```
################  STEP 1, processing the raw data and gettting summaries of the structure, correlation in the data
############     determining what periods have high fecundities (bonanazas), and the frequency of these events
#######          While this first step of initial data processing the theory could be looped through all populations, at the moment
######          There are some populations that have population specific steps that limit the ability to simply loop through the entire code.
#####           After step 1 the code is entirely automated.
#########################################################################################################################################

# Clean up the memory in R
  rm(list=ls(all=T))
# Load in the database
load("D:/Dropbox/ASD/Analyses/Processed_DB.RData")



## Bring in the packages I will need
library(fields)
library(scatterplot3d)
library(rgl)
library(nlme)
library(grid)
library(fBasics)
library(stats4)
library(MASS)
library(matrixcalc)
library(abind)
library(matlab)
library(PVA) # note that this is a series of functions needed to develop a PVA

##### Convert  to matrices...

############################################################## STOCK DEPENDENT PROCESSING     ##########################################################################################
######
#########                                    This section gets each population into a standardized form for use in PVA

############################################################## STOCK DEPENDENT PROCESSING     ##########################################################################################


# There are the populations I am looking at.

nm <- c("COD3NO","COD3Pn4RS","COD4TVn","COD4X","CODBA2224","CODBA2532","CODCOASTNOR","CODCS","CODFAPL","CODGB-GARM","CODGOM","CODICE","CODIS",
       "CODNEAR","CODNS","CODVIa","HERR4TFA","HERR4TSP","HERR30","HERR2224IIIa","HERR2532","HERRCS","HERRNS","HERRRIGA")

# Here we loop through each population we have data for.
p=1
for(p in 1:length(nm))
{
  # pull out the natural mortalities
  Nat_mort <- NM[stocks == nm[p],]
  # Clean up the data
  rm <- which(colSums(Nat_mort,na.rm=T) ==0)
  rmr <- which(rowSums(Nat_mort,na.rm=T) ==0)
  ifelse(length(rmr) > 0,Nat_mort <- Nat_mort[-rmr,-rm],Nat_mort <- Nat_mort[,-rm])

  # This is the Fishing mortality given in the stock assesment, I calculate this, but it isn't used in the PVA itself, the
  # raw survivals (directly from the changes in abundance in the stock asssessment) are what is used.
  CFM <- FM[stocks == "nm[p]",]
  rm <- which(colSums(CFM,na.rm=T) ==0)
  rmr <- which(rowSums(CFM,na.rm=T) ==0)
```

```
ifelse(length(rmr) >0, CFM <- CFM[-rmr,-rm],CFM <- CFM[,-rm])

num_years <- length(CFM[,1])
# Warning message if Natural mortality and Fishing mortalities aren't lining up.
ifelse(dim(Nat_mort)[1] == dim(CFM)[1], print("Same number of years"),print("Damn we got a different # of years"))
ifelse(dim(Nat_mort)[2] == dim(CFM)[2], print("and/or same age classes"),print("and/or a different # of age classes"))
if(dim(Nat_mort)[1] != dim(CFM)[1] || dim(Nat_mort)[2] != dim(CFM)[2]) ifelse(length(rmr) >0,Nat_mort <- NM[stocks == "CODBA2224",][-rmr,-rm],Nat_mort <- NM[stocks == "CODBA2224",][,-rm])

# So add in that natural mortality

CFM_Mort <- 1 - exp(-(CFM + Nat_mort))
CFM_Mort

# THis is needed if there is a step change in natural mortality...
#CFM_Mort <-1- exp(-(rbind(CFM[1:nat_change,]+Nat_mort, CFM[(nat_change+1):num_years,]+Nat_mort2)))
#CFM_Mort

# Turn mortality into survival for PVA purposes
CFM_Surv <- 1-CFM_Mort
CFM_Surv

# Get the average survival and the variance for the population
mean_CFM_surv <- colMeans(CFM_Surv,na.rm=T)
var_CFM_surv <- apply(CFM_Surv,2,function(x) var(x,na.rm=T))

# Now bring in the number of fish.
num <- Num[stocks == nm[p],]
num
# And clean it up.
rm <- which(colSums(num,na.rm=T) ==0)
rmr <- which(rowSums(num,na.rm=T) ==0)

ifelse(length(rmr) >0, num <- num[-rmr,-rm], num <- num[,-rm])



# Get the weight at age for each population and clean it up.
Weight <- WA[stocks == nm[p],]
rm <- which(colSums(Weight,na.rm=T)==0)

ifelse(length(rmr) >0, Weight <- Weight[-rmr,-rm], Weight <- Weight[,-rm])


#CFM_Surv <- cbind(matrix(1-exp(-.2),nrow = length(CFM[,1]),ncol=2),CFM_Surv[,-5])
#Weight <- cbind(zeros(length(CFM[,1]),1),Weight[,c(-6,-7)])


# Initial data, the below loop is used to determine the mortality calculated from
# the actual numbers of individuals (includes both fishing and natural mortality)

data_years <- length(Weight[,1])
num_classes <-length(Weight[1,])

#plot(rowSums(num),type="l")
classy <- array(0,dim=c(length(num[,1])-1,4,length(num[1,])))
dim(classy)
```

```
# These loops iare just organizing the matrix for use in the next loop.
for (j in 1:(length(num[1,])-1))
{
 for(i in 1:(length(num[,1])-1))
 {
  classy[i,,j] <- cbind(j,i,num[i,j],num[i+1,j+1])
 }
}
dim(classy)

# Fill in the final age class with the stuff in the second last, i_e_ assume the
# same vital rates for the oldest individuals, which seems a close approximation to reality___
classy[,,length(classy[1,1,])] <- classy[,,(length(classy[1,1,])-1)]

rates = classy    # rates is the data set used below (you
# could input several different data sets above here and
# this allows a choice of which to use)_
classes = length(classy[1,1,]) # how many different rates or classes for the
# rates calculations are there?******
# What are the total number of classes
total_classes <- length(num[1,])

# makes all the sets of variance values to search over



use <- 0
mean_surv_calc <- 0
var_surv_calc <- 0
var_header <- 0
raw_ratio <- zeros(length(rates[,1,1]),classes)

########################################  PVA survival calculations ##################################################################
### This loop calculates the raw estimate of survival from the abundance data as opposed to using the fishing mortality above.
# These are the survival data that are used in the model.  This is the survival data used in the PVA.
########################################  PVA survival calculations ##################################################################

for (class in 1:classes) # loop through each rate or class
{
 #   new_log_lik <- 0
 #   find the min and max rows of the data matrices to use
 #  min_row = (class-1)*times +1
 #   max_row =  (class)*times
 use = rates[,,class] # fetch the data to use
 mean_surv_calc[class] = mean(use[,4]/use[,3],na.rm=T)  # raw estimate of mean
 raw_ratio[,class] <- use[,4]/use[,3] # This gets the raw data which might be interesting to look at___
 var_surv_calc[class] = var(use[,4]/use[,3],na.rm=T)  # raw est_ of variance
}



# If we have CFM data but not raw ratio (or vice versa) we can combine both.
ifelse(na.omit(length(CFM_Surv[1,])) == length(raw_ratio[1,]) && length(CFM_Surv[,1]) == (length(raw_ratio[,1])+1),
      raw_ratio <- rbind(raw_ratio,as.numeric(CFM_Surv[length(CFM_Surv[,1]),])),raw_ratio <- rbind(raw_ratio,raw_ratio[length(raw_ratio[,1]),]))

#apply(raw_ratio,2,function(x) mean(x,na.rm=T))
#apply(raw_ratio,2,function(x) mean(x,na.rm=T))
#apply(CFM_Surv,2,mean)
#mean_surv_calc <- c(mean_surv_calc[1:11],0_39,0_39)
#var_surv_calc<- c(var_surv_calc[1:11],0_15,0_15)
# Same idea as above
```

```
### Now set up each survival calc so it has the right header and I don't get confused later!!!

# First get the appropriate header!!
# I will need to change the i's below on a case by case basis, depending on what age the first year
# class I will use this the matrix is gonna be_
# i = first age class is 1, i+1 = first age class is 2, i+2 = first age class is 3, etc___)

CFM_surv_header <- 0
mean_CFM_surv_header <- 0
var_CFM_surv_header <- 0


var_surv_calcs_header <- 0
mean_surv_calcs_header <- 0
Calcs_surv_header <- 0

for (i in 1:total_classes)
{
 CFM_surv_header[i] <- paste("surv_CFM",(i),sep="_")
 mean_CFM_surv_header[i] <- paste("mean_surv_CFM",(i),sep="_")
 var_CFM_surv_header[i] <- paste("var_surv_CFM",(i),sep="_")

 if(i <= classes)
 {

   var_surv_calcs_header[i] <- paste("var_surv_calcs",(i),sep="_")
   mean_surv_calcs_header[i] <- paste("mean_surv_calcs",(i),sep="_")
   Calcs_surv_header[i] <- paste("surv_calcs",(i),sep="_")
 } # end if loop
} # end for loop

# Set them up as lists so I can combine them with the data___
CFM_surv_header <- list(CFM_surv_header)
mean_CFM_surv_header <- list(mean_CFM_surv_header[1:length(mean_CFM_surv)])
var_CFM_surv_header <- list(var_CFM_surv_header[1:length(mean_CFM_surv)])

var_surv_calcs_header <- list(var_surv_calcs_header[1:length(mean_surv_calc)])
mean_surv_calcs_header <-list(mean_surv_calcs_header[1:length(mean_surv_calc)])
Calcs_surv_header <-    list(Calcs_surv_header[1:length(mean_surv_calc)])


# Now combine each of the survival estimates and their headers___
CFM_Surv <- as.matrix(CFM_Surv)

# Again first using the survivorship based on the Calculated Fishing mortality
raw_surv_CFM <- t(matrix(t(matrix(CFM_Surv,ncol=total_classes,byrow=F)),nrow=total_classes,dimnames=CFM_surv_header))
mean_surv_CFM <- t(matrix(mean_CFM_surv,dimnames = mean_CFM_surv_header))
var_surv_CFM <- t(matrix(var_CFM_surv,dimnames = var_CFM_surv_header))


# Now using the calculated data, NOTE THAT I COULD USE THE FIRST Age-class results OF THIS DATA TO SUPPLIMENT THE non-varialbe CFM data for age 1 fish___

var_surv_calcs <- t(matrix(var_surv_calc,dimnames=var_surv_calcs_header))
mean_surv_calcs <- t(matrix(mean_surv_calc,dimnames=mean_surv_calcs_header))

# NOTE THAT THIS ONLY HAS data_years-1 entries
raw_surv_calcs <-  t(matrix(t(matrix(raw_ratio,ncol=classes,byrow=F)),nrow=classes,dimnames=Calcs_surv_header))
```

```
paste(message("Here are the mean survival rates: Column 1)  calculated from the number of survivors, or Column 2) taken from the CFM in the VPA calcs"),
    round(mean_surv_calcs,digits=4),round(mean_surv_CFM,digits=4))
paste(message("Here are the variances of the survival rates: Column 1)  calculated from the number of survivors, or Column 2) taken from the CFM in the VPA calcs"),
    round(var_surv_calcs,digits=4),round(var_surv_CFM,digits=4))

# Given these data for CODIS I will use the CFM data (with natural mortality included) as it is much cleaner data..


######################################### END SURVIVAL END SURVIVAL ##########################################################################33




################################### FECUNDITITES FECUNDITIES FECUNDITIES ##################################################
################################### FECUNDITITES FECUNDITIES FECUNDITIES ##################################################

################################### NOW THE FECUNDITIES ###############################



## Now for the fecundities I need to get the weights at age data___

## Each individual produces some average number of fish
# I_e_ recruits in year 2 divided by NUMBER OF adults in year one_
# But these adults are different size, so to correct for this I need
# to scale their sizes so that large fish have more eggs and
# small fish have fewer using the weights

