



## Problem Set -- Assignment 2

Complete the problems below. You will be graded on the LOs specified in each problem. Some of the problems have **optional challenge questions** which you may skip. If you complete the optional challenge very well, you **may** score an extra ⑤ on the LO specified in the question.

Note: "very well" means answering the whole optional question with no errors or bugs and, it needs to be good enough that it can be published on something like Medium (perhaps with a few minor tweaks). See the [CS166 Grading Policy](#) for details.

### Problem 1: Design a model

- LOs: #cs166-Modeling
- Suggested word count: 300–600 (please provide the actual word count)

(Adapted from Exercise 2.4 in H. Sayama, *Introduction to the Modeling and Analysis of Complex Systems*, p. 299.)

Question 1 of 11

Write a few paragraphs (no math or code required) describing a real-world system of your choice with many interacting components/agents.

- At which scale do you choose to describe the microscopic components, and what are those components? Briefly describe other scales (larger and/or smaller) at which you could have chosen to model the microscopic state and why your chosen scale is the most appropriate.
- What states can the microscopic components take on and why?
- What are the relationships between the components within the system? How does studying these contribute to the understanding of the overall system?
- How do the states of the components change over time? Some of these changes may depend on interactions between the components, while others may not.

After answering all of the questions above, predict what kind of macroscopic behaviors would arise if you run a computational simulation of your model.

Normal ⌵ **B** *I* U A

## Topic: outbreak of ebola in Liberia in 2014

This system represents an analysis of a spread of a disease through a system. One way to think about it is on the city level (macro scale) of the country where each city would be a node in the network. This would assume that each city can be described in terms of three probability distributions: how likely it is to become an infected node, the time it takes to recover after becoming infected, and a third one representing which other city nodes are most likely to become infected because of it. Breaking down the system in this way would abstract away many of the details. It would give us insight on the most vulnerable geographical regions of the country. It could be used to plan allocation of resources that would be needed in such a medical supplies for treatment, doctors, and food. The government could also use it to plan the timing and order of interventions like quarantines or education workshops on ebola. Still, the model relies on some questionable assumptions. For example, it assumes that there is a particular distribution for the infection and recovery rates of the city nodes which is a big generalization. It would also be hard to model the relations between the nodes. Another way to model this could be in terms of groups (a smaller scale) - i.e. neighborhoods or households. However this would still make an inaccurate assumption that groups behave the same way when in reality they do not. A more appropriate way would be to model this system on the individual agent because the way ebola is transmitted. It is not an air transmitted disease unlike a cold. The Ebola virus is spread through direct contact with blood or other body fluids. Logically situations like that would not occur on large scale and are more likely to be incidents when there was a direct interactions between individual people. This is why I will focus on an agent based model for disease diffusion, specifically the SIR model.

The model classifies people in terms of 3 different state variables:

- susceptible (S): the number of people who can get infected with ebola
- infected (I): the number of people who are currently infected
- recovered (R): the number of people who had the disease but no longer have it. They are either dead or recovered.

They are all quantitative discrete variables as they describe number of people so we can perform valid mathematical operations with them. This comes in handy as the SIR model is governed by mathematical formulas (differential equations) that describe the rules of the system.

The model analyzes how these 3 variables evolve through time (they are a function of time). They are interconnected as agents transition between the groups and define the dynamics of the spread of ebola. Time is the independent variable measured in days (quantitative continuous).

The reason is because we want to model the spread of the disease through time. In real life this is how ebola spreads as well - from infected individuals on to susceptible ones. After being infected they either recover, develop immunity (for about 10 years or until life - it's unknown) and survive or die but in either case they would no longer be susceptible ("Survivors | Treatment | Ebola (Ebola Virus Disease) | CDC").

- People in (S) can become infected with some probability if they are in contact with a currently infected person (S->I).
- (I) can only pass on the disease WHILE they are infected . There is a parameter for the recovery rate (I->R).
- (R) cannot transition between the other groups.

