Mathematical Modeling Assignment 3

Alyssa Obermayer

10/3/2020

Question 1: Loggerhead Turtle Polulation Dynamics

Part A

Part A of question one takes the Leslie matrix provided to find the stable growth rate of the population of sea turtles presented. To find this growth rate, I generated a matrix and found the eigenvalues and vectors, then took the absolute value of the largest eigenvalue that RStudio computed, which is shown below as R.

```r
#Generating the Leslie Matrix
A <- matrix(c(0,0.675,0,0,0,0,0.703,0.047,0,0,0,0,0.657,0.019,0,
              4.665,0,0,0.682,0.061,61.896,0,0,0,0.8091), nrow = 5, ncol = 5)

#Find the eigenvalue and vector of the matrix (A)
EVs <- eigen(A)

#Assign largest eigenvalue to big R (Geometric Growth Factor>Stable population growth rate)
R <- abs(eigen(A)$values[1])
R
```

```
## [1] 0.9515817
```

Part B

For Part B of question 1 we want to adjust two different parameters separately to see their affect on growth rate of the sea turtle population.

The first parameter we are adjusting is increasing the survival rate of the juvenile sea turtles to 100%. To do this I took the P value located in the original matrix, 0.675 and adjusted it to be 1.000. This signifies that there is a 100% probability that the juvenile sea turtles will grow to the second stage of their life.

The next influence I made on the original matrix was decreasing the large mature clutch size by half. This was done by taking the original clutch size 61.896 and decreasing it by half.

After these changes were made I generated two new matrices and solved for their new eigenvalues and vectors to find a new growth rate, the largest eigenvalue, for each possibility. These eigenvalues were compared to the original eigenvalues by taking their difference to find which matrix change influence the growth rate the most. In the end it was found that reduction in clutch size by half was show to make the largest difference by decreasing the R value by 0.0293956.

```r
#Juvenile Change
#Generate Leslie Matrix with increased survival of juvenile stage to 100%
AJ <- matrix(c(0, 1.000,0,0,0,0,0.703,0.047,0,0,0,0,0.657,0.019,0,
               4.665,0,0,0.682,0.061,61.896,0,0,0,0.8091),
             nrow = 5, ncol = 5)

#Find the eigenvalue and vector of the matrix (AJ)
EVsJ <- eigen(AJ)

#Assign largest eigenvalue to big R (Geometric Growth Factor>Stable population growth rate)
RJ <- abs(eigen(AJ)$values[1])
RJ
```

```
## [1] 0.9744496
```

```r
#Clutch size change
#Decrease clutch size for large mature size class by half
MCS <- (A[1,5]/2)

#Generate Leslie Matrix with large mature size class by half
AC <- matrix(c(0, 0.675,0,0,0,0,0.703,0.047,0,0,0,0,0.657,0.019,0,
               4.665,0,0,0.682,0.061,30.948,0,0,0,0.8091),
             nrow = 5, ncol = 5)

#Find the eigenvalue and vector of the matrix (AC)
EVsC <- eigen(AC)

#Assign largest eigenvalue to big R (Geometric Growth Factor>Stable population growth rate)
RC <- abs(eigen(AC)$values[1])
RC
```

```
## [1] 0.9221861
```

```r
#Find change in new R values from original
ChangeWithRJ <- eval(abs(R-RJ))
ChangeWithRJ
```

```
## [1] 0.02286786
```

```r
ChangeWithRC <- eval(abs(R-RC))
ChangeWithRC
```

```
## [1] 0.0293956
```

Part C

In Part C we are attempting to achieve positive growth by way of decreasing mortality in stages 4 and 5, or can also be looked at as increasing survivability in stages 4 and 5. With the original eigenvalue, 0.9515817, being lower than 1, it shows that the population is decreasing over time to its eventual extinction. It is our goal to adjust the surviability of stages 4 and 5 just enough to produce a positive growth rate, some number greater than 1.

