoid ~ Pop\*Stage +  
 (1|famil/ID),  
 family = "poisson", data = CourtMCMC)  
 #Bad things? I don't understand

Examining results

summary(fit\_gaussian\_Sig)

##   
## Iterations = 3001:19991  
## Thinning interval = 10  
## Sample size = 1700   
##   
## DIC: 1518.392   
##   
## G-structure: ~idv(1):famil  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.famil 0.1586 2.755e-08 0.4783 63.24  
##   
## ~idv(1):ID  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.ID 0.1118 2.6e-07 0.4792 23.55  
##   
## R-structure: ~units  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## units 2.464 2.048 2.906 102.2  
##   
## Location effects: Sigmoid ~ Pop \* Stage   
##   
## post.mean l-95% CI u-95% CI eff.samp pMCMC   
## (Intercept) -0.03453 -0.97858 0.90321 1700 0.9259   
## PopCQ 0.50699 -0.50828 1.48694 1700 0.3082   
## PopQC 0.49212 -0.55502 1.48739 1700 0.3506   
## PopQH 0.70091 -0.53817 1.93267 1141 0.2871   
## StageBefore -0.01275 -1.31317 1.21999 1700 0.9929   
## PopCQ:StageBefore 0.17050 -1.14988 1.45254 1700 0.8200   
## PopQC:StageBefore 0.35366 -0.97673 1.81009 1700 0.6247   
## PopQH:StageBefore 1.68084 0.03494 3.25965 1700 0.0341 \*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(fit\_gaussian\_Cop)

##   
## Iterations = 3001:19991  
## Thinning interval = 10  
## Sample size = 1700   
##   
## DIC: 2054.719   
##   
## G-structure: ~idv(1):famil  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.famil 0.1438 7.572e-17 0.8564 49.46  
##   
## ~idv(1):ID  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.ID 4.561 2.592 6.577 1121  
##   
## R-structure: ~units  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## units 7.645 6.225 9.493 1494  
##   
## Location effects: Forced\_Cop\_Attempt ~ Pop \* Stage   
##   
## post.mean l-95% CI u-95% CI eff.samp pMCMC  
## (Intercept) 0.2807 -1.6684 2.3403 1700 0.774  
## PopCQ 0.9867 -0.9674 3.1084 1700 0.364  
## PopQC 1.7430 -0.3954 3.9425 1700 0.106  
## PopQH 1.5033 -1.2835 4.0492 1700 0.256  
## StageBefore -0.3548 -2.6584 2.0096 1700 0.742  
## PopCQ:StageBefore 0.7488 -1.6168 3.2105 1700 0.542  
## PopQC:StageBefore 0.1481 -2.3697 2.7960 1700 0.919  
## PopQH:StageBefore 0.6062 -2.7580 3.4453 1578 0.673

summary(fit\_gaussian\_Swing)

##   
## Iterations = 3001:19991  
## Thinning interval = 10  
## Sample size = 1700   
##   
## DIC: 2711.433   
##   
## G-structure: ~idv(1):famil  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.famil 6.989 1.466e-05 17.14 554.8  
##   
## ~idv(1):ID  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.ID 26.89 17.28 38.74 776  
##   
## R-structure: ~units  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## units 38.66 31.02 46.71 1585  
##   
## Location effects: G\_Swing ~ Pop \* Stage   
##   
## post.mean l-95% CI u-95% CI eff.samp pMCMC   
## (Intercept) 3.14579 -2.09677 8.23883 1700 0.2365   
## PopCQ 5.34257 0.08355 11.00603 1700 0.0529 .  
## PopQC 7.04074 1.57739 12.48793 1700 0.0165 \*  
## PopQH 7.96502 1.73895 14.77560 1700 0.0165 \*  
## StageBefore -1.68657 -6.84476 3.44550 1700 0.5294   
## PopCQ:StageBefore 1.05705 -4.51731 6.44877 1700 0.6906   
## PopQC:StageBefore -0.30958 -5.95830 5.62062 1700 0.9035   
## PopQH:StageBefore 2.03010 -4.26035 8.69443 1700 0.5624   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

str(fit\_gaussian\_Sig)

