NRES 779 Lab 1

Jerrod Merrell, Griffin Shelor

February 1, 2024

1 Exercise 1.1

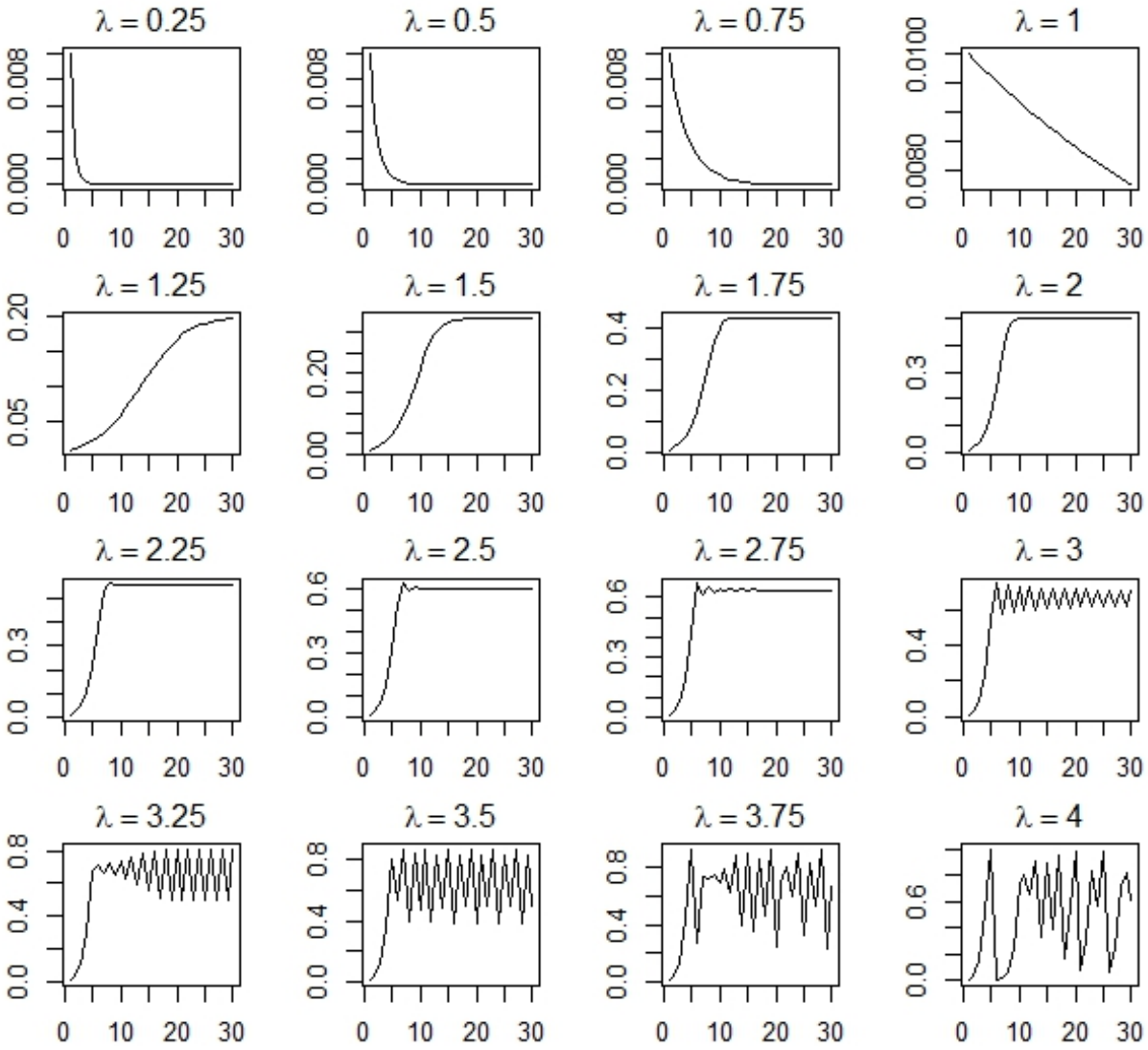

Figure 1: Population as a function of time for each value of λ

As λ increases, the population will also grow to a higher level. The line of population values also becomes more variable, fluctuating more rapidly around what appears to be some sort of central value. For $\lambda = 4$,

the line displays a wider variability than it does for $\lambda = 3.75$ or 3.50, and for λ which are less than 2.50, the line is reasonably smooth (figure 1).