Studying these contributes to the understanding of the overall system because it helps us understand how the disease spreads. This relationship can be described by a set of differential equations. Here is a photo of them because Forum workbooks don't provide a nice way to format this:

[https://drive.google.com/file/d/1gKPVBNWZoStbNc0XhE\\_tM65PfJNRuUHA/view?usp=share\\_link](https://drive.google.com/file/d/1gKPVBNWZoStbNc0XhE_tM65PfJNRuUHA/view?usp=share_link)

In the equations  $b$  represents the infection rate and  $k$  the recovery rate. These values should be chosen based on data about the particular disease (ebola) and the context in the country (Liberia). For example if there is a lack of available condoms, the value of  $b$  would increase. If there are a limited number of hospitals, then the value of  $k$  would decrease since it would be harder for people to receive medical assistance and recover.

Word count: 600 words

I wanted to add additional clarifications on the parameters and assumptions below:

----

The parameters will influence how the disease develops.

- If  $b$ 's value is high, then the Infected population will grow faster (people moving from susceptible to infected group). If  $b$ 's value is lower, then we would observe the opposite.
- If  $k$ 's value is high, then people will move faster from infected to recovered so the recovered group will grow at a higher rate and the pandemic lifecycle would be shorter. The system would eventually converge to the state of everyone being in that group and no more disease (all dead or immune). If  $k$  is smaller, then it means that people would be sick with the disease for a longer period of time so the recovered population will grow more slowly.

These factors would impact the dynamics and give different insights because different policies would be appropriate for different cases (i.e. if  $k$  is low, then we need more hospitals and doctors).

One assumption that the model makes is that the population is big which is met for the case of Liberia as we consider the population of the whole country. Another assumption is that it is homogenous and that people mix at random - it makes no differentiation between people who are more or less susceptible based on factors such as age. This is an oversimplification that allows the model to abstract the system.

References:

"Survivors | Treatment | Ebola (Ebola Virus Disease) | CDC." *W*[www.cdc.gov](http://www.cdc.gov), 6 Nov. 2019, [www.cdc.gov/vhf/ebola/treatment/survivors.html#:~:text=Those%20who%20do%20recover%20develop](http://www.cdc.gov/vhf/ebola/treatment/survivors.html#:~:text=Those%20who%20do%20recover%20develop). Accessed 24 Feb. 2023.

## Problem 2: Predator-prey model

A predator-prey model describes the dynamics of a predator-prey system, describing how the populations of the predator and prey species change over time based on the interactions between the two species

### 2.1. Creating a Simulation

- LOs: `#PythonImplementation`, `#CodeReadability`
- Words: no specific word count; mostly code

Consider a predator-prey system in which a population of predators and preys interact over time. The predator-prey system is simulated on a two-dimensional grid of cells, with each cell representing a patch of habitat. Each cell can be in one of three states: empty, occupied by a predator, or occupied by prey. The population of predators and prey will change over time according to certain rules and parameters.

To start the simulation, you will be using a grid of 35x35 cells, and assume a Moore neighborhood of radius 1 and periodic boundary conditions. The initial population of prey and predators will be randomly placed on the grid, with a 20% probability of the cell being prey and 20% of being the predator.

The state of each cell is updated at each time step according to the following rules:

- The predator will eat neighboring prey and reproduce at the location of the consumed prey with a certain probability of predator birth rate.
- If there are fewer than 2 prey cells in the neighborhood, the predator cell becomes empty.
- The prey will reproduce into empty neighboring cells.
- The predator dies with a certain probability of predator death rate.
- The prey dies with a certain probability of prey death rate.

**Note:** It is important to apply the rules in the following order:

2. then prey reproduces,
3. then predator and prey die randomly.

This will ensure that the rules for each type of cell are applied correctly and in the correct order, preventing errors such as a prey reproducing in a cell that has already been occupied by a predator that reproduces earlier.

You will be expected to determine what a reasonable time period designated to run the simulation for.

Your main goal for this question is to create an animation of the simulation showing the 2D grid state and a graph of the population sizes over time. Include your code below. (You will see several code cells which are included in the workbook should you wish to use them.)