# Now the number of recruits...
# Again this is variable for each stock!!

recruits <- Rec.df2$Recruits[Rec.df2$Stock_ID == nm[p]] # Extracts all recruits

length(recruits)==length(num[,1])
rm <- as.numeric(attr(na.omit(recruits[1:10]),"na.action"))
recruits <- num[,1] # Extracts all recruits
recruits[rm] <- NA


# O.K., so I need an equation to relate the fecundity to the Biomass, via weight at age___
# From Murawski 2001 in Table 1 shows fecundity is approximately the cube of Length
# and von-Bert says Weight is the cube of length, therefore fucunidity is approximately
# a linear function of weight, makes life easy, going to assume the linear relationship
# is a 1:1, i_e_ a fish weight 10 kg produces 10 times the eggs of a 1 kg fish___
# Now using the weights for each mature age class get the fecundity of each individual
# in an age class, oldest age classes should have much higher fecundities___

# Now we figure out what age the fish start to mature.
rm <- which(colSums(AM[stocks == nm[p],],na.rm=T)==0)
rmr
Age_mature <- max(rm)


# Now figure out what age the fish recruit to fishery
first_class <- unique(Rec.df2$Rec_age[Rec.df2$Stock_ID == nm[p]])
Draw_mature <- Age_mature - first_class +1
```

```
# This gets us only the mature ages, and stops at the same age at the number of individuals, the grep is needed to pick out
# only the num age classes that are mature, only important if we have data for immature individuals (see CODIS)
ifelse(length(rmr) > 0,mat_ogive <- AM[stocks == nm[p],-rm][-rmr,1:length(num[,grep(Age_mature,names(num))[1]:length(num[1,])])],
      mat_ogive <- AM[stocks == nm[p],-rm][,1:length(num[,grep(Age_mature,names(num))[1]:length(num[1,])])])

# Now for some stocks in the first age class there are years that suggests that no individuals in the first mature
# ogive are mature, while in other years there is data.  In these cases I am going to assume there is
# some maturation but it is at a rate of 50% of the minimum in other years, this way we can do the rest of the analysis
# using the log transformed data.
i=1
for(i in 1:length(mat_ogive[1,]))
{
  if(any(mat_ogive[,i] ==0) == T) mat_ogive[mat_ogive[,i] == 0,i] = 0.5*min(mat_ogive[mat_ogive[,i] > 0,i],na.rm=T)
}
mat_ogive <- cbind(zeros(length(mat_ogive[,1]),1),mat_ogive)


#############################################################################################
##############################    HERE I get it right!!!!                    ###############
## Here I adjust fertility by linearly increasing fertility with individual mass ######
####          THIS IS WHERE WE WOULD ADJUST THE FERTILITY IF THERE IS AN AGE EFFECT    ##########
#############################################################################################

# So this is the weight of mature individuals,
#the grep makes sure we start at the right age class only needed if we have immature age classes
# needed when weight and number are different lengths...
#mature_weight <- Weight[1:(data_years-first_class),grep(Age_mature,names(mat_ogive))[1]:length(num[1,])]
# For CODIS
mature_weight <- Weight[1:(data_years-first_class),1:length(num[1,])]

# the grep is needed to pick out
# only the num age classes that are mature, only important if we have data for immature individuals
mature_num <- num[1:(data_years-first_class),1:length(num[1,])]*mat_ogive[1:(data_years-first_class),]
#mature_num <- cbind(0,0,num[[1:(data_years-first_class),],3:13])

# This is how much bigger the oldest fish are comparted to youngest
# This could be done by making reference the smallest fish ever, rather
# then the smallest fish in a given year, but doing it that way you wouldn't be accounting for
# changes in size of the smallest fish since you always divide by that number, shouldn't make much
# difference, and this way makes more sense I believe

ratios <-  mature_weight[,]*mat_ogive[1:(data_years-first_class),] / min(mature_weight[,2],na.rm=T)

# This is the Number of mature individuals in each mature age class times this ratio
# so this tells us how much more important an individual in age class 15 is than the
# same indivudial in age class 5 for example___

# Summing this number gives us a measure of the potential fecundity... basically a combo of
# Mature number and the relative size of the fish.
adj_numbs <- num[1:(data_years-first_class),] * ratios

# This give us the number of recruits in the correct year, and divides it by the
# total value of numbers adjusted for size differences_ So This says
# how many fish are produced per individual unit, basline being the youngest age class
# Thus the youngest age class shouldn't change at all between x and ratios
# The denominator here is the total "Relative Biomass" that produces the recruits found in a given year
# With the assumption that each of these units is equivalent (i_e_ one old individual 16 times larger than
# one young individual will produce 16 times the number of recruits) we get the number of recruits
# produced per "Relative Biomass"
```

```
  # Divide Number of recruits by potential fecundity
  z  <- recruits[(first_class+1):data_years]/apply(adj_numbs[1:(data_years-first_class),],1,sum)

  # Then this assigns how many offspring 1 individual in each age class will produce, is essentially a realized fertility...
  fert  <- as.matrix(z*ratios)


 # Now we can pull out the mean fertility and variance

  mean_fert <- matrix(apply(fert,2,function(x) mean(x,na.rm=T)))
  var_fert <- apply(fert,2,function(x) var(x,na.rm=T))

  # now make some headers so I know what the heck this data is

  var_fert_header <- 0
  mean_fert_header <- 0
  raw_ferts_header <- 0

  for (i in 1:total_classes)
  {
   raw_ferts_header[i] <- paste("raw_ferts",(i+first_class-1),sep="_")
   var_fert_header[i] <- paste("var_fert",(i+first_class-1),sep="_")
   mean_fert_header[i] <- (paste("mean_fert",(i+first_class-1),sep="_"))
  }
  mean_fert_header <- list(mean_fert_header[1:total_classes])
  var_fert_header <- list(var_fert_header[1:total_classes])
  raw_ferts_header <- list(raw_ferts_header[1:total_classes])

  var_fert_calcs <- t(matrix(var_fert,dimnames=var_fert_header))

  mean_fert_calcs <- t(matrix(mean_fert,dimnames=mean_fert_header))

  colnames(fert) <- raw_ferts_header[[1]]
  ferts <- fert

 ######################################################  BONANZAS BONANZAS BONANZAS BONANZAS BONANZAS BONANZAS ##########################################################
 #
 #                                    It's at this point I noticed there was a bi-modal pattern in the fertilities
 #                                    It was either normal are very elevated, to get a good sampling distribution for the data
 #
 ######################################################  BONANZAS BONANZAS BONANZAS BONANZAS BONANZAS BONANZAS ##########################################################


  ### Now produce each annual matrix___
  #survs <- mean_surv_calcs
  survs <- mean_surv_calcs
  fert_classes <- length(ferts[1,])
  surv_classes <- length(survs[1,])
  years_data <- length(ferts[,1])


  mean(ferts[,1])
  sd(ferts[,1])


  fert_outlier <- zeros(years_data,fert_classes)
  fert_normal <- zeros(years_data,fert_classes)
  fert <- 0
```

```
temp <- 0

## I THINK I SHOULD TAKE THE GEOMETRIC MEANS AND THEN BACK TRANSFORM FOR THESE
## THAT MAY GET RID OF THE CRAZY OUTLIERS!!!
## BUT GIVES GRIEF WHEN WE HAVE 0's, see adjustment above!!
log_ferts <- log(ferts)



############################################################PICK YOUR OUTLIERS ##############################################
##      IMPORTANT STEP HERE!!!   IMPORTANT STEP HERE!!! IMPORTANT STEP HERE!!! IMPORTANT STEP HERE!!! IMPORTANT STEP HERE!!!
##     I am picking the "bonanza periods" as points in which the LOGNORMAL fertilities are way out in the tail
##     That is, if the fertility is greater than 2 *sd the mean (on the log scale), then I consider that year to be a bonanza_
############################################################PICK YOUR OUTLIERS ##############################################


outlier_fert <- exp(colMeans(log_ferts) + apply(log_ferts,2,sd))
outlier_fert <- c(0.1,outlier_fert[-1])

for(i in 1:fert_classes)
{

 for (j in 1:years_data)
 {
  temp <- ferts[j,i]
  if(temp >= outlier_fert[i])
  {
   fert_outlier[j,i] <- temp
   fert_normal[j,i] <- -9999
  } else
  {
   fert_normal[j,i] <- temp
   fert_outlier[j,i] <- -9999
  }
 }
}


# Now I want to look at the fertilities for each cohort___

cohort_fert <- zeros(length(ferts[,1]),length(ferts[1,]))

for(i in 1:(length(ferts[1,])))
{
 for(j in i:(length(ferts[,1])))
 {
  cohort_fert[j,i] <- ferts[j,i]

  if (i > 1) {
   cohort_fert[j-i+1,i] <- ferts[j,i]

  }
 }
}

cohort_fert <- cohort_fert[1:(length(ferts[,1])-length(ferts[1,])+1),]

matplot(t(cohort_fert[1:10,]),type='l',log="y")
```

```
legend("topleft",c(as.character(seq(1,10))),lty=c(1:10),col=c(1:10))

## NOW I WILL TAKE IT THAT ANY YEAR WITH AN OUTLIER IN IT IS AN UNUSUAL YEAR, AND
## THE FREQUENCIES, CORRELATION BETWEEN YEARS, AND DISTRIBUTION OF THESE
## OUTLIER YEARS MUST BE ACCOUNTED FOR

## I THINK TO GET AT THE CORRELATION BETWEEN YEARS, I WILL TAKE THE LONGEST RUN
## OF DATA WITHOUT A BONANAZA, FOR GULF COD THIS IS FOR EXAMPLE YEARS 10-35
## SO STILL USING ABOUT 25 YEARS OF DATA_
## I may need to think of a bonanaza in some cases more of a change in state of the
## ecosystem in which the population will grow well for a number of years_
## This has to be based upon what we have seen in the past though!

## I THINK THE BONANAZA'S NEED TO BE 1: DENSITY DEPENDENT, SEE B0 DISCUSSION WITH JEFF (essentially we put a ceiling on the bonanzas)
## 2: THEY SHOULD DEPEND ON WHETHER THERE WAS A BONANAZA PREVIOUSLY, THE DATA CAN
## BE USED TO DETERMINE IF BONANAZAS ARE CORRELATED, FOR THE GULF COD CLEARLY THEY ARE
## WHICH SHOULD MAKE THE STOCK QUIET RESILLENT, WHILE FOR CELTIC SOLE IT APPEARS TO
## BE MORE OF A FLUKE!

#  Here are my assumptions/caviates to this analysis___
#  1:  I am assuming that bonanza's in the current fished context only apply to recruitment events, I will check, but it
#  certainly looks like this is the case, which makes sense, but is interesting when you consider that the
#  sensistivity and elasticitiy analysis suggests that it is changes in survival that should have the largest effect_
#  2:  Ya know I should probably be thinking about the elasticity of total fecundity rather than the elasticitiy of each years class
#      at this stage as I am essentially just integrating the total recruitment, though if I get into the decline in weight
#     something else may pop out___

#  A:  If the bonanza is a unique event, or happens repeatedly at a low frequency (i_e_ every 10ish years) then
#  i:  I remove that year from the main dataset
#  ii:  I calculate the correlations for the longest stretch of data between bonanazas, normal times = longest time series without a large recruitment event
#  iib: I should also calculate the correlations for other shorter time series if applicable and compare them, if really different
#      then I need to think some about what to do (make a model with differing enviros, or just use the longest, or other ideas???)