To do this, I generated a "while" loop that takes the original matrix and calculates the R value continuously while extracting the P values of 4 and 5 and incrementally increasing them by 0.01 and then putting them back into the matrix until the R value exceeds 1, and then it stops. Once the R value is greater that 1 I pulled the new, positive growth rate, eigenvalue and presented the new matrix with the adjusted stage 4 and 5 P values. This can be done in much smaller sequential increments if chosen, but my "while" loop ended up increasing the P values by 0.09. This shows that the mortality needs to be reduced by 9% in stage 4 and 5 P values to achieve a positive growth rate.

```
#While loop to find where the largest eigenvalue exceeds 1
AcM <- matrix(c(0,0.675,0,0,0,0,0.703,0.047,0,0,0,0,0.657,0.019,0,
                4.665,0,0,0.682,0.061,61.896,0,0,0,0.8091), nrow = 5, ncol = 5)
while (abs(eigen(AcM)$values[1]) < 1) {
  AcM[4,4] = (AcM[4,4] + 0.01)
  AcM[5,5] = (AcM[5,5] + 0.01)
}

#Find largest eigenvalue and view new (R>1) matrix
AcMeigen <- eigen(AcM)$values[1]
AcMeigen
```

```
## [1] 1.004368+0i
```

```
print(AcM)
```

```
##       [,1]  [,2]  [,3]  [,4]    [,5]
## [1,] 0.000 0.000 0.000 4.665 61.8960
## [2,] 0.675 0.703 0.000 0.000  0.0000
## [3,] 0.000 0.047 0.657 0.000  0.0000
## [4,] 0.000 0.000 0.019 0.772  0.0000
## [5,] 0.000 0.000 0.000 0.061  0.8991
```

```
#Find difference between probability of survival in stages 4 & 5
#of the original matrix and new (R>1) matrix
diff44 <- abs(A[4,4]-AcM[4,4])
diff55 <- abs(A[5,5]-AcM[5,5])
print(c(diff44, diff55))
```

```
## [1] 0.09 0.09
```

Part D

Lastly, we want to project the population over 50 years with the new projection matrix to see if the overall population will double with the adjustment made to stages 4 and 5. To do this I defined my variables to be used, created a storage matrix to place each years population projection for each stage, and then ran a "for" loop to continually multiply the new population by the projection matrix for each year over 50 years. I ran the storage matrix to see the population for all stages over time in numbers. Then I took each stage in that storage matrix and graphed it over the 50 years to see the tend. With the overall starting population being 1250 individuals and the overall population after 50 years projected to be just over 8900 individuals, it can be said that the 9% reduction in mortality would more than double the population over 50 years.