Code Cell 1 of 37

```
In [3] 1 # Code was adapted from session 8 (percolation model)
        2 import matplotlib.pyplot as plt
        3 import numpy as np
        4 import random
        5
        6 class PredatorPreySimulation:
        7     """
        8     This class defines a simulation for a simple predator and prey model
        9     using a cellular automaton.
       10
       11     The rules of the model used to implement the simulation are the
       12     following:
       13     - The state of each cell is updated at each time step according to
       14       the following rules:
       15     - The predator will eat neighboring prey and reproduce at the
       16       location of the consumed prey with a certain probability of predator
       17       birth rate.
       18     - If there are fewer than 2 prey cells in the neighborhood, the
       19       predator cell becomes empty.
       20     - The prey will reproduce into empty neighboring cells.
       21     - The predator dies with a certain probability of predator death
       22       rate.
       23     - The prey dies with a certain probability of prey death rate.
       24     - Moore neighborhood
       25
       26     Attributes:
```

```

22         The size of the grid
23     state: list of lists of int
24         Represents the current state of each cell (empty = 0, prey = 1,
and predator = 2)
25     next_state: list of lists of int
26         Represents the state at the next time step of each cell (empty =
0, prey = 1, and predator = 2)
27     predator_death_rate: float
28         Natural mortality rate of predators
29     predator_birth_rate: float
30         Probability of predators to reproduce in a cell where they
consumed prey
31     prey_density: float
32         The initial probability of a cell containing a prey
33     predator_density: float
34         The initial probability of a cell containing a predator
35     step_counter: int
36         The current step. Simulation (initial state) begins at step 0
37     """
38
39     empty, prey, predator = [0, 1, 2] # Setup state values
40
41     def __init__(self, predator_death_rate, prey_death_rate,
predator_birth_rate, prey_density=0.2, predator_density=0.2, size=35):
42         self.size = size
43         self.state = np.zeros((size, size))
44         self.predator_death_rate = predator_death_rate
45         self.prey_death_rate = prey_death_rate
46         self.predator_birth_rate = predator_birth_rate
47         self.prey_density = prey_density
48         self.predator_density = predator_density
49         self.step_counter = 0
50         self.next_state = np.zeros((size, size))
51
52     def initialize(self):
53         """ A function that initializes the initial state of the
simulation (of the grid) at step 0. """
54
55         # Initialize all zeros
56         self.state.fill(self.empty)
57         # Place random prey and predators with the requested probability.
58         for i in range(self.size):

```

```

59         for j in range(self.size):
60             if random.random() < self.prey_density:
61                 self.state[i][j] = self.prey
62             elif random.random() < self.predator_density:
63                 self.state[i][j] = self.predator
64             else:
65                 self.state[i][j] = self.empty
66
67         self.changed = True # Track whether the state changed between
the
68                             # previous time step and the current one
69
70         # For plotting the simulation later.
71         self.figure, self.axes = plt.subplots()
72
73
74     def observe(self):
75         """A method used to plot the state of the configuration on the
plot at each step"""
76         plot = self.axes.imshow(
77             self.state, vmin=0, vmax=2, cmap = 'binary')
78         self.axes.set_title(f'State at step {self.step_counter}')
79         return plot
80
81     def update(self):
82         """ A method that updates the current state to the next state at
each step.
83
84         It follows the rules of the model in this order:
85         1. predator eats and reproduces,
86         2. then prey reproduces,
87         3. then predator and prey die randomly
88         """
89
90         # Predators remain the same (until rule 3 where they may die
randomly).
91         predators = np.where(self.state == self.predator)
92         for i in range(len(predators[0])):
93             self.next_state[predators[0][i], predators[1][i]] =
self.predator
94
95         ### RULE 1