#  iii:  Use the entire non-bonanaza data to determine the vital rates/matrix elements
##################################
#  B:  If the bonanza appears to be some sort of state change, where recruitment is elevated for some number of years, then___
#  i:  split the dataset into the two periods, my criteria for a change is recruitment periods in which recruitment in 50% of age classes of a given year is > 2 sd
#      larger than the geometric mean recuritment for that age class, over the whole time series
#  ii:  I calculate the correlations for the longest stretch of data without bonanzas, normal times = longest time series without a large recruitment event
#  iii:  Calculate the correlations for the bonanza data_
#####################################




############################################## CORRELATION#############################################################################
##########                Get the correlation structure for the data, both bonanza and non-bonanza
##########
##########
#######
#########################################################################################################################################
#  First, I reclassify all year classes with an "NA" in them, i_e_ any year with a really large recruitment event, and if they appear to be case 2 any year between as well

# To actually "qualify" as a bonanza period I don't want just 1 or 2 high value putting us into a bonanza period
## I want most of the time series to be high, so lets go for if 50% of timeseries is high, we consider it
# a bonanza!  The 50% is rather arbitrary, but seems to do a nice job picking out the years that are outliers


bonz <- apply(fert_normal,1,function(x) length(which(x == -9999)))/length(fert_normal[1,-1])
bonz_years <- which(bonz >= 0.5)
non_bonz <- which(bonz < 0.5)
```

```
count=0
countb = 0
for(i in 1:(data_years-first_class))
{
 if(i ==1 && bonz[i] < 0.5) count[i] = 1
 if(i ==1 && bonz[i] >= 0.5) count[i] = 0
 if(i > 1 && bonz[i] < 0.5) count[i] = count[i-1] +1
 if(i > 1 && bonz[i] >= 0.5) count[i] = 0

 if(i ==1 && bonz[i] < 0.5) countb[i] = 0
 if(i ==1 && bonz[i] >= 0.5) countb[i] = 1
 if(i > 1 && bonz[i] < 0.5) countb[i] = 0
 if(i > 1 && bonz[i] >= 0.5) countb[i] = countb[i-1] +1
}

long_non_bonz <- which(count == max(count))[1]
len_non_bonz <- max(count)
non_bonz_cor <- seq((long_non_bonz - len_non_bonz + 1),long_non_bonz)

long_bonz <- which(countb == max(countb))[1]
len_bonz <- max(countb)

if(max(countb) > 1) bonz_cor <- seq((long_bonz - len_bonz + 1),long_bonz)
# Now if there is no multi-year bonanzas in the data I am going to take the first
# Two bonanza's and use those as the data for a multi-year bonanza, not perfect
# But I believe it is better than having perfect correlation when the model gives a 2 year bonanza
# which it will from time to time...
if(max(countb) == 1) bonz_cor <- long_bonz <- which(countb == max(countb))[1:2]



# The longest stretch of continuous data without a bonanza is used to get the correlation matrix
cor_fert <- ferts[non_bonz_cor,]
# For the raw data I still want to use all the data we have, so I will use a different length of data to caluclate these...
raw_fert <- ferts[non_bonz,]


# And this extracts the survial and raw data for the same years as used for the fertilitity calcs_
# For the raw data I still want to use all the data we have, so I will use a different length of data to caluclate these...
cor_surv <-  raw_surv_calcs[non_bonz_cor,]
raw_surv <-  raw_surv_calcs[non_bonz,]
#cor_surv <-  raw_surv_CFM[non_bonz_cor,]
#raw_surv <-  raw_surv_CFM[non_bonz,]


#### NOW THE BONANZA DATA

cor_fert_bonanza <- ferts[bonz_cor,]

raw_fert_bonanza <-  ferts[bonz_years,]
#raw_surv_bonanza <- raw_surv_CFM[bonz_years,]
#cor_surv_bonanza <- raw_surv_CFM[bonz_cor,]
raw_surv_bonanza <- raw_surv_calcs[bonz_years,]
cor_surv_bonanza <- raw_surv_calcs[bonz_cor,]
### NO LONGER IMPLEMENTED IDEA, BUT I COULD RESURECT IT IF I WANT ########################################
# If I want to see if there is any correlation in the bonanza I should look at the data for the year before and year after a
# bonanza happens to see if this gives me any indication that a bonanza is coming, or how strongly the temporal correlation is for a bonanza, could start
# by picking the year before and after bonanza to look for correlations, if a multi-year bonanza occurs this could yield a decent time series, at least in PVA terms___
##########################################################################################################
```

```
  # Some final processing!
  # I WILL USE THE raw data to calculate the means and variances, i_e_ the data that was defined using the median + 2 * sd calculations___
  ### NOW THE VITAL RATES THEMSELVES, to get the mean and variance!
  ## MAKE NOTE!!!!  Because of the skew in the fertilities, I will use the geometric mean to caluculate the means!!
  # First the survivals___
  # I want to calculate the geometric means for the fecundities!!

  gm <- function(x) exp(mean(log(x)))
  gvar <- function(x) exp(var(log(x))) # Is the the variance then???

  mean_surv <- apply(raw_surv,2,mean)
  var_surv <- apply(raw_surv,2,var)
  mean_surv_bonanza <- apply(raw_surv_bonanza,2,mean)
  var_surv_bonanza <- apply(raw_surv_bonanza,2,var)

  # Now the fecundities___

  mean_fert <- apply(raw_fert,2,gm)
  var_fert <- apply(raw_fert,2,var)
  mean_fert_bonanza <- apply(raw_fert_bonanza,2,gm)
  var_fert_bonanza <- apply(raw_fert_bonanza,2,var)
  ######################################################  END END CORRELATION CORRELATION ######################################


  ## NOW OUTPUT THE DATA FOR USE IN STEPS 2-5___


  # Now output this
  save.image(paste("d:/Dropbox/My_Papers/PVA/Results/Step_1_",nm[p],".RData",sep=""))
}



######################################################  Here is the main part of the PVA code  ###########################################################
######################################################  This one performs the sensistivity analysis for the bonanza's  ##################################
######################################################  With a couple simple changes shown in the code you can do the same for non-bonanza years  #####

# Clear the memory
rm(list=ls(all=T))


library(fields)
library(scatterplot3d)
library(rgl)
library(nlme)
library(grid)
library(fBasics)
library(stats4)
library(MASS)
library(matrixcalc)
library(abind)
library(matlab)
library(PVA) #  A library of useful PVA code based on Morris and Doak, converted from their Matlab into R

#############################################################################################################################
```

```
# Here are the names of the stocks that we are using in the analysis
#################################################################################

nm <- c("COD3NO","COD3Pn4RS","COD4TVn","COD4X","CODBA2224","CODBA2532","CODCOASTNOR","CODCS","CODFAPL","CODGB-GARM","CODGOM","CODICE","CODIS",
      "CODNEAR","CODNS","CODVIa","HERR4TFA","HERR4TSP","HERR30","HERR2224IIIa","HERR2532","HERRCS","HERRNS","HERRRIGA")


#######################################################
# This runs the model for each population
#######################################################

for(g in 1:length(nm))
{
   load(paste("d:/Dropbox/My_Papers/PVA/Results/Step_1_",nm[g],".RData",sep=""))

# THe cod stocks!


library(matlab)
library(PVA)


#########################################################################################################################################################
# OK, so we need to rejig all of these when we have no variation in some of the age classes, I had done this earlier
# but this automates the process for all of the populations

# I can't have constant rates in the correlation, these include year classes with constant natural mortality (i.e. N = 0.2), and FM =0
# or when there are no adults in an age class, what I will do is put a tiny tiny amount of variation in these rates so we can use them
# but they don't influence the models.  These are age classes that we aren't interested in either.
#########################################################################################################################################################

if(any(apply(raw_surv_bonanza,2,var) == 0))
{
   for(i in c(which(apply(raw_surv_bonanza,2,var) == 0)))
   {
      len <- length(raw_surv_bonanza[,i])
      mn <- mean(raw_surv_bonanza[,i])
      sd <- mn*0.01
      raw_surv_bonanza[,i] <- rnorm(len,mn,sd)
   }
}

if(any(apply(raw_surv,2,var) == 0))
{
   for(i in c(which(apply(raw_surv,2,var) == 0)))
   {
      len <- length(raw_surv[,i])
      mn <- mean(raw_surv[,i])
      sd <- mn*0.01
      raw_surv[,i] <- rnorm(len,mn,sd)
   }
}

if(any(apply(cor_surv_bonanza,2,var) == 0))
{
   for(i in c(which(apply(cor_surv_bonanza,2,var) == 0)))
   {
      len <- length(cor_surv_bonanza[,i])
      mn <- mean(cor_surv_bonanza[,i])
      sd <- mn*0.01
```

```
      cor_surv_bonanza[,i] <- rnorm(len,mn,sd)
    }
 }

 if(any(apply(cor_surv,2,var) == 0))
 {
    for(i in c(which(apply(cor_surv,2,var) == 0)))
    {
       len <- length(cor_surv[,i])
       mn <- mean(cor_surv[,i])
       sd <- mn*0.01
       cor_surv[,i] <- rnorm(len,mn,sd)
    }
 }

 # Here we just organize the data for later use.
 data_Surv <-  cor_surv
 data_Surv_bonanza <- cor_surv_bonanza
 mean_surv <- apply(raw_surv,2,mean)
 var_surv <- apply(raw_surv,2,var)
 mean_surv_bonanza <- apply(raw_surv_bonanza,2,mean)
 var_surv_bonanza <- apply(raw_surv_bonanza,2,var)

 fert_max <- apply(raw_fert,2,max)
 fert_max_bon <- apply(raw_fert_bonanza,2,max)


 ##########################################################################################################################################
 #  Here we do the same with the fertilites
 #  Now we also need to see how the fertilities are correlated to other vital rates, so we grab the fertilities
 #
 #
 ##########################################################################################################################################

 if(any(apply(raw_fert_bonanza,2,var) == 0))
 {
    mny <- min(raw_fert_bonanza[raw_fert_bonanza >0])
    for(i in c(which(apply(raw_fert_bonanza,2,var) == 0)))
    {
       len <- length(raw_fert_bonanza[,i])
       mn <- mny*i*0.001
       sd <- mn*0.01
       raw_fert_bonanza[,i] <- rnorm(len,mn,sd)
    }
 }

 if(any(apply(raw_fert,2,var) == 0))
 {
    mny <- min(raw_fert[raw_fert >0])

    for(i in c(which(apply(raw_fert,2,var) == 0)))
    {
       len <- length(raw_fert[,i])
       mn <- mny*i*0.001
       sd <- mn*0.01
       raw_fert[,i] <- rnorm(len,mn,sd)
    }
 }
```

```
if(any(apply(cor_fert_bonanza,2,var) == 0))
{
   mny <- min(cor_fert_bonanza[cor_fert_bonanza >0])
   for(i in c(which(apply(cor_fert_bonanza,2,var) == 0)))
   {
      len <- length(cor_fert_bonanza[,i])
      mn <- mny*i*0.001
      sd <- mn*0.01
      cor_fert_bonanza[,i] <- rnorm(len,mn,sd)
   }
}

if(any(apply(cor_fert,2,var) == 0))
{
   mny <- min(cor_fert[cor_fert >0])

   for(i in c(which(apply(cor_fert,2,var) == 0)))
   {
      len <- length(cor_fert[,i])
      mn <- mny*i*0.001
      sd <- mn*0.01
      cor_fert[,i] <- rnorm(len,mn,sd)
   }
}


# Here we tidy up the data and get it ready for further processing.
mean_fert <- apply(raw_fert,2,gm)
var_fert <- apply(raw_fert,2,var)
mean_fert_bonanza <- apply(raw_fert_bonanza,2,gm)
var_fert_bonanza <- apply(raw_fert_bonanza,2,var)
data_Fert <-  cor_fert
data_Fert_bonanza <- cor_fert_bonanza


# Now to get the correlation in survival for the survivals rates that vary in the data
# This is it for the non-bonanza periods
corr_vital_rates <- cbind(data_Surv,data_Fert)
corr_mat_within <- cor(corr_vital_rates)

# This is it for the bonanza periods
corr_vital_rates_bonanza <- cbind(data_Surv_bonanza,data_Fert_bonanza)
corr_mat_within_bonanza <- cor(corr_vital_rates_bonanza)


# Here we are settig up the between year correlations
# Some initialization and categorizations of use here

years <- length(corr_vital_rates[,1])
years_bonanza <- length(corr_vital_rates_bonanza[,1])
cats <- length(corr_mat_within[1,])
corr_mat_between <- zeros(cats,cats)
corr_mat_between_bonanza <- zeros(cats,cats)
row_name <- list(names(corr_vital_rates))
col_name <- list(names(corr_vital_rates))


# Here is the loop for the between year correlations
for(i in 1:cats)
{
```

```
    for (j in 1:(cats))
    {

       corr_mat_between[i,j] <- ccf(corr_vital_rates[,j],corr_vital_rates[,i],lag=1,plot=F)$acf[1]
       corr_mat_between_bonanza[i,j] <- ccf(corr_vital_rates_bonanza[,j],corr_vital_rates_bonanza[,i],lag=1,plot=F)$acf[1]
    } # end j loop
}# End i loop

# Now I'm just putting the header back on the corr_mat_between so I can look at it a bit more__
corr_mat_between <-  matrix(corr_mat_between,ncol=cats,dimnames=c(row_name,col_name))
corr_mat_between_bonanza <-  matrix(corr_mat_between_bonanza,ncol=cats,dimnames=c(row_name,col_name))


#  Here we will have the vital rates for each stock as calculated previously (as per methods)

vr_mean_surv <- mean_surv
vr_vars_surv <- var_surv
vr_mean_surv_bonanza <- mean_surv_bonanza
vr_vars_surv_bonanza <- var_surv_bonanza

# Again Drop any variables that don't vary, see above!! First the survivals
vr_mean_surv <- vr_mean_surv[vr_vars_surv != 0]
vr_vars_surv <- vr_vars_surv[vr_vars_surv != 0]
vr_mean_surv_bonanza <- vr_mean_surv_bonanza[vr_vars_surv_bonanza != 0]
vr_vars_surv_bonanza <- vr_vars_surv_bonanza[vr_vars_surv_bonanza != 0]

# Same thing but fertilities
vr_mean_fert <- mean_fert
vr_vars_fert <- var_fert

vr_mean_fert_bonanza <- mean_fert_bonanza
vr_vars_fert_bonanza <- var_fert_bonanza

# Again Drop any variables that don't vary, see above!!
vr_mean_fert <- vr_mean_fert[vr_vars_fert != 0]
vr_vars_fert <- vr_vars_fert[vr_vars_fert != 0]

vr_mean_fert_bonanza <- vr_mean_fert_bonanza[vr_vars_fert_bonanza != 0]
vr_vars_fert_bonanza <- vr_vars_fert_bonanza[vr_vars_fert_bonanza != 0]


# Now that we got rid of all the constant values put them back together...

vr_means <- c(vr_mean_surv,vr_mean_fert)
vr_vars <- c(vr_vars_surv,vr_vars_fert)

vr_means_bonanza <- c(vr_mean_surv_bonanza,vr_mean_fert_bonanza)
vr_vars_bonanza <- c(vr_vars_surv_bonanza,vr_vars_fert_bonanza)



#########################################################################################################################################
# Now we simulate a load of vital rates so that we have a realistic set of vital rates to chose from.
# This is the development of the between year and within year correlations.


#########################################################################################################################################

# Some stuff to get the right sized loops below
```

```
tmax <- 2000  # Number of sets of vital rates to simulate
np_surv <- length(vr_mean_surv) # Number of survival vital rates
np_fert <- length(vr_mean_fert) # Number of fertility vital rates

np <- np_surv+np_fert # Total number of vital rates.

