# Exploring Wildlife Simulation Models

by Marcus R. Szwankowski

A Thesis
Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Mathematics

Northern Arizona University
May, 2005

Approved:

---

Terence R. Blows, Ph.D., Chair

---

Paul Beier, Ph.D.

---

Brent D. Burch, Ph.D.

# Abstract

## Exploring Wildlife Simulation Models

## Marcus R. Szwankowski

We introduce a brief description of wildlife population simulation modeling. This serves as a primer for those who are unfamiliar with the topic. We then describe, in detail, a simulation model used to answer questions about modeling techniques. This model considers each population member separately when determining the yearly population fluctuations of a species and allows for dynamic density dependence effects.

The first question we answer is whether or not the inclusion of males in the simulation effects the output. We compare a female only model, which uses just the female population size to determine the number of breeders, to a model which considers male and female population sizes before determining the number of breeders during a given time step. We find that female only models provide output which is considerably higher than the male and female model. We then discuss the ramifications of using a female only model with respect to three specific papers which use that type.

The second question deals with the introduction of density dependence effects and which vital rate, birth or survival, is better at creating the desired curve for a *time vs. average population* graph. We want a curve which mimics a logistic equation's curve. Both survival and birth rate driven models produced graphs which behave similar to the logistic equation's curve, although the survival model performed marginally better in our trials. We also explore the effect of introducing density dependence effects at a lower intensity before the true carrying capacity is reached. This produces results in which the carrying capacity is never obtained by the average population size.

The third and final question explores the use of a uniform distribution based catastrophe timing generator. We understand that a Poisson based distribution should be better at predicting the catastrophe occurrence. We discover that the geometric and exponential distributions are linked to the uniform and Poisson distributions, respectively, and see that they make a better comparison. This is due to their ability to determine whether a catastrophe will occur in the next time step during the simulation. A comparison of the means and variances of the distributions proves that little practical difference exists between the two.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Overview of Wildlife Simulation Models

Wildlife Simulation Models are used extensively in wildlife population management studies. A variety of functions, techniques and species specific algorithms can be used to form the simulation models, which are implemented on a computer based platform to help gain some understanding of a population's possible future growth. There are two main underlying approaches to models that simulate population growth: deterministic and stochastic.

In a deterministic model the demographic, environmental and spatial data values that are used to initiate the model are held constant throughout the simulation process. The result of these fixed values is that subsequent runs of the model always yields the same output. This approach is useful when erratic yearly changes in the environment or spatial make-up are not present. Deterministic models have an advantage in that they take less time to develop, less computing power to run and, subsequently, less time to return their output. In many cases, however, unpredictable yearly fluctuations in the environment and spatial make-up do occur and a deterministic model's output is not representative enough of possible trends.

A stochastic model allows the demographic, environmental and/or spatial data values to vary randomly throughout the simulation process. Typically, in a stochastic model, a range of observed values are entered (demographic, environmental and spatial data values). As the stochastic model runs, yearly random changes of the values are allowed to occur. This approach does introduce the inherent problem of multiple possible outcomes, so large numbers of simulations are run in order to get a clearer picture of the distribution of the most likely outcomes. Stochastic simulation models require more time and computing power to implement and run, but their end results are preferred when fluctuations in demographic, environmental and/or spatial data are observed.

Important measures of simulation models include the ending population size and the extinction risk.

- Ending population size is the number of subjects present at the end of one multi-year simulation (it should be noted that even though simulations can give the expected population size for a future hundreds of years from now, extrapolating more than fifty years is not recommended due to the amount of predictive error that can occur). Running multiple simulations in a stochastic model allows the average and variance of the ending population size to be found and analyzed.

- The extinction risk can also be found when multiple simulations of a stochastic model are run. Anytime the population goes to zero during a simulation, it is counted as an extinction. Dividing the number of extinctions by the total number of simulations will give the extinction risk for the species in question.

These outcomes of importance are then analyzed in order to help make decisions about wildlife population management. One such analysis is known as Population Viability Analysis (PVA). Lacy writes that PVA's are the estimation of extinction probabilities by analysis that incorporate identifiable threats to population survival into models of the extinction process [5].

In both the deterministic and stochastic models, there are choices for the complexity of the functions, techniques and algorithms used to model yearly growth. The following sections give a brief overview of some of the possibilities, beginning with the simplest type.

### 1.1.1 The Headcount Approach

The simplest type of model only keeps track of an overall population total. It is basically just a function that uses the empirically derived yearly growth rate along with an initial population setting as input. The function's output is the next year's population, which in turn is used as the input for the year after that. This process is then continued for a pre-determined amount of iterations.

The functions used are chosen for the shape that their *population at time t vs. population at time t+1* and *time vs. population* graphs produce, given the value of other present parameters, and for how well they represent the species in question. Note that many variations and types of these functions exist and that only the basic shapes will be presented here. For further information on wildlife simulation functions and their graphs, see "An Illustrative Guide to Theoretical Ecology" by Case [3].

The following graphs illustrate some of the shapes for both *population at time t vs. population at time t+1* and *time vs. population* graphs. The *population at time t vs. population at time t+1* graphs are showing the input of the model. That is, they show how the population changes with respect to itself from year to year, and the numerical representations of these changes are used to parameterize the model. Whereas, the *time vs. population* graphs show the output of the model which results from the *population at time t vs. population at time t+1* input parameters that were used.

Figure 1.1 shows the *population at time t vs. population at time t+1* graph for uninhibited geometric growth, using $N_{t+1} = (R+1)N_t$, where $R$ is the yearly growth rate.

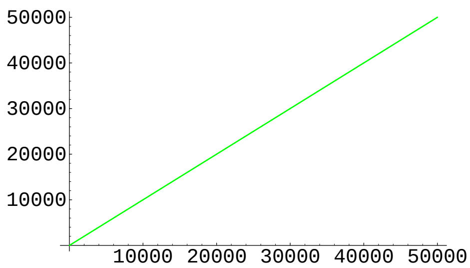


Figure 1.1: $R = 0.00$

And, figure 1.2 shows the related *time vs. population* output, or uninhibited exponential growth, using the equation $N_t = N_0 e^{rt}$ where $r$ is the rate of growth, $t$ is the time of growth, $N_0$ is the population size at time zero, and $N_t$ is the population at time $t$. It should be noted that $r$ was set to 1.00 for the figure, but all $r > 0$ produce roughly the same result, and most species in fact have $r < 0.10$.

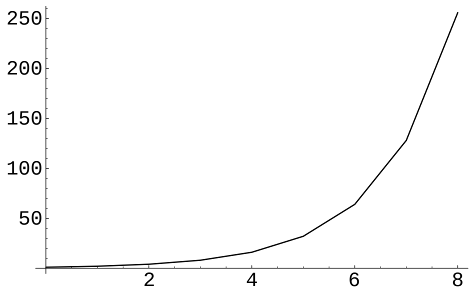


Figure 1.2: $r = 1.00$

Figure 1.3 shows the *population at time t vs. population at time t+1* graph for density dependent growth, using Beverton Holt equation $N_{t+1} = \frac{N_t(R+1)}{1+\frac{R}{K}N_t}$, where the new variable $K$ is the carrying capacity. Again, this is a visual representation of the input parameters that would be used to initiate the model.

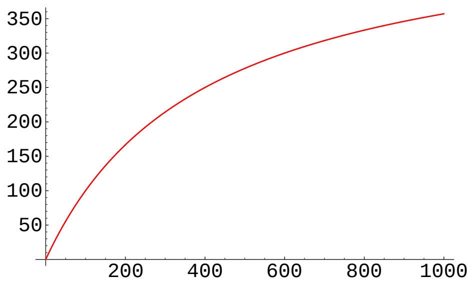


Figure 1.3: $R = 0.25, K = 250$

Figure 1.4 shows the graph for the *time vs. population* output, using the logistic equation, $N_t = \frac{K}{1+(\frac{K-N_0}{N_0})e^{-rt}}$, which results from figure 1.3's input.

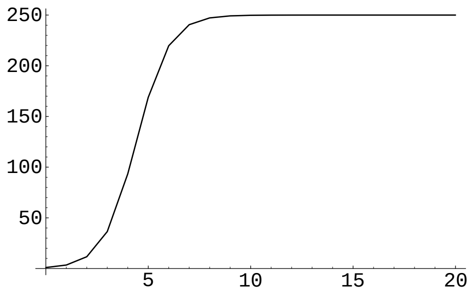


Figure 1.4: $r = 1.25, K = 250$

Figures 1.5-1.7 show the *population at time t vs. population at time t+1* input graphs for three parameter variations of density dependent growth, using the Ricker equation $N_{t+1} = N_t e^{(r(1-\frac{N_t}{K}))}$.

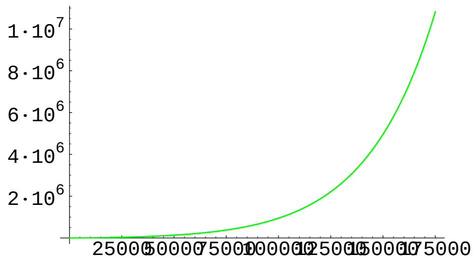


Figure 1.5: $r = -0.25, K = 10000$

The related *time vs. population* output for the Ricker equation input can vary from exponential curves to periodic curves to chaotic curves, depending on the input values.

All of the previous examples have been deterministic in nature. If the user wanted to, they could vary the yearly growth for each time step using a stochastic process. This would change the look of both the input and output graphs. The graphs would tend to be less smooth and, in cases where the growth rate was allowed to vary greatly from time step to time step, may look very little like the examples presented above.

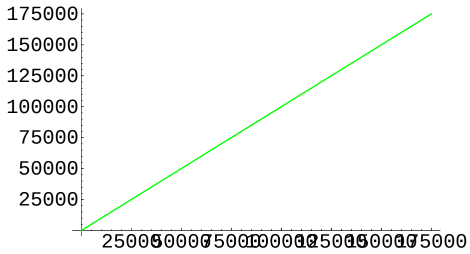


Figure 1.6: $r = 0.00, K = 10000$

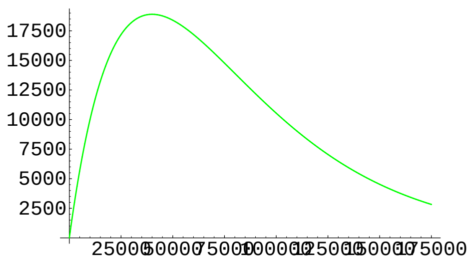


Figure 1.7: $r = 0.25, K = 10000$

### 1.1.2 Age Structure

Usually, a population will have different survival and fecundity rates for different age groups, as opposed to one simple growth rate value, and the combination of these rates determines the overall growth rate. Transition matrix techniques are commonly used to facilitate populations with these more complicated dynamics. So, instead of a single value representing a population's growth rate, transition matrices use multiple values for each of the age classes' survival and fecundity rates in order to produce yearly population change. An initial population vector is loaded from top to bottom with the size of each age group from juveniles to adults and is then multiplied by the transition matrix to determine the next year's population. The resulting vector is then multiplied by the transition matrix to get the population for the year after that, and the process is continued for the predetermined amount of iterations. Typically, only the female side of the population is tracked when using transition matrices.

There are two main types of transition matrix setup: the pre-breeding and the post-breeding. A pre-breeding setup, as described by Case [3], has each year, $t$, beginning before the birth pulse. Post-breeding has each year, $t$, beginning after the birth pulse [3]. Examples of each type, along with two different approaches for handling age classes follow.

Pre-Breeding:

Leslie style matrices (see equation 1.1) are used when all of the age class populations ($n_x$) have different survival ($s_x$) and fecundity ($f_x$) rates.

$$\begin{bmatrix} n_0 \\ n_1 \\ n_2 \end{bmatrix} (t+1) = \begin{bmatrix} f_0 s_0 & f_1 s_0 & f_2 s_0 \\ s_1 & 0 & 0 \\ 0 & s_2 & 0 \end{bmatrix} \cdot \begin{bmatrix} n_0 \\ n_1 \\ n_2 \end{bmatrix} (t) \tag{1.1}$$

Lefkovich style matrices (see equation 1.2) are used when certain age classes exhibit nearly equal survival and fecundity rates. The idea is to lump segments of the population together into what's called a stage class (in this example $n_a$ replaces the age classes $n_1$ & $n_2$).

$$\begin{bmatrix} n_0 \\ n_a \end{bmatrix} (t+1) = \begin{bmatrix} f_0 s_0 & f_a s_0 \\ s_a & s_a \end{bmatrix} \cdot \begin{bmatrix} n_0 \\ n_a \end{bmatrix} (t) \tag{1.2}$$

Post-Breeding:

Equation 1.3 shows a post-breeding Leslie style matrix.

$$\begin{bmatrix} n_0 \\ n_1 \\ n_2 \end{bmatrix} (t+1) = \begin{bmatrix} f_0 s_0 & f_1 s_1 & f_2 s_2 \\ s_0 & 0 & 0 \\ 0 & s_1 & s_2 \end{bmatrix} \cdot \begin{bmatrix} n_0 \\ n_1 \\ n_2 \end{bmatrix} (t) \tag{1.3}$$

And equation 1.4 shows a post-breeding Lefkovich style matrix.

$$\begin{bmatrix} n_0 \\ n_a \end{bmatrix} (t+1) = \begin{bmatrix} f_a s_0 & f_a s_a \\ s_0 & s_a \end{bmatrix} \cdot \begin{bmatrix} n_0 \\ n_a \end{bmatrix} (t) \tag{1.4}$$

These matrices serve as the input for the model and are the counterpart to the *population at time t vs. population at time t+1* graphs from simple headcount approach from before. The output of these matrices is dependent on the dominant eigenvalue, or Perron root, of each matrix. In order for the population to have positive growth, the Peron root must be greater than 1. This will then give a *time vs. population* graph that is exponential in nature, even though there may be some early transition.

The transition matrix can be adjusted before time steps to mimic environmental and spatial effects, and the Perron root, as well as the yearly growth rate, will subsequently be affected. One adjustment would be to introduce a carrying capacity element to some (or all) of the terms. Another adjustment would be the introduction of a stochastic process which would allow the vital rates to change from time step to time step. These adjustments allow the other non-linear types of *time vs. population* output graphs to be possible.

### 1.1.3 Other Possibilities

Available demographic, environmental & spacial data drives the choice and complexity of the model used for the study. The more detailed the data, the more intricate a model can be, in terms of input parameters. The most basic deterministic model uses only the species' yearly growth rate for the region being studied.

If data collected on a species provides more specific information for other parameters which influence the yearly growth rate, such as, female/male ratio, litter sizes per breeding pair, yearly survival rates, age class divisions in survival and breeding performance, environmentally motivated fluctuations in the vital rates (i.e. birth and survival),and/or density dependence motivated fluctuations in the vital rates, then more intricate models can be used.

Also, if the species interacts between two or more patches of habitats, it is called a metapopulation. Metapopulations will have patch-to-patch movement within them as a species searches for food, shelter and/or a mate(s). Patch-to-patch movement is sometimes an important factor in the survival of a species, and models can be created which incorporate this movement.

At some point the decision must be made between using either a deterministic or a stochastic approach, with respect to the various parameters used. When yearly fluctuations are observed in the vital rates, a stochastic approach should be considered.

Often, the biggest problem faced by the people who do wildlife simulation modeling is the lack of pertinent data. Wildlife studies in the wild are not abundant or in-depth enough for many species. So, the empirical data, which parameters are based on, may be incomplete or from a small time frame. This leads to uncertainty about the results produced by the model, and, more specifically, the importance of different parameters in relation to their effect on the overall outcome.

To give insight into the parameters which have the most affect on a population's growth rate, systematic "tweaking" of the parameters is performed as a sensitivity analysis. Sensitivity analysis can be performed in the deterministic setting as well, without the need of multiple simulations to invoke statistic's law of large numbers.

## 1.2 Questions of Interest

As one can imagine, the topic of wildlife population modeling is often debated due to the sheer volume of modeling options that are available, the dearth of reliable data for many species, and the difficulty of predicting the trends of mother nature. Questions were raised as we explored some of the modeling options, and wildlife population simulation modeling in general. Being motivated by these questions, we decided to create, or at least attempt to create, a "better mousetrap". The model's code, to be detailed later, was created from scratch. It builds on the ideas and techniques of wildlife modeling that have been developed previously by others, but it varies from the mainstream codes that we saw (i.e. RAMAS/metapop [1], VORTEX [5], and ALEX [9]) in that it handles each creature's life "independently" through a stochastic process and allows for dynamic density dependence effects. After working out many bugs, we used the code to answer three main questions of interest:

1. Does a model which considers both male and female parts of a population when determining the number of breeding pairs exhibit significantly different outcomes than a model which just considers the female part of the population? The motivation for this question comes directly from some of the papers we read. Some models [8], [9] chose to ignore the male parts of the population while determining population growth. We felt this could lead to misleading results, especially in monogamous breeding populations. A female only model would not be able to factor in the possibility that male breeding partners may not be available for all the females.

2. Does the use of different vital rates, different timing for the introduction of density dependence effects (i.e. will the effects be felt only when the population is at 100% of the carrying capacity, or earlier), and different equations to handle the logistics of the effects greatly affect the outcome

when density dependence effects are used? Motivation for this question came when we were incorporating density dependence into our model. Due to the model's make-up (which will be discussed in chapter two), we had to make choices, such as which vital rate and which equation to use as a basis. This led us to question whether or not a fecundity based or a survival based density dependence model (or both) would produce results where the average population size grows to its carrying capacity, then flat-lines there until the end of the simulation run.

