Name: Abdullah Shahzad
Course: CS 152
Section: B
Date: 10/31/202

Project 6 Report:

Title: *Simulating the Growth of the Elephant Population in Kruger National Park*

Note: Interpretations of results are given in the follow-up questions sections to avoid repetition.

Abstract:

This is the third installment of a three-part project about controlling the elephant population in Kruger National Park. In the previous two projects, we've already covered simulating the population of elephants with a default set of parameters with great complexity, using culling or darting as methods of controlling the population in each simulation. In this project, we're researching the effect of changing certain parameters, such as the mortality rate of an elephant of a particular age group, on the number of elephants that need to be darted (probability of darting).
This project makes use of two main coding concepts, binary search and computational complexity. Binary search was used to automatically find the best probability of darting to control a population of elephants with specific characteristics. In binary search we take a sample and keep cutting out half of it depending on which half the answer we're looking for is in, making the sample smaller, eventually reaching an answer. Computational complexity, on the other hand, is the relationship between the resources it takes to execute a function and the input of that function, it's represented as a function of n where n is the input value. When I use the word resources, I mean time and storage space. Since we were dealing with large amounts of data, depending on what the carrying capacity for the park was or the amount of time the simulation was run for, the time and space it took to execute functions was significant.

Results: Required Elements 1 through 5 (All simulations run with 'Carrying Capacity' = 1000)

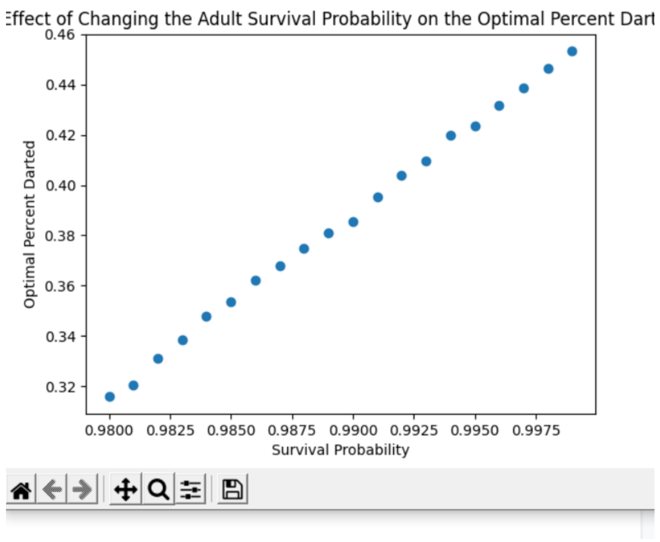


Fig 1.1

In Fig 1.1, the program is checking for the optimum darting percentage (percentage of elephants to dart with a contraceptive to maintain the elephant population at carrying capacity) while varying the probability of an adult elephant dying from 0.98 to a probability of 1.0 with steps of 0.001 (this means that the program records the optimum darting percentage everytime it changes the survival probability of adult elephants by 0.001).

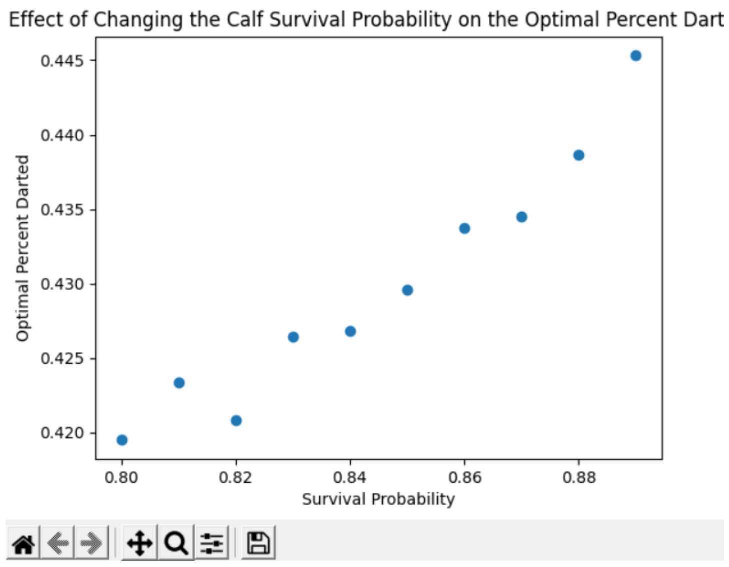


Fig 1.2

In Fig 1.2, the program is checking for the optimum darting percentage while varying the probability of a calf surviving. The program varies the probability of calf survival from 0.80 to 0.90 in steps of 0.01.