## The minimum and maximum values for each vital rate
# Zeros are place holders for rates that aren't stretched betas


##############################################  IMPORTANT STOCK DEPENDENT STEP!!!!! ##########################################################################
# NOW the min and max for using either the stretched beta or the log-normal distributions___
# The maximum the maximum and minimum are stock depdendent values.
#########################################################################################################################################################

vr_mins <- zeros(1,np)
#vr_mins <- c(apply(raw_surv,2,min),apply(raw_fert,2,min))*ones(1,np)
#vr_maxs <- c(ones(1,np_surv),apply(raw_fert,2,max))*ones(1,np)
vr_maxs <- max(raw_fert)*ones(1,np)
vr_mins_bonanza <- zeros(1,np)
#vr_mins_bonanza <- c(apply(raw_surv_bonanza,2,min),apply(raw_fert_bonanza,2,min))*ones(1,np)
vr_maxs_bonanza <- c(ones(1,np_surv),apply(raw_fert_bonanza,2,max))*ones(1,np)
#vr_maxs_bonanza[vr_maxs_bonanza < 0.01] <- vr_maxs_bonanza[vr_maxs_bonanza < 0.01]*100
vr_maxs_bonanza <- max(raw_fert_bonanza)*ones(1,np)
# What type of vital rates are these??
# In most case the survivals will be beta distributed, while I will make the survivals stretched betas.
# Note that when you use the log normal distribution some fertilities are just re-dick-ulously big
# So really can't use it at all, need to use strectched beta's and that make sense
vr_types <- c(rep(1,np_surv),rep(2,np_fert))
vr_types[c(ones(np_surv,1),fert_max)<1] <- 1


vr_types_b <- c(rep(1,np_surv),rep(2,np_fert))
vr_types_b[c(ones(np_surv,1),fert_max_bon)<1] <- 1


# Get the eigen values and eigen vectors...

D <- diag(eigen(corr_mat_within)$values)
W <- eigen(corr_mat_within)$vectors
D_bonanza <-diag(eigen(corr_mat_within_bonanza)$values)
W_bonanza <- eigen(corr_mat_within_bonanza)$vectors

# Now calculate the matrix (c12) used to make correlated standard normal variates from uncorrelated ones
# This is kind of like PCA in reverse, see Quan. Cons. Biol. by Doaks and Morris
# as this certainly not straightforward, theory and code is in Chapter 8, around  page 282.
# One for normal period and one for bonanzas.
C12 <- W %*% (sqrt(abs(D))) %*% t(W)
C12_bonanza <-W_bonanza %*% (sqrt(abs(D_bonanza))) %*% t(W_bonanza)

# Initialize the variables
norm_vals <- 0
corr_norms <- 0
corr_norms_bonanza <- 0

v_rate <- zeros(np,tmax)
v_rate_bonanza <- zeros(np,tmax)

# And start the loop
for(tt in 1:tmax) # This will loop through each years vital rates
```

```
{
   norm_vals <- rnorm(np) # just getting some random normals...
   corr_norms <- C12%*%norm_vals # now make the correlated normals_
   corr_norms_bonanza <- C12_bonanza%*%norm_vals # and for the bonanza
   for(i in 1:np)
   {
      # Each if statement here picks the appropriate distribution given the type of vital rate chosen.
      if(vr_types[i]==1)
      {
         # now get the beta vital rate with the same Fx as the first normal
         v_rate[i,tt] <- (beta.val(vr_means[i],vr_vars[i]^.5,stnormfx(corr_norms[i])))
      }
      if(vr_types_b[i]==1)
      {
         v_rate_bonanza[i,tt] <- (beta.val(vr_means_bonanza[i],vr_vars_bonanza[i]^.5,stnormfx(corr_norms_bonanza[i])))

      }
      if(vr_types[i]==2)
      {
         v_rate[i,tt] <- stretch.beta.val(vr_means[i],vr_vars[i]^.5,vr_mins[i],vr_maxs[i],stnormfx(corr_norms[i]))  # or the stretched beta
      }
      if(vr_types_b[i]==2)
      {

         v_rate_bonanza[i,tt] <- stretch.beta.val(vr_means_bonanza[i],vr_vars_bonanza[i]^.5,vr_mins_bonanza[i],vr_maxs_bonanza[i],stnormfx(corr_norms_bonanza[i]))  # or the stretched beta
      }
      if(vr_types[i]==3)
      {
         v_rate[i,tt] <- l.norm(vr_means[i],vr_vars[i],corr_norms[i]) # this one is the log normal
      }
      if(vr_types_b[i]==3)
      {
         v_rate_bonanza[i,tt] <- l.norm(vr_means_bonanza[i],vr_vars_bonanza[i],corr_norms_bonanza[i]) # this one is the log normal
      }
   } # end i loop
} # end tt loop

# Make it a matrix of appropriate dimensions
v_rate <- matrix(as.numeric(v_rate),ncol=tmax,nrow=np)
v_rate_bonanza <- matrix(as.numeric(v_rate_bonanza),ncol=tmax,nrow=np)
#hist(v.rate[1,])

#apply(ferts,2,median)

# Now we are making the cross correlation matrix for the within year vital rates.
cor_out <- zeros(np,np)
cor_out_bonanza <- zeros(np,np)
for(i in 1:np)
{
   for(j in 1:np)
   {
      cor_out[i,j] <- cor(v_rate[i,],v_rate[j,])
      cor_out_bonanza[i,j] <- cor(v_rate_bonanza[i,],v_rate_bonanza[j,])
   }}

#cor_out
#corr_mat_within

# Make these all data frames.
```

```
cor_within <- data.frame(cor_out)
cor_between <- data.frame(corr_mat_between)
cor_within_bonanza <- data.frame(cor_out_bonanza)
cor_between_bonanza <- data.frame(corr_mat_between_bonanza)


# I want to output these means and variances for the PVA itself so I don't waste time reprocessing them again and again.

processed_survival_data <- cbind(vr_mean_surv,vr_vars_surv)
processed_fertility_data   <- cbind(vr_mean_fert,vr_vars_fert)
processed_survival_data_bonanza <- cbind(vr_mean_surv_bonanza,vr_vars_surv_bonanza)

processed_fertility_data_bonanza <- cbind(vr_mean_fert_bonanza,vr_vars_fert_bonanza)

# Option to save everything at this step.
#save.image("d:/Dropbox/My_Papers/PVA/Results/Step_2.RData")


#########################################################################################################################################################
####################  Now we need to make sure our correlation matrices can actually be solved.
## This step is used to check if a correlation matrix you developed has impossible combinations of values
## due to missing data, or small numbers of observations_
#########################################################################################################################################################

################################### FIRST CHECK THE NON BONANZA DATA ###################################################


corr_within <- cor_within
corr_between <- cor_between

np <- min(dim(corr_within)) # Get the dimensions of the matrix___


corr_matrx_within <- corr_within

# This step necessary only if matrix is a lower-triangular, will fill up the upper
# or lower corners and make it symmetric___
# corr_matrx_within <- corr_matrx_within + t(corr_matrx_within)- eye(np)

# Get the eigen values
Eigens <- (eigen(corr_matrx_within)$values)
# Get the eigen vectors
W <- eigen(corr_matrx_within)$vectors

# Now we put this together to make our correlation matrix so we can make sure it is positive definite.
corr_matrx_within <- W %*% diag(Eigens) %*% t(W)

print("Now in the eigen value matrix (D) all elements should be positive, if not ")
print("Then some remedial action is required, lets check___")


# Define new matrix element
new_corr_matrix <- zeros(np,np)
# Warn me that we have a problem and we need to massage the matrix.
if(min(Eigens) < 00)
{
   print("Crap there is a negative, need to adjust the matrix")
   # Now I have to automate the code to drop all the negative elements
   # First get the number of 0's to drop
```

```
    drop <- length(Eigens[Eigens<0]) # How many of the Eigen values are negative

    Eigens_pos <- Eigens[1:(np-drop)]  # Drop them

    D <- diag(c(Eigens_pos,zeros(drop,1))) # Get a D eigen matrix, and pad the end of it with 0's

    # Now make a new matrix off a decomposition, note that this is a covariance matrix, not a correlation matrix as above___

    Cov_new <-  W %*% D %*% t(W)


    # Now take this covariance matrix and turn it into a "proper" correlation matrix"

    for(i in 1:np)
    {
       for(j in 1:np)
       {
          new_corr_matrix[i,j] <- Cov_new[i,j]/((Cov_new[i,i]*Cov_new[j,j])^0.5)
       }
    }

    # Now we can go back and get the corrected eigen values and vectors to do a proper decomposition

    D <- diag(eigen(new_corr_matrix)$values)

    W <- eigen(new_corr_matrix)$vectors

    C12_new <- W %*% sqrt(abs(D)) %*% t(W)

    # "Now how different are the two correlation matrices, i_e_ what is the difference between the old and the new?")

    diff_matrix <- (corr_matrx_within -new_corr_matrix) /corr_matrx_within

 } else print("All is good, no negative values for the bonanza, Woot et. Woot!")  # This is the end of the if loop___
 ##


 # Now determine which correlation matrix I want to save!!
 if(max(new_corr_matrix) > 0 )
 {
    print("and I saved the new correlation matrix")
    save.image("d:/Dropbox/My_Papers/PVA/Results/Step_3.RData")
    #write.table(new_corr_matrix,"/home/keithdm/Documents/PhD/R_programs/PVA/Demographic_PVA/Gulf_cod/Results/Neg_removed_Gulf_cod_corr_matrix_txt",row_names=F)

 } else
    print("So I just re-saved the old NON-BONANZA correlation matrix")
 #save.image("d:/Dropbox/My_Papers/PVA/Results/Step_3.RData")


 ##################################### END THE NON BONANZA DATA ##################################################################
 ###############################################################################################################################
 # NOW CHECK THE BONANZA DATA EXACT SAME PROCEDURE!
 ###############################################################################################################################

 corr_within_bonanza <- cor_within_bonanza
 corr_between_bonanza <- cor_between_bonanza

 np <- min(dim(corr_within_bonanza)) # Get the dimensions of the matrix___
```

```
corr_matrx_within_bonanza <- corr_within_bonanza



# This step necessary only if matrix is a lower-triangular, will fill up the upper
# or lower corners and make it symmetric___

# corr_matrx_within <- corr_matrx_within + t(corr_matrx_within)- eye(np)

Eigens <- (eigen(corr_matrx_within_bonanza)$values)

W <- eigen(corr_matrx_within_bonanza)$vectors

corr_matrx_within_bonanza <- W %*% diag(Eigens) %*% t(W)

print("Now in the eigen value matrix (D) all elements should be positive, if not ")
print("Then some remedial action is required, lets check___")



new_corr_matrix_bonanza <- zeros(np,np)


if(min(Eigens) < 00)
{
   print("Crap there is a negative IN THE BONANZA MATRIX, need to adjust the matrix")
   # Now I have to automate the code to drop all the negative elements
   # First get the number of 0's to drop

   drop <- length(Eigens[Eigens<0]) # How many of the Eigen values are negative

   Eigens_pos <- Eigens[1:(np-drop)]  # Drop them

   D <- diag(c(Eigens_pos,zeros(drop,1))) # Get a D eigen matrix, and pad the end of it with 0's

   # Now make a new matrix off a decomposition, note that this is a covariance matrix, not a correlation matrix as above___

   Cov_new <-  W %*% D %*% t(W)


   # Now take this covariance matrix and turn it into a "proper" correlation matrix"

   for(i in 1:np)
   {
      for(j in 1:np)
      {
         new_corr_matrix[i,j] <- Cov_new[i,j]/((Cov_new[i,i]*Cov_new[j,j])^0.5)
      }
   }

   # Now we can go back and get the corrected eigen values and vectors to do a proper decomposition

   D <- diag(eigen(new_corr_matrix)$values)

   W <- eigen(new_corr_matrix)$vectors

   C12_new <- W %*% sqrt(abs(D)) %*% t(W)
```