3. Does a Poisson distribution based counter produce more accurate spacing between catastrophes during a simulation than the more commonly used uniform distribution based method? This question was motivated by the stat and probability courses that I took in college. I was always taught that the best way to predict the occurrence of naturally occurring events was to use an Poisson distribution. So, when I read that VORTEX and RAMAS/metapop didn't determine spacing between events with a Poisson distribution, but instead used a Uniform distribution to determine whether or not a catastrophe happened each time step [5] [1], I wondered if a Poisson approach would be better.

The simulation model that was used to answer the posed questions differs from most of the other common, readily available simulation models in that it uses cumulative arrays instead of transition matrices to determine the year to year population changes, and it allows for density dependence effects to be applied in a manner other than just truncating the population.

# Chapter 2

# The Cumulative Array Based Simulation Model

A common approach to simulation modeling is to determine the yearly changes of the whole population of subjects, gender groups, and/or age classes en masse. For example, if a stochastic process produces an adult survival rate of 70% for a particular time-step, then exactly 70% of the adults will survive that time-step, no more, no less.

Two models in particular, RAMAS/metapop [1] and ALEX [9], used the en masse approach. RAMAS/metapop uses a Leslie matrix based to determine yearly population change, which as discussed in chapter one, applies one set of vital rates to the entire population at each time step. The nature of this approach is that the individuals are not treated independently.

ALEX is also based on a matrix. A matrix based approach was taken because it "significantly increases the speed at which the model runs, particularly where the population size is large because pseudo-random numbers are not being generated for each individual" [9]. This indicates that ALEX's matrices do not treat the population as independent individuals in the code make-up. We did not use these models because we desired a model which used an independent approach to determine the yearly population fluctuations.

Another model, VORTEX, uses an independent approach and treats members of the population separately [5]. That is, if the adult survival is 70%, then each subject within the population has an independent 70% chance of surviving. The end result of an "independent" simulation model may not be exactly 70% adult survival for the time-step. We would have used

VORTEX to answer our questions, but it does not use a dynamic method to control density dependence effects. Instead, VORTEX just truncates the population if it goes over the maximum carrying capacity. RAMAS/metapop uses a method that is based on eigenvalues of the transition matrix it does not treat the members of the population independently. A model that treated the population independently, while also controlling density dependence effects in a more dynamic manner was desired.

One difficulty in choosing a model of this type was the void of available shareware software downloads which implement these techniques. As a result, the code had to be written from scratch. We chose the C++ programming language to write the code for our cumulative array based simulation model.

## 2.1 The Algorithm Used

The first step was to develop an algorithm for the code that would be implemented in C++. We decided to incorporate a multitude of options in the algorithm that would handle 1) independent input probabilities for every possible litter size, 2) a multi-age class structure in which the last, or oldest, age class will be the only breeders, 3) environmental stochasticity, including rarer catastrophe occurrences, in the demographic values, 4) metapopulation dynamics for cases where the species interacts between two or more disjoint patches & 5) density dependence effects, with the ability to hamper growth if the population is approaching, but not yet at maximum population density.

Input values for the algorithm consist of the following items. Demographic data values, i.e. the max litter size, litter size probabilities, the number of age classes, the male/female birth ratio, and survival probabilities. Environmental probabilities and their effects on vital rates. The number of patches and the probability of moving between patches (if more than one patch is used), which allow for metapopulation dynamics. Catastrophe probabilities and their effects on vital rates. Density dependence factors, i.e. a value for maximum population capacity, which will trigger the full amount of user defined density dependence effects, and a lower threshold value, which will trigger muted effects of density dependence before the maximum capacity is reached. Note that the lower threshold and maximum capacity can be the same value, in essence bringing on the full effect without ramping up to it. And finally, initial population sizes, the number of independent simulations to be run, and the number of years to be extrapolated during each simulation.

**Cumulative Array Simulation Model Algorithm**[1]

- Input: the number of patches

  Then for each patch:

- Input: patch to patch movement (yes or no), max litter size, the number of age classes, litter size probabilities, M/F birth ratio, survival probabilities, environmental probabilities and their effects on vital rates (optional), catastrophe probabilities and their effects on vital rates (optional), density dependence cap and lower threshold (optional), patch to patch movement information (optional), and initial population sizes.
- Initialize the density dependence modifiers.
- Create birth, mortality, environmental, catastrophe, and patch to patch movement pdf and/or cumulative arrays.
- Initialize the number of independent simulations to be run.
- Initialize the number of years to be extrapolated during each simulation.
- While simulation counter $\geq$ number of simulations ...

  1. While yearly counter $\geq$ number of years ...
     - i) Update population of age groups.
     - ii) If the density dependence option is on, then determine density dependence effects, if any.
     - iii) Determine the amount of breeders.
     - iv) Use pseudo-randomly generated numbers to determine the environment type (good, normal, bad, or catastrophe) for the current time step.
     - v) Determine the number of newborns by using the relevant birth cumulative arrays in conjunction with pseudo-randomly generated numbers.
     - vi) Determine the number of survivors by using the relevant mortality cumulative arrays in conjunction with pseudo-randomly generated numbers.

[1]The exact process of how the pseudo-random numbers determine the simulated life cycle of a species will be developed in the next section.

vii) Check to see if species has gone extinct.

viii) If there is more than one patch being modeled, determine creature movement from patch to patch using the movement cumulative array and pseudo-randomly generated numbers.

ix) Update yearly counter.

2. Update simulation counter.

- Output: average ending population size, standard deviation and variation of ending population size, extinction risk percentage, the pdf and cumulative arrays which were created and used by the program to generate the results.

## 2.2 Cumulative Array Formation & Usage

### 2.2.1 The Cumulative Array

The main feature of this model is its use of cumulative arrays and pseudo-randomly generated numbers. This section is devoted to an example of how the different cumulative arrays are created and used by the program during the simulation process.

For this example, the fictional No-No Bird is being used. The only age-class which breeds is adults. There are four yearly environment types built into the code: Good, Normal, Bad, and Catastrophe. Normal environmental years use the vital rates as entered by the user, without modification. First, we will demonstrate how the normal environmental year birth array is formed.

Table 2.1 shows the relative parameters that are used as input for the example.

| Litter Size | Probability |
|---|---|
| 0 | .2 |
| 1 | .3 |
| 2 | .5 |

| Female/Male Birth Ratio |
|---|
| 55/45 |

| Survival Rates | |
|---|---|
| Newborn | 0.5 |
| Juvenile | 0.7 |
| Adult | 0.8 |

Table 2.1: No-No Bird vital rates.

With the information from Table 2.1 the birth pdf is formed ...

$$\begin{aligned}
P(0\ newborns) &= 0.2 \\
P(1\ female\ newborn) &= 0.3 \cdot \left[\binom{1}{1} \cdot 0.55^1 \cdot 0.45^0\right] = 0.165 \\
P(1\ male\ newborn) &= 0.3 \cdot \left[\binom{1}{0} \cdot 0.55^0 \cdot 0.45^1\right] = 0.135 \\
P(2\ female\ newborns) &= 0.5 \cdot \left[\binom{2}{2} \cdot 0.55^2 \cdot 0.45^0\right] = 0.15125 \\
P(1\ female\ \&\ 1\ male\ newborn) &= 0.5 \cdot \left[\binom{2}{1} \cdot 0.55^1 \cdot 0.45^1\right] = 0.2475 \\
P(2\ male\ newborns) &= 0.5 \cdot \left[\binom{2}{0} \cdot 0.55^0 \cdot 0.45^2\right] = 0.10125
\end{aligned}$$

Then, bins in a new array are formed with the pdf information, using a cumulative approach...

$$\begin{array}{rcl}
0 \leq & P(0\ newborns) & \leq 0.2 \\
0.2 < & P(1\ female\ newborn) & \leq 0.365 \\
0.365 < & P(1\ male\ newborn) & \leq 0.5 \\
0.5 < & P(2\ female\ newborns) & \leq 0.65125 \\
0.65125 < & P(1\ female\ \&\ 1\ male\ newborn) & \leq 0.89875 \\
0.89875 < & P(2\ male\ newborns) & \leq 1
\end{array}$$

The result is an array where the first entry is always greater than or equal to zero (in this case 0.2) and the last entry is always one...

$$[0.2,\ 0.365,\ 0.5,\ 0.65125,\ 0.89875,\ 1]$$

The Cumulative Birth Array

Once the birth array is created, it is used each time-step, with each breeding pair in conjunction with pseudo-randomly generated numbers (uniform on $[0, 1)$) to determine how many newborns should be added to the population. The code will first initialize a loop counter with the number of breeding pairs (in the case of the model which considers both female and male population sizes before determining the number of breeders, the minimum of breeding age males or females is used, whereas the female only approach will always use the number of breeding age females). Then, each time through the loop, a pseudo-random number is generated. The pseudo-random number is systematically checked against each element of the array, starting with the first entry. If the pseudo-random number is less than or equal to the array element, then the population is adjusted accordingly. For the No-No Bird example, a value of 0.65 would mean that two female newborns would be added to the total newborn population. This loop is repeated until the breeding pair counter is decremented to zero.

The normal environmental year survival array uses a similar technique, except that each element of the array represents the probability of survival for its respective age-class[2]...

[2]The model also allows for gender specific rates so a survival array is created for both genders during parameterization, and each gender's population is subjected to the survival process.

$$[0.5,\ 0.7,\ 0.8]$$

The Survival Array

To determine the number of survivors for the first age class, a loop counter is initialized with the number of individuals currently "alive" in the first age class, then, each time through the loop, a pseudo-random number is generated and checked against the age class's parameter value. A pseudo-random number larger than the parameter value indicates a failure to survive to the next time-step, and the population is then decremented by one. In this example, if 0.612 is the first pseudo-random number generated for the newborn population, then the newborn population is decremented by one. The process repeats until the loop counter of the particular age class decrements to zero; the model performs the same procedure for each age class present in the array.

### 2.2.2 Environmental Effects

Three separately initialized environmental condition arrays (denoted as Good ($g$), Normal ($n$), and Bad ($b$)) as well as a separately initialized Catastrophe ($c$) array are allowed for in the code (more arrays could be added easily to the code, if necessary). The $g, b$, & $c$ arrays start out equal to the user-initialized birth and survival normal year arrays (as detailed above). Non-negative "environmental effect" multiplicative scalars that are input by the user are then applied to the birth pdf and survival arrays. In the $g, b$, & $c$ survival arrays, if the multiplicative scalars are less than 1, then they will reduce the age class's survival probability value and if the scalars are greater than 1, then the age class's probability value will increase. For example, assume the user entered the survival array:

$$[0.5,\ 0.7,\ 0.8]$$

Then, assume that the user decided that, in a good year, the 1st age class survives at a rate that's 120% of the normal rate, the 2nd age class survives at a rate that's 130% of the normal rate, and the 3rd age class survives at a rate that's 120% of the normal rate. The user would enter 1.2, 1.3 and 1.2 for the respective age class survival modifiers, and the resulting good year survival array would be...

$$[0.5 \cdot 1.2,\ 0.7 \cdot 1.3,\ 0.8 \cdot 1.2]$$

or, after simplifying...

$$[0.6,\ 0.91,\ 0.96]$$

Birth pdf arrays, excluding the first element, are affected similarly by the scalars, with the difference being how the first element is handled. The first element of the birth pdf array is adjusted separately, so that the sum of all the elements in the pdf array will equal 1, and it is found by summing the elements $2, \ldots, n,\ n \in N$ and then subtracting the sum from one. Clearly, a scalar greater than 1 could produce a negative value for the first element of the birth array. For this reason, the code will stop and present the problematic array[3] when this scenario is encountered. Assume the user entered a Female to Male ratio of 55:45 and the following probabilities for births: $P(0) = 0.2,\ P(1) = 0.3, \&\ P(2) = 0.5$. This gives the birth array:

$$[0.2,\ 0.365,\ 0.5,\ 0.65125,\ 0.89875,\ 1]$$

Then, assume the user decided that in a good year the probability of a litter size of 1 is 110% of the normal rate and the probability of a litter size of 2 is 105% of the normal rate. The user would enter 1.1 and 1.05 as the respective litter size modifiers, and the resulting good year birth probabilities would be $P(1) = 0.3 \cdot 1.1 = 0.33, \&\ P(2) = 0.5 \cdot 1.05 = 0.525$. The good year birth pdf is formed...

[3] A list of all the arrays used is included in the output for each simulation so that the user can check for any potential input errors and be made aware of all numbers being used

$$
\begin{aligned}
P(1\ female\ newborn) &= 0.33 \cdot \left[\binom{1}{1} \cdot 0.55^1 \cdot 0.45^0\right] = 0.1815 \\
P(1\ male\ newborn) &= 0.33 \cdot \left[\binom{1}{0} \cdot 0.55^0 \cdot 0.45^1\right] = 0.1485 \\
P(2\ female\ newborns) &= 0.525 \cdot \left[\binom{2}{2} \cdot 0.55^2 \cdot 0.45^0\right] = 0.1588125 \\
P(1\ female\ \&\ 1\ male\ newborn) &= 0.525 \cdot \left[\binom{2}{1} \cdot 0.55^1 \cdot 0.45^1\right] = 0.259875 \\
P(2\ male\ newborns) &= 0.525 \cdot \left[\binom{2}{0} \cdot 0.55^0 \cdot 0.45^2\right] = 0.1063125 \\
P(0) &= 1 - 0.855 = 0.145
\end{aligned}
$$

And then the cumulative good year birth array is formed:

$$[0.145,\ 0.3265,\ 0.475,\ 0.6338125,\ 0.8936875,\ 1]$$

### 2.2.3 Eigenvalue Driven Density Dependence Adjustment

The goal of our density dependence effect was to have a *time vs. population* curve representative of a logistic equation's, where the population grows to maximum density and remains there for the duration of the simulation. A stable growth rate at maximum density with logistic equations is obtained because a population equal to the carrying capacity effectively cancels out the growth term in the equation. Due to the array based nature of our model, and the presence of multiple variables which could not easily be squeezed into a normal logistic equation, we had to approach the concept of density dependence differently.

After some trial (and much error), we chose to use the fact that "the dominant eigenvalue of the (deterministic) Leslie matrix yields an age-structured population's ultimate geometric growth rate" [3] as a basis for our design. And we knew that if we could adjust vital rates as the simulation ran so that the dominant eigenvalue would equal 1 at carrying capacity, the population would maintain stable growth. We had to find a way to have the code use this information to inact density dependence effects. Two different approaches were designed: birth rate driven and survival rate driven. In the birth rate driven approach, only the birth rates were modified to enact density dependence. The survival rate approach calls for only the survival rates to be modified to enact density dependence.

### Birth (or fecundity) Driven Model

The first step in the process was transforming the user defined values for birth and survival into Leslie matrix entries. Survival rates required no change in order to be useful, but the code had to be modified in order to allow it to take individual litter sizes and their respective probabilities and transform the information into a general fecundity rate. Once this was done, we applied a scalar, $m$, to the fecundity values in the matrix. After finding the characteristic equation of the matrix, we set $\lambda = 1$, where $\lambda$ is an eigenvalue, and solved for $m$. This process was done for square Post-Breeding Leslie matrices of size 2, 3, 4, 5, & 6 [4]. A square matrix of size 2 has a unique equation (2.1) for a scalar $m$, where $s$ is the survival rate and $b$ represents the simplified $f_n s_n$ terms.

$$m = \frac{s_1 - 1}{-bs_0} \tag{2.1}$$

Square matrices of size 3 through 6 produced equations for $m$ which can be described by the general form of equation 2.2

$$m = \frac{s_{n-1} - 1}{s_0 s_1 \cdots s_{n-3} b(-1 - s_{n-2} + s_{n-1})} \tag{2.2}$$

For example, the fictional No-No Bird's Post-Breeding Leslie matrix would be...

$$\begin{bmatrix} 0 & 1.3 & 1.3 \\ 0.5 & 0 & 0 \\ 0 & 0.7 & 0.8 \end{bmatrix}$$

where $b = 1.3,\ s_0 = 0.5,\ s_1 = 0.7, s_2 = 0.8, \&\ \lambda \approx 1.26$. Using the above equation, the scalar $m$ needed for the birth elements in order to give $\lambda = 1$ would be...

---

[4]For future work, we'd like to prove the conjecture that the formulas work for $n > 6$, where $n \in N$.

$$\frac{0.8-1}{0.5\cdot 1.3(-1-0.7+0.8)} \approx 0.3419$$

Each $b$ in the Leslie matrix would then be multiplied by $m$, and the new Leslie matrix with $\lambda = 1$ would be...

$$\begin{bmatrix} 0 & 0.44447 & 0.44447 \\ 0.5 & 0 & 0 \\ 0 & 0.7 & 0.8 \end{bmatrix}$$

**Survival Driven Model**

Determining scalar values for survival followed a similar approach, with the scalar $m$ applied to all of the survival entries in the matrix instead of the fecundity values. And again, a special case was found for a square matrix of size 2 (2.3).

$$m = \frac{1}{s_1 + s_0 b} \tag{2.3}$$

It also became clear that square matrices of even and odd sizes had different general forms.