```

```

96         # Rule 1: predator eats and reproduces
97         # Check which prey should be eaten (all those with predator
neighbors)
98         ys, xs = np.where(self.state == self.prey)
99         predator_neighbor = np.zeros(len(xs), dtype=bool)
100
101         # Moore neighborhood with radius of 1
102         for dx in [-1, 0, 1]:
103             for dy in [-1, 0, 1]:
104                 # Boolean array indicates whether there is a predator.
105                 predator_neighbor = predator_neighbor | (
106                     self.state[(ys + dy) % self.size, (xs + dx) %
self.size] == self.predator)
107
108         # Prey with predator neighbors get eaten.
109         self.next_state[(ys[predator_neighbor], xs[predator_neighbor])] =
self.empty
110
111         # Prey without predator neighbors stay the same (for now until
rule 3 random dying)
112         self.next_state[(ys[~predator_neighbor], xs[~predator_neighbor])]
= self.prey
113
114         # Predator reproduce
115         # Traverse all cells of predators and randomly reproduce at eaten
prey cells.
116         ys, xs = np.where(self.state == self.predator)
117
118         # For each predator, go through the neighborhood
119         for y, x in zip(ys, xs):
120             for dx in [-1, 0, 1]:
121                 for dy in [-1, 0, 1]:
122                     # If the cell in the neighborhood is prey, reproduce
with a chance of predator_birth_rate
123                     if self.state[(y + dy) % self.size, (x + dx) %
self.size] == self.prey and random.random() < self.predator_birth_rate:
124                         self.next_state[(y + dy) % self.size, (x + dx) %
self.size] = self.predator
125
126         ### RULE 2
127         # Rule 2: prey reproduces
128         # Moore neighborhood with radius of 1
129         prey_indices = np.argwhere(self.next_state == self.prey)

```



```

131         # Traverse each prey cell
132         for i in range(len(preying_indices)):
133             x, y = preying_indices[i]
134             # Check each neighbor of the prey
135             for dx in [-1, 0, 1]:
136                 for dy in [-1, 0, 1]:
137                     # If the neighboring cell is empty, prey reproduces
138                     there.
139                     if self.next_state[(x + dx) % self.size, (y + dy) %
140 self.size] == self.empty:
141                         self.next_state[(x + dx) % self.size, (y + dy) %
142 self.size] = self.prey
143
144     ### RULE 3
145     # Rule 3: predator and prey die randomly
146     # Part 1: Prey die
147     ys, xs = np.where(self.state == self.prey) # get prey cells
148     dead_preying = np.zeros(len(xs), dtype=bool) # list to update
149     whether that prey cell died (got eaten so it's empty)
150     for i, (y, x) in enumerate(zip(ys, xs)):
151         if not predator_neighbor[i] and random.random() <
152 self.preying_death_rate and self.next_state[y, x] != self.predator:
153             dead_preying[i] = True
154
155     # Set the state to 0 of the prey who got eaten during this
156     update.
157     self.next_state[(ys[dead_preying], xs[dead_preying])] = self.empty
158
159     # Part 2 of third rule: Predators die randomly
160     ys, xs = np.where(self.state == self.predator) # get predators
161     self.next_state[(ys, xs)] = self.predator
162
163     dead_predator = np.zeros(len(xs), dtype=bool) # list to update
164     whether that predator cell died randomly (empty)
165
166     # Traverse predator cells
167     for i, (y, x) in enumerate(zip(ys, xs)):
168         if random.random() < self.predator_death_rate:
169             dead_predator[i] = True
170
171     self.next_state[(ys[dead_predator], xs[dead_predator])] =
172 self.empty # set the dead ones to empty
173
174

```

```

166         # Part 3: Predators die when not enough prey in the neighborhood
167         # Traverse all cells of predators.
168         ys, xs = np.where(self.state == self.predator)
169         for y, x in zip(ys, xs):
170             # Get number of prey neighbors for each predator.
171             current_preys_neighbors = 0
172             for dx in [-1, 0, 1]:
173                 for dy in [-1, 0, 1]:
174                     if self.state[(y + dy) % self.size, (x + dx) %
self.size] == self.prey :
175                         current_preys_neighbors+=1
176             # If 0 or 1 prey only, the predator dies
177             if current_preys_neighbors < 2:
178                 self.next_state[y, x] = self.empty
179
180         # Update the next state and the step counter
181         self.changed = (self.next_state != self.state).any()
182         self.state, self.next_state = self.next_state,
np.zeros((self.size, self.size))
183
184         self.step_counter += 1
185

```

Run Code

Code Cell 2 of 37

```

In [1] 1 import matplotlib.pyplot as plt
        2 import numpy as np
        3
        4 def make_animation(sim, total_frames, steps_per_frame=1, interval=100):
        5     """
        6         This function creates a visualization for the behavior of the
        7         predator prey model.
        8
        9         Attributes
10         -----
11         sim: instance of a class
12             The simulation to be visualized