```
    # "Now how different are the two correlation matrices, i_e_ what is the difference between the old and the new?")

    diff_matrix <- (corr_matrx_within -new_corr_matrix) /corr_matrx_within

} else print("All is good, no negative values for the bonanza, woot et_ woot!")  # This is the end of the if loop___
##


# Now determine which correlation matrix I want to save!!
if(max(new_corr_matrix) > 0 )
{
    print("and I saved the new BONANZA correlation matrix")
#    save.image("d:/Dropbox/My_Papers/PVA/Step_3.RData")
    #write.table(new_corr_matrix,"/home/keithdm/Documents/PhD/R_programs/PVA/Demographic_PVA/Gulf_cod/Results/Neg_removed_Gulf_cod_corr_matrix_txt",row_names=F)

} else
    print("So I just re-saved the old BONANZA correlation matrix")


#########################################################################################################################################################################
    # This is box 9_5 from Morris and Doak___
    # This program performs simulations to estimate stochastic growth rate and extinction risk
    # sensistivity analyses for vital rate means, variances, and correlations
    # Part of this comes from Box 8_9, Between year correlations, so see file
    # autocorrelations_cross_correlations_within_year_correlations_ in_vital_rates.R
    #rm(list=ls(all=T))

    library(matlab)
    library(PVA)
    library(snow)
    library(Rmpi)
    # First up load up the PVA function____

    source("d:/Dropbox/R_functions/PVA_Step_5_Sens_Elas_new.R")

    ###################################PARAMETERS ################################

    vr_fertility <- processed_fertility_data #read_table("/home/keithdm/Documents/PhD/R_programs/PVA/Demographic_PVA/Gulf_cod/Results/processed_fertility_data_step2_csv",header=T)
    vr_survival <- processed_survival_data #read_table("/home/keithdm/Documents/PhD/R_programs/PVA/Demographic_PVA/Gulf_cod/Results/processed_survival_data_step2_csv",header=T)


    vital_rate_fert <- data.frame(vr_fertility)
    vital_rate_surv <- data.frame(vr_survival)

    # Now the population data___
    pop_data <- num

    # Extract the means and variances of each vital rate seperately__

    vr_mean_surv <- as.matrix(vital_rate_surv[grep("mean",names(vital_rate_surv))])
    vr_mean_fert <- as.matrix(vital_rate_fert[grep("mean",names(vital_rate_fert))])
    vr_vars_surv <- as.matrix(vital_rate_surv[grep("var",names(vital_rate_surv))])
    vr_vars_fert <- as.matrix(vital_rate_fert[grep("var",names(vital_rate_fert))])

    # And also put them together into one
    vr_means <- c(vr_mean_surv,vr_mean_fert)
    vr_vars <- c(vr_vars_surv,vr_vars_fert)
```

```
# The number of transitions (survivals), and fecundities
num_trans <- length(vr_mean_surv)
fec_classes <- length(vr_mean_fert)
np <- num_trans+fec_classes


# Make sure the correlation "corr_within", and the between year correlation "corr_between" are matrices

corr_between <- as.matrix(corr_between)
corr_between_b <- as.matrix(corr_between_bonanza)

cov_within <- as.matrix(corr_within)
cov_within_b <- as.matrix(corr_within_bonanza)

# Here we define the type of distribution used to estimate each of these rates
# "beta" <- beta, "st.beta" <- stretched beta, "lnorm" <- log normal


#vr_types <- c(rep("beta",num_trans),rep("lnorm",fec_classes))
#vr_types


# In most stocks there will be some variables that are constant (e.g. M =0.2, or Fertility of 0), we need to add these back into the correlation matricies.
# Are their any more variables in the vr_ means than the covariance matrix (i.e. terms that are invariant??)

num_const <- (length(vr_means)-length(cov_within[1,]))
if(num_const > 0)
{
   zero_pad_col <-zeros(length(cov_within[1,]),num_const)
   zero_pad_row <- t(zeros((length(cov_within[1,])+num_const),num_const))
   cov_matrix_adj1 <- as.matrix(cbind(cov_within,zero_pad_col))

   cov_matrix <- as.matrix(rbind(cov_matrix_adj1,zero_pad_row))
} else cov_matrix <- cov_within  # end little if loop

# Now for the bonanza are their any more variables in the vr_ means than the covariance matrix (i.e. terms that are invariant?)

num_const_b <- (length(vr_means)-length(cov_within_b[1,]))
if(num_const_b > 0)
{
   zero_pad_col <-zeros(length(cov_within_b[1,]),num_const_b)
   zero_pad_row <- t(zeros((length(cov_within_b[1,])+num_const_b),num_const_b))
   cov_matrix_adj1_b <- as.matrix(cbind(cov_within_b,zero_pad_col))

   cov_matrix_b <- as.matrix(rbind(cov_matrix_adj1_b,zero_pad_row))
} else cov_matrix_b <- cov_within_b  # end little if loop


# Using the mean of the survival and fertility matrices I need to make the mean matrix.  First for non-bonanzas
# mean.matrix is found in the PVA package and is used to make the Leslie matrix for each population.
mx <- mean.matrix(vr_mean_surv,vr_mean_fert)
# and for bonanzas
da_mx_bonanza <- mean.matrix(mean_surv_bonanza,mean_fert_bonanza)
# just for curiosity what are they?
eigen(mx)$values[1]
eigen(da_mx_bonanza)$values[1]

#### PREMADE FILES - #2 Random values matrix, use the function "Beta_values_calculator.R" to get this matrix___
```

```
## Now I will load in a set of premade Beta values_ Can be re-made using box 8_10, or file Beta_values_calculator.R, using
## the vital rates for the species of interest, for this question I need 14 of them___
## THEY SHOULD BE MADE AND SAVED FROM STEP 4

# Now get the random values matrix, it says "beta_calcs", but also can be used to generate log normal or stretched beta along with just a beta distribution.
betas <- beta_calcs(vr_means,vr_vars,vr_types,vr_mins,vr_maxs)
betas_b <- beta_calcs(vr_means_bonanza,vr_vars_bonanza,vr_types_b,vr_mins_bonanza,vr_maxs_bonanza)

########################################################## Set up modle parameters and initial conditions ###################################################################
## Now the model parameters___
nnodes <- 7                          # The number of processors to run on
runs <- 715                          # How many trajectories to do on each processor

n0 <- pop_data[1,1:num_trans]              # Initial Population structure
N_start <- sum(pop_data[length(pop_data[,1]),])        # Total Initial Population size

# HERE WE GET THE set up for the Survivals, MSY, and collapse
tot_num <- rowSums(num)
N_ext <- 0.07*max(tot_num)                 # Our estimate of collapse
tmax <- 100                          # Number of years to simulate
dims <- length(mx[1,])                   # Length of the projection matrix
perts <- 0.05                        # The magnitude of the perturbations used to estimate sensistivities/elasticities
c_row <- 1                           # Used to choose the correlation matrix elements
c_col <- 2                           # Used to choose the correlation matrix elements
pert_num <- (2*np)+2                     # Number of parameters to run the sensistivity/elasticity analysis on
#pert_num <- 10
nt <- n0                             # Initialize the total populatin at 0 to the initial population


################################################ NOW START THE BONANZA PARAMETERIZATION   ######################
#########
#########################################################################################################################

# Here is the mean and variance for the bonanza period___

# Here is the max population size in which you can have a bonanza, we use, on average, 2 times the maximum population.

# This makes the ceiling a random number so that we aren't pinned to the same maximum every time there is a bonanza
# sd I made 10% of maximum which is completely arbitrary, but seems to bound this to vary from hist max to 3 times hist max
# which if anything is very un-conservative, assumes populations can get pretty big, and the historic maximum we've
# ever seen is at the low end of the curve
ceiling_bonanza <- rnorm(runs,2*max(apply(pop_data,1,sum)),0.1*2*max(apply(pop_data,1,sum)))

# This tells me what cells in data are bonanzas, if it is just one period of bonanza
# Then the result will be a string of 1's, thus I know it was just one bonanza
# if there are a bunch of short duration good years then we will know how many of these
# short duration bonanza's there are.

################################  Here I figure out how many bonanza's are in the dataset...

if(length(bonz_years) > 0)
{
   num_bonanza =1
   for(i in 1:(length(bonz_years)-1))
   {
      if(bonz_years[i+1] - bonz_years[i] > 1) num_bonanza <- num_bonanza+1
      if(bonz_years[i+1] - bonz_years[i] == 1) num_bonanza <- num_bonanza

   }
```

```
}
if(length(bonz_years) == 0) num_bonanza  <-0



av_bonanza_duration <- length(raw_fert_bonanza[,1])/num_bonanza # this is based solely on the 2*sd criteria!
total_bonanza_duration <- length(raw_fert_bonanza[,1])  # this is based solely on the 2*sd criteria!
years_total <- length(pop_data[,1]) # number of years in the data
freq_bonanza <- num_bonanza/years_total # frequency of the bonanza
percent_time_in_bonanza <- total_bonanza_duration/years_total # How long the population was in a bonanza over life of simulation

# I use a binomial distribution to determine how long we are going to be in a bonanza
freq_b <- rbinom(runs,tmax,percent_time_in_bonanza)
# The final bit is to do a number of runs to get the growth rate and collapse risks
# Here I can a-priori set up what the bonanza years will be, if population is too high they won't enter
# the bonanza, so actual number of bonanza years will be slightly below this...
#########################################################################################################
bonanza <- zeros(tmax,runs) # This will tell me if in a bonanza, and how long i've been in the bonanza___
# Now I need to be careful, was there only one correlated bonanza, or were there several bonanzas
# that were correlated during this period??

##############################################  End Bonanza Parameterization ##########################################



############################################## END INITIAL PARAMETERS ##################################################

# Now get the M^0_5 matrix___
D <- eigen(cov_matrix)$values
D <- diag(D)
W <- eigen(cov_matrix)$vectors

M_0_5 <- W %*% (sqrt(abs(D))) %*% t(W)

# Now for the bonanza'ed bits___

D_b <- eigen(cov_matrix_b)$values
D_b <- diag(D_b)
W_b <- eigen(cov_matrix_b)$vectors

M_0_5 <- W %*% (sqrt(abs(D))) %*% t(W)
M_0_5_b <- W_b %*% (sqrt(abs(D_b))) %*% t(W_b)

# Now get the eigen values and right vectors for the mean matrix, above mx___

uu <- eigen(mx)$vectors
lambda <- eigen(mx)$values
lambda1 <- lambda[1] # get the leading eigen value___
u_vec1 <- uu[,1]/sum(uu[,1]) # and it's standardized associated right eigen vector
#initial population size
n0 <- as.numeric(N_start*sum(u_vec1)*u_vec1)


# HERE WE GET THE set up for the Survivals, persistance, and collapse
tot_num <- rowSums(num)

N_surv <- 0.1*max(tot_num) # I don't en up using this term.