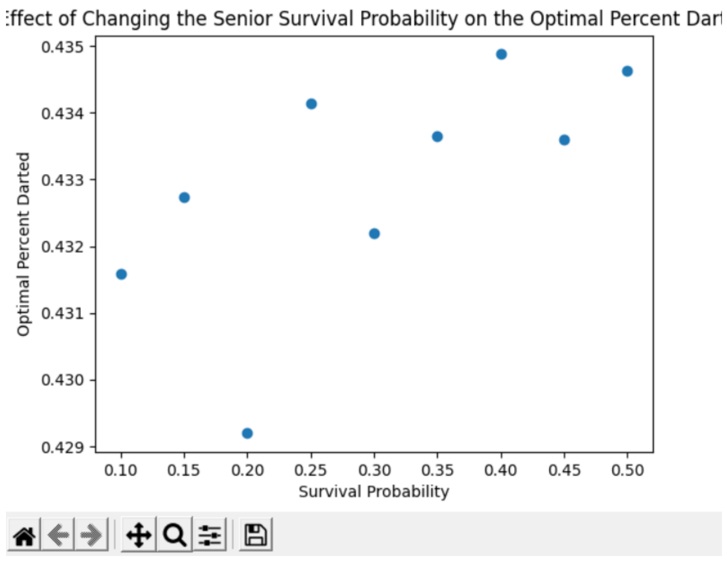


Fig 1.3

In Fig 1.3, the program is checking for the optimum darting percentage while varying the probability of a senior elephant surviving. The program varies the probability of senior survival from 0.10 to 0.50 in steps of 0.05.

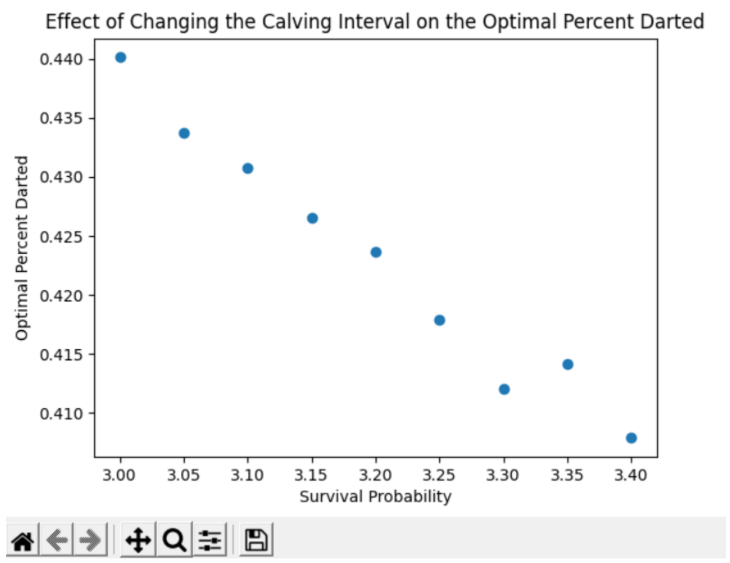


Fig 1.4

In fig 1.4, we're exploring the effect of varying the calving interval on the amount of elephants that need to be darted to control the population. We're changing the calving interval from 3.0 to 3.4 in steps of 0.05.

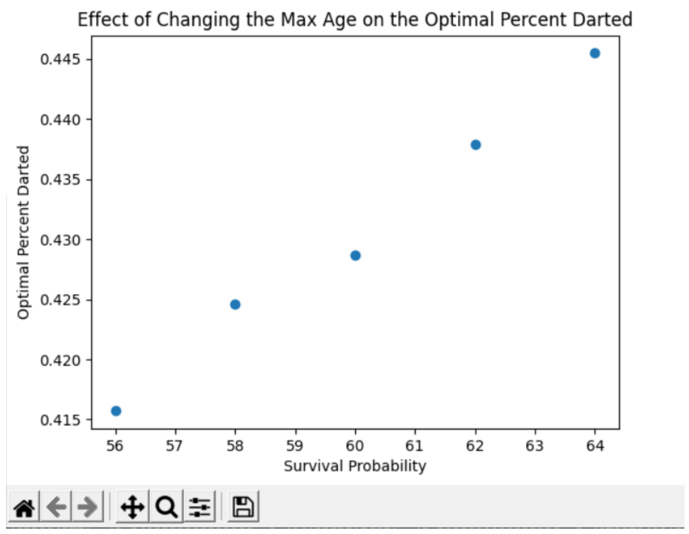


Fig 1.5

In this last required image (Fig 1.5) we're varying the Max Age of the elephants (this is the age after which adult elephants qualify as senior elephants) and seeing how it affects the optimum darting percentage. We're varying the max age from 56 to 64 in increments of 2 years.

Reflection:

Lecture concepts used in this project are making programs in small increments and testing out each increment to see if it works properly, this makes sure that we don't make mistakes with our code and that we don't have to look at absurd amounts of code if something goes wrong.

Another lecture concept utilized in this project is binary search that allows us to search for a particular object by slicing the sample we're looking through in halves depending on where the answer is. Real life applications of these processes would be looking through huge directories or making a really large and complex python program in general. When looking through a directory of, say, ID numbers, we can find the desired user faster if we use binary search (more real life applications are listed in the follow-up question that asks the same thing).

Follow-up Reflection Questions

1. **What does an import statement do?**

An import statement lets us access a python file from another python file or module.