## List of 20  
## $ Sol : 'mcmc' num [1:1700, 1:8] -0.935 0.732 -0.756 0.153 0.315 ...  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : NULL  
## .. ..$ : chr [1:8] "(Intercept)" "PopCQ" "PopQC" "PopQH" ...  
## ..- attr(\*, "mcpar")= num [1:3] 3001 19991 10  
## $ Lambda : NULL  
## $ VCV : 'mcmc' num [1:1700, 1:3] 0.582 0.238 0.176 0.11 0.618 ...  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : NULL  
## .. ..$ : chr [1:3] "1.famil" "1.ID" "units"  
## ..- attr(\*, "mcpar")= num [1:3] 3001 19991 10  
## $ CP : NULL  
## $ Liab : NULL  
## $ Fixed :List of 3  
## ..$ formula:Class 'formula' language Sigmoid ~ Pop \* Stage  
## .. .. ..- attr(\*, ".Environment")=<environment: R\_GlobalEnv>   
## ..$ nfl : int 8  
## ..$ nll : num 0  
## $ Random :List of 5  
## ..$ formula:Class 'formula' language ~idv(1):famil + idv(1):ID  
## .. .. ..- attr(\*, ".Environment")=<environment: R\_GlobalEnv>   
## ..$ nfl : num [1:2] 1 1  
## ..$ nrl : int [1:2] 20 216  
## ..$ nat : num [1:2] 0 0  
## ..$ nrt : int [1:2] 1 1  
## $ Residual :List of 6  
## ..$ formula :Class 'formula' language ~units  
## .. .. ..- attr(\*, ".Environment")=<environment: 0x0000000029418398>   
## ..$ nfl : num 1  
## ..$ nrl : int 398  
## ..$ nrt : int 1  
## ..$ family : chr "gaussian"  
## ..$ original.family: chr "gaussian"  
## $ Deviance : 'mcmc' num [1:1700] 1466 1472 1492 1503 1494 ...  
## ..- attr(\*, "mcpar")= num [1:3] 3001 19991 10  
## $ DIC : num 1518  
## $ X :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots  
## .. ..@ i : int [1:1157] 0 1 2 3 4 5 6 7 8 9 ...  
## .. ..@ p : int [1:9] 0 398 622 742 772 970 1082 1142 1157  
## .. ..@ Dim : int [1:2] 398 8  
## .. ..@ Dimnames:List of 2  
## .. .. ..$ : chr [1:398] "1.1" "2.1" "3.1" "4.1" ...  
## .. .. ..$ : chr [1:8] "(Intercept)" "PopCQ" "PopQC" "PopQH" ...  
## .. ..@ x : num [1:1157] 1 1 1 1 1 1 1 1 1 1 ...  
## .. ..@ factors : list()  
## $ Z :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots  
## .. ..@ i : int [1:796] 0 10 11 14 15 35 36 53 54 61 ...  
## .. ..@ p : int [1:237] 0 68 137 148 160 168 183 204 210 222 ...  
## .. ..@ Dim : int [1:2] 398 236  
## .. ..@ Dimnames:List of 2  
## .. .. ..$ : NULL  
## .. .. ..$ : chr [1:236] "(Intercept).famil.401" "(Intercept).famil.404" "(Intercept).famil.405" "(Intercept).famil.409" ...  
## .. ..@ x : num [1:796] 1 1 1 1 1 1 1 1 1 1 ...  
## .. ..@ factors : list()  
## $ ZR :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots  
## .. ..@ i : int [1:398] 0 1 2 3 4 5 6 7 8 9 ...  
## .. ..@ p : int [1:399] 0 1 2 3 4 5 6 7 8 9 ...  
## .. ..@ Dim : int [1:2] 398 398  
## .. ..@ Dimnames:List of 2  
## .. .. ..$ : NULL  
## .. .. ..$ : chr [1:398] "units.1" "units.2" "units.3" "units.4" ...  
## .. ..@ x : num [1:398] 1 1 1 1 1 1 1 1 1 1 ...  
## .. ..@ factors : list()  
## $ XL : NULL  
## $ ginverse : NULL  
## $ error.term : int [1:398] 1 1 1 1 1 1 1 1 1 1 ...  
## $ family : chr [1:398] "gaussian" "gaussian" "gaussian" "gaussian" ...  
## $ Tune : num [1, 1] 1  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : chr "1"  
## .. ..$ : chr "1"  
## $ meta : logi FALSE  
## $ y.additional: num [1:398, 1:2] 0 0 0 0 0 0 0 0 0 0 ...  
## - attr(\*, "class")= chr "MCMCglmm"

summary(fit\_poisson\_Sig)