2 Exercise 1.2

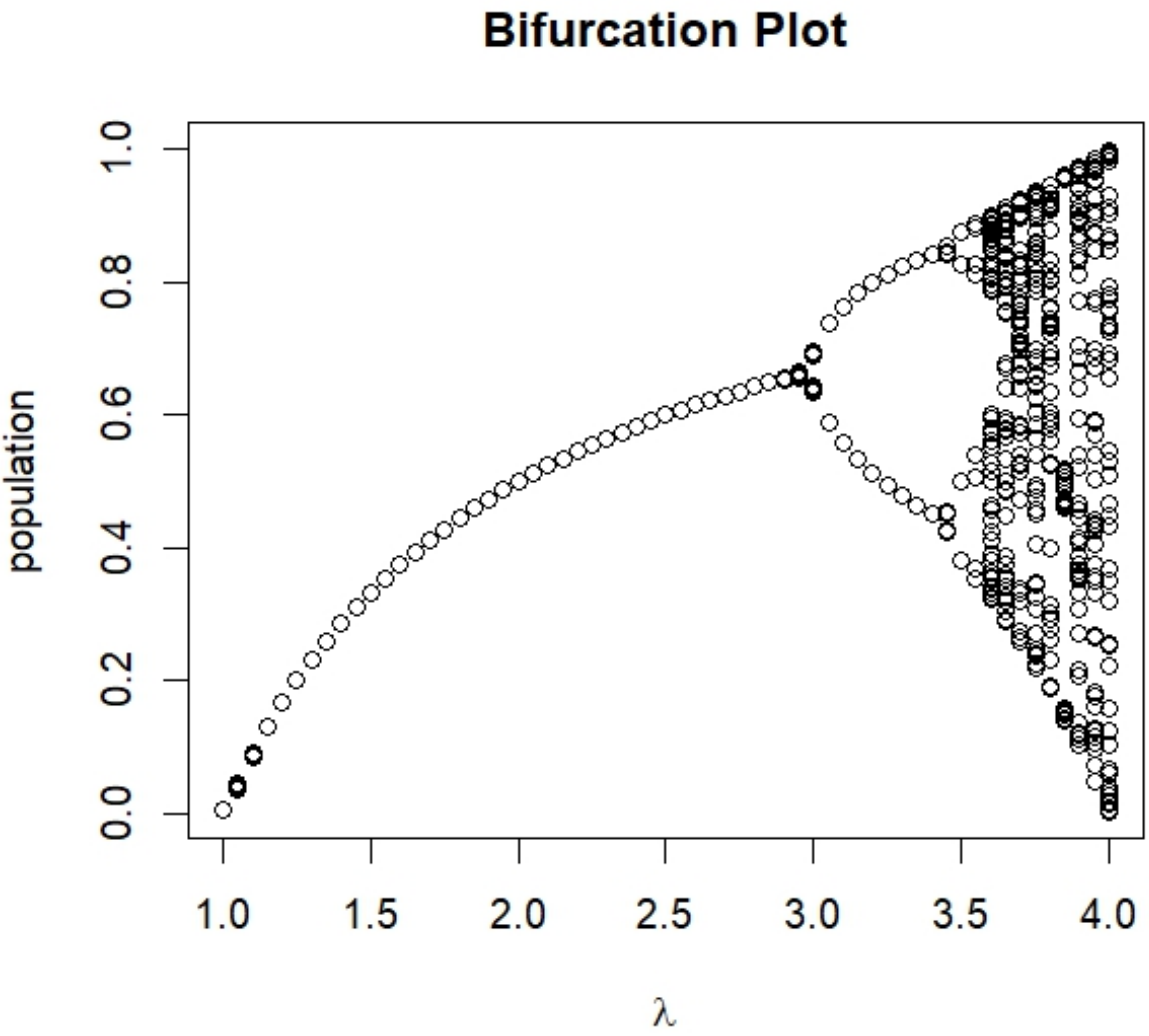

Figure 2: Bifurcation Plot

The bifurcation diagram (figure 2) describes the changes in trajectory of x_t as it interacts with λ. We see the sample pattern here as we did in the plot from the previous exercise. As λ increases, we see greater variability in x_t, implying that high values of λ do not produce stable population values.