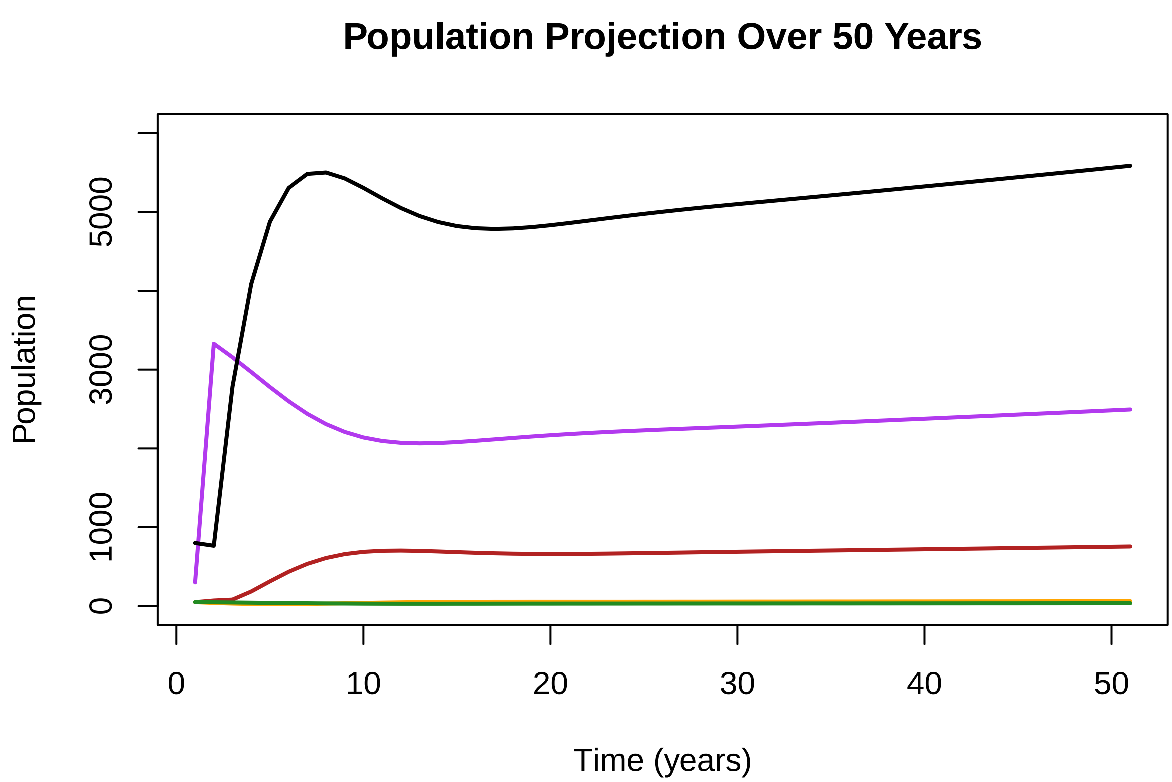
Additionally, I decided to try out a package "popbio" that has numerous function to analyze population data. I was able to call the "pop.projection" function with the given matrix, starting population, and projection years, and found similar result to the function I ran and plotted.

```r
pMat <- AcM                 #Projection matrix to be used
pYears <- 50                #Number of years we are projecting over
n0 <- c(300,800,50,50,50) #Define initial population which was given in the text

StorYears <- matrix(0, nrow = nrow(pMat), ncol = (pYears + 1)) #Storage array for abundances
StorYears[,1] <- n0                                  #Setting initial abundance at year zero
for (t in 2:(pYears + 1)) {                          #Create loop to project 50 years
  StorYears[,t] <- pMat %*% StorYears[,t-1] #Multiplies the projection matrix by the previous years pop
}

StorYears
```

```
##      [,1]     [,2]       [,3]       [,4]       [,5]       [,6]       [,7]
## [1,]  300 3328.050 3155.81823 2969.51766 2778.61972 2597.92687 2439.06464
## [2,]  800  764.900 2784.15845 4087.44070 4877.89523 5304.72865 5482.82488
## [3,]   50   70.450   82.23595  184.88447  313.57881  435.28235  535.30275
## [4,]   50   39.550   31.87115   26.16701   23.71374   24.26500   27.00295
## [5,]   50   48.005   45.57385   42.91958   40.18519   37.57704   35.26568
##            [,8]       [,9]      [,10]      [,11]      [,12]      [,13]
## [1,] 2308.77333 2209.20787 2139.03074 2094.66637 2071.41534 2064.31128
## [2,] 5500.79452 5425.48054 5305.32813 5173.49143 5050.86428 4948.96294
## [3,]  609.38668  658.90439  687.89777  701.29926  703.90771  699.85799
## [4,]   31.01703   35.52349   39.94332   43.90630   47.22035   49.82836
## [5,]   33.35455   31.88112   30.83125   30.15692   29.79237   29.66676
##           [,14]      [,15]      [,16]      [,17]      [,18]      [,19]
## [1,] 2068.70297 2080.59318 2096.78650 2114.90111 2133.29438 2150.94234
## [2,] 4872.53106 4821.76384 4794.10038 4785.58345 4791.82342 4808.62557
## [3,]  692.40795  683.92099  675.95899  669.42777  664.73647  661.94756
## [4,]   51.76479   53.11817   54.00173   54.53255   54.81826   54.94969
## [5,]   29.71291   29.87253   30.09860   30.35576   30.61935   30.87377
##           [,20]      [,21]      [,22]      [,23]      [,24]      [,25]
## [1,] 2167.30314 2182.18425 2195.62572 2207.80512 2218.96543 2229.36438
## [2,] 4832.34986 4860.07157 4889.60468 4919.43946 4948.63440 4976.69165
## [3,]  660.90495  661.33499  662.92046  665.35016  668.34871  671.69092
## [4,]   54.99816   55.01578   55.03754   55.08447   55.16687   55.28745
## [5,]   31.11054   31.32637   31.52150   31.69827   31.86007   32.01057
##           [,26]      [,27]      [,28]      [,29]      [,30]      [,31]
## [1,] 2239.24209 2248.80315 2258.20939 2267.57990 2276.99542 2286.50503
## [2,] 5003.43519 5028.90335 5053.26118 5076.73395 5099.56040 5121.96287
## [3,]  675.20544  678.77143  682.31129  685.78179  689.16513  692.46083
## [4,]   55.44404   55.63170   55.84433   56.07574   56.32032   56.57343
## [5,]   32.15324   32.29106   32.42643   32.56110   32.69631   32.83279
##           [,32]      [,33]      [,34]      [,35]      [,36]      [,37]
## [1,] 2296.13346 2305.88805 2315.76485 2325.75340 2335.84041 2346.01228
## [2,] 5144.13079 5166.21403 5188.32290 5210.53227 5232.88773 5255.41235
## [3,]  695.67902  698.83526  701.94683  705.03024  708.09989  711.16735
## [4,]   56.83144   57.09177   57.35272   57.61329   57.87303   58.13188
## [5,]   32.97094   33.11089   33.25260   33.39593   33.54069   33.68669
##           [,38]      [,39]      [,40]      [,41]      [,42]      [,43]
## [1,] 2356.25657 2366.56294 2376.92336 2387.33207 2397.78533 2408.28101
## [2,] 5278.11317 5300.98674 5324.02367 5347.21190 5370.53912 5393.99410
## [3,]  714.24133  717.32787  720.43079  723.55214  726.69272  729.85245
## [4,]   58.38999   58.64766   58.90522   59.16302   59.42134   59.68044
## [5,]   33.83375   33.98171   34.13046   34.27992   34.43002   34.58073
```