##   
## Iterations = 3001:19991  
## Thinning interval = 10  
## Sample size = 1700   
##   
## DIC: 546.1294   
##   
## G-structure: ~idv(1):famil  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.famil 1.373 0.006223 3.547 508.3  
##   
## ~idv(1):ID  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.ID 0.8269 0.0009544 2.488 47.27  
##   
## R-structure: ~units  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## units 3.197 1.298 4.943 144.2  
##   
## Location effects: Sigmoid ~ Pop \* Stage   
##   
## post.mean l-95% CI u-95% CI eff.samp pMCMC   
## (Intercept) -110.93 -153.06 -41.79 3.690 <6e-04 \*\*\*  
## PopCQ 107.97 39.28 150.40 3.733 <6e-04 \*\*\*  
## PopQC 108.09 39.02 150.00 3.704 <6e-04 \*\*\*  
## PopQH 108.74 40.78 151.48 3.813 <6e-04 \*\*\*  
## StageBefore 97.01 35.81 133.99 5.139 <6e-04 \*\*\*  
## PopCQ:StageBefore -96.29 -133.82 -35.54 5.252 <6e-04 \*\*\*  
## PopQC:StageBefore -96.45 -132.41 -34.04 5.265 <6e-04 \*\*\*  
## PopQH:StageBefore -95.44 -132.46 -34.28 5.342 <6e-04 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(fit\_poisson\_Cop)

##   
## Iterations = 3001:19991  
## Thinning interval = 10  
## Sample size = 1700   
##   
## DIC: 796.4752   
##   
## G-structure: ~idv(1):famil  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.famil 0.29 1.186e-16 1.498 42.48  
##   
## ~idv(1):ID  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.ID 4.161 2.297 6.55 196.8  
##   
## R-structure: ~units  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## units 1.884 0.971 2.979 302.8  
##   
## Location effects: Forced\_Cop\_Attempt ~ Pop \* Stage   
##   
## post.mean l-95% CI u-95% CI eff.samp pMCMC   
## (Intercept) -3.5019 -5.7673 -1.1282 528.148 <6e-04 \*\*\*  
## PopCQ 1.5539 -0.7815 3.8465 623.195 0.1976   
## PopQC 2.0661 -0.4404 4.3774 669.785 0.0941 .   
## PopQH 1.4837 -1.4697 4.2927 764.876 0.3106   
## StageBefore -33.9066 -93.1631 -2.9337 2.137 <6e-04 \*\*\*  
## PopCQ:StageBefore 34.3893 3.4209 93.4006 2.169 <6e-04 \*\*\*  
## PopQC:StageBefore 33.7165 1.7160 92.1045 2.172 <6e-04 \*\*\*  
## PopQH:StageBefore 34.8785 3.6649 94.2011 2.121 <6e-04 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(fit\_poisson\_Swing)

##   
## Iterations = 3001:19991  
## Thinning interval = 10  
## Sample size = 1700   
##   
## DIC: 2005.628   
##   
## G-structure: ~idv(1):famil  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.famil 0.3398 4.177e-16 0.9225 35.58  
##   
## ~idv(1):ID  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## 1.ID 0.6388 0.3364 0.9941 51.73  
##   
## R-structure: ~units  
##   
## post.mean l-95% CI u-95% CI eff.samp  
## units 0.6325 0.4329 0.8562 837.8  
##   
## Location effects: G\_Swing ~ Pop \* Stage   
##   
## post.mean l-95% CI u-95% CI eff.samp pMCMC   
## (Intercept) 0.51988 -0.48011 1.36040 1567.3 0.26588   
## PopCQ 1.10187 0.19599 2.04789 1500.3 0.03059 \*   
## PopQC 1.35317 0.46085 2.27022 1700.0 0.00588 \*\*  
## PopQH 1.22204 0.09248 2.39091 582.9 0.03294 \*   
## StageBefore -0.83234 -1.72429 0.19409 1473.6 0.08588 .   
## PopCQ:StageBefore 0.66447 -0.26026 1.68638 1485.5 0.19412   
## PopQC:StageBefore 0.50610 -0.51789 1.47263 1489.2 0.31882   
## PopQH:StageBefore 0.68974 -0.50605 1.90308 1509.7 0.24824   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(fit\_glmer)