```

```
# Next we figure out what percent of populations are in a healthy state, that is their population is above 40\% of maximum, for
# populations that declined significantly this would represent a return to a healthy state.  This will be compared to an average
# over a number of years (10-20 years)
N_msy <- 0.4*max(tot_num)
# Finally, what percent of the populations are getting absolutely hammered
N_ext <- 0.07*max(tot_num)

########################################## INITIALIZATION ##########################################################################3
# Initialize the variables for the loops___

MSY_sens <- zeros(tmax,pert_num)
MSY_Elast <- zeros(tmax,pert_num)
Surv_sens <- zeros(tmax,pert_num)
Surv_Elast <- zeros(tmax,pert_num)


Ext_sens <- zeros(tmax,pert_num)
Ext_Elast <- zeros(tmax,pert_num)
L_sens <- zeros(pert_num,1)
L_elsat <- zeros(pert_num,1)
Temp_vr_means <- 0 ; Temp_vr_vars <- 0 ; Temp_betas <- 0
vr_diff <- 0
Temp_M_0_5 <- 0
Temp_corr_mx <- cov_matrix
Temp_corr <- cov_matrix
Pr_ext <- zeros(tmax,1)
log_lambda <- 0        # stochastic log lambda
m <- 0
raw_elemts <- 0
vrs <- 0
index = 0
Lambda_sens <- 0
Lambda_elast <- 0
Extinct <- 0                # Initial number of extinct pop'ns_
log_Lambda <- 0             # Initialize log_lambda
base_ext <- 0
base_ext2 <- zeros(tmax,runs)
base_ext3 <- zeros(tmax,runs)
cum_ext <- cumsum(Pr_ext/runs)
orig_cum_ext <- 0
cum_ext2 <- 0
Ext_Results <- zeros(tmax,pert_num)
Surv_Results  <- zeros(tmax,pert_num)
MSY_Results   <- zeros(tmax,pert_num)

smooth_Ext_Elast <- zeros(tmax,pert_num)
smooth_MSY_Elast <- zeros(tmax,pert_num)
smooth_Surv_Elast <- zeros(tmax,pert_num)

Lambda_Results <- 0
P_msy <- zeros(tmax,pert_num)
P_surv <-zeros(tmax,pert_num)
P_ext <- zeros(tmax,pert_num)
diff_vr <- 0
fec <- 0
fec2 <- 0
vr_surv <- 0
all_nt <- array(0,dim=(c(dims,tmax,runs)))
nt <- 0
```

```
fecundities <- zeros(fec_classes,runs)
# Bonanza variables to initialize___
Temp_vr_means_b <-0
Temp_vr_vars_b <-0
Temp_betas_b <- 0
Temp_M_0_5_b <- 0
vr_diff_b <- 0
Temp_corr_mx_b <- cov_matrix_b
W_cor_b <- 0
D_cor_b <- 0
D_cor_b <- 0
diff_vr_b <- 0
Temp_corr_b <- cov_matrix_b
x <- 1

##########################################SIMULATION BEGINS ##################################################################

# Now loop one by one through each perturbation of each mean, variance, and correlation
# For each type of rate, a change is made of size 'perts', the appropriate set of parameters are re-made
# and the model is then simulated stochasitically, should be fun___
#pert <- 0 # For code testing only___
#pert_num =2
for(pert in 0:pert_num)

{
 # Temporary variables
   Temp_vr_means <- vr_means
   Temp_vr_vars <- vr_vars
   Temp_betas <- betas
   Temp_M_0_5 <- M_0_5
   # Here are the bonanzas___
   Temp_vr_means_b <- vr_means_bonanza
   Temp_vr_vars_b <- vr_vars_bonanza
   Temp_betas_b <- betas_b
   Temp_M_0_5_b <- M_0_5_b
   # Now depending on what is changed, modify the appropriate values,
   if(pert > 0 && pert < np+1) # This if statement essentially takes care of the mean survival/fertility elasticities
   {
      Temp_vr_means[pert] <- vr_means[pert] # here we change the mean rates___
      Temp_vr_means_b[pert] <- (1+perts)*vr_means_bonanza[pert] # Here we go for the bonanza___
      if(Temp_vr_means[pert]*(1-Temp_vr_means[pert]) <= vr_vars[pert]) # make sure the mean doesn't get too large
      {
         Temp_vr_means[pert] <- vr_means[pert]
      } # End if looop ensuring mean doesn't get too high!
      if(Temp_vr_means_b[pert]*(1-Temp_vr_means_b[pert]) <= vr_vars_bonanza[pert]) # make sure the mean doesn't get too large
      {
         Temp_vr_means_b[pert] <- (1-perts)*vr_means_bonanza[pert]
      } # End if looop ensuring mean for the bonanza loop doesn't get too high either!

      vr_diff <- Temp_vr_means[pert] - vr_means[pert]
      vr_diff_b <- Temp_vr_means_b[pert] - vr_means_bonanza[pert] # And for the bonanza___
      # Now using the function "function_to_call_beta_or_strected_beta_functions.R" to get beta/stretched beta values_

      Temp_betas[,pert] <- betaset(vr_types[pert],Temp_vr_means[pert],Temp_vr_vars[pert],vr_mins[pert],vr_maxs[pert])
      Temp_betas_b[,pert] <- betaset(vr_types_b[pert],Temp_vr_means_b[pert],Temp_vr_vars_b[pert],vr_mins_bonanza[pert],vr_maxs_bonanza[pert])
   } else # End the 'if' pert section for the means

      # Now lets change up the variances___
      if(pert > np && pert < 2*np+1)  # This if statement essentially takes care of the variance of survival/fertility elasticities
```

```
    {
      Temp_vr_vars[pert-np] <- vr_vars[pert-np]
      vr_diff <- Temp_vr_vars[pert-np] - vr_vars[pert-np]
      Temp_betas[,pert-np] <- betaset(vr_types[pert-np],Temp_vr_means[pert-np],Temp_vr_vars[pert-np],
                                      vr_mins[pert-np],vr_maxs[pert-np])

      # And for the bonanzas.__
      Temp_vr_vars_b[pert-np] <- (1+perts)*vr_vars_bonanza[pert-np]
      vr_diff_b <- Temp_vr_vars_b[pert-np] - vr_vars_bonanza[pert-np]
      Temp_betas_b[,pert-np] <- betaset(vr_types_b[pert-np],Temp_vr_means_b[pert-np],Temp_vr_vars_b[pert-np],
                                        vr_mins_bonanza[pert-np],vr_maxs_bonanza[pert-np])

    } else # End the 'if' pert section for the variances
      # Here I modify the occurance of bonanzas, note that I only change total time in bonanza, and the duration of an individual bonanza
      # The number of bonanza's is not independent of these two factors so adding it in would be silly!!
      # This is the percent time in bonanza's
      if(pert == (2*np+1))
      {
        percent_time_in_bonanza <- percent_time_in_bonanza*(1-perts)

      }
# This is the average duration of an individual bonanza
if(pert == (2*np+2))
{
  av_bonanza_duration <- av_bonanza_duration*(1-perts)
}


diff_vr[pert] <- vr_diff # Just want to look at the different vr_diff's___
diff_vr_b[pert] <- vr_diff_b # Just want to look at the different vr_diff's___
#} # TEMPORARY to test the pert loop!!!!!!!!!!

## Now run the step5_real function, basically the replicates of the above, run "runs" times___
## The rest of the code is to run everything in parallel.  I had 8 processors so ran this on 7 of them.


########## initialize cluster ############
if( (runs>1) & (nnodes>1) ) {
  if(!require(snow))
    stop("You have to install SNOW package to use cluster")
  # make cluster object
  cl <- makeCluster(nnodes,type="SOCK")
  # correct the possible correlation problem
  clusterApplyLB(cl, runif(length(cl), max=1000000000), set.seed)
}
# permutation - use MPI cluster if specified
if(nnodes > 1) { # use MPI cluster
  # use cluster call to do permutation
  # calculate the number of permutation needed in each node
  runs_cluster <- rep(floor((runs-1)/nnodes), nnodes)
  # maybe some leftovers
  leftover <- runs - 1 - sum(runs_cluster)
  if(leftover > 0)
    runs_cluster[1:leftover] <- runs_cluster[1:leftover] + 1

  # load library on all nodes
  clusterEvalQ(cl, library(matlab))
  clusterEvalQ(cl, library(PVA))
```

```
    # And now get the cluster running and solve this bugger.

    cat(paste("Doing permutation on", nnodes, "cluster nodes ___ \n"))

    cluster_run <- clusterCall(cl, step5_real,runs=runs,tmax=tmax,n0=n0,pert=pert,np=np,Temp_M_0_5=Temp_M_0_5,num_trans=num_trans,fec_classes=fec_classes,
                               vr_types=vr_types,Temp_betas=Temp_betas,Temp_vr_means=Temp_vr_means,Temp_vr_vars=Temp_vr_vars,vr_types_b=vr_types_b,
                               N_start = N_start,Temp_M_0_5_b=Temp_M_0_5_b,Temp_betas_b=Temp_betas_b,Temp_vr_means_b=Temp_vr_means_b,
                               Temp_vr_vars_b=Temp_vr_vars_b,bonanza=bonanza,ceiling_bonanza=ceiling_bonanza,
                               years_total=years_total,all_nt=all_nt,percent_time_in_bonanza=percent_time_in_bonanza,
                               av_bonanza_duration=av_bonanza_duration,tot_num=tot_num,N_msy=N_msy,N_surv=N_surv,N_ext=N_ext)

    stopCluster(cl)
}
#}

# Now we need to initialize some variables and run a loop to organize all the data of interest from the analysis.
log_Lambda <- 0
Pr_msy<- 0
Pr_surv<- 0
Pr_ext<- 0
bonanza_base<- zeros(tmax,nnodes*runs)
all_nt_base <- array(0,dim=(c(dims,tmax,nnodes*runs)))
i=2
# Here's the loop to get the data from each cluster into something usable.
for(i in 1:nnodes)
{
    #    if(i == 1) Pr_ext <- unlist(cluster_run[[i]][2]) else Pr_ext <- Pr_ext + unlist(cluster_run[[i]][2])
    if(i == 1)
    {
        log_Lambda <- unlist(cluster_run[[i]][1])
        Pr_msy    <- unlist(cluster_run[[i]][4])
        Pr_surv <- unlist(cluster_run[[i]][5])
        Pr_ext     <- unlist(cluster_run[[i]][6])
    }
    if(i > 1)
    {
        log_Lambda<- log_Lambda + unlist(cluster_run[[i]][1])
        Pr_msy<- Pr_msy + unlist(cluster_run[[i]][4])
        Pr_surv <-Pr_surv+ unlist(cluster_run[[i]][5])
        Pr_ext     <- Pr_ext+ unlist(cluster_run[[i]][6])
    }

    if(pert==0)
    { # This will give the results for the initial run only, not the elasticities cause that'd be way alot of data!
        bonanza_base[,(runs*i-(runs-1)):(runs*i)] <- matrix(unlist(cluster_run[[i]][2]),ncol=runs) # This is the bonanza data for base model
        all_nt_base[,,(runs*i-(runs-1)):(runs*i)] <- array(unlist(cluster_run[[i]][3][[1]]),c(num_trans,tmax,runs)) # This is the entire dataset for base model
    }
}

rm(cluster_run) # I've extracted everything I want, so get rid of this to save some space!

Lambda_Results[pert] <- exp(log_Lambda/(nnodes*runs))
P_msy[,pert] <- Pr_msy /(nnodes*runs)
P_surv[,pert] <-Pr_surv/(nnodes*runs)
P_ext[,pert] <-Pr_ext/(nnodes*runs)

if(pert == 0) # This gets the baseline results___
{
    base_ext <- cumsum(unlist(Pr_ext)/(nnodes*runs))
```

```
      base_msy <- cumsum(unlist(Pr_msy)/(nnodes*runs))
      base_surv <- cumsum(unlist(Pr_surv)/(nnodes*runs))
      base_lambda <- exp(log_Lambda/(nnodes*runs))
      orig_base_lambda <- base_lambda
      # now smooth the extinction CDF function____
      base_ext2 <- (base_ext + c(base_ext[1],base_ext[1:tmax-1]) + c(base_ext[2:tmax],base_ext[tmax]))/3 # 3 year running average, tis o_k_
      base_msy2 <- (base_msy + c(base_msy[1],base_msy[1:tmax-1]) + c(base_msy[2:tmax],base_msy[tmax]))/3 # 3 year running average, tis o_k_
      base_surv2 <- (base_surv + c(base_surv[1],base_surv[1:tmax-1]) + c(base_surv[2:tmax],base_surv[tmax]))/3 # 3 year running average, tis o_k_

    } else # End the pert == 0 if statement___

      # Now summarize the results!
      cum_ext <- cumsum(Pr_ext/(nnodes*runs))
    cum_surv <- cumsum(Pr_surv/(nnodes*runs))
    cum_msy <- cumsum(Pr_msy/(nnodes*runs))
    orig_cum_ext <- cum_ext
    # Again smooth the extinction CDF function___
    cum_ext2 <- (cum_ext + c(cum_ext[1],cum_ext[1:tmax-1]) + c(cum_ext[2:tmax],cum_ext[tmax]))/3 # 3 year running average
    Ext_Results[,pert]<- cum_ext2
    cum_surv2 <- (cum_surv + c(cum_surv[1],cum_surv[1:tmax-1]) + c(cum_surv[2:tmax],cum_surv[tmax]))/3 # 3 year running average
    Surv_Results[,pert]<- cum_surv2
    cum_msy2 <- (cum_msy + c(cum_msy[1],cum_msy[1:tmax-1]) + c(cum_msy[2:tmax],cum_msy[tmax]))/3 # 3 year running average
    MSY_Results[,pert]<- cum_msy2


    # Now for the vital rates that actually vary we can look at the extinction rates.
    if(vr_diff_b != 0)