Odd sizes (beginning at size 3) had the form of equation 2.4.

$$m^{n-1}(s_0 \cdots s_{n-2}b - s_0 \cdots s_{n-3}s_{n-1}b) + m^{n-2}(s_0 \cdots s_{n-3}b) + ms_{n-1} - 1 = 0 \tag{2.4}$$

Whereas even sizes (beginning at 4) had the form of equation 2.5.

$$m^{n-1}(s_0 \cdots s_{n-3}s_{n-1}b - s_0 \cdots s_{n-2}b) - m^{n-2}(s_0 \cdots s_{n-3}b) - ms_{n-1} + 1 = 0 \tag{2.5}$$

The fact that polynomials greater than degree 1 were represented left us with the problem of solving for multiple $m$ values. We knew that $m$ needed to lie between 0 and 1 to be effective. After plugging 0 and 1 into equations 2.4 & 2.5 for $m$, we found that the signs for the two bounds were opposite in both equations. This knowledge, along with a conjecture that only one real root would exist between 0 and 1, allowed us to use the bisection method of finding roots and subsequently find scalars $m$ for the survival rates.

For example, again using the No-No Bird information, we would use the equation for odd sizes...

$$m^2(0.5\cdot 0.7\cdot 1.3 - 0.5\cdot 0.8\cdot 1.3) + m(0.5\cdot 1.3) + 0.8m - 1 = -0.065m^2 + 1.45m - 1$$

After using the bisection method, the scalar $m \approx 0.717$ is obtained and multiplied by all of the survival elements. This then produces the modified survival array:

$$[0.3585,\ 0.5019,\ 0.5736]$$

The Density Dependence Survival Array

**Implementation**

Algorithms were then created and implemented into the code which would use the equations (2.1-2.5) along with user defined data to find the appropriate scalar $m$ necessary for reducing the fecundity or survival rate to a level which stops population growth at carrying capacity. The code was also modified to allow the user the option of using a lower cap threshold, in order to enact carrying capacity effects in a gradual manner as the population grows towards maximum density. Linear interpolation was used to scale $m$ appropriately, depending on where the population lies between the lower threshold value and the maximum capacity value. Both approaches produced *time vs. population* graphs that grow towards maximum density at an exponential rate and then flatline at maximum density. If a lower cap threshold is used, then the graph experienced a concavity change from up to down before flatlining at maximum density. As mentioned earlier, we questioned the difference in approaching density dependence from a birth or survival perspective. The results of the testing of the two approaches, as well as other remarks, is saved for chapter 5.

### 2.2.4 Metapopulation

The code also allows for multiple population patches. Each patch is allowed to have separate vital rates, environmental effects and density dependence effects. Movement is facilitated between the patches by way of arrays. For example, in a three patch system, the user would enter the probability of moving from patch one to patches two and three, the probability of moving from patch two to patches one and three, and the probability of moving from

$$\begin{bmatrix} - & 0.3 & 0.5 \\ 0.2 & - & 0.4 \\ 0.15 & 0.6 & - \end{bmatrix}$$

Figure 2.1: In this three patch movement array, patch one would have a 30% chance of moving to patch two and a 20% chance of moving to patch three. Patch 2 would have a 20% chance of moving to patch one and a 20% chance of moving to patch three Patch 3 would have a 15% chance of moving to patch one and a 45% chance of moving to patch two.

patch three to patches one and two. The probability of moving from a patch to itself is always 0. Then a cumulative array is formed, as in figure 2.1.

Finally, pseudo-random numbers are generated and compared to the array to determine patch to patch movement. For example, if a pseudo-randomly generated number for patch 1 was 0.55, then it would be checked against the 0.3 in row 1 first. The first element is skipped because the code is designed to skip entries in which the departure patch number equals the arrival patch number. Since 0.55 is greater than 0.3, the creature does not move to patch 2. Then, 0.55 is checked against 0.5, and since it is greater, the creature does not move to patch 3 either. Instead, the creature remains in patch 1 as a result.

## 2.3 The Random Number Generator Used

This simulation model relies heavily on the generation of pseudo-random numbers. The generation of pseudo-random numbers in this model complies with the ANSI C standards for "randomness" and uses a method proposed by Lehmer in 1949 [7] known as the Linear Congruence Generator (L.C.G.). Common notation for L.C.G.'s is $LCG(m, a, c, Z_0)$. The parameters $m$, $a$, $c$, & $Z_0$ are used to parameterize an iterative process that uses equation 2.6 to produce a range of $n$ numbers, $[0, m-1)$, or, as in our model, each $Z_i$ can be divided by $m$ to produce a range of uniformly distributed values on $[0, 1)$.

$$Z_i = (a \cdot Z_{i-1} + c) \bmod m \tag{2.6}$$

The choice of $m$, $a$, & $c$ determine whether every possible number in the range 0 & $m-1$ (a full period) will be generated. Theorem 2.3.1, proven by Hull and Dobell in 1966 [4], lists conditions which guarantee a full period.

**Theorem 2.3.1** (Hull & Dobell) $Z_i = (aZ_{i-1} + c) \bmod m$ *has a full period iff:*

*i) m & c are relatively prime*

*ii) if q is a prime number which divides m, then q divides a-1*

*iii) if 4 divides m, then 4 divides a-1*

The period length is one result of the choice of $m$, $a$, & $c$. The choice of $m$, $a$, & $c$ also determines whether the generation of pseudo-random numbers falls into a certain pattern. That is, sequences of numbers may repeat themselves often, which could taint stochastically driven results, and yet the overall distribution would appear to be uniform.

## 2.4 Model Validation

Once the code was implemented in C++, we compared its performance to that of a deterministic probability model implemented in Microsoft Excel. A deterministic model was used for comparison because we felt that running 1000 simulations of 50 years each in our stochastic model would invoke the law of large numbers necessary to allow the average ending population size to approach the expected value produced by the deterministic model. The stochastic model was run without any environmental, density dependence, or metapopulation effects. No-No bird (pg. 15) birth & survival rates were used, in conjunction with four initial population sizes.

Table 2.2 shows the effect of different initial population sizes on the stochastic and deterministic ending population size ratio. Note that each population size uses three trials.

The first item to note is that with smaller population sizes there is a greater discrepancy between the stochastic output and the deterministic output. This is to be expected as the stochastic output mimics the idea that smaller populations will have a harder time growing and flourishing.

Another item which arose with this type of model was the effect of different survival rates on the correlation between the stochastic and deterministic model's resultant ending population size. Data from a study on Northern

| Initial Population Size | Stochastically Obtained Average & Standard Deviation | | Stochastic/Deterministic Ratio |
|---|---|---|---|
| 120 | 135.352 | 61.347 | 0.8154 |
| | 143.299 | 63.797 | 0.8632 |
| | 142.274 | 64.777 | 0.8571 |
| 600 | 823.967 | 166.846 | 0.9909 |
| | 815.740 | 163.168 | 0.9810 |
| | 827.630 | 168.221 | 0.9953 |
| 1200 | 1645.110 | 234.430 | 0.9892 |
| | 1656.670 | 234.640 | 0.9962 |
| | 1653.890 | 249.234 | 0.9945 |
| 6000 | 8277.210 | 528.141 | 0.9955 |
| | 8217.790 | 558.170 | 0.9883 |
| | 8265.350 | 543.774 | 0.9940 |

Table 2.2: Average & standard deviation of ending population sizes and their respective stochastic/deterministic ratios using No-No bird data.

Spotted Owls [6] indicated that the newborn survival rate was a low 0.159 (the juvenile and adult rates were 0.83 & 0.84, respectively). And when output from the stochastic and deterministic models was compared, using what was thought to be a sufficiently high initial population size (5000 subjects), it was observed that the stochastic model's ending population size was roughly 70% of the deterministic ending population size. This was not in the 90% range that was expected.

We theorized that the low survival rate was the cause of such a low percentage. In order to test this, we ran the simulation with higher values of newborn survival. Once we set newborn survival at 40%, we observed that the stochastic model's ending population size was 80% of the deterministic ending population size. And when we set the newborn survival at 49%, we observed that the stochastic model's ending population size was 90% of the deterministic ending population size.

Clearly, different newborn survival rates affect the stochastic results' relation to their deterministic counterparts. The reasoning for the discrepancy is that, at low rates of newborn survival, the stochastic process is not given a chance to "rebound" from a string of failures.

So, the model does have its deficiencies. It needs larger initial populations to avoid ending population sizes that are below those predicted by a deterministic model. The ability for only one age class to be the breeding class may not fit well with some species[5]. And, the patch to patch movement process only allows for adults to travel from patch to patch, which again, may not fit well with some species[6].

All-in-all, we felt confident in using the CDF model approach to answer the our first two questions of interest.

[5]The code can be modified to allow breeding for other age classes.

[6]The code can also be modified to allow for more, or different, age classes to move.

# Chapter 3

# Female Only Models vs. Female & Male Models

## 3.1 Background Information

The question as to whether there is a difference between "female only" and "female & male" models, in terms of ending population size, was borne out of a concern that population predictions could be adversely affected if the male population was ignored. We felt "female only" models could affect monogamous species the most, because a shortage of male partners means that not all of the females would be able to mate and produce offspring.

The models in three of the papers we read did not track the male population during simulation runs. In a paper written by H.P. Possingham & I. Davies [9], the ALEX model is a model which tracks only one gender. The authors did consider this trait to be a weakness of the model, and states that "species in which both sexes can limit population growth rate are not modeled accurately". Some mention is also made about the scenario where there are no breeding males for any of the females, and how this might affect the results.

Another paper, written by R.H. Lamberson, et al [6], also uses a female only model. In this paper, no defense or explanation, whether explicit or implicit, is given for choosing this approach.

Finally, a third paper, written by M.T. Murphy [8], also describes a model that ignores the male population. In this paper, the author only implicitly suggests that the approach is acceptable when he states "I am confident that my 11 year demographic study provides reliable estimates of $S_a$ (Adult Survival), $F/2$ (female offspring per year), and dispersal behavior". No explicit statements are given about why the male population numbers and their potential affects on population growth were ignored.

These three papers in particular drove our desire to determine the affect (if any) of using a "female only" model for simulating a species' population growth.

## 3.2 Method

We compared the ending population size of a "female only" model to a model designed to use both female and male population numbers, effectively simulating a monogamous species. In order to simulate monogamy, the model considers both female and male population sizes when determining the number of breeding pairs. Specifically, the number of breeding age females or males, whichever is less, is used as the number of breeding pairs for each breeding cycle. The "female only" model just uses the number of breeding age females as the number of breeding pairs.

Data from Lamberson, et al.'s paper on Northern Spotted Owls was used. The parameters used included a female/male ratio of 50/50 & 1.42 offspring per breeding pair [1]. Subadult and adult survival rates were set at 0.83 and 0.84 respectively. We used three juvenile survival rates (0.159, 0.509, & 0.735) in order to obtain decreasing, stable, and increasing yearly growth rates. Several different population sizes were chosen for each of the three scenarios.

The focus was on the mean and standard deviation of the ending population size. We ran trials incorporating 1000 independent simulation runs of 50 years apiece.

Some of the models in question, namely the ones in Murphy's and Lamberson's papers, were female based and included metapopulation dynamics. To determine if the discrepancy in the ending population sizes for single patch environments also carried over to multiple patch environments, we ran multiple patch simulations using two and three patch metapopulations. Again, we ran 1000 simulations of 50 years each for the female only and male & female based models. All of the patches used the same vital rates. The survival rate of 0.509 was used for the juveniles, because the output in the single patch

[1] individual litter probabilities of 0.37, 0.5, 0.13 were used as input for the program

scenario produced more reasonable numbers for a comparison. The other vital rates were kept the same as in the single patch situation.

Both the two patch and three patch metapopulations were run with four different movement scenarios. For the two patch metapopulation, the movement scenarios used were:

1) A 24% chance of moving from patch 1 to patch 2, and a 24% chance of moving from patch 2 to patch 1.

2) A 50% chance of moving from patch 1 to patch 2, and a 24% chance of moving from patch 2 to patch 1.

3) A 24% chance of moving from patch 1 to patch 2, and a 50% chance of moving from patch 2 to patch 1.

4) a 50% chance of moving from patch 1 to patch 2, and a 50% chance of moving from patch 2 to patch 1.

For the three patch metapopulation, the movement scenarios used were:

1) A 24% chance of moving from patch 1 to patch 2 and 3, a 24% chance of moving from patch 2 to patch 1 and 3, and a 24% chance of moving from patch 3 to patch 1 and 2.

2) A 24% chance of moving from patch 1 to patch 2, a 50% chance of moving from patch 1 to patch 3, a 24% chance of moving from patch 2 to patch 1, a 50% chance of moving from patch 2 to patch 3,and a 24% chance of moving from patch 3 to patch 1 and 2.

3) a 50% chance of moving from patch 1 to patch 2, a 24% chance of moving from patch 1 to patch 3, a 24% chance of moving from patch 2 to patch 1 and 3, a 24% chance of moving from patch 3 to patch 1, and a 50% chance of moving from patch 3 to patch 2.

4) a 12% chance of moving from patch 1 to patch 2 and 3, a 12% chance of moving from patch 2 to patch 1 and 3, and a 12% chance of moving from patch 3 to patch 1 and 2.

## 3.3 Results

The results from our single patch trials are shown in table 3.1.

| Initial Population Size | Female Only | | Female and Male | |
|---|---|---|---|---|
| Juvenile Survival=0.159 | | | | |
| | $\overline{x}$ | $s$ | $\overline{x}$ | $s$ |
| 100 | 0.423 | 0.977 | 0.136 | .0440 |
| 250 | 0.999 | 1.546 | 0.445 | 0.862 |
| 500 | 1.977 | 2.113 | 1.108 | 1.426 |
| 1000 | 4.115 | 3.034 | 2.726 | 2.277 |
| 2000 | 8.196 | 4.549 | 6.056 | 3.566 |
| Juvenile Survival=0.509 | | | | |
| | $\overline{x}$ | $s$ | $\overline{x}$ | $s$ |
| 100 | 101.408 | 50.950 | 54.448 | 27.696 |
| 250 | 258.561 | 77.053 | 174.418 | 50.866 |
| 500 | 513.981 | 113.296 | 393.763 | 77.097 |
| 1000 | 1028.150 | 157.796 | 854.516 | 113.432 |
| 2000 | 2059.320 | 220.751 | 1806.400 | 169.278 |
| Juvenile Survival=0.735 | | | | |
| | $\overline{x}$ | $s$ | $\overline{x}$ | $s$ |
| 100 | 1797.040 | 519.216 | 1173.390 | 334.126 |
| 250 | 4466.450 | 864.825 | 3476.530 | 578.927 |
| 500 | 8910.890 | 1123.980 | 7474.130 | 864.913 |
| 1000 | 17900.600 | 1683.130 | 15826.500 | 1224.490 |
| 2000 | 35601.800 | 2328.460 | 32678.200 | 1711.420 |

Table 3.1: Female only vs. Female & Male based average ending population and standard deviation results

The data from table 3.1 was used to create 95% conficence intervals for $\mu_{Female\ Only}$-$\mu_{Female\ and\ Male}$ in the single patch scenario. If 0 is in the confidence interval, then we would have evidence that the two models produced essentially the same output. Formula 3.1 was used to create the confidence intervals, with $\overline{x}_i$ and $s_i$ from table 3.1 and $n_i = 1000$ (note that subscripts 1 and 2 refer to the female only and female & male models, respectively).

$$(\overline{x}_1 - \overline{x}_2) \pm 1.96 \cdot \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \tag{3.1}$$

The resulting confidence intervals for the single patch trials are given in table 3.2. Because, in each case, 0 lies outside the confidence interval, we conclude that the female only model produces ending population size results that are greater than the model that includes the male population.

| Population Size | 95% Confidence Interval |
|---|---|
| Juvenile Survival=0.159 | |
| 100 | (0.226, 0.348) |
| 250 | (0.444, 0.664) |
| 500 | (0.711, 1.027) |
| 1000 | (1.154, 1.624) |
| 2000 | (1.782, 2.498) |
| Juvenile Survival=0.509 | |
| 100 | (43.366, 50.554) |
| 250 | (78.420, 89.866) |
| 500 | (111.724, 128.712) |
| 1000 | (161.589, 185.679) |
| 2000 | (235.678, 270.162) |
| Juvenile Survival=0.735 | |
| 100 | (585.381, 661.919) |
| 250 | (925.416,1054.424) |
| 500 | (1348.857, 1524.663) |
| 1000 | (1945.092, 2203.108) |
| 2000 | (2744.491, 3102.709) |

Table 3.2: Female only vs. Female & Male based 95% confidence intervals

For the metapopulations, the total average ending population size across all patches was found by summing the individual average ending population sizes for each patch. The standard deviation of the total ending population size was found by taking the square root of the combined variance of the individual patches. Before we combined the variances, we determined if there was a correlation between the individual patches' outcomes by performing a multivariate analysis on the ending population sizes of the individual patch outcomes. We used the statistical software package JMP to plot the ending population sizes in the two patch model for patch 1 vs patch 2 for the 1000 simulations and discovered a correlation existed between them. Using JMP, we found that there was also a correlation between the ending population sizes for the three patch model. So, in order to take into account the correlation and combine the variances properly, we included covariance terms in the combined variance formulas.