```
##            [,44]      [,45]      [,46]      [,47]      [,48]      [,49]
## [1,] 2418.81821 2429.39690 2440.01762 2450.68116 2461.38849 2472.14054
## [2,] 5417.56753 5441.25227 5465.04325 5488.93730 5512.93270 5537.02892
## [3,]  733.03078  736.22690  739.43993  742.66907  745.91363  749.17309
## [4,]   59.94049   60.20165   60.46398   60.72755   60.99238   61.25848
## [5,]   34.73204   34.88395   35.03646   35.18958   35.34334   35.49773
##            [,50]      [,51]
## [1,] 2482.93817 2493.78214
## [2,] 5561.22620 5585.52528
## [3,]  752.44708  755.73536
## [4,]   61.52583   61.79444
## [5,]   35.65277   35.80849
```

```r
#Plot population projection over 50 years
plot(StorYears[1,], ylim = c(0, 6000), type = "l", lwd = 2, ylab = "Population",
     xlab = "Time (years)", main = "Population Projection Over 50 Years",
     col = "darkorchid2")
lines(StorYears[2,], lwd = 2, ylab = "Population", xlab = "Time (years)",
      main = "Population Projection Over 50 Years", col = "black")
lines(StorYears[3,], lwd = 2, ylab = "Population", xlab = "Time (years)",
      main = "Population Projection Over 50 Years", col = "firebrick")
lines(StorYears[4,], lwd = 2, ylab = "Population", xlab = "Time (years)",
      main = "Population Projection Over 50 Years", col = "orange")
lines(StorYears[5,], lwd = 2, ylab = "Population", xlab = "Time (years)",
      main = "Population Projection Over 50 Years", col = "forestgreen")
```



```r
#Compare found numbers to "PopBio" R package
library(popbio)
pop.projection(AcM, c(300,800,50,50,50), 51)
```

```
## $lambda
## [1] 1.004369
##
## $stable.stage
## [1] 0.279176206 0.625293498 0.084603754 0.006917820 0.004008721
##
## $stage.vectors
##        0        1          2          3          4          5          6
## [1,] 300 3328.050 3155.81823 2969.51766 2778.61972 2597.92687 2439.06464
## [2,] 800  764.900 2784.15845 4087.44070 4877.89523 5304.72865 5482.82488
## [3,]  50   70.450   82.23595  184.88447  313.57881  435.28235  535.30275
## [4,]  50   39.550   31.87115   26.16701   23.71374   24.26500   27.00295
## [5,]  50   48.005   45.57385   42.91958   40.18519   37.57704   35.26568
##               7          8          9         10         11         12
## [1,] 2308.77333 2209.20787 2139.03074 2094.66637 2071.41534 2064.31128
## [2,] 5500.79452 5425.48054 5305.32813 5173.49143 5050.86428 4948.96294
## [3,]  609.38668  658.90439  687.89777  701.29926  703.90771  699.85799
## [4,]   31.01703   35.52349   39.94332   43.90630   47.22035   49.82836
## [5,]   33.35455   31.88112   30.83125   30.15692   29.79237   29.66676
##              13         14         15         16         17         18
## [1,] 2068.70297 2080.59318 2096.78650 2114.90111 2133.29438 2150.94234
## [2,] 4872.53106 4821.76384 4794.10038 4785.58345 4791.82342 4808.62557
## [3,]  692.40795  683.92099  675.95899  669.42777  664.73647  661.94756
## [4,]   51.76479   53.11817   54.00173   54.53255   54.81826   54.94969
## [5,]   29.71291   29.87253   30.09860   30.35576   30.61935   