## Generalized linear mixed model fit by maximum likelihood (Laplace  
## Approximation) [glmerMod]  
## Family: poisson ( log )  
## Formula: Sigmoid ~ Pop \* Stage + (1 | famil/ID)  
## Data: CourtMCMC  
##   
## AIC BIC logLik deviance df.resid   
## 774.4 814.3 -377.2 754.4 388   
##   
## Scaled residuals:   
## Min 1Q Median 3Q Max   
## -2.2393 -0.3828 -0.2891 0.0000 3.6287   
##   
## Random effects:  
## Groups Name Variance Std.Dev.  
## ID:famil (Intercept) 2.7550 1.6598   
## famil (Intercept) 0.5205 0.7215   
## Number of obs: 398, groups: ID:famil, 216; famil, 20  
##   
## Fixed effects:  
## Estimate Std. Error z value Pr(>|z|)  
## (Intercept) -2.072e+01 7.622e+03 -0.003 0.998  
## PopCQ 1.845e+01 7.622e+03 0.002 0.998  
## PopQC 1.841e+01 7.622e+03 0.002 0.998  
## PopQH 1.894e+01 7.622e+03 0.002 0.998  
## StageBefore -1.982e-02 1.131e+04 0.000 1.000  
## PopCQ:StageBefore 5.725e-01 1.131e+04 0.000 1.000  
## PopQC:StageBefore 5.808e-01 1.131e+04 0.000 1.000  
## PopQH:StageBefore 1.450e+00 1.131e+04 0.000 1.000  
##   
## Correlation of Fixed Effects:  
## (Intr) PopCQ PopQC PopQH StgBfr PCQ:SB PQC:SB  
## PopCQ -1.000   
## PopQC -1.000 1.000   
## PopQH -1.000 1.000 1.000   
## StageBefore -0.674 0.674 0.674 0.674   
## PpCQ:StgBfr 0.674 -0.674 -0.674 -0.674 -1.000   
## PpQC:StgBfr 0.674 -0.674 -0.674 -0.674 -1.000 1.000   
## PpQH:StgBfr 0.674 -0.674 -0.674 -0.674 -1.000 1.000 1.000  
## convergence code: 0  
## unable to evaluate scaled gradient  
## Model failed to converge: degenerate Hessian with 2 negative eigenvalues

#Note: summary.MCMCglmm(fit\_gaussian\_Swing) returns the same results as the regular summary() function

Trying emmeans to find pairwise comparisons (Doesn’t run yet. Use Stat 511 notes to construct the function)

#emmeans(fit\_gaussian\_Cop)

### GENERATING FIGURES

Function for getting summary, created by Laura Stein. Gives count, mean, standard deviation, standard error of the mean, and confidence interval (default 95%). Arguments: data: a data frame. measurevar: the name of a column that contains the variable to be summariezed groupvars: a vector containing names of columns that contain grouping variables na.rm: a boolean that indicates whether to ignore NA’s conf.interval: the percent range of the confidence interval (default is 95%)

summarySE <- function(data=NULL, measurevar, groupvars=NULL, na.rm=FALSE,  
 conf.interval=.95, .drop=TRUE) {  
 library(plyr)  
   
 # New version of length which can handle NA's: if na.rm==T, don't count them  
 length2 <- function (x, na.rm=FALSE) {  
 if (na.rm) sum(!is.na(x))  
 else length(x)  
 }  
   
 # This does the summary. For each group's data frame, return a vector with  
 # N, mean, and sd  
 datac <- ddply(data, groupvars, .drop=.drop,  
 .fun = function(xx, col) {  
 c(N = length2(xx[[col]], na.rm=na.rm),  
 mean = mean (xx[[col]], na.rm=na.rm),  
 sd = sd (xx[[col]], na.rm=na.rm)  
 )  
 },  
 measurevar  
 )  
   
 # Rename the "mean" column   
 datac <- rename(datac, c("mean" = measurevar))  
   
 datac$se <- datac$sd / sqrt(datac$N) # Calculate standard error of the mean  
   
 # Confidence interval multiplier for standard error  
 # Calculate t-statistic for confidence interval:   
 # e.g., if conf.interval is .95, use .975 (above/below), and use df=N-1  
 ciMult <- qt(conf.interval/2 + .5, datac$N-1)  
 datac$ci <- datac$se \* ciMult  
   
 return(datac)  
}

Changing the position of a geom layer in ggplot, using the “dodge” function

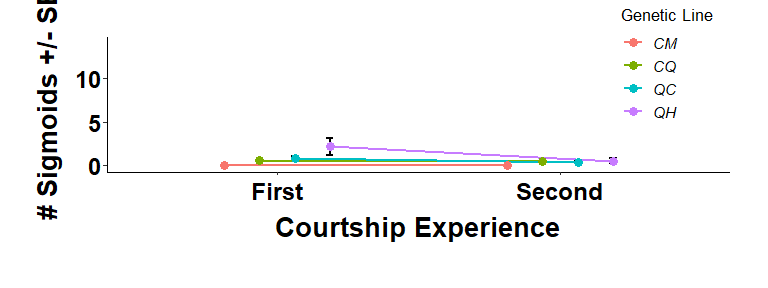
pd <- position\_dodge(0.5) # move them 0.05 to the left and right