For two patches we used formula 3.2, and for three patches we used formula 3.3. In both formulas, the subscripts refer to the patch number, the $\overline{x}_i$'s are the average ending population sizes for the individual patches, the $s_i^2$'s are the variances of the ending population size for the individual patches, the $s_{ij}$'s are the estimated covariance terms and $n$ is the number of simulations. JMP was used to determine the estimated covariance values from the raw data for the metapopulation scenarios.

$$v\hat{a}r(\overline{x}_1 + \overline{x}_2) = \frac{1}{n} \cdot (s_1^2 + s_2^2 + 2 \cdot s_{12}) \tag{3.2}$$

$$v\hat{a}r(\overline{x}_1 + \overline{x}_2 + \overline{x}_3) = \frac{1}{n} \cdot (s_1^2 + s_2^2 + s_3^2 + 2 \cdot (s_{12} + s_{13} + s_{23})) \tag{3.3}$$

The results for the combined mean and standard deviation of the two and three patch models are shown in table 3.3 on page 33.

The table values were used to create 95% confidence intervals for $\mu_{Female\ Only}$-$\mu_{Female\ and\ Male}$ using formula 3.4, where $\overline{y}_1$ and $\overline{y}_2$ are the summed patch means and $s_1^2$ & $s_2^2$ are the combined patch variances for each model, and $n$ is the number of simulations.

$$(\overline{y}_1 - \overline{y}_2) \pm 1.96 \cdot \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \tag{3.4}$$

Table 3.4 on page 34 shows the resulting confidence intervals for the metapopulation scenarios.

Again, 0 does not lie in the confidence interval, so we conclude that the female only model produces higher values than the male & female based model.

## 3.4 Conclusion

Overall, the consistent and relatively large discrepancy between the two models' ending population sizes makes a strong argument that tracking the

| Movement Scenario | Initial Population Size | Female Only combined $\overline{x}$ | Female Only combined $s$ | Female and Male combined $\overline{x}$ | Female and Male combined $s$ |
|---|---|---|---|---|---|
| Two Patches with Juvenile Survival=0.509 | | | | | |
| 1 | 250 | 252.251 | 2.378 | 140.863 | 1.391 |
| 2 | 250 | 259.257 | 2.426 | 142.123 | 1.373 |
| 3 | 250 | 259.034 | 2.391 | 148.028 | 1.418 |
| 4 | 250 | 257.449 | 2.396 | 143.541 | 1.347 |
| 1 | 1000 | 1032.696 | 4.869 | 783.921 | 3.196 |
| 2 | 1000 | 1034.664 | 4.975 | 784.350 | 3.401 |
| 3 | 1000 | 1030.597 | 4.885 | 793.604 | 3.163 |
| 4 | 1000 | 1019.992 | 4.532 | 778.539 | 3.328 |
| Three Patches with Juvenile Survival=0.509 | | | | | |
| 1 | 250 | 253.343 | 2.357 | 123.756 | 1.297 |
| 2 | 250 | 251.758 | 2.378 | 122.213 | 1.311 |
| 3 | 250 | 253.768 | 3.457 | 127.415 | 1.079 |
| 4 | 250 | 255.251 | 2.466 | 123.932 | 1.264 |
| 1 | 1000 | 1019.989 | 4.738 | 738.162 | 3.103 |
| 2 | 1000 | 1030.606 | 4.930 | 743.663 | 3.313 |
| 3 | 1000 | 1028.808 | 4.638 | 745.203 | 3.197 |
| 4 | 1000 | 1033.118 | 5.011 | 738.396 | 3.274 |

Table 3.3: Female only vs. Female & Male based combined means and standard deviations

male population during simulation modeling could be of vital importance. This is especially true for monogamous species where there may not be enough males to breed with all of the present females, but it could also be important in smaller polygynous populations where extremely low male population numbers would still leave some females without a breeding partner.

Notice in the two and three patch models that the confidence intervals indicate an even larger discrepancy between the two approaches.

In the case of the ALEX model, the authors claim that the strengths of the model are related to its multiple patch capability, while at the same time, listing the single sex approach as a weakness. However, as seen in

| Movement Scenario | Population Size | 95% Confidence Interval |
|---|---|---|
| Two Patches | | |
| 1 | 250 | (111.217, 111.559) |
| 2 | 250 | (116.961, 117.307) |
| 3 | 250 | (110.834, 111.178) |
| 4 | 250 | (113.738, 114.078) |
| 1 | 1000 | (248.414, 249.136) |
| 2 | 1000 | (249.940, 250.688) |
| 3 | 1000 | (236.632, 237.354) |
| 4 | 1000 | (241.105, 241.801) |
| Three Patches | | |
| 1 | 250 | (129.420, 129.754) |
| 2 | 250 | (129.377, 129.713) |
| 3 | 250 | (127.415, 127.747) |
| 4 | 250 | (131.147, 131.491) |
| 1 | 1000 | (281.476, 282.178) |
| 2 | 1000 | (286.575, 287.311) |
| 3 | 1000 | (283.256, 283.954) |
| 4 | 1000 | (294.351, 295.093) |

Table 3.4: Female only vs. Female & Male based test statistics and related P-values

our metapopulation results, single sex, multiple patch models could produce results that are even more disparate than the single patch model. It is especially important then to only use ALEX when the user is sure that there will be enough males present in the population to ensure that the females can produce offspring.

Lamberson's paper "conclude(d) that a conservation plan that provides for clusters of territories above some minimal size should greatly increase the persistence likelihood of Spotted Owls"[6]. But, as our results show, the female only model can produce overly optimistic results, and the minimal size determined by the simulation may not be enough to truly ensure persistence.

In Murphy's paper, "population size after 10 years was predicted accurately when (he) used the empirically observed, habitat-specific rates of adult survival, productivity, and adult dispersal among habitats, and assumed a

5% rate of immigration into the upland"[8]. In this case, it would seem that ignoring the females was acceptable. So, the results of our exploration should not be looked at as the be all, end all of what can happen when males are ignored, but based on our results, careful consideration should be given before choosing to ignore the male population in a simulation model.

# Chapter 4

# Density Dependence Exploration: Birth Rate Basis vs. Survival Rate Basis

## 4.1 Background Information

The next question of interest to be explored deals with density dependence. Density dependence is the idea that as a population grows, available resources necessary for survival remain constant or decline, and, as a result, the population's size will plateau, or peak, after time. Different approaches exist to inact this effect during simulation. RAMAS/metapop [1], VORTEX [5], and the method used in a model developed by Beier [2] illustrate the range of approaches taken to inact density dependence effects in simulation models.

RAMAS/metapop is a matrix based simulation model that uses an approach similar to our model. The model allows for three basic types of density dependence: 1) Scramble (based on the Ricker or logistic model), 2) Contest (based on the Beverton-Holt model), 3) Ceiling (a truncation approach), and allows for an Allee effect [1] on each of the basic types if the user desires. For scramble and contest density dependence, with or without Allee effects, the

[1] An Allee effect is a formula adjustment that causes the population to decline if the population is too high, and if it is too low. This effectively models the situation where, with relatively small population numbers over a vast territory, potential mates may not find each other and produce offspring.

RAMAS/metapop model modifies matrix elements by way of a scalar, $m$, which is calculated from functions based on the population size. These functions are pre-determined by the model at the start of the simulation. When $m$ is found, the model adjusts the matrix so that a target eigenvalue is obtained. The target eigenvalue is designed to match the growth rate predicted by a Ricker equation equivalent or a Beverton-Holt equation equivalent. This causes the population growth to follow the curve predicted by the Ricker and Beverton-Holt based equations. Ceiling density dependence, with or without Allee effects, just returns the population level to the carrying capacity set by the user if the population exceeds the maximum. So, as far as density dependence choices, the RAMAS/metapop model is extremely versatile.

In the VORTEX model, the population is allowed to grow normally until the carrying capacity is reached. After the carrying capacity is exceeded, the model truncates the population so that it is equal to the carrying capacity [2]. So, even though VORTEX allows for independence in the simulation, it does not give nearly the same amount of choices for inacting density dependence as RAMAS/metapop.

The model described in Beier's paper on cougar conservation was built from scratch to be used specifically for the purpose of doing a population viability analysis. It uses separate carrying capacities for the female and male populations and includes "subroutines to simulate density dependence, including an Allee effect, inhibition of reproduction for the youngest females when the population exceeded carrying capacity, enhancement of survival rates at low density and decline in survival rates (especially for juveniles and dispersers) at high density" [2]. The subroutines lowered survival rates by multiplying the survival rates by a scalar. The breeding probabilities were adjusted by assigning the youngest females to a non-breeding status and only allowing 20% of the females in excess of the female carrying capacity to breed when the female carrying capacity is exceeded. When the male population was below the male carrying capacity, an Allee effect was introduced, reflecting the difficulty that some females may have in finding mates.

As detailed in chapter 2, our model uses an eigenvalue based system. In effect, our model is a hybrid of Vortex and RAMAS/metapop with respect to the use of independence and eigenvalues, respectively. Our model does not have an Allee effect built into it but considering the question we were trying

[2]This approach is equivalent to RAMAS/metapop's ceiling density dependence approach.

to answer, its inclusion in our code seemed unnecessary.

We used our model to answer the question as to whether a fecundity based or a survival based density dependence model (or both) would produce results where qualitative differences in behavior between the two approaches could be observed.

## 4.2 Method

We used Northern Spotted Owl data, with juvenile survival set at 0.735 to ensure strong positive growth. One thousand simulations of 50 years each were run for each of 6 different starting population sizes, with the carrying capacity for each of the six populations set at twice the initial population size, and with 5 different lower cap threshold values: 100%, 95%, 90%, 85%, & 80% of maximum carrying capacity. We used the 1000 simulated population values to obtain end of year average population sizes for each of the years 1 through 50. The averaged yearly population sizes were used to create figures 4.1 & 4.2 on pages 39 and 40, respectively. These figures show the overlay of the different outcomes for each of the 5 threshold values, for a starting population size of 200. They shape of these graphs are indicative of what the other initial population sizes produce.

## 4.3 Results

The main reason for allowing for lower cap thresholds, where linearly interpolated $m$ values were applied to the appropriate fecundity or survival value, was the desire to closely mimic a logistic graph's tendencies. That is, we wanted the *time vs. population* graphs to have a smooth transition from the normal growth rate to the stable carrying capacity population level. This is opposed to the sharp transition produced by a deterministic ceiling model. We still expected the average ending population size in the simulations with lower cap thresholds set below 80% to reach the carrying capacity level. But, as the results show, the lower cap thresholds produce average population sizes that stay below the carrying capacity during all time steps.

We investigated this phenomena by looking at the yearly range of population sizes for 30 simulations of 50 years each for both the survival and birth model, for both 80% and 100% lower cap thresholds. The initial population

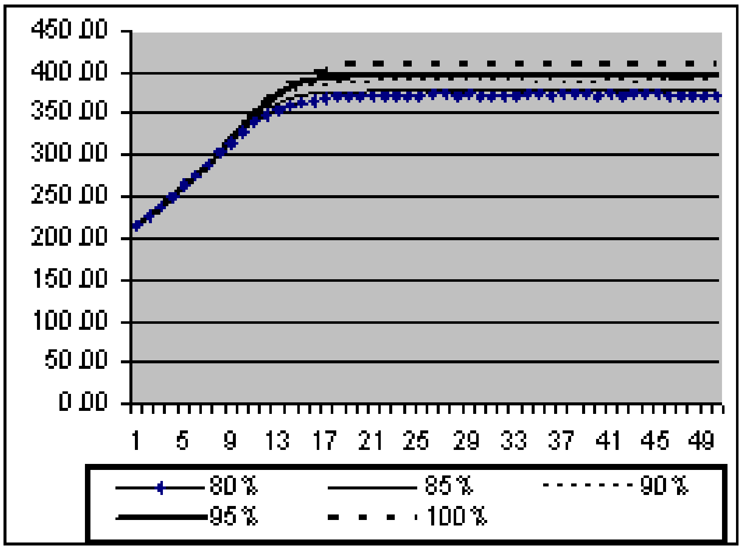


Figure 4.1: Initial Pop=200, Fecundity Based

sizes were 100 and the carrying capacity equaled 200. As seen in figures 4.3 and 4.4 (pages 41 and 42), the ranges for the 80% lower cap threshold indicated that the population can climb to the maximum carrying capacity, and in some cases exceed it.

However, the 80% lower threshold slowed the growth early enough so that population size did not exceed the cap often, or by much. Therefore, the largest population sizes in the simulations, when balanced out with the smaller simulated population size values, produced averages which were less than the maximum carrying capacity. The 100% lower threshold setting, on the other hand, did not slow down the population growth quick enough to keep the population size from going over the carrying capacity more often, and to a larger value than the 80% setting. As figures 4.5 and 4.6 (page 43) show, the range of the 100% lower threshold yearly population sizes is higher and thus average population size values for multiple simulations are higher, and subsequently closer to the desired preset maximum carrying capacity level.

One thing that stood out in the graphs was the smoothness and stability of the average population size curves. It was surprising how well the graphs

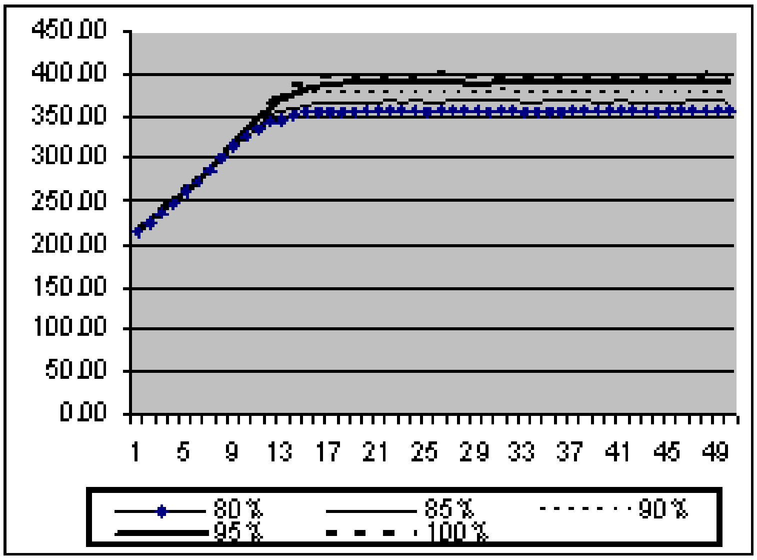


Figure 4.2: Initial Pop=200, Survival Based

maintained an almost level average population size once carrying capacity was reached. Also of interest was the change in concavity that the graphs exhibit. Logistic graphs have this property, but we had expected a much more rigid change from exponential growth to no growth for lower cap thresholds set at 100% of the carrying capacity. This expectation was based on the fact that the density dependence effects are not in effect until the maximum carrying capacity is reached. Once it is reached, the effects turn on and the population is instantly regulated so that its growth is stopped. However, our graphs show that even for lower cap thresholds set at 100% of the carrying capacity, the transition from exponential growth to no growth is smooth and gradual. This is caused by the averaging of values; individual simulations produce more pronounced fluctuations from year to year. The stochastic nature causes the population size to grow slower in some years, faster in others, and this variance allows for average population sizes which produce the curvature that is evident in the graphs.

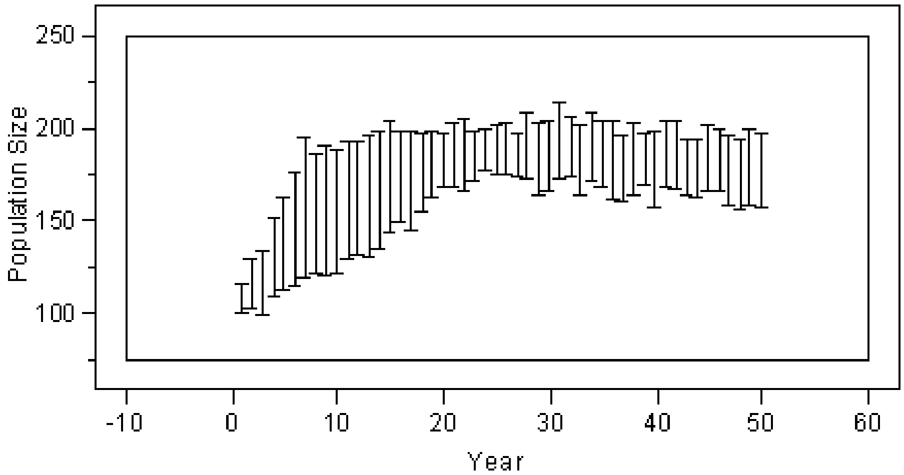


Figure 4.3: Birth rate based density dependence & 80% lower cap threshold.

## 4.4 Conclusion

We investigated whether a birth rate based or survival rate based method was better at keeping the average population size stable at the user defined carrying capacity. In order to determine this, we looked at the model's ability to produce average population sizes which, when plotted, tended to grow to the exact carrying capacity, without exceeding it, and stay there with little fluctuation. We also investigated whether or not lower cap thresholds were useful in respect to providing smooth graphs which also meet the aforementioned criteria.

Survival based models run with lower thresholds set at 100% of the maximum capacity generally produced superior results for populations greater than 26 in terms of allowing the average population size to grow to close to the carrying capacity, without exceeding it. Both models produced average population sizes which grew to within $\pm 3\%$ of the carrying capacity in this scenario. However, the survival based model's average ending population size never exceeded the carrying capacity as the fecundity based model's did, and this was one of the main concerns. So, the fecundity based model's performance for larger populations was considered slightly worse than the survival based model.