30.87377
##              19         20         21         22         23         24
## [1,] 2167.30314 2182.18425 2195.62572 2207.80512 2218.96543 2229.36438
## [2,] 4832.34986 4860.07157 4889.60468 4919.43946 4948.63440 4976.69165
## [3,]  660.90495  661.33499  662.92046  665.35016  668.34871  671.69092
## [4,]   54.99816   55.01578   55.03754   55.08447   55.16687   55.28745
## [5,]   31.11054   31.32637   31.52150   31.69827   31.86007   32.01057
##              25         26         27         28         29         30
## [1,] 2239.24209 2248.80315 2258.20939 2267.57990 2276.99542 2286.50503
## [2,] 5003.43519 5028.90335 5053.26118 5076.73395 5099.56040 5121.96287
## [3,]  675.20544  678.77143  682.31129  685.78179  689.16513  692.46083
## [4,]   55.44404   55.63170   55.84433   56.07574   56.32032   56.57343
## [5,]   32.15324   32.29106   32.42643   32.56110   32.69631   32.83279
##              31         32         33         34         35         36
## [1,] 2296.13346 2305.88805 2315.76485 2325.75340 2335.84041 2346.01228
## [2,] 5144.13079 5166.21403 5188.32290 5210.53227 5232.88773 5255.41235
## [3,]  695.67902  698.83526  701.94683  705.03024  708.09989  711.16735
## [4,]   56.83144   57.09177   57.35272   57.61329   57.87303   58.13188
## [5,]   32.97094   33.11089   33.25260   33.39593   33.54069   33.68669
##              37         38         39         40         41         42
## [1,] 2356.25657 2366.56294 2376.92336 2387.33207 2397.78533 2408.28101
## [2,] 5278.11317 5300.98674 5324.02367 5347.21190 5370.53912 5393.99410
## [3,]  714.24133  717.32787  720.43079  723.55214  726.69272  729.85245
## [4,]   58.38999   58.64766   58.90522   59.16302   59.42134   59.68044
## [5,]   33.83375   33.98171   34.13046   34.27992   34.43002   34.58073
```

```
##              43          44          45          46          47          48
## [1,] 2418.81821 2429.39690 2440.01762 2450.68116 2461.38849 2472.14054
## [2,] 5417.56753 5441.25227 5465.04325 5488.93730 5512.93270 5537.02892
## [3,]  733.03078  736.22690  739.43993  742.66907  745.91363  749.17309
## [4,]   59.94049   60.20165   60.46398   60.72755   60.99238   61.25848
## [5,]   34.73204   34.88395   35.03646   35.18958   35.34334   35.49773
##              49          50
## [1,] 2482.93817 2493.78214
## [2,] 5561.22620 5585.52528
## [3,]  752.44708  755.73536
## [4,]   61.52583   61.79444
## [5,]   35.65277   35.80849
##
## $pop.sizes
##  [1] 1250.000 4250.955 6099.658 7310.929 8033.993 8399.780 8519.461 8483.326
##  [9] 8360.997 8203.031 8043.520 7903.200 7792.627 7715.120 7669.269 7650.946
## [17] 7654.801 7675.292 7707.339 7746.667 7789.933 7834.710 7879.377 7922.975
## [25] 7965.045 8005.480 8044.401 8082.053 8118.732 8154.738 8190.335 8225.746
## [33] 8261.140 8296.640 8332.325 8368.242 8404.411 8440.835 8477.507 8514.414
## [41] 8551.539 8588.869 8626.389 8664.089 8701.962 8740.001 8778.205 8816.571
## [49] 8855.099 8893.790 8932.646
##
## $pop.changes
##  [1] 3.4007640 1.4348911 1.1985803 1.0989017 1.0455299 1.0142481 0.9957586
##  [8] 0.9855801 0.9811068 0.9805546 0.9825549 0.9860091 0.9900537 0.9940570
## [15] 0.9976109 1.0005038 1.0026769 1.0041754 1.0051026 1.0055852 1.0057481
## [22] 1.0057012 1.0055332 1.0053098 1.0050766 1.0048618 1.0046805 1.0045384
## [29] 1.0044348 1.0043652 1.0043235 1.0043029 1.0042972 1.0043012 1.0043105
## [36] 1.0043221 1.0043339 1.0043446 1.0043535 1.0043603 1.0043652 1.0043685
## [43] 1.0043703 1.0043712 1.0043714 1.0043711 1.0043706 1.0043700 1.0043694
## [50] 1.0043689
```