3 Exercise 3

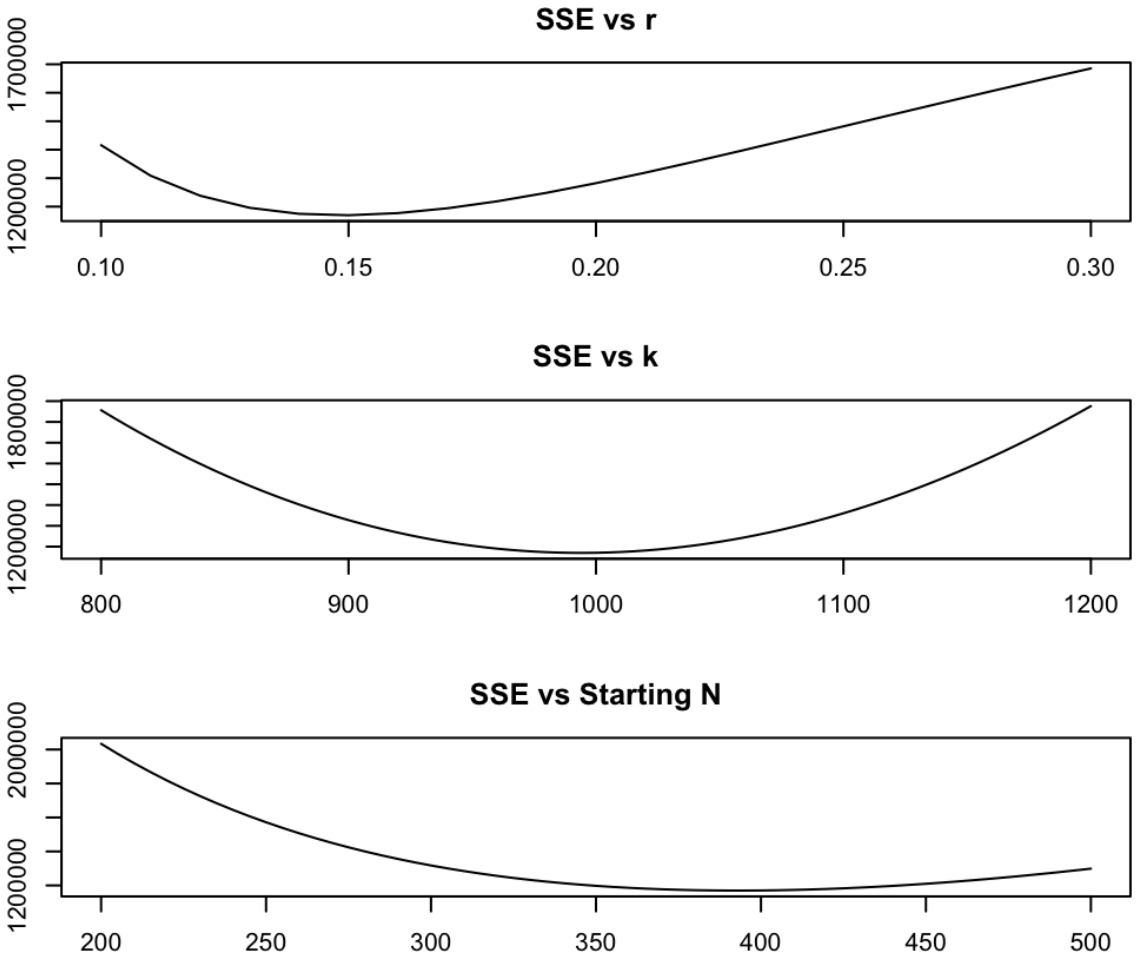

Figure 3: SSE as a function of each parameter with the other 2 parameters kept constant at optimal values