The fecundity based model did perform marginally better than the sur-

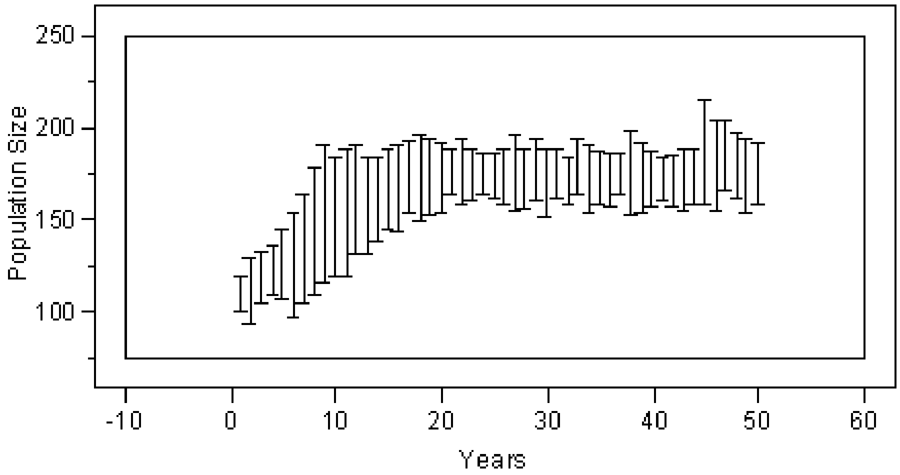


Figure 4.4: Survival rate based density dependence & 80% lower cap threshold.

vival based model with the initial population size of 26. It allowed the average population to grow to 92% of the maximum carrying capacity, whereas the survival based model reached a slightly smaller 88%.

Overall, the survival driven models seemed to be the best choice for simulating populations with density dependence issues in our model.

We investigated simulated output from the two models' with lower cap thresholds set less than and equal to 100%. Those which were less than 100% generally produced less superior results. The introduction of the lower cap threshold scalar before the carrying capacity is reached seems unnecessary. That is, the lower cap threshold kept the average population size from ever equaling the preset carrying capacity. As seen before in the 80% lower cap threshold setting (figures 4.3 & 4.4), individual simulations did go over the carrying capacity in some years. However, the average ending population size is used in our simulation model because of the underlying stochastic process. And, the 100% lower cap threshold produces average ending population size values more in line with is expected from a simulation model; the carrying capacity is eventually reached and maintained by a species with positive population growth. Therefore, a lower cap threshold which inacts density dependence effects earlier is not recommended in models like ours.

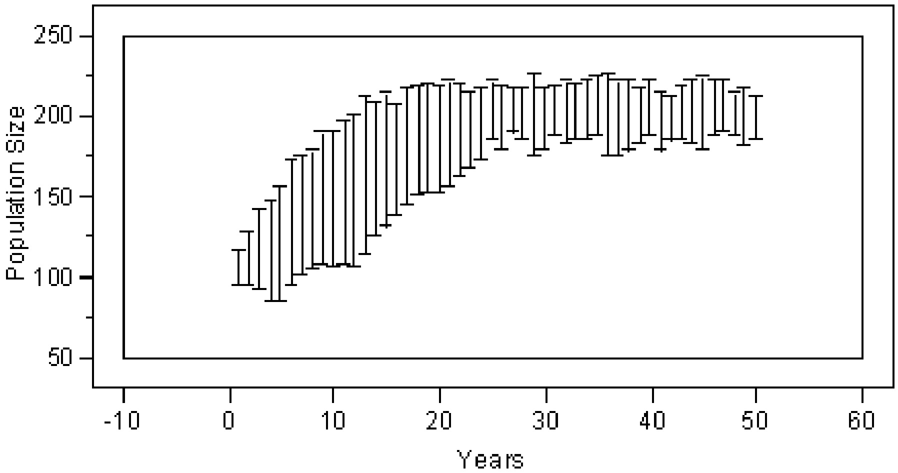


Figure 4.5: Birth rate based density dependence & 100% lower cap threshold.

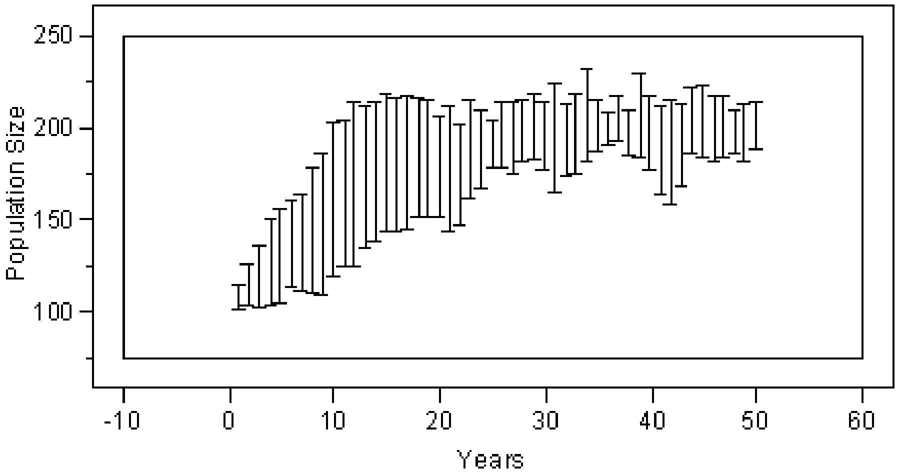


Figure 4.6: Survival rate based density dependence & 100% lower cap threshold.

# Chapter 5

# The Uniform vs Poisson Probability Question

## 5.1 Background Information

We had a question about the method used to determine catastrophe occurrence in some of the PVA packages. The question was why VORTEX and RAMAS/metapop used a uniform distribution based technique [5], [1]. The literature indicates that, in VORTEX and RAMAS/metapop, a uniform random variable is chosen at each time step to determine whether the time step will have a catastrophe. However, the Poisson distribution is used for determining the occurrence and frequency of naturally occurring events in queuing theory and statistical process control [11]. In order to address this difference, we looked at the underlying principles of the distributions themselves.

### 5.1.1 The Uniform Distribution

In determining when a catastrophe will occur, a uniform random variable is chosen stochastically and compared against the yearly probability of a catastrophe occurrence at each time step during the simulation. This yearly probability is given the value $1/c$, where $c$ the average number of years between catastrophes. This ensures that, on average, a catastrophe will occur once every $c$ years.

### 5.1.2 The Poisson Distribution

The formula for Poisson is $P_n(t) = (\lambda t)^n \cdot e^{-\lambda t}/n!$, which determines the number, $n$, occurrences of some event during time, $t$, at a rate of $\lambda$. In the catastrophe occurrence context, the formula would be used to determine the number of catastrophes in a given length of time. The desire was to use the formula to determine the probability of at least one occurrence in the next time step. By setting $n = 0$ & $t = 1$ in $P_n(t)$, the probability of no occurrences in the next time step is $e^{-\lambda}$, and the complement, $1 - e^{-\lambda}$, gives the desired probability of at least one occurrence in the next time step. So, with $\lambda = 1/c$, we obtain a yearly probability value of occurrence of $1 - e^{-1/c}$.

## 5.2 Method

We chose to compare the uniform and Poisson derived probabilities in the context of catastrophe occurrences using the value $c$ to denote the average number of years between catastrophes. We used a range of $c$ values from 5 to 100, incremented by 5's. Ratios of uniform to Poisson were formed once the probabilities were found. After comparing the yearly probability values, we will look at simulations run in Mathematica with the value of $c$ set at 25. Each simulation will record the time to an occurrence based on the yearly probabilities for 1000 simulations. The results are then used to infer why the uniform model is chosen over the Poisson in simulation modeling.

## 5.3 Results

The values obtained for the yearly uniform and Poisson based probabilities, as well as for the relative relation between them is shown in table 5.1.

| c | uniform | Poisson | $\frac{uniform}{Poisson}$ |
|---|---|---|---|
| 5 | 0.2000 | 0.1813 | 1.1033 |
| 10 | 0.1000 | 0.0952 | 1.0508 |
| 15 | 0.0667 | 0.0645 | 1.0337 |
| 20 | 0.0500 | 0.0488 | 1.0252 |
| 25 | 0.0400 | 0.0392 | 1.0201 |
| 30 | 0.0333 | 0.0328 | 1.0168 |
| 35 | 0.0286 | 0.0282 | 1.0144 |
| 40 | 0.0250 | 0.0247 | 1.0126 |
| 45 | 0.0222 | 0.0220 | 1.0112 |
| 50 | 0.0200 | 0.0198 | 1.0100 |
| 55 | 0.0182 | 0.0180 | 1.0091 |
| 60 | 0.0167 | 0.0165 | 1.0084 |
| 65 | 0.0154 | 0.0153 | 1.0077 |
| 70 | 0.0143 | 0.0142 | 1.0072 |
| 75 | 0.0133 | 0.0132 | 1.0067 |
| 80 | 0.0125 | 0.0124 | 1.0063 |
| 85 | 0.0118 | 0.0117 | 1.0059 |
| 90 | 0.0111 | 0.0110 | 1.0056 |
| 95 | 0.0105 | 0.0105 | 1.0053 |
| 100 | 0.0100 | 0.0100 | 1.0050 |

Table 5.1: Uniform and Poisson based yearly occurrence probabilities, with their relative relation.

The results from the Mathematica simulations for $c = 25$ are typical of the results for the other values of $c$ and are given in the figures 5.1 and 5.2.

## 5.4 Conclusion

The results in table 5.1 show that the values for the two approaches are nearly identical. The relative difference for values of $c$ that were chosen starts at 10.33% and decreases rapidly to less than 2% when $c$ is greater than 25. In fact, as the value of $c$ approaches infinity, the relative difference will decrease to 0%. So, the relative difference between the two approaches' yearly probabilities indicates that their performance in a simulation model

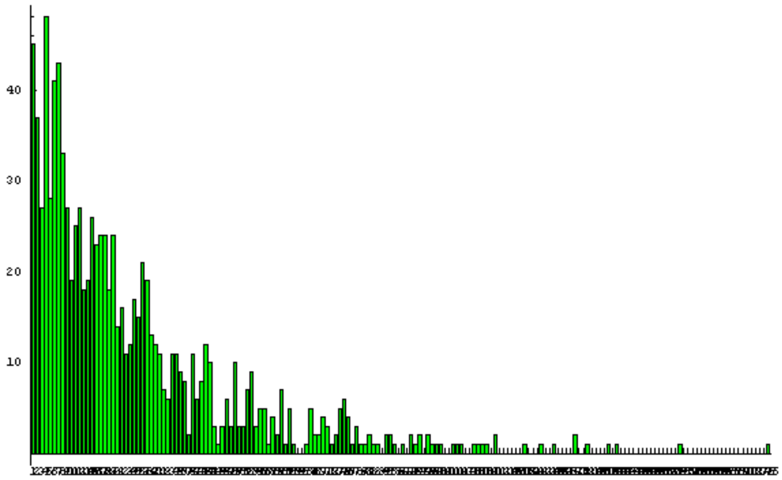


Figure 5.1: Results from 1000 simulations of the Poisson approach, where the number of years to an occurrence is tracked and recorded.

should not differ that much. Especially as typical values for $c$ are 50 or 100.

The simulations support this conclusion. As seen in the figures, there is little practical difference between the two distributions. The means & standard deviations of the uniform and Poisson distributions were reported to be 25.59 & 24.58 and 25.18 & 24.41, respectively. Both of the values are close to the expected value of 25 years before an occurrence, and the standard deviations are also nearly identical. The results from the other values of $c$ also show little difference in their shape, means, or standard deviations. So, as expected, the performance was not notably different.

From a programming perspective the uniform based model is simpler, if only in that the programmer needs to type $1/c$ instead of $1 - e^{-1/c}$ as in the Poisson case. And, in terms of computing power, $1/c$ is less demanding than $1 - e^{-1/c}$. Overall, it is now understood why the uniform distribution approach is used in simulation modeling over the Poisson distribution approach to determine catastrophe spacing.

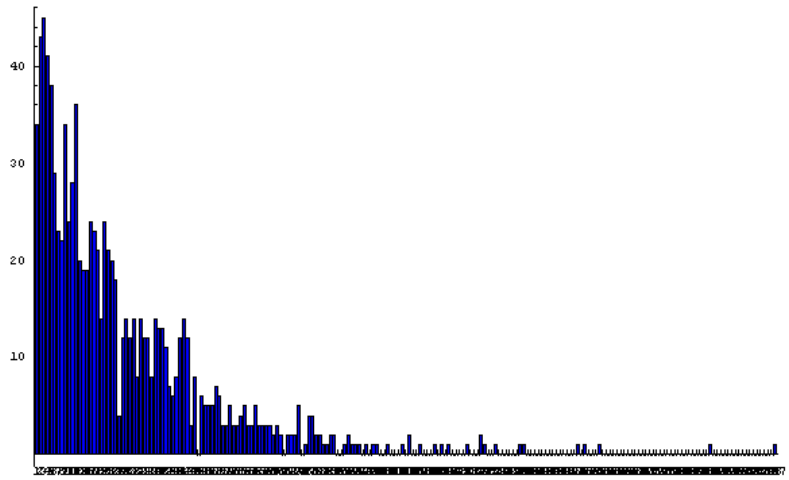


Figure 5.2: Results from 1000 simulations of the uniform approach, where the number of years to an occurrence is tracked and recorded.