```

```

13         The number of frames we want to visualize
14     steps_per_frame: int
15         The number of steps (updates of the state of the simulation) per
    frame
16     interval: int
17         Delay between frames in milliseconds
18     """
19     # Import libraries
20     from matplotlib.animation import FuncAnimation
21     from IPython.display import HTML
22     from tqdm import tqdm
23
24     def update(frame_number):
25         """ This function generates and updates the state of the
    simulation and plot"""
26         for _ in range(steps_per_frame):
27             sim.update()
28             progress_bar.update(1)
29             return [sim.observe()]
30
31     sim.initialize() # initialize the simulation
32     progress_bar = tqdm(total=total_frames) # include a progress tracker
33     # Make animation
34     animation = FuncAnimation(
35         sim.figure, update, init_func=lambda: [], frames=total_frames,
    interval=interval)
36     output = HTML(animation.to_html5_video())
37     sim.figure.clf()
38     return output

```

Run Code

Code Cell 3 of 37

```

In [212] 1 sim = PredatorPreySimulation(predator_death_rate=0, prey_death_rate=0.5,
    predator_birth_rate=1, prey_density=0.6, predator_density=0.2)
    2 make_animation(sim, total_frames=10, steps_per_frame=1)

```

Run Code

Out [212] 100%|██████████| 10/10 [00:00<00:00, 16.65it/s]

<Figure size 432x288 with 0 Axes>

1.00

0:00 / 0:01



Code Cell 4 of 37

```
In [213] 1 sim = PredatorPreySimulation(predator_death_rate=0, prey_death_rate=0.5,
    2 predator_birth_rate=1, prey_density=0.2, predator_density=0.2, size = 1)
    3 make_animation(sim, total_frames=50, steps_per_frame=1)
```

Run Code

Out [213] 100%|██████████| 50/50 [00:05<00:00, 8.80it/s]

<Figure size 432x288 with 0 Axes>

1.00

0:00 / 0:05

Code Cell 5 of 37

```
In [214] 1 sim = PredatorPreySimulation(predator_death_rate=0, prey_death_rate=0,
    predator_birth_rate=1, prey_density=0.6, predator_density=0.4)
    2 make_animation(sim, total_frames=5, steps_per_frame=1)
```

Run Code

Out [214] 100%|██████████| 5/5 [00:00<00:00, 16.22it/s]

<Figure size 432x288 with 0 Axes>

1.00

0:00 / 0:00

## 2.2. Analyzing the simulation

- LOs: #EmpiricalAnalysis, #TheoreticalAnalysis, #cs166-Modeling

Select three distinct sets of parameter values for the predator birth rate, predator death rate, and prey death rate of your choice so that you can study interesting regimes in this system. Run the simulation multiple times for each set of parameters, and then answer the following questions.

Question 2 of 11

**(a)** Identify at least two edge cases where you can argue that the simulation is implemented as intended.

Normal  **B** *I* U  ” ‹›     A

There are many different edge cases that we can study. I will select two specific ones that show that the python implementation of the model works as intended.

1) In the initial state, there are no prey and no predators. We expect the whole grid to just remain empty no matter how long we run it for. This is because no cells can reproduce and everything should just remain as empty.

The parameters that correspond to this edge case are: predator\_death\_rate=0, prey\_death\_rate=0, predator\_birth\_rate=1, prey\_density=0, predator\_density=0

2) In the initial state, begin with only prey. Eventually this should converge to a grid full of prey.

The parameters that correspond to this edge case are: predator\_death\_rate=0, prey\_death\_rate=0, predator\_birth\_rate=1, prey\_density=0.01, predator\_density=0

3) In the initial state, begin with only predators. They will all die at the next step because there are not enough prey in the neighborhood.