2. **What is binary search? How does it work differently than a linear search algorithm?**

Binary search is when python finds the middle of a set of data and eliminates the half of the data where the answer cannot lie. Python then repeats this process until it gets to the answer. A linear search algorithm on the other hand, is an algorithm where python goes over every item in the sample we're looking for one by one.

3. **Why is binary search faster than a linear search (e.g. going page by page to find a word in a dictionary)?**

Binary search is faster than linear search because in binary search, we eliminate wrong answers faster. Half of the data in a large data set is automatically skipped in binary search while in linear search, we'd have to go over all the values.

4. **For full credit, the data must be interpreted for the reader in clear sentences within the writeup. <u>What</u> do these numbers mean? <u>Why</u> does the darting percentage go up/down when you vary each parameter? <u>How</u> should Kruger National Park use this information**

1- In figure 1.1, we're varying the probability of survival of adult elephants and we see that the optimal darting probability increases as we increase the probability of survival. Increasing the probability of survival means that fewer elephants die as we get closer to the value of 1.0 and so, naturally, we'd need to dart more elephants because there are, a) more female adults to give birth to calves, and b) more elephants in general.

2- In figure 1.2, we're varying the probability of a calf surviving and its effect on the optimal darting percentage. Similar to the analysis above, as we get closer to 1, more calves survive. This would mean that we would naturally have to control the amount of calves that are born and to do that, we'd have to increase the amount of elephants that are darted, which is exactly what the graph shows.

3- In figure 1.3, we're varying the probability that a senior elephant survives and testing its effect on the optimum darting probability. Interestingly, although more senior elephants survive, the darting probability remains almost similar. This may be because senior females can't have children. Something else that is also interesting is that we have an anomalous result for the darting probability at the value 0.20 on the x-axis; the darting probability there is shown to be impossibly low

4- In figure 1.4, we're varying the calving interval and exploring its effect on the optimal darting probability. In this graph, the optimal darting probability is decreasing as the calving interval increases. This is probably because the probability of a new elephant being born is dependent on the calving interval. The larger the calving interval, the lesser the probability that a female elephant is/gets pregnant. This would automatically reduce the elephant population significantly.

5- In the last required image, we're varying the max age parameter of the elephants and monitoring its effect on the optimal darting probability. The optimal darting probability increases as the max age increases. This is because elephants of greater age can also give birth since they would be considered adults and not seniors, and more elephants would survive since they would live according to the probability of adult survival which is significantly higher than the probability of senior survival.

Kruger National Park can use this information to change the darting probability if the behavior of the elephants changes or they need to account for more elephants or a different breed of elephants with slightly different age-brackets. This program could also be used for other animals in the park since we can obviously change variables within a given range and then call the function.

5. **How might you apply this type of optimized search algorithm approach to something you are really interested in?**

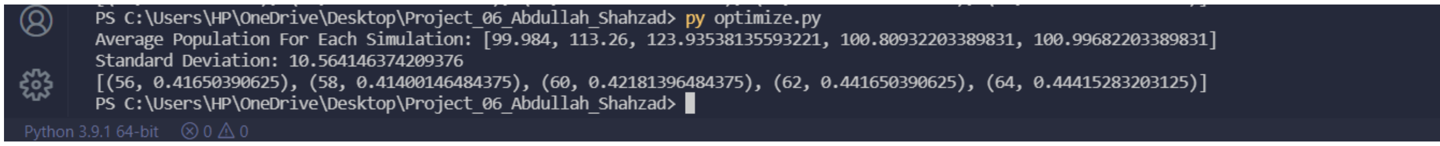
If we're looking through a registry of cars, we can locate a car with a particular registration number using binary search. Or if we're looking through medical records, we can find a patient really easily and quickly using binary search. The point with these applications is that they involve large sets of data and we want to find a single value from them.

Extension (optional):

1) **Automatically making graphs using matplotlib:**

Whenever the optimize function is run, the matplotlib import statement is used to automatically generate a plot of the graph

2) **Standard deviation for average population of each simulation:**



```
PS C:\Users\HP\OneDrive\Desktop\Project_06_Abdullah_Shahzad> py optimize.py
Average Population For Each Simulation: [99.984, 113.26, 123.93538135593221, 100.80932203389831, 100.99682203389831]
Standard Deviation: 10.564146374209376
[(56, 0.41650390625), (58, 0.41400146484375), (60, 0.42181396484375), (62, 0.441650390625), (64, 0.44415283203125)]
PS C:\Users\HP\OneDrive\Desktop\Project_06_Abdullah_Shahzad>
```

Fig 1.6

This extension involves me using the stats.py file to calculate the standard deviation of the values calculated by varying one of the parameters. I started by adding the average population to the returns of the previous functions so that I could pass that value between my functions and use it where appropriate.I had to reassign some variables in the optimize.py file to accommodate for the functions now having multiple outputs. I then accessed the standard deviation function from my stats.py file and plugged in the average population as the argument. After this I just had to issue a call to it in the main function and print it out cleanly in a string. The result can be seen in Fig 1.6 above.

Acknowledgements: Professor Allen Harper and Narit Trikasemsak
Imported libraries: matplotlib, the random library, and the sys library