# Appendix A

# The Code

The implementation code that was used to answer the questions is given below in the C++ programming language. In an effort to use less paper, the smallest script was used. A larger font version of the code is available upon request.

```
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <time.h>

using namespace std;

int C(int n, int r);
float R();
int Poisson(int t, float lambda, float CatRnd);
double fimod(int, int);

main() {
//seed random number generator
 srand(time(NULL));
 int ENV = 0;
 cout << "Will you use environmental effects? (Enter 1 for yes, 0 for no): ";
 cin >> ENV;
 cout << ENV << endl;
 int CATAS = 0;
 cout << "Will you include catastrophe effects? (Enter 1 for yes, 0 for no): ";
 cin >> CATAS;
 cout << CATAS << endl;
 int DEN = 0;
 cout << "Will you include density dependence effects? (Enter 1 for yes, 0 for no): ";
 cin >> DEN;
 cout << DEN << endl;
 int DENTYPE = 0;
 if(DEN == 1)
 {
 cout << "Will you use the birth or survival rate as a basis of density dependent adjustment?";
 cout << "(enter 0 for births or 1 for survival): ";
 cin >> DENTYPE;
 cout << DENTYPE << endl;
 }
//initialize breeding approach
//initialize the choice of breeding options
 int BREEDINGTYPE = 1;
 cout << "Do you want the model to use female population numbers only as the basis for determining " << endl;
 cout << "the number of breeders each year (enter 0), or do you want the model to base the number " << endl;
 cout << "of breeders on the minimum of the male and female population numbers (enter 1)?: ";
```

```
 cin >> BREEDINGTYPE;
 cout << BREEDINGTYPE << endl;
//random number counter
 int RandomCount=0;
//initialize the number of patches
 int PATCH = 0;
 cout << "Enter the number of patches: ";
 cin >> PATCH;
 cout << PATCH << endl;
 int PATCHMOVE = 0;
 if(PATCH > 1)
 {
  cout << "Will you include patch to patch movement? (Enter 1 for yes, 0 for no): ";
  cin >> PATCHMOVE;
  cout << PATCHMOVE << endl;
 }
//initialize maximum litter size
 int ML = 0;
 cout << "Enter the maximum litter size over all patches: ";
 cin >> ML;
 cout << ML << endl;
//determine size for litter arrays
 int LS = ML+1;
//determine birth pdf & cdf array size
 int AS = 0;
 for(int i = LS; i > 0; i--)
 {
  AS = AS + i;
 }
//intitialize the size of the survival arrays
 int AC = 0;
 cout << "Enter the number of age classes (incl. newborns and breeding adults): ";
 cin >> AC;
 cout << AC << endl;
//patch specific initialization for litter size probabilities, male/female ratio, & environment
//initialize the temporary probabilities of each litter size array
 float TEMPNYPL[PATCH][LS];
 float TEMPNYBPDF[PATCH][AS];
 float TEMPNYBCDF[PATCH][AS];
 float TEMPGYPL[PATCH][LS];
 float TEMPGYBPDF[PATCH][AS];
 float TEMPGYBCDF[PATCH][AS];
 float TEMPBYPL[PATCH][LS];
 float TEMPBYBPDF[PATCH][AS];
 float TEMPBYBCDF[PATCH][AS];
 float TEMPCYPL[PATCH][LS];
 float TEMPCYBPDF[PATCH][AS];
 float TEMPCYBCDF[PATCH][AS];
//initialize the temporary survival probabilities
 float TEMPNYFS[PATCH][AC];
 float TEMPNYMS[PATCH][AC];
 float TEMPGYFS[PATCH][AC];
 float TEMPGYMS[PATCH][AC];
 float TEMPBYFS[PATCH][AC];
 float TEMPBYMS[PATCH][AC];
 float TEMPCYFS[PATCH][AC];
 float TEMPCYMS[PATCH][AC];
//initialize the temporary expected birth rate
 float TEMPEXPBIRTH[PATCH];
//initialize the temporary female and male survival
 float TEMPFS[PATCH][AC];
 float TEMPMS[PATCH][AC];
//initialize the normal year probability of each litter size array
 float NYPL[PATCH][LS];
 float EXPBIRTH[PATCH];
 float VARBIRTH[PATCH];
//initialize the chance of being born female
 float FR[PATCH];
//initialize the  chance of being born male
 float MR[PATCH];
//initialize the environmental effect pdf and cdf for good, normal, and bad years
 float EEPDF[PATCH][3];
 float EECDF[PATCH][3];
//initialize good year effects for litter sizes other than zero
 float GYBE[PATCH][ML];
//initialize bad year effects for litter sizes other than zero
 float BYBE[PATCH][ML];
```

```
//initialize the normal year survival for each age class arrays
 float NYFS[PATCH][AC];
 float NYMS[PATCH][AC];
//initialize the good year survival for each age class arrays
 float GYFS[PATCH][AC];
 float GYMS[PATCH][AC];
 float GYPL[PATCH][LS];
 float GYBPDF[PATCH][AS];
 float GYBCDF[PATCH][AS];
//initialize the effects of a good year on survival for each age class arrays
 float GYFSE[PATCH][AC];
 float GYMSE[PATCH][AC];
//initialize the bad year survival for each age class arrays
 float BYFS[PATCH][AC];
 float BYMS[PATCH][AC];
 float BYPL[PATCH][LS];
 float BYBPDF[PATCH][AS];
 float BYBCDF[PATCH][AS];
//initialize the effects of a bad year on survival for each age class arrays
 float BYFSE[PATCH][AC];
 float BYMSE[PATCH][AC];
//Catastrophe occurrance
 float CAT[PATCH];
//initialize catastrophe year effects for litter sizes other than zero
 float CYBE[PATCH][ML];
 float CYPL[PATCH][LS];
//initialize the catastrophe year survival for each age class arrays
 float CYFS[PATCH][AC];
 float CYMS[PATCH][AC];
//initialize the effects of a catastrophe year on survival for each age class arrays
 float CYFSE[PATCH][AC];
 float CYMSE[PATCH][AC];
 float CYBCDF[PATCH][AS];
 float CYBPDF[PATCH][AS];
//initialize capacity variables
 int CAPHIGH[PATCH];
 int CAPLOW[PATCH];
 int FLOATERS[PATCH];
 float CARCAPBE[PATCH];
 float CARCAPFS[PATCH][AC];
 float CARCAPMS[PATCH][AC];
 float CARCAPMOD[PATCH];
 float TEMPONE = 0;
 float TEMPTWO = 1;
 float TEMPTHREE = 1;
 float POLYONE = 0;
 float POLYTWO = 0;
 float POLYTHREE = 0;
 float BEG = 0;
 float MID = 0;
 float END = 0;
 float CAREXPBIRTH[PATCH];
 float CARCAPFSE[PATCH][AC];
//initialize movement variables
 int TESTM[PATCH];
 int TESTF[PATCH];
 float PATCHMOVEMENTPDF[PATCH][PATCH];
 float PATCHMOVEMENTCDF[PATCH][PATCH];
 int e = AC-2;
 float MOVEMENT[PATCH];
//create array of litter probabilities
 for(int k = 0; k < PATCH; k++)
 {
  for (int i = 0; i <= ML; i++)
  {
   cout << "Enter the probability for a litter of size " << i << " in patch " << k+1 << ": ";
   float t = 0;
   cin >> t;
   cout << t << endl;
   NYPL[k][i] = t;
  }
  EXPBIRTH[k] = 0;
  for(int i = 0; i <= ML; i++)
  {
   EXPBIRTH[k] += i*NYPL[k][i];
  }
  VARBIRTH[k] = 0;
```

```
  for(int i = 0; i <= ML; i++)
  {
   VARBIRTH[k] += pow(i,2.0)*NYPL[k][i];
  }
  VARBIRTH[k] = VARBIRTH[k]-pow(EXPBIRTH[k],2);
  float t = 0;
  cout << "Enter the percentage chance of being born a female in patch " << k+1 << ": ";
  cin >> t;
  cout << t << endl;
  FR[k] = t;
  MR[k] = 1 - FR[k];
  if(ENV == 0)
  {
   EEPDF[k][1] = 1;
  }
  else
  {
//fill in the environmental effect pdf and cdf for good, normal, and bad years
   float n = 0;
   cout << "Enter the probability of a normal environmental year in patch " << k+1 << ": ";
   cin >> n;
   cout << n << endl;
   EEPDF[k][1] = n;
  }
//fill in the normal year survival arrays
  for(int i = 0; i < AC; i++)
  {
   cout << "Enter the normal year probability of survival for females in age class " <<i << " in patch " <<k+1 << ": ";
   float t = 0;
   cin >> t;
   cout << t << endl;
   NYFS[k][i] = t;
   cout << "Enter the normal year probability of survival for males in age class " << i << " in patch " <<k+1 << ": ";
   t = 0;
   cin >> t;
   cout << t << endl;
   NYMS[k][i] = t;
  }
  if(ENV == 0)
  {
   EEPDF[k][0] = 0;
   EEPDF[k][2] = 0;
  }
  else
  {
   float g = 0;
   cout << "Enter the probability of a good environmental year in patch " << k+1 << ": ";
   cin >> g;
   cout << g << endl;
   EEPDF[k][0] = g;
   for(int i = 1; i <= ML ; i++)
   {
    cout << "Enter the good year modifier for a litter of size " << i << " in patch " << k+1 << ": ";
    float t = 0;
    cin >> t;
    cout << t << endl;
    GYBE[k][i-1] = t;
   }
//fill in the effect arrays and adjust the survival rates for a good year
   for(int i = 0; i < AC; i++)
   {
    cout << "Enter the good yr effect on survival for females in age class " << i << " in patch " << k+1 << ": ";
    float t = 0;
    cin >> t;
    cout << t << endl;
    GYFSE[k][i] = t;
    cout << "Enter the good yr effect on survival for males in age class " << i << " in patch " << k+1 << ": ";
    t = 0;
    cin >> t;
    cout << t << endl;
    GYMSE[k][i] = t;
   }
   for(int i = 0; i < AC; i++)
   {
    GYFS[k][i] = NYFS[k][i]*GYFSE[k][i];
    GYMS[k][i] = NYMS[k][i]*GYMSE[k][i];
   }
```

```
    float b = 0;
    cout << "Enter the probability of a bad environmental year in patch " << k+1 << ": ";
    cin >> b;
    cout << b << endl;
    EEPDF[k][2] = b;
    for(int i = 1; i <= ML ; i++)
    {
     cout << "Enter the bad year modifier for a litter of size " << i << " in patch " << k+1 << ": ";
     float t = 0;
     cin >> t;
     cout << t << endl;
     BYBE[k][i-1] = t;
    }
//fill in the effect arrays and adjust the survival rates for a bad year
    for(int i = 0; i < AC; i++)
    {
     cout << "Enter the bad yr effect on survival for females in age class " << i << " in patch " << k+1 << ": ";
     float t = 0;
     cin >> t;
     cout << t << endl;
     BYFSE[k][i] = t;
     cout << "Enter the bad yr effect on survival for males in age class " << i << " in patch " << k+1 << ": ";
     t = 0;
     cin >> t;
     cout << t << endl;
     BYMSE[k][i] = t;
    }
    for(int i = 0; i < AC; i++)
    {
     BYFS[k][i] = NYFS[k][i]*BYFSE[k][i];
     BYMS[k][i] = NYMS[k][i]*BYMSE[k][i];
    }
   }
//cdf's
  for(int i = 2; i >= 0; i--)
  {
   float t = 0;
   for(int j = 0; j <= i; j++)
   {
    t = t + EEPDF[k][j];
   }
   EECDF[k][i] = t;
  }
//catastrophes
  if(CATAS == 1)
  {
   cout << "Enter the average number of catastrophes per year in patch " << k+1 << ": ";
   cin >> CAT[k];
   cout << CAT[k] << endl;
   for(int i = 1; i <= ML ; i++)
   {
    cout << "Enter the catastrophe year modifier for a litter of size " << i << " in patch " << k+1 << ": ";
    float t = 0;
    cin >> t;
    cout << t << endl;
    CYBE[k][i-1] = t;
   }
//fill in the effect arrays and adjust the survival rates for a catastrophe year
   for(int i = 0; i < AC; i++)
   {
    cout << "Enter the cat. yr effect on survival for females in age class " << i << " in patch " << k+1 << ": ";
    float t = 0;
    cin >> t;
    cout << t << endl;
    CYFSE[k][i] = t;
    cout << "Enter the cat. yr effect on survival for males in age class " << i << " in patch " << k+1 << ": ";
    t = 0;
    cin >> t;
    cout << t << endl;
    CYMSE[k][i] = t;
   }
   for(int i = 0; i < AC; i++)
   {
    CYFS[k][i] = NYFS[k][i]*CYFSE[k][i];
    CYMS[k][i] = NYMS[k][i]*CYMSE[k][i];
   }
  }
```

```
//carrying capacity initialization
  if(DEN == 1)
  {
   cout << "Enter the carrying capacity for patch " << k+1 << ": ";
   cin >> CAPHIGH[k];
   cout << CAPHIGH[k] << endl;
   cout << "Enter the population size when the first effects of the carrying capacity will be inacted ";
   cout << "in patch " << k+1 << ": ";
   cin >> CAPLOW[k];
   cout << CAPLOW[k] << endl;
   if(DENTYPE == 0)
   {
    TEMPONE = EXPBIRTH[k]/2;
    if(AC == 2)
    {
     CARCAPMOD[k] = (1-NYFS[k][AC-1])/(NYFS[k][AC-2]*TEMPONE);
     CAREXPBIRTH[k] = CARCAPMOD[k]*EXPBIRTH[k];
    }
    else
    {
     for(int i = 0; i < e; i++)
     {
      TEMPTWO = TEMPTWO*NYFS[k][i];
     }
     CARCAPMOD[k] = (1-NYFS[k][AC-1])/(TEMPTWO*TEMPONE*(1+NYFS[k][AC-2]-NYFS[k][AC-1]));
     CAREXPBIRTH[k] = CARCAPMOD[k]*EXPBIRTH[k];
    }
   }
   if(DENTYPE == 1)
   {
    TEMPONE = EXPBIRTH[k]/2;
    if(AC == 2)
    {
     CARCAPMOD[k] = 1/(NYFS[k][AC-1]+NYFS[k][AC-2]*TEMPONE);
     for(int i = 0; i < AC; i++)
     {
      CARCAPFS[k][i] = NYFS[k][i]*CARCAPMOD[k];
     }
    }
    else
    {
     for(int i = 0; i < e; i++)
     {
      TEMPTWO = TEMPTWO*NYFS[k][i];
     }
     for(int i = 0; i < (AC-1); i++)
     {
      TEMPTHREE = TEMPTHREE*NYFS[k][i];
     }
     if(fimod(AC,2) == 1)
     {
      BEG = 0;
      MID = 0.5;
      END = 1;
      POLYONE = pow(BEG,AC-1)*(TEMPTHREE*TEMPONE-TEMPTWO*NYFS[k][AC-1]*TEMPONE);
      POLYONE = POLYONE+pow(BEG,AC-2)*(TEMPTWO*TEMPONE)+BEG*NYFS[k][AC-1]-1;
      POLYTWO = pow(MID,AC-1)*(TEMPTHREE*TEMPONE-TEMPTWO*NYFS[k][AC-1]*TEMPONE);
      POLYTWO = POLYTWO+pow(MID,AC-2)*(TEMPTWO*TEMPONE)+MID*NYFS[k][AC-1]-1;
      POLYTHREE = pow(END,AC-1)*(TEMPTHREE*TEMPONE-TEMPTWO*NYFS[k][AC-1]*TEMPONE);
      POLYTHREE = POLYTHREE+pow(END,AC-2)*(TEMPTWO*TEMPONE)+END*NYFS[k][AC-1]-1;
      while(POLYTWO > 0.0001 || POLYTWO < -0.0001)
      {
       if(POLYONE*POLYTWO > 0)
       {
        BEG = MID;
        MID = (BEG+END)/2;
        POLYONE = pow(BEG,AC-1)*(TEMPTHREE*TEMPONE-TEMPTWO*NYFS[k][AC-1]*TEMPONE);
        POLYONE = POLYONE+pow(BEG,AC-2)*(TEMPTWO*TEMPONE)+BEG*NYFS[k][AC-1]-1;
        POLYTWO = pow(MID,AC-1)*(TEMPTHREE*TEMPONE-TEMPTWO*NYFS[k][AC-1]*TEMPONE);
        POLYTWO = POLYTWO+pow(MID,AC-2)*(TEMPTWO*TEMPONE)+MID*NYFS[k][AC-1]-1;
        POLYTHREE = pow(END,AC-1)*(TEMPTHREE*TEMPONE-TEMPTWO*NYFS[k][AC-1]*TEMPONE);
        POLYTHREE = POLYTHREE+pow(END,AC-2)*(TEMPTWO*TEMPONE)+END*NYFS[k][AC-1]-1;
       }
       else
       {
        END = MID;
        MID = (BEG+END)/2;
```

```
       POLYONE = pow(BEG,AC-1)*(TEMPTHREE*TEMPONE-TEMPTWO*NYFS[k][AC-1]*TEMPONE);
       POLYONE = POLYONE+pow(BEG,AC-2)*(TEMPTWO*TEMPONE)+BEG*NYFS[k][AC-1]-1;
       POLYTWO = pow(MID,AC-1)*(TEMPTHREE*TEMPONE-TEMPTWO*NYFS[k][AC-1]*TEMPONE);
       POLYTWO = POLYTWO+pow(MID,AC-2)*(TEMPTWO*TEMPONE)+MID*NYFS[k][AC-1]-1;
       POLYTHREE = pow(END,AC-1)*(TEMPTHREE*TEMPONE-TEMPTWO*NYFS[k][AC-1]*TEMPONE);
       POLYTHREE = POLYTHREE+pow(END,AC-2)*(TEMPTWO*TEMPONE)+END*NYFS[k][AC-1]-1;
      }
     }
     CARCAPMOD[k] = MID;
     for(int i = 0; i < AC; i++)
     {
      CARCAPFS[k][i] = NYFS[k][i]*CARCAPMOD[k];
     }
    }
    else
    {
     BEG = 0;
     MID = 0.5;
     END = 1;
     POLYONE = pow(BEG,AC-1)*(TEMPTWO*NYFS[k][AC-1]*TEMPONE-TEMPTHREE*TEMPONE);
     POLYONE = POLYONE-pow(BEG,AC-2)*(TEMPTWO*TEMPONE)-BEG*NYFS[k][AC-1]+1;
     POLYTWO = pow(MID,AC-1)*(TEMPTWO*NYFS[k][AC-1]*TEMPONE-TEMPTHREE*TEMPONE);
     POLYTWO = POLYTWO-pow(MID,AC-2)*(TEMPTWO*TEMPONE)-MID*NYFS[k][AC-1]+1;
     POLYTHREE = pow(END,AC-1)*(TEMPTWO*NYFS[k][AC-1]*TEMPONE-TEMPTHREE*TEMPONE);
     POLYTHREE = POLYTHREE-pow(END,AC-2)*(TEMPTWO*TEMPONE)-END*NYFS[k][AC-1]+1;
     while(POLYTWO > 0.0001 || POLYTWO < -0.0001)
     {
      if(POLYONE*POLYTWO > 0)
      {
       BEG = MID;
       MID = (BEG+END)/2;
       POLYONE = pow(BEG,AC-1)*(TEMPTWO*NYFS[k][AC-1]*TEMPONE-TEMPTHREE*TEMPONE);
       POLYONE = POLYONE-pow(BEG,AC-2)*(TEMPTWO*TEMPONE)-BEG*NYFS[k][AC-1]+1;
       POLYTWO = pow(MID,AC-1)*(TEMPTWO*NYFS[k][AC-1]*TEMPONE-TEMPTHREE*TEMPONE);
       POLYTWO = POLYTWO-pow(MID,AC-2)*(TEMPTWO*TEMPONE)-MID*NYFS[k][AC-1]+1;
       POLYTHREE = pow(END,AC-1)*(TEMPTWO*NYFS[k][AC-1]*TEMPONE-TEMPTHREE*TEMPONE);
       POLYTHREE = POLYTHREE-pow(END,AC-2)*(TEMPTWO*TEMPONE)-END*NYFS[k][AC-1]+1;
      }
      else
      {
       END = MID;
       MID = (BEG+END)/2;
       POLYONE = pow(BEG,AC-1)*(TEMPTWO*NYFS[k][AC-1]*TEMPONE-TEMPTHREE*TEMPONE);
       POLYONE = POLYONE-pow(BEG,AC-2)*(TEMPTWO*TEMPONE)-BEG*NYFS[k][AC-1]+1;
       POLYTWO = pow(MID,AC-1)*(TEMPTWO*NYFS[k][AC-1]*TEMPONE-TEMPTHREE*TEMPONE);
       POLYTWO = POLYTWO-pow(MID,AC-2)*(TEMPTWO*TEMPONE)-MID*NYFS[k][AC-1]+1;
       POLYTHREE = pow(END,AC-1)*(TEMPTWO*NYFS[k][AC-1]*TEMPONE-TEMPTHREE*TEMPONE);
       POLYTHREE = POLYTHREE-pow(END,AC-2)*(TEMPTWO*TEMPONE)-END*NYFS[k][AC-1]+1;
      }
     }
     CARCAPMOD[k] = MID;
     for(int i = 0; i < AC; i++)
     {
      CARCAPFS[k][i] = NYFS[k][i]*CARCAPMOD[k];
     }
    }
   }
  }
 }
//movement initialization
 if(PATCHMOVE == 1)
 {
  for(int i = 0; i < PATCH; i++)
  {
   if(k == i)
   {
    PATCHMOVEMENTPDF[k][i] = 0;
   }
   else
   {
    cout << "Enter the probability of moving from patch " << k+1 << " to patch " << i+1 << ": ";
    cin >> PATCHMOVEMENTPDF[k][i];
    cout << PATCHMOVEMENTPDF[k][i] << endl;
   }
  }
 }
}
```

```
//**************The Birth Cycle Initialization***************************************
//NORMAL YEAR BIRTH
//initialize the normal year birth PDF array
 float NYBPDF[PATCH][AS];
//initialize the array which will be used to get the male & female babies per litter
 int BABIES[PATCH][2*AS];
 for(int k = 0; k < PATCH; k++)
//fill in the normal year birth array & the babies per litter array
 {
  NYBPDF[k][0] = NYPL[k][0];
  TEMPNYBPDF[k][0] = NYPL[k][0];
  BABIES[k][0] = 0;
  BABIES[k][1] = 0;
  int c1 = 1;
  for(int i = 1; i <= ML; i++)
  {
   for(int j = 0; j <= i; j++)
   {
    float t = 0;
    t = NYPL[k][i]*C(i,j)*pow(MR[k],j)*pow(FR[k],(i-j));
    NYBPDF[k][c1] = t;
    TEMPNYBPDF[k][c1] = t;
    BABIES[k][2*c1] = j;
    BABIES[k][2*c1+1] = i-j;
    c1++;
   }
  }
 }
//create Normal Year CDF array
 float NYBCDF[PATCH][AS];
 for(int k = 0; k < PATCH; k++)
 {
  for(int i =(AS-1); i >= 0; i--)
  {
   float t = 0;
   for(int j = 0; j <= i; j++)
   {
    t = t + NYBPDF[k][j];
   }
   NYBCDF[k][i] = t;
   TEMPNYBCDF[k][i] = t;
  }
 }

//GOOD YEAR EFFECTS ON BIRTH
 if(ENV == 1)
 {
//initialize and modify good year litter probability array
  for(int k = 0; k < PATCH; k++)
  {
   float t1 = 0;
   for(int i = ML; i > 0; i--)
   {
    GYPL[k][i] = GYBE[k][i-1]*NYPL[k][i];
    t1 += GYPL[k][i];
   }
   GYPL[k][0] = 1-t1;
  }
//initialize and fill in the good year birth PDF array
  for(int k = 0; k < PATCH; k++)
  {
   GYBPDF[k][0] = GYPL[k][0];
   TEMPGYBPDF[k][0] = GYPL[k][0];
   int c2 = 1;
   for(int i = 1; i <= ML; i++)
   {
    for(int j = 0; j <= i; j++)
    {
     float t = 0;
     t = GYPL[k][i]*C(i,j)*pow(MR[k],j)*pow(FR[k],(i-j));
     GYBPDF[k][c2] = t;
     TEMPGYBPDF[k][c2] = t;
     c2++;
    }
   }
  }
```

```
//create good year CDF array
  for(int k = 0; k < PATCH; k++)
  {
   for(int i =(AS-1); i >= 0; i--)
   {
    float t = 0;
    for(int j = 0; j <= i; j++)
    {
     t = t + GYBPDF[k][j];
    }
    GYBCDF[k][i] = t;
    TEMPGYBCDF[k][i] = t;
   }
  }

//BAD YEAR EFFECTS ON BIRTH
//initialize and modify bad year litter probability array
  for(int k = 0; k < PATCH; k++)
  {
   float t2 = 0;
   for(int i = ML; i > 0; i--)
   {
    BYPL[k][i] = BYBE[k][i-1]*NYPL[k][i];
    t2 += BYPL[k][i];
   }
   BYPL[k][0] = 1-t2;
  }
//initialize and fill in the bad year birth PDF array
  for(int k = 0; k < PATCH; k++)
  {
   BYBPDF[k][0] = BYPL[k][0];
   TEMPBYBPDF[k][0] = BYPL[k][0];
   int c3 = 1;
   for(int i = 1; i <= ML; i++)
   {
    for(int j = 0; j <= i; j++)
    {
     float t = 0;
     t = BYPL[k][i]*C(i,j)*pow(MR[k],j)*pow(FR[k],(i-j));
     BYBPDF[k][c3] = t;
     TEMPBYBPDF[k][c3] = t;
     c3++;
    }
   }
  }
//create bad year CDF array
  for(int k = 0; k < PATCH; k++)
  {
   for(int i =(AS-1); i >= 0; i--)
   {
    float t = 0;
    for(int j = 0; j <= i; j++)
    {
     t = t + BYBPDF[k][j];
    }
    BYBCDF[k][i] = t;
    TEMPBYBCDF[k][i] = t;
   }
  }
 }
//CATASTROPHE YEAR EFFECTS ON BIRTH
 if(CATAS == 1)
 {
//initialize and modify catastrophe year litter probability array
  for(int k = 0; k < PATCH; k++)
  {
   float t3 = 0;
   for(int i = ML; i > 0; i--)
   {
    CYPL[k][i] = CYBE[k][i-1]*NYPL[k][i];
    t3 += CYPL[k][i];
   }
   CYPL[k][0] = 1-t3;
  }
//initialize and fill in the catastrophe year birth PDF array
  for(int k = 0; k < PATCH; k++)
  {
```

```
    CYBPDF[k][0] = CYPL[k][0];
    TEMPCYBPDF[k][0] = CYPL[k][0];
    int c4 = 1;
    for(int i = 1; i <= ML; i++)
    {
     for(int j = 0; j <= i; j++)
     {
      float t = 0;
      t = CYPL[k][i]*C(i,j)*pow(MR[k],j)*pow(FR[k],(i-j));
      CYBPDF[k][c4] = t;
      TEMPCYBPDF[k][c4] = t;
      c4++;
     }
    }
   }
 //create catastrophe year CDF array
   for(int k = 0; k < PATCH; k++)
   {
    for(int i =(AS-1); i >= 0; i--)
    {
     float t = 0;
     for(int j = 0; j <= i; j++)
     {
      t = t + CYBPDF[k][j];
     }
     CYBCDF[k][i] = t;
     TEMPCYBCDF[k][i] = t;
    }
   }
  }