Line graph of Sigmoids vs Experience by Population, with standard error bars

#Creating summary statistics table to use as basis for graph  
dSig <- summarySE(Court, measurevar="Sigmoid",  
 groupvars=c("Pop","Stage"))  
  
dSig

## Pop Stage N Sigmoid sd se ci  
## 1 CM After 13 0.0000000 0.000000 0.0000000 0.0000000  
## 2 CM Before 11 0.0000000 0.000000 0.0000000 0.0000000  
## 3 CQ After 112 0.4821429 1.407715 0.1330165 0.2635811  
## 4 CQ Before 112 0.6339286 1.494709 0.1412367 0.2798699  
## 5 QC After 60 0.4333333 1.332344 0.1720049 0.3441810  
## 6 QC Before 60 0.8000000 2.031948 0.2623234 0.5249079  
## 7 QH After 15 0.5333333 1.125463 0.2905933 0.6232606  
## 8 QH Before 15 2.2000000 3.802255 0.9817380 2.1056186

#Re-ordering levels before and after  
dSig$Stage <- factor(dSig$Stage, levels = c("Before", "After"))  
  
#Creating graph as object  
graph.Sig <- ggplot(dSig, aes(x=Stage, y=Sigmoid, group=Pop, col=Pop)) +  
 geom\_errorbar(aes(ymin=Sigmoid-se, ymax=Sigmoid+se), position = pd, width=0.1, colour = "black", size=1) +   
 geom\_line(size=1, position = pd) +  
 geom\_point(position = pd, size=3) +  
 scale\_fill\_brewer(palette="Set1") +  
 labs(x="Courtship Experience", y="# Sigmoids +/- SE") +  
 scale\_x\_discrete(breaks=c("Before","After"),  
 labels=c("First","Second")) +  
 guides(color=guide\_legend("Genetic Line")) +  
 theme\_classic() +  
 expand\_limits(y=c(0,14)) +  
 theme(axis.title.x = element\_text(face = "bold", colour = "black", size = 20, vjust = -1),  
 axis.text.x = element\_text(vjust=0.5, size=18, face = "bold", colour = "black"),  
 axis.title.y = element\_text(face = "bold", colour = "black", size = 20, vjust = 3),  
 axis.text.y = element\_text(vjust = 0.5, size = 18, face = "bold", colour = "black"),  
 legend.title = element\_text(size = 13),  
 legend.text = element\_text(size = 11, face = "italic"),  
 legend.position=c(0.9, 0.8),  
 plot.margin=unit(c(1,1,1.5,1.2),"cm"))  
  
graph.Sig

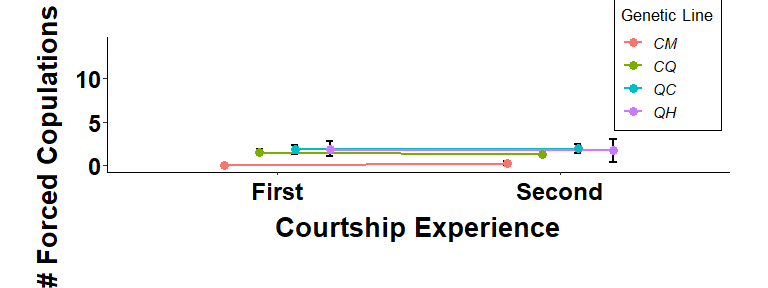


Doing the same thing for forced copulations

#Creating summary statistics table to use as basis for graph  
dForced <- summarySE(Court, measurevar="Forced\_Cop\_Attempt",  
 groupvars=c("Pop","Stage"))  
# Re-ordering levels Before and After  
dForced$Stage <- factor(dForced$Stage, levels = c("Before", "After"))  
  