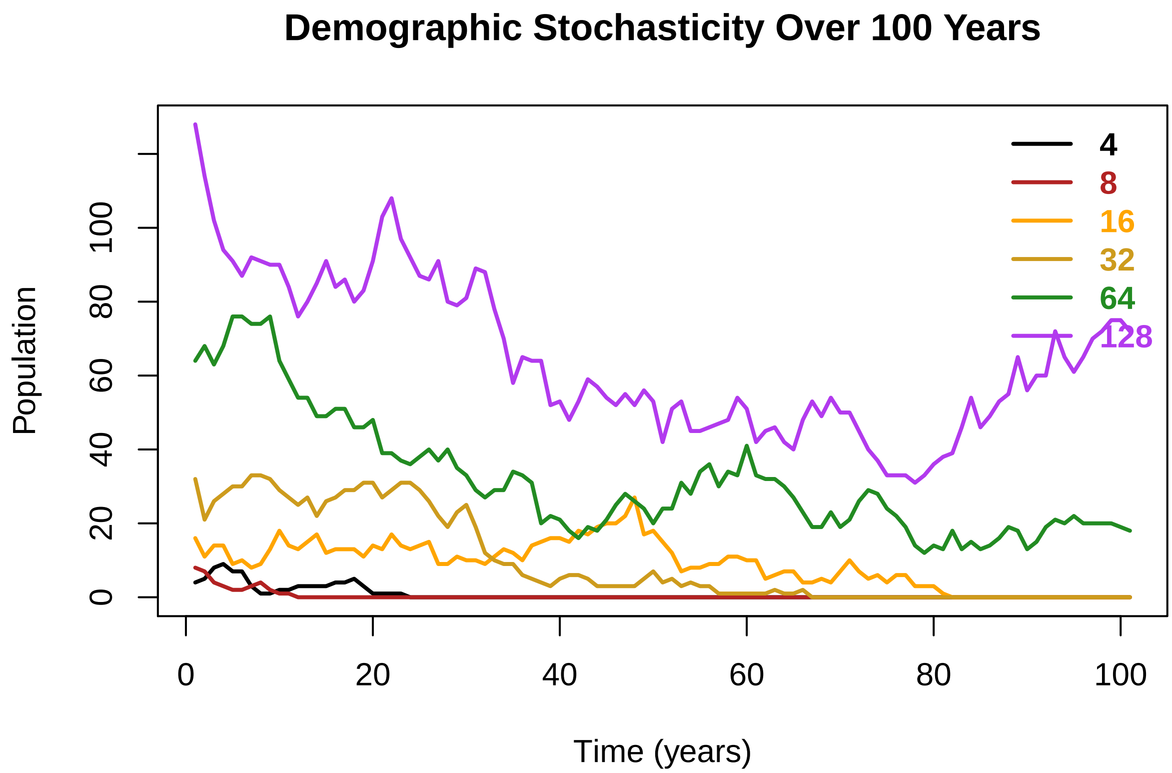
Question 2: Minimum Viable Population Size