The parameters that correspond to this edge case are: predator\_death\_rate=0, prey\_death\_rate=0, predator\_birth\_rate=1, prey\_density=0, predator\_density=1

4) An interesting regime: predators have enough prey to eat until the prey population declines. Predators start declining while prey start reproducing more. Which means the

The parameters that correspond to this edge case are: predator\_death\_rate=0.04, prey\_death\_rate=0.005, predator\_birth\_rate=0.08, prey\_density=0.2, predator\_density=0.2

Code Cell 6 of 37

```
In [215] 1 # edge case 1 - no one
          2 sim = PredatorPreySimulation(predator_death_rate=0, prey_death_rate=0,
          3 predator_birth_rate=1, prey_density=0, predator_density=0)
          4 make_animation(sim, total_frames=50, steps_per_frame=1)
          5
          6 # edge case 2 - only prey
          7 sim = PredatorPreySimulation(predator_death_rate=0, prey_death_rate=0,
          8 predator_birth_rate=1, prey_density=0.01, predator_density=0)
          9 make_animation(sim, total_frames=10, steps_per_frame=1)
          10
```

Run Code

```
Out [215] 100%|██████████| 50/50 [00:05<00:00, 8.57it/s]
          100%|██████████| 10/10 [00:00<00:00, 16.37it/s]
```

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

1.00

0:00 / 0:01



## Code Cell 7 of 37

```

In [81] 1 # edge case 3 - only predators and they all die
2 sim = PredatorPreySimulation(predator_death_rate=0, prey_death_rate=0,
3 predator_birth_rate=1, prey_density=0, predator_density=1)
4
5 # There are only 2 code cells and I couldn't add more cases. Please
6 comment or uncomment one of the lines to see the output for both of the
7 time steps. Forum workbooks only show the latest line of code (so you
8 will only see the white grid at time step 1).
9
10 make_animation(sim, total_frames=1, steps_per_frame=0) # All black at
11 step 0 - all cells are predators
12
13 make_animation(sim, total_frames=1, steps_per_frame=1) # All white at
14 step 1 - all predators died and all cells are empty
15
16 """
17
18 # An interesting regime: predators have enough prey to eat until the prey
19 population declines. Predators start declining while prey start
20 reproducing more. Which means the predators have enough food again. Forms
21 different patterns.
22
23 sim = PredatorPreySimulation(predator_death_rate=0.04,
24 prey_death_rate=0.005, predator_birth_rate=0.08, prey_density=0.2,
25 predator_density=0.2)
26
27 make_animation(sim, total_frames=50, steps_per_frame=1)
28
29 """

```

Run Code

```

Out [81] 100%|██████████| 1/1 [00:00<00:00, 9.15it/s]
100%|██████████| 1/1 [00:00<00:00, 6.75it/s]

'\n# An interesting regime: predators have enough prey to eat until the prey
population declines. Predators start declining while prey start reproducing
more. Which means the predators have enough food again. Forms different
patterns. \nsim = PredatorPreySimulation(predator_death_rate=0.04,
prey_death_rate=0.005, predator_birth_rate=0.08, prey_density=0.2,
predator_density=0.2)\nmake_animation(sim, total_frames=50,
steps_per_frame=1)\n'

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

```



Question 3 of 11

**(b)** Create a graph of the population size over time, including the 95% confidence interval of the mean for each of the three sets of parameters you have identified. Analyze and interpret the empirical results by comparing the population dynamics and the behavior of the system under different parameter values.

Normal  **B** *I* U  ” ‹/›     A

I have made plots for each of the 4 cases that I studied and described above empirically.

Case 1: we expect that the proportions of both the prey and predator populations will remain at 0% relevant to all grid cells. If we look at the first histogram below we can see that both means of the prey and predator fractions are centered at 0 %.



#### PREY STATISTICS

Sample mean: 0.0

95% confidence interval of population mean: [0.0, 0.0]



#### PREDATOR STATISTICS

Sample mean: 0.0

95% confidence interval of population mean: [0.0, 0.0]

It makes sense that both sample means are 0 and that both CIs are narrow and just point to 0 - there shouldn't be ANY other measurements in the data. This also corresponds with the trend we see in the animated grid for the previous question.

Please note that the histograms might be a bit hard to read. Again, here we are analyzing EDGE cases. So many of the measurements overlap on the histograms (i.e. the distributions, mean, and confidence intervals). This might make it a bit hard to read but that's why you can see the statistics printed out as well. In case 4 when we see different patterns you can properly see the CI.