 //create patch to patch movement cdf
  if(PATCHMOVE == 1)
  {
   for(int k = 0; k < PATCH; k++)
   {
    for(int i = (PATCH-1); i >= 0; i--)
    {
     float t = 0;
     for(int j = 0; j <= i; j++)
     {
      t = t + PATCHMOVEMENTPDF[k][j];
     }
     PATCHMOVEMENTCDF[k][i] = t;
    }
   }
  }
 //************************THE MAIN SIMULATION LOOP**************************************
 //initialize the population-by-age-class arrays for both males and females
  int FPOPINIT[PATCH][AC];
  int MPOPINIT[PATCH][AC];
 //fill in the population arrays
  for(int k = 0; k < PATCH; k++)
  {
   for(int i = 0; i < AC; i++)
   {
    int t = 0;
    cout << "Enter the number of females in age class " << i << " in patch " << k+1 << ": ";
    cin >> t;
    cout << t << endl;
    FPOPINIT[k][i] = t;
    cout << "Enter the number of males in age class " << i << " in patch " << k+1 << ": ";
    cin >> t;
    cout << t << endl;
    MPOPINIT[k][i] = t;
   }
  }
 //initialize the number of times to be simulated
  int SIMCNT = 0;
  cout << "Enter the number of independent simulations to be run: ";
  cin >> SIMCNT;
  cout << SIMCNT << endl;
 //initialize the years to be simulated
  int YRCNT = 0;
  cout << "Enter the number of years to be simulated in each simulation run: ";
  cin >> YRCNT;
  cout << YRCNT << endl;
  int YEARLYPOPTOT[PATCH][SIMCNT][YRCNT];
```

```
 for(int k = 0; k < PATCH; k++)
 {
  for(int b = 0; b < SIMCNT; b++)
  {
   for(int d = 0; d < YRCNT; d++)
   {
    YEARLYPOPTOT[k][b][d]=0;
   }
  }
 }
 float YEARLYPOPAVE[PATCH][YRCNT];
 for(int k = 0; k < PATCH; k++)
 {
  for(int d = 0; d < YRCNT; d++)
  {
   YEARLYPOPAVE[k][d]=0;
  }
 }
 int EXT = 0;
 int TOTALPOP = 0;
 int TOTPOP[PATCH];
 int FTOT[PATCH];
 int MTOT[PATCH];
 float AVERAGEPOP[PATCH];
 float POPAVE[PATCH][SIMCNT];
 float AVEFEMPOP[PATCH];
 float AVEMALPOP[PATCH];
 float FEMACPOP[AC];
 float MALACPOP[AC];
 int FPOP[PATCH][AC];
 int MPOP[PATCH][AC];
 for(int k = 0; k <PATCH; k++)
 {
  for(int l = 1; l <= SIMCNT; l++)
  {
   POPAVE[k][l-1] = 0;
  }

  AVERAGEPOP[k] = 0;
  AVEFEMPOP[k] = 0;
  AVEMALPOP[k] = 0;
  TOTPOP[k] = 0;
  FTOT[k] = 0;
  MTOT[k] = 0;
 }
//enter simulation loop
 for(int l = 1; l <= SIMCNT; l++)
 {
  for(int k = 0; k < PATCH; k++)
  {
   TOTPOP[k] = 0;
   for(int i = 0; i < AC; i++)
   {
    FPOP[k][i] = FPOPINIT[k][i];
    MPOP[k][i] = MPOPINIT[k][i];
    TOTPOP[k] += FPOPINIT[k][i];
    TOTPOP[k] += MPOPINIT[k][i];
   }
  }
  int CATCNT = 1;
  float CATRAN = R();
//enter yearly loop
  for(int i = 1; i <= YRCNT; i++)
  {
   float RNDNUM;
//patch specific population adjustments
   for(int k = 0; k < PATCH; k++)
   {
    for(int j = (AC-1); j > 0; j--)
    {
     FPOP[k][j] += FPOP[k][j-1];
     FPOP[k][j-1] = 0;
     MPOP[k][j] += MPOP[k][j-1];
     MPOP[k][j-1] = 0;
    }
    for(int j = 0; j < AC; j++)
    {
```

```
     TEMPNYFS[k][j] = NYFS[k][j];
     TEMPNYMS[k][j] = NYMS[k][j];
     TEMPGYFS[k][j] = GYFS[k][j];
     TEMPGYMS[k][j] = GYMS[k][j];
     TEMPBYFS[k][j] = BYFS[k][j];
     TEMPBYMS[k][j] = BYMS[k][j];
     TEMPCYFS[k][j] = CYFS[k][j];
     TEMPCYMS[k][j] = CYMS[k][j];
    }
    for(int j = 0; j < AS; j++)
    {
     TEMPNYBCDF[k][j] = NYBCDF[k][j];
     TEMPGYBCDF[k][j] = GYBCDF[k][j];
     TEMPBYBCDF[k][j] = BYBCDF[k][j];
     TEMPCYBCDF[k][j] = CYBCDF[k][j];
    }
    //determine carrying capacity effects on births and survival
    if(TOTPOP[k] > CAPLOW[k])
    {
     if(DEN == 1)
     {
//birth rate based
      if(DENTYPE == 0)
      {
       if(TOTPOP[k] < CAPHIGH[k])
       {
        TEMPEXPBIRTH[k] = (EXPBIRTH[k]-CAREXPBIRTH[k])/(CAPLOW[k]-CAPHIGH[k])*(TOTPOP[k]-CAPHIGH[k])+CAREXPBIRTH[k];
        CARCAPBE[k] = TEMPEXPBIRTH[k]/EXPBIRTH[k];
       }
       else
       {
        CARCAPBE[k] = CAREXPBIRTH[k]/EXPBIRTH[k];
       }
      }
//survival based
      else
      {
       if(TOTPOP[k] < CAPHIGH[k])
       {
        for(int a = 0; a < AC; a++)
        {
         TEMPFS[k][a] = (NYFS[k][a]-CARCAPFS[k][a])/(CAPLOW[k]-CAPHIGH[k])*(TOTPOP[k]-CAPHIGH[k])+CARCAPFS[k][a];
         CARCAPFSE[k][a] = TEMPFS[k][a]/NYFS[k][a];
        }
       }
       else
       {
        for(int a = 0; a < AC; a++)
        {
         CARCAPFSE[k][a] = CARCAPFS[k][a]/NYFS[k][a];
        }
       }
       for(int a = 0; a < AC; a++)
       {
        TEMPNYFS[k][a] = CARCAPFSE[k][a]*TEMPNYFS[k][a];
        TEMPGYFS[k][a] = CARCAPFSE[k][a]*TEMPGYFS[k][a];
        TEMPBYFS[k][a] = CARCAPFSE[k][a]*TEMPBYFS[k][a];
        TEMPCYFS[k][a] = CARCAPFSE[k][a]*TEMPCYFS[k][a];
        TEMPNYMS[k][a] = CARCAPFSE[k][a]*TEMPNYMS[k][a];
        TEMPGYMS[k][a] = CARCAPFSE[k][a]*TEMPGYMS[k][a];
        TEMPBYMS[k][a] = CARCAPFSE[k][a]*TEMPBYMS[k][a];
        TEMPCYMS[k][a] = CARCAPFSE[k][a]*TEMPCYMS[k][a];
       }
      }
//normal
      if(DENTYPE == 0)
      {
       float ct = 0;
       for(int a = 1; a <= ML; a++)
       {
        TEMPNYPL[k][a] = CARCAPBE[k]*NYPL[k][a];
        ct += TEMPNYPL[k][a];
       }
       TEMPNYPL[k][0] = 1 - ct;
       TEMPNYBPDF[k][0] = TEMPNYPL[k][0];
       int c = 1;
       for(int g = 1; g <= ML; g++)
```

```
        {
         for(int j = 0; j <= g; j++)
         {
          float t = 0;
          t = TEMPNYPL[k][g]*C(g,j)*pow(MR[k],j)*pow(FR[k],(g-j));
          TEMPNYBPDF[k][c] = t;
          c++;
         }
        }
//create carrying capacity CDF array
        for(int g =(AS-1); g >= 0; g--)
        {
         float t = 0;
         for(int j = 0; j <= g; j++)
         {
          t = t + TEMPNYBPDF[k][j];
         }
         TEMPNYBCDF[k][g] = t;
        }
//good
        ct = 0;
        for(int a = 1; a <= ML; a++)
        {
         TEMPGYPL[k][a] = CARCAPBE[k]*GYPL[k][a];
         ct += TEMPGYPL[k][a];
        }
        TEMPGYPL[k][0] = 1 - ct;
        TEMPGYBPDF[k][0] = TEMPGYPL[k][0];
        c = 1;
        for(int g = 1; g <= ML; g++)
        {
         for(int j = 0; j <= g; j++)
         {
          float t = 0;
          t = TEMPGYPL[k][g]*C(g,j)*pow(MR[k],j)*pow(FR[k],(g-j));
          TEMPGYBPDF[k][c] = t;
          c++;
         }
        }
//create carrying capacity CDF array
        for(int g =(AS-1); g >= 0; g--)
        {
         float t = 0;
         for(int j = 0; j <= g; j++)
         {
          t = t + TEMPGYBPDF[k][j];
         }
         TEMPGYBCDF[k][g] = t;
        }
//bad
        ct = 0;
        for(int a = 1; a <= ML; a++)
        {
         TEMPBYPL[k][a] = CARCAPBE[k]*BYPL[k][a];
         ct += TEMPBYPL[k][a];
        }
        TEMPBYPL[k][0] = 1 - ct;
        TEMPBYBPDF[k][0] = TEMPBYPL[k][0];
        c = 1;
        for(int g = 1; g <= ML; g++)
        {
         for(int j = 0; j <= g; j++)
         {
          float t = 0;
          t = TEMPBYPL[k][g]*C(g,j)*pow(MR[k],j)*pow(FR[k],(g-j));
          TEMPBYBPDF[k][c] = t;
          c++;
         }
        }
//create carrying capacity CDF array
        for(int g =(AS-1); g >= 0; g--)
        {
         float t = 0;
         for(int j = 0; j <= g; j++)
         {
          t = t + TEMPBYBPDF[k][j];
         }
```

```
        TEMPBYBCDF[k][g] = t;
       }
//catastrophe
       ct = 0;
       for(int a = 1; a <= ML; a++)
       {
        TEMPCYPL[k][a] = CARCAPBE[k]*CYPL[k][a];
        ct += TEMPCYPL[k][a];
       }
       TEMPCYPL[k][0] = 1 - ct;
       TEMPCYBPDF[k][0] = TEMPCYPL[k][0];
       c = 1;
       for(int g = 1; g <= ML; g++)
       {
        for(int j = 0; j <= g; j++)
        {
         float t = 0;
         t = TEMPCYPL[k][g]*C(g,j)*pow(MR[k],j)*pow(FR[k],(g-j));
         TEMPCYBPDF[k][c] = t;
         c++;
        }
       }
//create carrying capacity CDF array
       for(int g =(AS-1); g >= 0; g--)
       {
        float t = 0;
        for(int j = 0; j <= g; j++)
        {
         t = t + TEMPCYBPDF[k][j];
        }
        TEMPCYBCDF[k][g] = t;
       }
      }
     }
    }
//determine breeding pair number (this is based on a monogamous species)
    int BREEDERS = 0;
    if(BREEDINGTYPE == 0)
    {
     BREEDERS = FPOP[k][AC-1];
    }
    else
    {
     if(FPOP[k][AC-1] < MPOP[k][AC-1])
     {
      BREEDERS = FPOP[k][AC-1];
     }
     else
     {
      BREEDERS = MPOP[k][AC-1];
     }
    }
//determine the type of environment for the year
    int ENVTYPE;
    if(ENV == 0)
    {
     ENVTYPE = 1;
    }
    else
    {
     int CATTEST = Poisson(CATCNT, CAT[k], CATRAN);
     if(CATTEST == 1)
     {
      ENVTYPE = 3;
      CATCNT = 1;
      CATRAN = R();
     }
     else
     {
      CATCNT++;
      RNDNUM = R();
      RandomCount++;
      for(int j = 0; j < 3; j++)
      {
       if(RNDNUM <= EECDF[k][j])
       {
        ENVTYPE = j;
```

```
        break;
       }
      }
     }
    }
//determine the number of newborns
    for(int j = 1; j <= BREEDERS; j++)
    {
     RNDNUM = R();
     RandomCount++;
     for(int h = 0; h < AS; h++)
     {
      if(ENVTYPE == 0)
      {
       if(RNDNUM <= TEMPGYBCDF[k][h])
       {
        FPOP[k][0] += BABIES[k][2*h+1];
        MPOP[k][0] += BABIES[k][2*h];
        break;
       }
      }
      if(ENVTYPE == 1)
      {
       if(RNDNUM <= TEMPNYBCDF[k][h])
       {
        FPOP[k][0] += BABIES[k][2*h+1];
        MPOP[k][0] += BABIES[k][2*h];
        break;
       }
      }
      if(ENVTYPE == 2)
      {
       if(RNDNUM <= TEMPBYBCDF[k][h])
       {
        FPOP[k][0] += BABIES[k][2*h+1];
        MPOP[k][0] += BABIES[k][2*h];
        break;
       }
      }
      if(ENVTYPE == 3)
      {
       if(RNDNUM <= TEMPCYBCDF[k][h])
       {
        FPOP[k][0] += BABIES[k][2*h+1];
        MPOP[k][0] += BABIES[k][2*h];
        break;
       }
      }
     }
    }
//determine the number of female and male survivors
    for(int j = 0; j < AC; j++)
    {
     int t = FPOP[k][j];
     for(int h = t; h > 0; h--)
     {
      RNDNUM = R();
      RandomCount++;
      if(ENVTYPE == 0)
      {
       if(RNDNUM >= TEMPGYFS[k][j])
       {
        FPOP[k][j] -= 1;
       }
      }
      if(ENVTYPE == 1)
      {
       if(RNDNUM >= TEMPNYFS[k][j])
       {
        FPOP[k][j] -= 1;
       }
      }
      if(ENVTYPE == 2)
      {
       if(RNDNUM >= TEMPBYFS[k][j])
       {
        FPOP[k][j] -= 1;
```

```
        }
       }
       if(ENVTYPE == 3)
       {
        if(RNDNUM >= TEMPCYFS[k][j])
        {
         FPOP[k][j] -= 1;
        }
       }
      }
      t = MPOP[k][j];
      for(int h = t; h > 0; h--)
      {
       RNDNUM = R();
       RandomCount++;
       if(ENVTYPE == 0)
       {
        if(RNDNUM >= TEMPGYMS[k][j])
        {
         MPOP[k][j] -= 1;
        }
       }
       if(ENVTYPE == 1)
       {
        if(RNDNUM >= TEMPNYMS[k][j])
        {
         MPOP[k][j] -= 1;
        }
       }
       if(ENVTYPE == 2)
       {
        if(RNDNUM >= TEMPBYMS[k][j])
        {
         MPOP[k][j] -= 1;
        }
       }
       if(ENVTYPE == 3)
       {
        if(RNDNUM >= TEMPCYMS[k][j])
        {
         MPOP[k][j] -= 1;
        }
       }
      }
     }
    }
//test for extinction of species and output of yearly data
   for(int w = 0; w < PATCH; w++)
   {
    TOTPOP[w] = 0;
    FTOT[w] = 0;
    MTOT[w] = 0;
   }
   for(int w = 0; w < PATCH; w++)
   {
    for(int h = 0; h < AC; h++)
    {
     TOTPOP[w] += FPOP[w][h];
     TOTPOP[w] += MPOP[w][h];
     FTOT[w] += FPOP[w][h];
     MTOT[w] += MPOP[w][h];
    }
   }
   TOTALPOP = 0;
   for(int w = 0; w < PATCH; w++)
   {
    TOTALPOP += TOTPOP[w];
    YEARLYPOPTOT[w][l-1][i-1] = TOTPOP[w];
   }
   if(TOTALPOP < 1)
   {
    EXT += 1;
    break;
   }
//patch to patch movement
   if(PATCHMOVE == 1)
   {
```

```
for(int w = 0; w < PATCH; w++)
{
 if(TOTPOP[w] > CAPHIGH[w])
 {
  RNDNUM = R();
  RandomCount++;
  for(int a = 0; a < PATCH; a++)
  {
   if(RNDNUM < PATCHMOVEMENTCDF[w][a])
   {
    if(TOTPOP[a] < CAPHIGH[a])
    {
     RNDNUM = R();
     RandomCount++;
     if(RNDNUM < MR[w])
     {
      MPOP[w][e]--;
      MPOP[a][e]++;
      TOTPOP[w]--;
      TOTPOP[a]++;
      MTOT[w]--;
      MTOT[a]++;
      break;
     }
     else
     {
      FPOP[w][e]--;
      FPOP[a][e]++;
      TOTPOP[w]--;
      TOTPOP[a]++;
      FTOT[w]--;
      FTOT[a]++;
      break;
     }
    }
   }
  }
 }
 else
 {
  for(int a = 0; a < PATCH; a++)
  {
   TESTM[a]=MPOP[a][e];
   TESTF[a]=FPOP[a][e];
  }
  for(int h = TESTM[w]; h > 0; h--)
  {
   RNDNUM = R();
   RandomCount++;
   for(int a = 0; a < PATCH; a++)
   {
    if(w == a)
    {
     a++;
    }
    if(RNDNUM < PATCHMOVEMENTCDF[w][a])
    {
     MPOP[w][e]--;
     MPOP[a][e]++;
     TOTPOP[w]--;
     TOTPOP[a]++;
     MTOT[w]--;
     MTOT[a]++;
     break;
    }
   }
  }
  for(int h = TESTF[w]; h > 0; h--)
  {
   RNDNUM = R();
   RandomCount++;
   for(int a = 0; a < PATCH; a++)
   {
    if(w == a)
    {
     a++;
    }
```

```
        if(RNDNUM < PATCHMOVEMENTCDF[w][a])
        {
         FPOP[w][e]--;
         FPOP[a][e]++;
         TOTPOP[w]--;
         TOTPOP[a]++;
         FTOT[w]--;
         FTOT[a]++;
         break;
        }
       }
      }
     }
    }
   }
  }
//find end of simulation run populations for each patch
  for(int w = 0; w < PATCH; w++)
  {
   POPAVE[w][l-1] = TOTPOP[w];
   AVERAGEPOP[w] += TOTPOP[w];
   AVEFEMPOP[w] += FTOT[w];
   AVEMALPOP[w] += MTOT[w];
  }
 }
//output of ending population sizes for each simulation run
 for(int w = 0; w < PATCH; w++)
 {
  for(int b = 0; b < YRCNT; b++)
  {
   for(int d = 0; d < SIMCNT; d++)
   {
    YEARLYPOPAVE[w][b] += YEARLYPOPTOT[w][d][b];
   }
  }
 }
 for(int w = 0; w < PATCH; w++)
 {
  for(int b = 0; b < YRCNT; b++)
  {
   YEARLYPOPAVE[w][b] = YEARLYPOPAVE[w][b]/SIMCNT;
  }
 }
 double POPVAR[PATCH];
 double POPSTDDEV[PATCH];
 for(int w = 0; w < PATCH; w++)
 {
  AVERAGEPOP[w] = AVERAGEPOP[w]/SIMCNT;
  POPVAR[w] = 0;
  for(int h = 0; h < SIMCNT; h++)
  {
   POPVAR[w] += pow((AVERAGEPOP[w]-POPAVE[w][h]),2);
  }
  POPVAR[w] = POPVAR[w]/SIMCNT;
  POPSTDDEV[w] = pow(POPVAR[w],0.5);
  AVEFEMPOP[w] = AVEFEMPOP[w]/SIMCNT;
  AVEMALPOP[w] = AVEMALPOP[w]/SIMCNT;
  cout << "The average end of the sim. pop. for patch " << w+1 << " is:" << AVERAGEPOP[w] << endl;
  cout << "The variance of the end of the sim. pop. for patch " << w+1 << " is:" << POPVAR[w] << endl;
  cout << "The standard deviation of the end of the sim. pop. for patch " << w+1 << " is:" << POPSTDDEV[w] << endl;
  cout << "The average end of the sim. female proportion for patch " << w+1 << " is:" << AVEFEMPOP[w]/AVERAGEPOP[w] << endl;
  cout << "The average end of the sim. male proportion for patch " << w+1 << " is:" << AVEMALPOP[w]/AVERAGEPOP[w] << endl;
 // cout << "End of year population averages for patch " << w+1 << ": " << endl;
 // for(int b = 0; b < YRCNT; b++)
 // {
 //  cout << YEARLYPOPAVE[w][b] << endl;
 // }
 }

//extinction risk
 cout << "The extinction risk is: " << EXT << "/" << SIMCNT << endl;
//# of rnd #'s
 cout << "There were " << RandomCount << " random numbers used during this simulation." << endl;
 for(int w = 0; w < PATCH; w++)
  {
   cout << "The end of 50 year population sizes for patch " << w+1 << ": " << endl;
   for(int d = 0; d < SIMCNT; d++)
```

```
    {
     cout << YEARLYPOPTOT[w][d][YRCNT-1] << endl;
    }
   }
 //**************************************************************************************
 //output of the various arrays and variables
  int d = 0;
  cout << "To see the pdf's, cdf's, and other arrays used for the simulation, type '1'." << endl;
  cin >> d;
  if(d == 1)
  {
   cout << "Newborn #'s" << endl;
   for(int i = 0; i < AS ; i++)
   {
    cout << BABIES[0][2*i] << ",";
    cout << BABIES[0][2*i+1] << "  ";
   }
   cout << endl;
   for(int k = 0; k < PATCH; k++)
   {
    char Temp;
    cout << "Hit any key, then ENTER to see patch " << k+1 << "." << endl;
    cin >> Temp;
    cout << "Normal Year PDF for patch " << k+1 << " :" << endl;
    cout << "[";
    for(int i = 0; i < AS ;i++)
    {
     cout << NYBPDF[k][i] << "  ";
    }
    cout << "]" << endl;
    cout << "Normal Year CDF for patch " << k+1 << " :" << endl;
    cout << "[";
    for(int i = 0; i < AS ; i++)
    {
     cout << NYBCDF[k][i] << "  ";
    }
    cout << "]" << endl;
    cout << "Good year probabilities for each litter for patch " << k+1 << " :" << endl;
    cout << "[";
    for(int i = 0; i <= ML; i++)
    {
     cout << GYPL[k][i] << "  ";
    }
    cout << "]" << endl;
    cout << "Good Year PDF for patch  " << k+1 << " :" << endl;
    cout << "[";
    for(int i=0; i < AS ;i++)
    {
     cout << GYBPDF[k][i] << "  ";
    }
    cout << "]" << endl;
    cout << "Good Year CDF for patch " << k+1 << " :" << endl;
    cout << "[";
    for(int i=0; i < AS ;i++)
    {
     cout << GYBCDF[k][i] << "  ";
    }
    cout << "]" << endl;
    cout << "Bad year probabilities for each litter for patch " << k+1 << " :" << endl;
    cout << "[";
    for(int i = 0; i <= ML; i++)
    {
     cout << BYPL[k][i] << "  ";
    }
    cout << "]" << endl;
    cout << "Bad Year PDF for patch " << k+1 << " :" << endl;
    cout << "[";
    for(int i=0; i < AS ;i++)
    {
     cout << BYBPDF[k][i] << "  ";
    }
    cout << "]" << endl;
    cout << "Bad Year CDF for patch " << k+1 << " :" << endl;
    cout << "[";
    for(int i=0; i < AS ;i++)
    {
     cout << BYBCDF[k][i] << "  ";
```

```
}
cout << "]" << endl;
cout << "Normal Year survival rates for each age class (Females) for patch " << k+1 << " :" <<endl;
cout << "[";
for(int i =0; i < AC; i++)
{
 cout << NYFS[k][i] << "  ";
}
cout << "]" << endl;
cout << "Normal Year survival rates for each age class (Males) for patch " << k+1 << " :" << endl;
cout << "[";
for(int i =0; i < AC; i++)
{
 cout << NYMS[k][i] << "  ";
}
cout << "]" << endl;
cout << "Good Year survival rates for each age class (Females) for patch " << k+1 << " :" << endl;
cout << "[";
for(int i =0; i < AC; i++)
{
 cout << GYFS[k][i] << "  ";
}
cout << "]" << endl;
cout << "Good Year survival rates for each age class (Males) for patch " << k+1 << " :" << endl;
cout << "[";
for(int i =0; i < AC; i++)
{
 cout << GYMS[k][i] << "  ";
}
cout << "]" << endl;
cout << "Bad Year survival rates for each age class (Females) for patch " << k+1 << " :" << endl;
cout << "[";
for(int i =0; i < AC; i++)
{
 cout << BYFS[k][i] << "  ";
}
cout << "]" << endl;
cout << "Bad Year survival rates for each age class (Males) for patch " << k+1 << " :" << endl;
cout << "[";
for(int i =0; i < AC; i++)
{
 cout << BYMS[k][i] << "  ";
}
cout << "]" << endl;
cout << "Catastrophe year probabilities for each litter for patch " << k+1 << " :" << endl;
cout << "[";
for(int i = 0; i <= ML; i++)
{
 cout << CYPL[k][i] << "  ";
}
cout << "]" << endl;
cout << "Catastrophe Year PDF for patch " << k+1 << " :" << endl;
cout << "[";
for(int i=0; i < AS ;i++)
{
 cout << CYBPDF[k][i] << "  ";
}
cout << "]" << endl;
cout << "Catastrophe Year CDF for patch " << k+1 << " :" << endl;
cout << "[";
for(int i=0; i < AS ;i++)
{
 cout << CYBCDF[k][i] << "  ";
}
cout << "]" << endl;
cout << "Catastrophe Year survival rates for each age class (Females) for patch " << k+1 << " :" << endl;
cout << "[";
for(int i =0; i < AC; i++)
{
 cout << CYFS[k][i] << "  ";
}
cout << "]" << endl;
cout << "Catastrophe Year survival rates for each age class (Males) for patch " << k+1 << " :" << endl;
cout << "[";
for(int i =0; i < AC; i++)
{
 cout << CYMS[k][i] << "  ";
```

```
   }
   cout << "]" << endl;
  }
 }
}
//combination subroutine
int C(int n, int r)
{
 int t1 = 1;
 int t2 = 1;
 int c = 0;
 if ((n-r) > r)
  c = r;
 else
  c = (n-r);
 for (int i = n; i >= n-c+1; i--)
 {
  t1 *= i;
 }
 for (int j = 1; j <= c; j++)
 {
  t2 *= j;
 }
 int t3 = t1/t2;
 return t3;
 }

//random number generator
float R()
{
 float t = rand()/(RAND_MAX+1.0);
 return t;
}

//catastrophe check
int Poisson(int t, float lambda, float CatRnd)
{
 float x = 1 - exp(-lambda*t);
 if(CatRnd <= x)
 {
  return 1;
 }
 else
 {
  return 0;
 }
}

//fmod rescale
double fimod(int x, int y)
{
 double v = x;
 double z = y;
 return fmod(v,z);
}
```