Question 2 set out to determine the minimum viable population size that an species can start with in an unchanging environment. To find this minimum population we applied a demographic stochasticity model to different starting populations there the function simulates random variation within the population to model the total population over 100 years. Due to this model being randomly generated every time it is calculated I cannot always confidently say which starting population or which set of starting populations will persist over 100 years. Although, throughout the numerous times I have run this simulation via checking code and knitting the starting population size of 128 individuals has been the only one I have not seen go extinct within 100 years yet.

To be transparent, I do wish I had been able to find more resources or ask for assistance in calculating the probability within the function. As the model is on sea turtles, a birthing individual will most likely have more than 1 or 2 offspring per year. If I could have, and maybe in the future, I will look more towards calculating probability of an individual dying or not each year to get a better fit model to the sea turtle population.

```
#Demographic Stochasticity Function
demogstoch <- function(p0,p1,n0,times) {
  n = n0
  for (i in 1:times) {
    new_n = 0
```

```r
    if (n[i] == 0) {
      n <- c(n, new_n) #Makes sure population cannot grow from zero
    } else {
      for (j in 1:n[i]) {
        luck = runif(1)
        if (luck < p0) {
          new_n = new_n
        } else {
          if (luck < p0 + p1) {
            new_n = new_n + 1
          } else {
            new_n = new_n + 2}
        }
      }
      n <- c(n, new_n)
    }
  }
  return(n)
}

#Run function for different starting population
n0.4 <- demogstoch(0.25,0.5,4,100)
n0.8 <- demogstoch(0.25,0.5,8,100)
n0.16 <- demogstoch(0.25,0.5,16,100)
n0.32 <- demogstoch(0.25,0.5,32,100)
n0.64 <- demogstoch(0.25,0.5,64,100)
n0.128 <- demogstoch(0.25,0.5,128,100)

#Plot all models
plot(n0.128,ylim = c(0,max(n0.128)),type = "l",ylab = "Population",
     xlab = "Time (years)",main = "Demographic Stochasticity Over 100 Years",
     lwd = 2,col = "darkorchid2")
lines(n0.4,ylab = "Population",xlab = "Time (years)",lwd = 2,col = "black")
lines(n0.8,ylab = "Population",xlab = "Time (years)",lwd = 2,col = "firebrick")
lines(n0.16,ylab = "Population",xlab = "Time (years)",lwd = 2,col = "orange")
lines(n0.32,ylab = "Population",xlab = "Time (years)",lwd = 2,col = "goldenrod3")
lines(n0.64,ylab = "Population",xlab = "Time (years)",lwd = 2,col = "forestgreen")
legend("topright",legend=c("4","8","16","32","64","128"),
       text.col = c("black","firebrick","orange","goldenrod3","forestgreen","darkorchid2"),
       lty = 1,col = c("black","firebrick","orange","goldenrod3","forestgreen","darkorchid2"),
       box.lty = 0,lwd = 2, bg = "transparent", text.font = 2)
```




```
#Generate data frames to see exact numerical variation from the graph
n0.4df <- data.frame(n0.4)
n0.8df <- data.frame(n0.8)
n0.16df <- data.frame(n0.16)
n0.32df <- data.frame(n0.32)
n0.64df <- data.frame(n0.64)
n0.128df <- data.frame(n0.128)
```