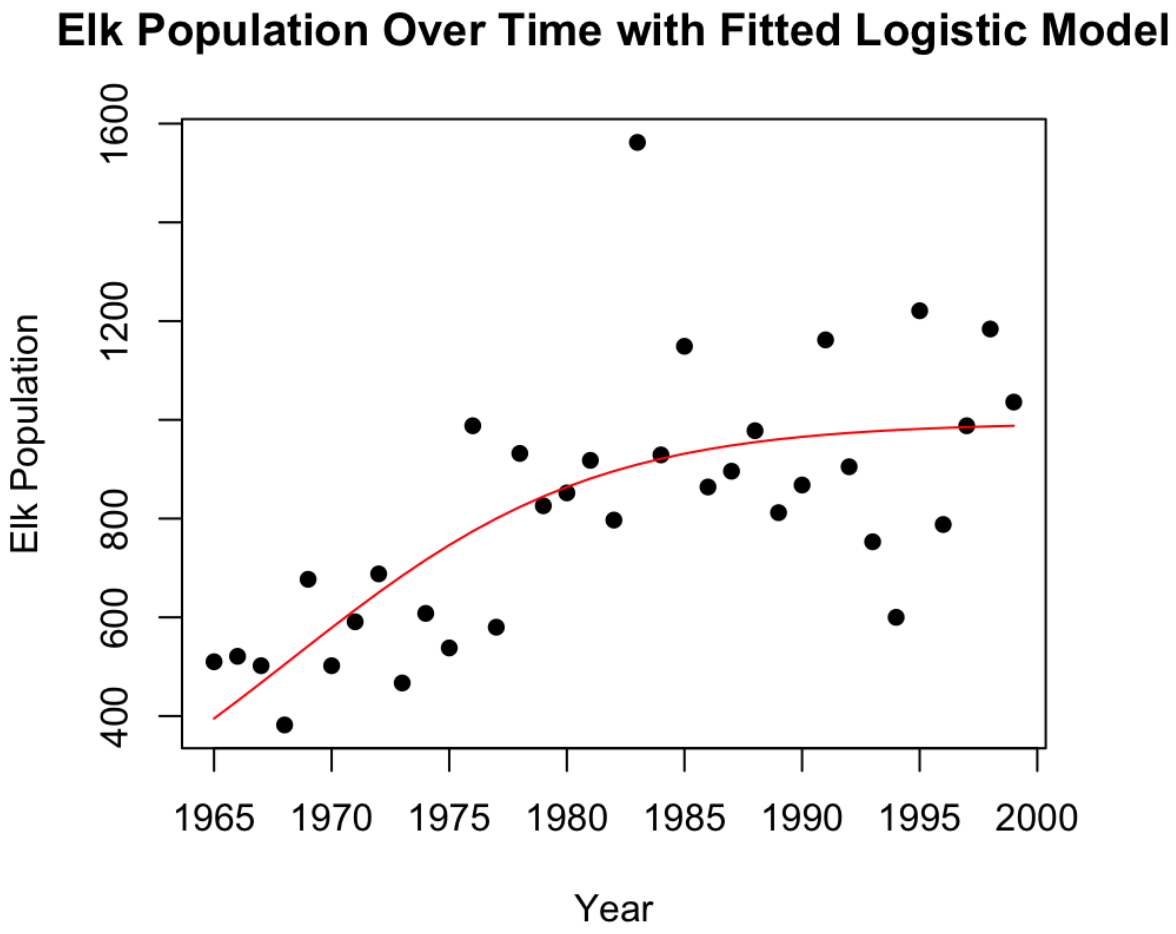

Figure 4: Elk population over time with the red line representing a fitted logistic model