# Checking that reordering worked  
head(dForced)

## Pop Stage N Forced\_Cop\_Attempt sd se ci  
## 1 CM After 13 0.3076923 0.8548504 0.2370928 0.5165809  
## 2 CM Before 11 0.0000000 0.0000000 0.0000000 0.0000000  
## 3 CQ After 112 1.2946429 3.0362328 0.2868970 0.5685056  
## 4 CQ Before 112 1.5446429 3.2156386 0.3038493 0.6020976  
## 5 QC After 60 1.9833333 4.1230714 0.5322862 1.0651023  
## 6 QC Before 60 1.8333333 4.2072039 0.5431477 1.0868360

#Creating graph as object  
graph.Forced <- ggplot(dForced, aes(x=Stage, y=Forced\_Cop\_Attempt, group=Pop, col=Pop)) +  
 geom\_errorbar(aes(ymin=Forced\_Cop\_Attempt-se, ymax=Forced\_Cop\_Attempt+se), position = pd, width=0.1, colour="black", size=1) +  
 geom\_line(size=1, position = pd) +   
 geom\_point(position = pd, size=3) +  
 scale\_fill\_brewer(palette="Set1") +  
 labs(x="Courtship Experience", y="# Forced Copulations +/- SE") +  
 guides(color=guide\_legend("Genetic Line")) +  
 scale\_x\_discrete(breaks=c("Before","After"),  
 labels=c("First","Second")) +  
 expand\_limits(y=c(0,14)) +  
 theme\_classic() +  
 theme(axis.title.x = element\_text(face = "bold", colour = "black", size = 20, vjust = -1),  
 axis.text.x = element\_text(vjust=0.5, size=18, face = "bold", colour = "black"),  
 axis.title.y = element\_text(face = "bold", colour = "black", size = 20, vjust = 3),  
 axis.text.y = element\_text(vjust = 0.5, size = 18, face = "bold", colour = "black"),  
 legend.title = element\_text(size = 13),  
 legend.text = element\_text(size = 11, face = "italic"),  
 legend.position=c(0.9, 0.8),  
 legend.background = element\_rect(size=0.5, color=1),  
 plot.margin=unit(c(1,1,1.5,1.2),"cm"))  
  
graph.Forced



Doing the same for gonopodium swings

#Creating summary statistics table to use as basis for graph  
dSwing <- summarySE(Court, measurevar="G\_Swing",  
 groupvars=c("Pop","Stage"))  
  
#Re-ordering levels before and after  
dSwing$Stage <- factor(dSwing$Stage, levels = c("Before", "After"))  
  
  
head(dSwing)

## Pop Stage N G\_Swing sd se ci  
## 1 CM After 13 3.307692 4.571371 1.2678703 2.762452  
## 2 CM Before 11 1.909091 3.176619 0.9577867 2.134082  
## 3 CQ After 112 8.285714 7.829752 0.7398420 1.466046  
## 4 CQ Before 112 7.589286 7.748936 0.7322056 1.450914  
## 5 QC After 60 10.100000 7.830189 1.0108731 2.022752  
## 6 QC Before 60 8.200000 8.676483 1.1201291 2.241373

graph.Swing <- ggplot(dSwing, aes(x=Stage, y=G\_Swing, group=Pop, col=Pop)) +  
 geom\_errorbar(aes(ymin=G\_Swing-se, ymax=G\_Swing+se), position = pd, width=0.1, colour="black", size=1) +  
 geom\_line(size=1, position = pd) +   
 geom\_point(position = pd, size=3) +  
 scale\_fill\_brewer(palette="Set1") +  
 labs(x="Courtship Experience", y="# Gonopodium Swings +/- SE") +  
 guides(color=guide\_legend("Genetic Line")) +  
 theme\_classic() +  
 scale\_x\_discrete(breaks=c("Before","After"),  
 labels=c("First","Second")) +  
 expand\_limits(y=c(0,14)) +  
 theme(axis.title.x = element\_text(face = "bold", colour = "black", size = 20, vjust = -1),  
 axis.text.x = element\_text(vjust=0.5, size=18, face = "bold", colour = "black"),  
 axis.title.y = element\_text(face = "bold", colour = "black", size = 20, vjust = 3),  
 axis.text.y = element\_text(vjust = 0.5, size = 18, face = "bold", colour = "black"),  
 legend.title = element\_text(size = 13),  
 legend.text = element\_text(size = 11, face = "italic"),  
 legend.position=c(0.15, 0.8),  
 legend.background = element\_rect(size=0.5, color=1),  
 plot.margin=unit(c(1,1,1.5,1.2),"cm"))  
  
graph.Swing