Question 3: Logistic Growth with Environmental Variation

The final question looks at logistic growth of the sea turtle population over discrete time which takes into account the resources available for the population, density, and other variables that can create conditions for a good, bad, or neutral year.

To start, I generated a basic logistic model with the original R value of 0.9515817, a carrying capacity of 1000, and a starting population of 2 individuals, and I simulated this over 20 years. Following this, I added possible environmental variation to the logistic model by creating "if/else" statements which would generate random numbers and if those numbers were within certain ranges it would assign them different R values for the specific range they were in. This environmental variation model was run similarly but I gave the function three different R values to choose from; rbad, rgood, and rneut, who's values were (4/5),(5/4),1 respectively.This simulation had a starting population size of 2 and a carrying capacity of 1000 and was also run over 20 years.

Like the demographic stochacticity model, the environmental variation model (seen in the cyan color on the graph) is run with random numbers and the exact output can vary each time. Overall, it has a very similar form to the basic logistic model, but the variations that are made with the random uniform number generator do not show a trend in constantly being a lower or higher logistic growth rate. This just shows that it can be very difficult to model simulation with so many variables, especially with a variable dealing with mother nature.

```
#Basic Logistic Model

#Basic Logistic Model Function
logistic <- function(r, k, n0, time){
  n <- n0
    for (t in 1:time){
      nprime <- n[t] + r*n[t]*(k-n[t])/k
      if (nprime < 0) nprime = 0 #Ensures the population cannot bounce back from zero
      n <- c(n, nprime)
    }
  return(n)
}

#Basic Logistic Model Function Values
logR <- logistic(R,1000,2,20)

#Logistic Model with Environmental Variation

#Logistic Model with Environmental Variation Function
logistic <- function(rbad, rgood, rneut, k, n0, time){
  n <- n0
    for (t in 1:time){
      if (runif(1) < 0.33) { #Random switch between bad, neutral, and good conditions
        r = rbad
      } else {
        if (0.33 > runif(1) && runif(1) < 0.66) {
          r = rneut
        } else {
          if (runif(1) > 0.66) {
            r = rgood
          }
        }
      }
      nprime <- n[t] + r*n[t]*(k-n[t])/k
      if (nprime < 0) nprime = 0
      n <- c(n, nprime) #Ensures the population cannot bounce back from zero
    }
  return(n)
}

#Logistic Model with Environmental Variation Function Values
envrlog <- logistic((4/5),(5/4),1,1000,2,20)

#Plotting Basic Logistics Model with and without Environmental Variation
plot(logR,
     type = "l",
     lwd = 2,
     ylab = "Population",
     xlab = "Time (years)",
     main = "Basic Logistic Model")
```

```r
lines(envrlog,
     type = "l",
     lwd = 2,
     ylab = "Population",
     xlab = "Time (years)",
     main = "Logistic Model with Environmental Variation",
     col = "cyan")
legend(12, 200,
       legend = c("Basic","Environmental Variant"),
       text.col = c("black","cyan"),
       lty = 1,
       lwd = 2,
       col = c("black","cyan"))
```