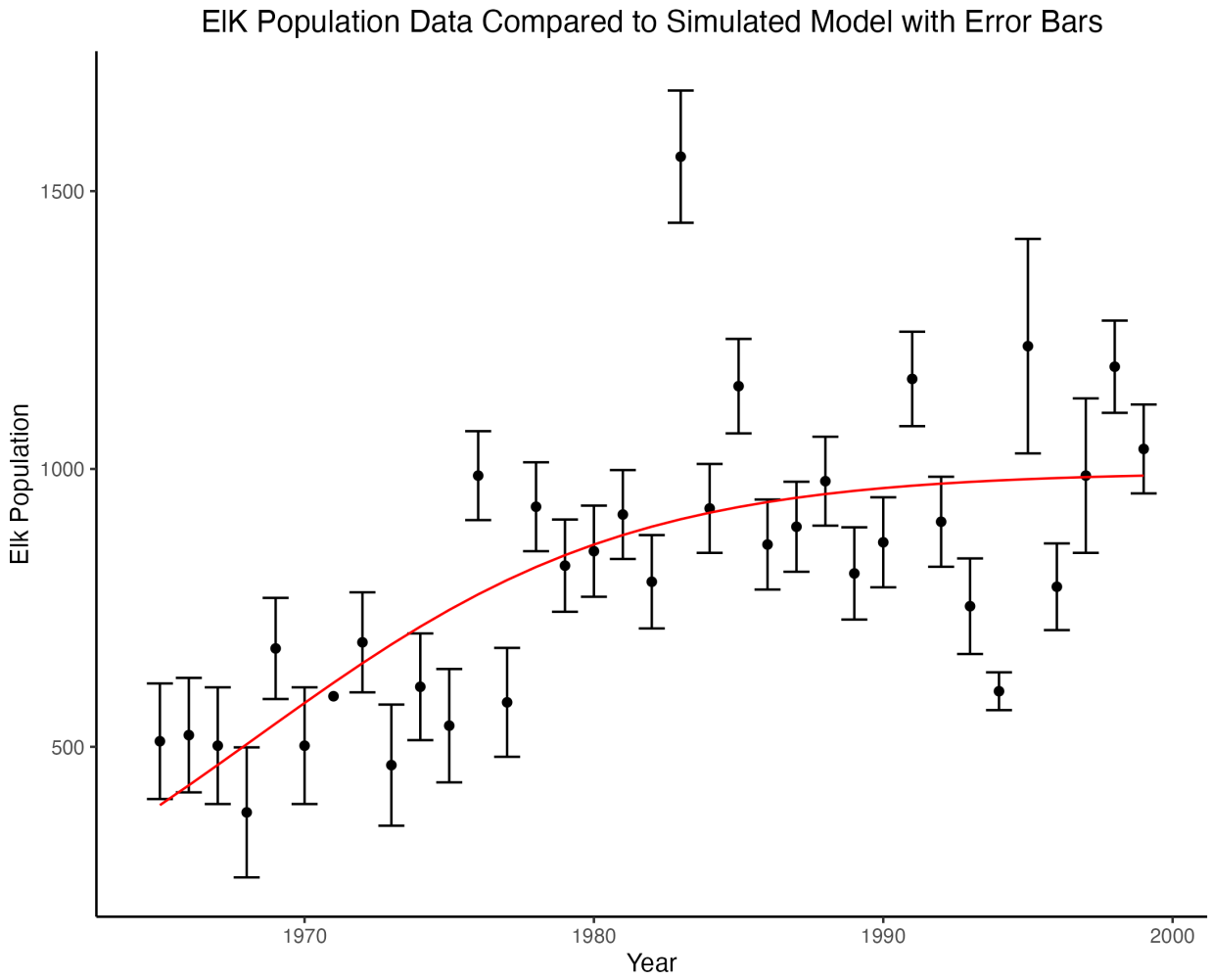

Figure 5: Elk model with standard error bars extending from data points

Some problems with the "brute force" approach include that it can be slow, imprecise, and can be difficult and take a long time to code. There are packages which can speed up this process, as well as helping with the imprecision of the brute force approach. When we use the brute force method, we are manually selecting intervals at which to test a certain parameter value. While this potentially opens us up to bias in terms of how these intervals and the ranges they span are selected, the bigger issue is that it can potentially result in us missing a more optimal value as we test different sets of parameters. Narrowing the parameter intervals after finding an initial optimal parameter value can certainly help with this. The brute force method also likely involves more nested for loops than methods which may make use of packages in R and many nested for loops could potentially get quite confusing both for the writer and anyone who reads the code later if we were to be working with more complex models.