    {
      # Calculate extinction time sensitivities and elasticities
      Ext_sens[,pert] <- (cum_ext2 - base_ext2) /vr_diff_b
      # Next line may yield divide by 0 errors, but that's o_k_!!!!!!!!!!!!
      Ext_Elast[,pert] <- ((cum_ext2 - base_ext2)/base_ext2)/(perts*vr_diff_b/abs(vr_diff_b))
      # Now for the MSY sensistivities
      MSY_sens[,pert] <- (cum_msy2 - base_msy2) /vr_diff_b
      # Next line may yield divide by 0 errors, but that's o_k_!!!!!!!!!!!!
      MSY_Elast[,pert] <- ((cum_msy2 - base_msy2)/base_msy2)/(perts*vr_diff_b/abs(vr_diff_b))
      # And finally Survival sens...
      Surv_sens[,pert] <- (cum_surv2 - base_surv2) /vr_diff_b
      # Next line may yield divide by 0 errors, but that's o_k_!!!!!!!!!!!!
      Surv_Elast[,pert] <- ((cum_surv2 - base_surv2)/base_surv2)/(perts*vr_diff_b/abs(vr_diff_b))
      # Now calculate stochastic lambda sensistivities and elasticities
      Lambda_sens[pert] <- (Lambda_Results[pert] -base_lambda)/vr_diff_b
      Lambda_elast[pert] <- ((Lambda_Results[pert]-base_lambda) / (base_lambda)) / (perts*vr_diff_b/abs(vr_diff_b))
    } # end the vr_diff if loop

    ## Now smooth the elasticities, basically running a smoother through the elasticities.

    smooth_Ext_Elast[,pert] <- (Ext_Elast[,pert]  +  c(Ext_Elast[1,pert],Ext_Elast[1:tmax-1,pert]) + c(Ext_Elast[2:tmax,pert],Ext_Elast[tmax,pert]))/3;
    smooth_MSY_Elast[,pert] <- (MSY_Elast[,pert]  +  c(MSY_Elast[1,pert],MSY_Elast[1:tmax-1,pert]) + c(MSY_Elast[2:tmax,pert],MSY_Elast[tmax,pert]))/3;
    smooth_Surv_Elast[,pert] <- (Surv_Elast[,pert]  +  c(Surv_Elast[1,pert],Surv_Elast[1:tmax-1,pert]) + c(Surv_Elast[2:tmax,pert],Surv_Elast[tmax,pert]))/3;

  } # End the pert loop


# Save this baby!
  save.image(paste("d:/Dropbox/My_Papers/PVA/Results/Sensistivity/Step_5_",nm[g],"_bon.RData",sep=""))

} # End the loop for each population
```

```
#########################################################################################################################################################
#########################################################################################################################################################
#########################################################################################################################################################
# The functions used in the analysis (PVA package) # The functions used in the analysis # The functions used in the analysis # The functions used in the analysis #
# The functions used in the analysis # The functions used in the analysis # The functions used in the analysis # The functions used in the analysis #
#########################################################################################################################################################
#########################################################################################################################################################
#########################################################################################################################################################

#########################################################################################################################################################################
########## Begin Log Normal function
#########################################################################################################################################################################
l.norm <- function(means,vars,raws)
{
  # Log.normal function, one built into R, but this might
  # be more user friendly with the code to follow with this PVA stuff
  # or maybe not...

  n.means <- log(means) - 0.5*log(vars/means^2+1)
  n.vars <- log(vars/means^2 +1)
  norms <- raws*sqrt(n.vars)+ n.means
  lns <- exp(norms)
}
#########################################################################################################################################################
########## End Log Normal function
#########################################################################################################################################################


#########################################################################################################################################################
########## Begin Beta random number generator
#########################################################################################################################################################
beta.val <- function(mn,sd,fx)
{
  library(zipfR)
  # This function caluclates a random number from a beta distribution with mean (mn), st. deviation (sd), and
  # cumulative distribution function (fx).  This function in MATLAB uses something called the "betainc" function
  # will try and use something analogous in R.., see page 277 in book...

  if(sd == 0) bb <- mn
  toler <- 1e-9 # how close the CDF value of the answer must be to the input value (Fx)
  var <- sd^2
  (if (var >= (1-mn)* mn) {
    return("You dumb SOB, your variance is too big.. jerk")
    break # End the run
  }
  else
  {
    #       return("I like cheese")
    vv <- mn*((mn*(1-mn)/var)-1) # Here we are getting the a and b parameters for the beta distribution
    ww <- (1-mn)*((mn*(1-mn)/var)-1)

    up.value <- 1
    low.value <- 0

    # Start using an initial guess for x, using the random number generator to adds some "wiggle"
    # to the start of the search, should help to avoid biases in this search...
```

```
    x <- 0.5 + 0.02*runif(1)

    i <- Rbeta(x,vv,ww) #This calculates the CDF value of x, THE Rbeta is RIGHT, IBeta is not the same function see R and Matlab help!!

    # Now the below loop iteratively searches for a value of x, trying to get the value of
    # x has a CDF that is within the tolerance set above.  Unless close to 0 or 1, this would cause problems
    # and would stop the search.
    #  }) # temporary end of the else statement from way above!!
    while((toler < abs(i-fx)) && ((x > 1e-6)) && ((1-x) > 1e-6))
    {
      (if (fx > i)
      {
        low.value <- x
        x <- (up.value+low.value)/2
      }
      else
      {
        up.value <- x
        x <- (up.value+low.value)/2
      })
      i <- Rbeta(x,vv,ww)
    }

    # Now the below makes values of x kinda random, this will get rid of problems associated with variances that are very small or very large
    # This will also truncate x, with small values = tolerance and large values = 1- tolerance.

    bbb <- x + toler*0.1*(0.5-runif(1))
    if(bbb< toler) bbb <- toler
    if(bbb>1) bbb <- 1-toler

    bb <- bbb

  }) # end the else statement from way above!!
  #return(i)
} # End of function...
#########################################################################################################################################
########## Begin Beta random number generator
#########################################################################################################################################


#########################################################################################################################################
#############  Begin Beta values calculator
#########################################################################################################################################

beta_calcs <- function(vr.means,vr.vars,vr.types,vr.mins,vr.maxs)
{
  library(matlab)

  ## This program is used to generate the beta values for each vital rate/matrix element so that you can do a stochastic PVA
  ## This is adapted from box 8.10 of Morris and Doaks, it is basically just the first part of that program turned into a function...

  ## NOTE THAT YOU WILL NEED THE FUNCTIONS stretched.beta.val, beta.val, l.norm, stnormfx LOADED to run this program, i.e. my library(PVA)

  # Now the minimum and maximum values for each vital rate ONLY NON-ZERO if USING the STRETCHED beta distribution
  # in the default example none are so...

  np <- length(vr.means)

  count <- 0
```

```
  # First make sure that the vr.types is in the right format, could be either 1-3 or the name of the fuction, otherwise
  # we'll get an error

  for(i in 1:np)
  {
   count[i] <- i
   if(vr.types[i] != "beta" && vr.types[i] != "st.beta" && vr.types[i] != "lnorm")
   {
    if(vr.types[i] != 1 && vr.types[i] != 2 && vr.types[i] != 3)
    {
     message(paste("Dude value '", count[i], "' of vr.types is '" ,vr.types[i], "' it gotta be either between 1 and 3 or
                one of \"beta\" \"st.beta\" or \"lnorm\" "))
    } else
     if(vr.types[i] == 1) vr.types[i] <- "beta"
    if(vr.types[i] == 2) vr.types[i] <- "st.beta"
    if(vr.types[i] ==3) vr.types[i] <- "lnorm"
   } # end vr.types if statement
  } # end i loop

  ################################### NOW THE FUN REALLY BEGINS ###########################################################

  ## First up this section makes a set of beta or stretched beta values to choose from
  ## during the simulations, it makes 99 vaues for 1% increments of Fx for each parameter of interest
  ## Hopefully I also figure out how to make an option to use values already calculated for a life history
  ## or to recalculate them...


  para.betas <- zeros(99,np)


  # Begin the calculation,
  for(i in 1:(np))
  {
   if(vr.types[i] != "lnorm")
   {
    for(fx99 in 1:99)
    {
     if(vr.types[i] == "beta")
     {
      para.betas[fx99,i] <- beta.val(vr.means[i],sqrt(vr.vars[i]),fx99/100)
     } # end vr.types == 1
     if(vr.types[i] =="st.beta")
     {
      para.betas[fx99,i] <- stretch.beta.val(vr.means[i],sqrt(vr.vars[i]),vr.mins[i],vr.maxs[i],fx99/100)
     }
    } # end fx99 loop
   } # end vr.types[i] != 3 loop
  } # end of i loop

  return(para.betas)
 } # End of the function
 ##########################################################################################################################################################################
 ##########3  End Beta values calculator #############
 ##########################################################################################################################################################################



 ################# ################# ################# ################# ################# ################# #################
```

```
################  Begin Matrix Maker
################ ################ ################ ################ ################ ################ ################


mean.matrix <- function(vrs,fec)
{
 # This is a routine to make the population matrix from
 # a set of values for the vital rates.
 #WARNING: this function uses whatever the current values
 #are of vrs to build the mx.

 #g21 g32 g43 g54 g65 g76 g87 g98 g109 etc...
 #1    2  3  4  5  6  7  8   9
 # fec3 fec4 fec5 fec6 fec7 fec8  fec9 fec10 etc...
 #   1   2   3   4   5   6   7    8

 dims = length(vrs)
 mature <- dims - length(fec)
 mx = zeros(dims,dims)
 for (i in 1:dims)
 {
  for(j in 1:dims)
  {

   if(i == 1 && j >mature) # First line, basically just fecundities!
   {
    mx[i,j] <- fec[j-mature] # This puts the fecundities into the right cells!
   } # End the fecundities

   if(i >1 && j == (i-1)) # This gets the off-diagonal
   {
    mx[i,j]=vrs[j]
   } # End the off-diagonal bit

   if(i ==dims && j == dims) # And this fills in the survivorship in last age class
   {
    mx[i,j] <- vrs[i-1]
   } # End last age class...
  } # End j
 } # end i

 return(mx)
}

################ ################ ################ ################ ################ ################ ################
################  END Matrix Maker
################ ################ ################ ################ ################ ################ ################



################ ################ ################ ################ ################ ################ ################
################  Begin standard normal to cumulative normal conversion
################ ################ ################ ################ ################ ################ ################


stnormfx <- function(xx)
{

 # stnormfx takes a value from a std. normal distr., xx,
```

```
  # and returns its cumulative distribution function value
  # (Abramowitz and Stegun 1964).
  ci <- 0.196854  # these are approximation constants
  cii <- 0.115194
  ciii <- 0.000344
  civ <- 0.019527

  if (xx >= 0) z<- xx  else z <- -xx
  a <- 1 + (ci*z) + (cii*z*z)
  b <- (ciii*z*z*z) + (civ*z*z*z*z)
  w <- a+b
  if (xx >= 0)    ff <- 1 - 1/(2*w*w*w*w)  else  ff <- 1 - (1 - 1/(2*w*w*w*w))
}


################ ################ ################ ################ ################ ################ ################
################ End standard normal to cumulative normal conversion
################ ################ ################ ################ ################ ################ ################




################ ################ ################ ################ ################ ################ ################
################ BEGIN PVA Sens/Elasticity Program  PVA Sens/Elasticity Program  PVA Sens/Elasticity Program PVA Sens/Elasticity Program
################ ################ ################ ################ ################ ################ ################