# Bibliography

[1] Akcakaya, H. Resit. RAMAS/metapop User Manual version 1.1. New York: Applied Biomathematics.

[2] Beier, Paul "Metapopulation Models, Tenacious Tracking, and Cougar Conservation." in D.R. McCullogh ed. Metapopulations and Wildlife Management, Island Press, 1996: 293-323

[3] Case, Ted J. An Illustrated Guide to Theoretical Ecology. New York: Oxford UP, 2000

[4] Knuth, D. E. The art of computer programming. Vol. 2, 3rd ed. Addison-Wesley, 1998. 17

[5] Lacy, Robert C. "VORTEX: A Computer Simulation Model for Population Viability Analysis." Wildlife Research Vol. 20 (1991): 45-65.

[6] Lamberson, Roland H., Noon, Barry R., Voss, Curtis, & McKelvey, Kevin, "Reserve Design for Territorial Species: The Effects of Patch Size and Spacing on the Viability of the Northern Spotted Owl." Conservation Biology, 1994: 185-195

[7] Lehmer, D.H. "Mathematical methods in large scale computing units." In Proc. 2nd Symposium on Large-Scale Digital Calculating Machinery. 1949. Cambridge, MA: Harvard UP, 1951, 141-146.

[8] Murphy, Michael T. "Source-Sink Dynamics of a Declining Eastern Kingbird Population and the Value of Sink Habitats." Conservation Biology Vol. 15 No. 3 June 2001: 737-748.

[9] Possingham, H.P. & Davies, I. "ALEX: A Model for the Viability Analysis of Spatially Structured Populations." Biological Conservation Vol. 73 1995: 143-150.

[10] Ross, Sheldon. A First Course in Probability, Fifth Edition. New Jersey: Prentice Hall, 1998.

[11] Taha, Hamdy A. Operations Research, Fourth Edition. New York: Macmillan Publishing Co., 1987