4 Appendix

R code

```
## Exercise 1.1
## storing empty matrix of population values
N <- matrix(nrow = 30, ncol = 16)
## setting initial population
```

```r
N[1,] <- 0.01
## setting time vector
time_vector <- seq(2, 30, 1)
## setting vector of lambda values
lambda_vector <- seq(0.25, 4, 0.25)
## writing a function
logistic <- function(n, R) {
  new_n <- R * n * (1 - n)
  return(new_n)
}

## Outer for loop
for (l in 1:length(lambda_vector)) {
  ## inner for loop
  for (t in 1:length(time_vector)) {
    N[t + 1, l] <- logistic(N[t,l], lambda_vector[l])
  }
}
N
N_df <- data.frame(N)
N_df <- N_df |>
  mutate(time = seq(1:30), .before = 1)

## plotting charts
# lambda_expression <- expression(lambda)
# lambda_text <- as.character(lambda_vector)
par(mfrow = c(4,4), mar = c(2,2,2,2))
for (p in seq(1:16)) {
  plot(N_df$time, N_df[,p + 1],
  type = "l", main = bquote(lambda == .(lambda_vector[p])))
}
```

R code

```r
## Exercise 1.2
## setting time vector
time_vector_2q <- seq(1, 100, 1)
## setting vector of lambda values
lambda_vector_2q <- seq(1, 4, 0.05)
## storing empty vector of population values
population <- NULL
## setting initial population
# population[1,] <- 0.01
## writing a function
# logistic <- function(n, R) {
#   new_n <- R * n * (1 - n)
#   return(new_n)
# }

## for loop
for (l in 1:length(lambda_vector_2q)) {
  ## inner for loop
  for (t in 1:length(time_vector_2q)) {
    if (t == 1) {
      h <- c(time_vector_2q[t], lambda_vector_2q[l],
```

```
      logistic(0.01, lambda_vector_2q[l]))
      population <- rbind(population, h)
    } else{
      h <- c(time_vector_2q[t], lambda_vector_2q[l],
      logistic(population[nrow(population),3], lambda_vector_2q[l]))
      population <- rbind(population, h)
    }
  }
}

population_df <- data.frame(population)
colnames(population_df) <- c("time_step", "lambda_val", "population")
population_df <- population_df |>
  filter(time_step >= 51)

par(mfrow = c(1,1), mar = c(4,4,4,2))
plot(population_df$lambda_val, population_df$population,
xlab = expression(lambda), ylab = "population", main = "Bifurcation Plot")
```