# Here is the function to loop through the PVA Sensistivity and Elasticity analysis using a paralell processor set up
# THis is a.k.a step 5 of the PVA analysis.

step5_real <- function(runs=runs,tmax=tmax,n0=n0,pert=pert,np=np,Temp_M_0_5=Temp_M_0_5,num_trans=num_trans,fec_classes=fec_classes,
                vr_types=vr_types,Temp_betas=Temp_betas,Temp_vr_means=Temp_vr_means,Temp_vr_vars=Temp_vr_vars,vr_types_b=vr_types_b,
                N_start = N_start, Temp_M_0_5_b=Temp_M_0_5_b,Temp_betas_b=Temp_betas_b,Temp_vr_means_b=Temp_vr_means_b,
                Temp_vr_vars_b=Temp_vr_vars_b,bonanza=bonanza,ceiling_bonanza=ceiling_bonanza,
                years_total=years_total,all_nt=all_nt,percent_time_in_bonanza=percent_time_in_bonanza,
                av_bonanza_duration=av_bonanza_duration,tot_num,N_msy,ten_per,N_surv,N_ext)


  # Define a geometric mean
{
  gm <- function(x) exp(mean(log(x)))


  # The final bit is to do a number of runs to get the growth rate and extintsion risks
  # First I initialize some vectors

  Pr_ext <- zeros(tmax,1)
  log_Lambda <- 0
  vr_surv <- 0
  fec <- 0
  fecundities <- zeros(fec_classes,runs)
  bonanza <- zeros(tmax,runs)
  freq_b <- rbinom(runs,tmax,percent_time_in_bonanza)
  N_total <- zeros(tmax,runs)

  # Now the main loop to simulate each stochastic run for the population
  for(x in 1:runs)
  {
    ############  HERE IS THE bonanza set up.. ##################################################################3
    # Here we get the number of years we are in a bonanza in a simulation...
    if(freq_b[x] != 0)
```

```
{
 # The bonanza duration is sampled from a poisson distribution.
 len_b <- rpois(freq_b[x],lambda = av_bonanza_duration)
 # If frequency =1, then make sure the length is at least 1!
 if(freq_b[x] ==1 && len_b ==0) len_b =1

 # This gets the initial length of the bonanza
 bon_lens <- len_b[1]

 # This gets the rest of the bonanza lengths
 for(j in 2:length(len_b))
 {
  if(sum(bon_lens) < freq_b[x]) bon_lens <- c(bon_lens,len_b[j])
  if(sum(bon_lens) >= freq_b[x]) bon_lens <- bon_lens
 }
 # And this puts them into a vector
 bon_lens[length(bon_lens)] <- bon_lens[length(bon_lens)] - (sum(bon_lens) - freq_b[x])
 bon_lens

 # Now initializing for loop to determine which years are in a bonanza
 bonz <- NA
 bonzo <- zeros(length(bon_lens),2)
 ex <- zeros(length(bon_lens),2)
 years <- zeros(tmax,length(bon_lens))
 bono <- NA
 k=1

 for(k in 1:length(bon_lens))
 {
  # Fill up the years matrix for the first bonanza
  if(k==1 && length(bon_lens) >1)
  {
   years[,k] <- 1:tmax
   #samp=1
   samp <- sample(years[,k],size=1,replace=T)
   bonz <- samp:(samp+bon_lens[k]-1)
   #Now, if any of these happen to be greater than tmax we need to truncate them...
   rm <- which(bonz> tmax | bonz < 1)
   if(length(rm) > 0) bonz <- bonz[-rm]
   bonzo[k,] <- c(min(bonz),max(bonz))
   ex[k,] <- c(bonzo[k,1]- bon_lens[k+1]-2,bonzo[k,2]+2)
   if(ex[k,1] <=0) ex[k,1] <- 1
   if(ex[k,2] >tmax) ex[k,2] <- tmax
  }
  # If there is just one bonanza period this is all we need..
  if(length(bon_lens) ==1)
  {
   years[,k] <- 1:tmax
   #samp=1
   samp <- sample(years[,k],size=1,replace=T)
   bonz <- samp:(samp+bon_lens[k]-1)
   #Now, if any of these happen to be greater than tmax we need to truncate them...
   rm <- which(bonz> tmax | bonz < 1)
   if(length(rm) > 0) bonz <- bonz[-rm]
   bonzo[k,] <- c(min(bonz),max(bonz))
  }
  # Now fill up the matrix (years) for the
  if(k > 1 & k < length(bon_lens))
  {
```

```
        #ex <- bonzo[k-1,1] - bon_lens[k]-2
        exc <- seq(ex[k-1,1],ex[k-1,2])

        years[,k] <- c(rep(NA,length(exc)),years[,k-1][-exc])
        #If we are out of years to choose from then no more bonanza's!
        if(length(na.omit(years[,k])) > 0)
        {
          samp <- sample(na.omit(years[,k]),size=1,replace=T)
          bonz <- samp:(samp+bon_lens[k]-1)
          rm <- which(bonz> tmax | bonz < 1)
          if(length(rm) > 0) bonz <- bonz[-rm]
          bonzo[k,] <- c(min(bonz),max(bonz))
          ex[k,] <- c(bonzo[k,1]- bon_lens[k+1]-2,bonzo[k,2]+2)
          if(ex[k,1] <=0) ex[k,1] <- 1
          if(ex[k,2] >tmax) ex[k,2] <- tmax
        }
      }
      # And this takes care of the final bonanza
      if(k == length(bon_lens) && length(bon_lens) >1)
      {
        #ex <- bonzo[k-1,1] - bon_lens[k]-2
        exc <- seq(ex[k-1,1],ex[k-1,2])

        years[,k] <- c(rep(NA,length(exc)),years[,k-1][-exc])
        #If we are out of years to choose from then no more bonanza's!
        if(length(na.omit(years[,k])) > 0)
        {
          samp <- sample(na.omit(years[,k]),size=1,replace=T)
          bonz <- samp:(samp+bon_lens[k]-1)
          rm <- which(bonz> tmax | bonz < 1)
          if(length(rm) > 0) bonz <- bonz[-rm]
          bonzo[k,] <- c(min(bonz),max(bonz))
        }
      }
      bono <- c(bono,bonz)
    } # End k loop...
    bono <- bono[-1]
    bonanza[bono,x] <- 1
  } # End the loop if we have a bonanza... bonanza[,x]
} #End the x loop for the bonanza
#############################################################################################################################
### END the bonanza warm up!
#############################################################################################################################

# Now we have the bonanza years determined for this simulation, we move to the heart of the PVA
for(x in 1:runs) # This loops through each simulation
{
  nt <- n0
  Extinct <- 0
  NMSY = 0
  NSurv = 0
  if(round(x/10) == x/10) print(c(pert,x)) # Tell me every time I finish 10 runs
  for(t in 1:tmax) # This runs the PVA over the number of years specified
  {

    ###########################################################################
    #### Here is the beginning of the bonanza loop, this gets our simulated vital rates for a given year during the bonanza
    ###########################################################################
```

```
if(bonanza[t,x] == 1) # Yep it's a bonanza year...
{

 # Now get the population size and determine if the population size is too large for a bonanza...
 if (t == 1) N_tot <- sum(nt)
 if (N_tot < ceiling_bonanza[x])    # The final crietira is met, we enter a bonanza...
 {
  m <- rnorm(np,0,1)
  raw_elemts <- as.numeric(Temp_M_0_5_b %*% m) # Just remove the complex bits, which should = 0
  for(y in 1:np)
  {
   if(y <= num_trans ) # This gets the survival probabilities!
   {
    if(vr_types_b[y] != 3) # This gets the vital rates for this year, either log-norm or beta dist'n
    {
     index <- round(100*stnormfx(raw_elemts[y]))
     if(index ==0) index=1
     if(index==100) index=99
     vr_surv[y] <- Temp_betas_b[index,y]

    } else # End the if vr_types != 3 loop
     vr_surv[y] <- l.norm(Temp_vr_means_b[y],Temp_vr_vars_b[y],raw_elemts[y])
   } # End the survival calcs...

   if(y > num_trans) # Otherwise I assume this is a fecundity, for these age structured this should be cool___
   {
    if(vr_types_b[y] == 1  || vr_types_b[y] ==2)  # If it is not a log normal do all this
    {
     index <- round(100*stnormfx(raw_elemts[y]))
     if(index == 0) index = 1 # This will round off the index at the extremes
     if(index == 100) index = 99 # ditto..
     fec[y] <- Temp_betas_b[index,y] # find the correct stored betas___

    } # end the if(vr_types != 3 loop)

    if(vr_types_b[y] == 3) fec[y] <- l.norm(Temp_vr_means_b[y],Temp_vr_vars_b[y],raw_elemts[y]); # If it is a log-normal than calculate the log normal rate, duh!
   }  # End the Fecundity calcs___

  }} # end y loop


 if(N_tot > ceiling_bonanza[x]) bonanza[t,x] = 0  # Get out of the bonanza the population size it too big!
}

#########################################################################################################################
#################################### Now we do the same thing for the non-bonanza periods
#########################################################################################################################

if(bonanza[t,x] == 0)
{

 m <- rnorm(np,0,1)
 raw_elemts <- as.numeric(Temp_M_0_5 %*% m) # Just remove the complex bits, which should = 0
 for(y in 1:np)
 {
  if(y <= num_trans ) # This gets the survival probabilities!
  {
   if(vr_types[y] != 3) # This gets the vital rates for this year, either log-norm or beta dist'n
   {
```

```
          index <- round(100*stnormfx(raw_elemts[y]))
          if(index ==0) index=1
          if(index==100) index=99
          vr_surv[y] <- Temp_betas[index,y]

        } else # End the if vr_types != 3 loop
          vr_surv[y] <- l.norm(Temp_vr_means[y],Temp_vr_vars[y],raw_elemts[y])
      } # End the survival calcs___
      if(y > num_trans) # Otherwise I assume this is a fecundity, for these age structured this should be cool___
      {
       if(vr_types[y] == 1  || vr_types[y] ==2)  # If it is not a log normal do all this shite2
       {
         index <- round(100*stnormfx(raw_elemts[y]))
         if(index == 0) index = 1 # This will round off the index at the extremes
         if(index == 100) index = 99 # ditto___

         fec[y] <- Temp_betas[index,y] # find the correct stored betas___

        } # end the if(vr_types != 3 loop)

        if(vr_types[y] == 3) fec[y] <- l.norm(Temp_vr_means[y],Temp_vr_vars[y],raw_elemts[y]); # If it is a log-normal than calculate the log normal rate, duh!
      }  # End the Fecundity calcs___

    } # end y loop
  } # End the NOT IN A BONANZA if statement!


  # Now update the matrix with the new vital rates!
  fec2 <- as.numeric(na.omit(fec[fec >0]))
  fecundities[,x] <- fec2

  new_matrix <- mean.matrix(vr_surv,fec2)




  nt <- new_matrix %*% nt    # And this updates the populatin size at each time step_
  all_nt[,t,x] <- nt         # This is a gigantic 3-D matrix with the entire age structure in each year and each run, you'll need a boat load of memory to store this!
  N_total[t,x] <- sum(nt)    # This is the total population size in each year in each simulation
  N_tot <- sum(nt)



  log_Lambda <- log_Lambda + (1/tmax)* log(sum(nt)/N_start) # Calculate log_lambda, N_start is the initial number of indivduals___
} # end the for x loop....



############################  Probability of Persistance ############################################################  Probability of Persistance ############################
# So here is the Probability of your abundance being above the persistance target in a given year
MSY <- NA
Surv <- NA
Ext <- NA
for(t in 1:tmax)
{

  if(t==1) MSY[t] <- length(which(N_total[1:t,] >= N_msy)) # Are we above Persistance target after 1 year?
  if(t > 1) MSY[t] <- length(which(apply(N_total[1:t,],2,gm) >= N_msy)) # Are we above it in subsequent years, using geometric mean
```

```
  ############################  End persistance ############################################################  End persistance ############################

  ############################  Probability of Survival (not used) ############################################################  Probability of Survival (not used) ############################
  # Now what is probability the population doesn't really decline any further...
  if(t==1) Surv[t] <- length(which(N_total[1:t,] >= N_surv))
  if(t > 1) Surv[t] <- length(which(apply(N_total[1:t,],2,gm) >= N_surv))# Are we above it in subsequent years, using geometric mean

  ############################  End Survival ############################################################  End Survival ############################

  ############################  Probability of Collapse ############################################################  Probability of Collapse ############################
  # Now what is probability the population collapses

  if(t==1) Ext[t] <- length(which(N_total[1:t,] <= N_ext))
  if(t > 1) Ext[t] <- length(which(apply(N_total[1:t,],2,min) <= N_ext)) # Has the population collapsed?

  ############################  End Collapse ############################################################  End Collapse ############################


 }


 # Here is the data to return from the model for subsequent analysis.
 db <- list(log_Lambda,bonanza,all_nt,MSY,Surv,Ext) # Pr_ext, This was in spot #2 before...
 return(db)
 #return(log_Lambda,Pr_ext,bonanza,all_nt)
} # end the step5_real function!


################  ################  ################  ################  ################  ################  ################
################  END PVA Sens/Elasticity Program  PVA Sens/Elasticity Program  PVA Sens/Elasticity Program PVA Sens/Elasticity Program
################  ################  ################  ################  ################  ################  ################
```