R code

```
## reading in Elk data
elk <- read.csv(here("Labs", "Lab1", "RMNPelktimeseries.csv"))

## writing a function with k
logistic_k <- function(n, R, k) {
  new_n <- n + (R * n * (1 - n / k))
  return(new_n)
}

## nested loops for elk
## setting up time, k, r and N1 vectors
time_vector_3q <- seq(1, 35, 1)
r_vector_3q <- seq(0.1, 0.3, 0.05)
k_vector_3q <- seq(800, 1200, 10)
starting_n_3q <- seq(200, 500, 10)

## iterating over different sets of paremeters to find set with lowest SSE
elk_pop_df <- expand.grid(r_val = r_vector_3q, k_val = k_vector_3q,
    starting_n = starting_n_3q)

for(i in 1:nrow(elk_pop_df)) {
  ## Iterate over each row of the matrix
  params <- elk_pop_df[i,]
  N <- rep(0, length(time_vector_3q))
  # Set the initial population size, retrieved from params
  N[1] <- params$starting_n
  for(t in 1:(length(N)-1)) {
    # For each t (time step), calculate pop. size for next year based on current pop. size
    N[t+1] <- N[t] + params$r_val * N[t] * (1 - N[t] / params$k_val)
  }
  ## calculating the SSE
  elk_pop_df$SSE[i] <- sum((elk$Population_size - N)^2)
}
```

```r
## finding best parameters
params_best <- elk_pop_df[which.min(elk_pop_df$SSE),]
params_best

## doing it again but with finer parameter differences
## setting up r, k, and starting n vectors
r_vector_3q_v2 <- seq(0.1, 0.3, 0.01)
k_vector_3q_v2 <- seq(800, 1200, 5)
starting_n_3q_v2 <- seq(200, 500, 5)
elk_pop_df_v2 <- expand.grid(r_val = r_vector_3q_v2, k_val = k_vector_3q_v2,
    starting_n = starting_n_3q_v2)

for(i in 1:nrow(elk_pop_df_v2)) {
  ## Iterate over each row of the matrix
  params <- elk_pop_df_v2[i,]
  N <- rep(0, length(time_vector_3q))
  # Set the initial population size, retrieved from params
  N[1] <- params$starting_n
  for(t in 1:(length(N)-1)) {
    # For each t (time step), calculate pop. size for next year based on current
    #pop. size
    N[t+1] <- N[t] + params$r_val * N[t] * (1 - N[t] / params$k_val)
  }
  ## calculating the SSE
  elk_pop_df_v2$SSE[i] <- sum((elk$Population_size - N)^2)
}

## finding best parameters
params_best_v2 <- elk_pop_df_v2[which.min(elk_pop_df_v2$SSE),]
params_best
params_best_v2

## filtering to plot SSE vs individual parameters with parameters
#not being plotted getting kept constant at their "best" values
elk_pop_df_r_SSE <- elk_pop_df_v2 |>
  filter(k_val == params_best_v2$k_val &
  starting_n == params_best_v2$starting_n)
elk_pop_df_k_SSE <- elk_pop_df_v2 |>
  filter(r_val == params_best_v2$r_val &
  starting_n == params_best_v2$starting_n)
elk_pop_df_n_SSE <- elk_pop_df_v2 |>
  filter(r_val == params_best_v2$r_val &
  k_val == params_best_v2$k_val)

## plots
par(mfrow = c(3,1), mar = c(3,3,3,3))
plot(elk_pop_df_r_SSE$r_val, elk_pop_df_r_SSE$SSE, xlab = "r",
    ylab = "SSE", main = "SSE vs r", type = "l")
plot(elk_pop_df_k_SSE$k_val, elk_pop_df_k_SSE$SSE, xlab = "k",
    ylab = "SSE", main = "SSE vs k", type = "l")
plot(elk_pop_df_n_SSE$starting_n, elk_pop_df_n_SSE$SSE, xlab = "Starting N",
     ylab = "SSE", main = "SSE vs Starting N", type = "l")
```

```
## creating model using best-fit parameters
# Assigning the best-fit parameters
N1_best <- 395
r_best <- 0.15
k_best <- 995

## Do predictions using the logistic growth model
N <- numeric(length(time_vector_3q)) # Create numerical vector N with the same
N[1] <- N1_best
for(t in 1:(length(N)-1)) { #iterate over time steps,
filling the N vector with pop predictions other than first
  N[t+1] <- N[t] + r_best * N[t] * (1 - N[t] / k_best)
  # Calculate N_t+1 using the logistic growth formula
}

## plot line overlaid on data points
par(mfrow = c(1,1), mar = c(4.5,5,4,3.5))
plot(elk$Year, elk$Population_size, pch = 16, xlab = "Year",
    ylab = "Elk Population",
        main = "Elk Population Over Time with Fitted Logistic Model")
lines(elk$Year, N, type = 'l', col = 'red')
# Overlay the model predictions, predicted against years


## plotting with error bars
elk_errorbar_plot <- ggplot(elk, aes(x = Year, y = Population_size)) +
  theme_classic() +
  geom_point() +
  geom_errorbar(aes(ymin = Population_size - SE,
    ymax = Population_size + SE)) +
  labs(x = "Year", y = "Elk Population") +
  ggtitle("ElK Population Data Compared to Simulated Model with Error Bars") +
  theme(plot.title = element_text(hjust = 0.5))

elk_errorbar_plot <- elk_errorbar_plot +
  geom_line(data = elk, aes(x = Year, y = N), color = "red")
elk_errorbar_plot
ggsave(here("Labs", "Lab1", "Elk_Errorbar_Plot.png"))
```