Case 2: In the initial state, begin with only prey. Eventually this should converge to a grid full of prey. We expect that the prey will represent all of the population in the end while the predators will represent 0. If we look at the second histogram below, this is exactly what we see. The prey population is centered at 1:1 ratio to the total grid while the predators are at 0.



#### PREY STATISTICS

95% confidence interval of population mean: [1.0, 1.0]

 PREDATOR STATISTICS

Sample mean: 0.0

95% confidence interval of population mean: [0.0, 0.0]

Again, this makes sense that the means are accurate and that both CIs are narrow and just point to 0 or 1.0 - there shouldn't be ANY other measurements in the data. The prey always converges to ALL the grid cells while the predators always disappear. This also corresponds with the trend we see in the animated grid for the previous question.

Case 3: In the initial state, begin with only predators. They will all die at the next step because there are not enough prey in the neighborhood.

This is similar to case 1. There's no prey and all predators ALWAYS disappear so ALL measurements are at 0%.

 PREY STATISTICS

Sample mean: 0.0

95% confidence interval of population mean: [0.0, 0.0]

 PREDATOR STATISTICS

Sample mean: 0.0

95% confidence interval of population mean: [0.0, 0.0]

Case 4: This is a more interesting case to analyze. Where we observe waves of different patterns.

This regime doesn't converge to one specific state. It oscillates in similar patterns.

We can see that in the pattern on the graph. The range of the prey is from 0.35 to about 0.85 of the whole grid. This makes sense - if we look at the trend in the previous question we can see that the prey population comes and goes in waves - it shrinks and then it takes up almost all of the grid. This wave pattern can also be seen through the mostly even distribution of the prey distribution.

 PREY STATISTICS

Sample mean: 0.56

95% confidence interval of population mean: [0.5404000000000001, 0.5796]

It makes sense that the mean would fall somewhere in the middle of the range. Since the pattern we see from the empirical results is this wave like pattern from shrinking to growing, then the mean is about 50% of the grid which we can also see being reflected in the statistics. The confidence interval leaves a small room for error as well

For the predator population the range is much narrower - approximately [0.05, 0.15]. The peak is at around 0.12. If we look back at the previous question and the patterns that emerge we can see how the predators 'chase' around the prey and eat it but they do not reproduce. When there isn't enough prey left, some of them die off. But for the most part the total number of predators on the grid remains constant.



## PREDATOR STATISTICS

Sample mean: 0.12

95% confidence interval of population mean: [0.12, 0.12]

From these observations we can conclude some points about the effects of the parameters on the system:

- predator\_death\_rate
  - A higher value means that the predator population will decrease faster -- they will eat less prey and leave more space for the prey population to reproduce.
  - A lower value would mean the opposite thing. The predators will survive for longer period of time, consuming more prey and having a negative effect on the growth rate of the prey.
- prey\_death\_rate
  - A higher value would mean that there are more random deaths (natural mortality rate). Therefore the prey population will shrink. There would be less prey for the predators to eat so it might mean that the predator population decreases as well.
  - A lower value would mean that there are fewer random deaths (prey lives longer naturally). This would mean that the predators have more food to consume so their population might increase.
- predator\_birth\_rate
  - A higher value will mean that the predator population would increase over time as long as there is prey in the system to be eaten. It will have a negative effect on the growth rate of the prey.
  - A lower predator birth rate will mean that the number of prey would stay more constant over time. They will be less likely to reproduce so they will consume less of the prey.

Code Cell 8 of 37

```
In [90] 1 def plot_hist(data1, data2, percentile_string, title, units,
        2             xlabel_string, bins):
        3     """ A function nthat is used to create histograms including the mean,
        4         95% confidence interval, and 99th percentile
```

```

5         The data to be plotted
6     title: str
7         The title of the histogram
8     units: str
9         The units of the variable
10    xlabel_string: str
11        The label for the x-axis
12    """
13
14    # Create the figure
15    plt.figure(figsize=(12, 6))
16    plt.hist(data1, bins=bins, density=True, color = 'blue', alpha =0.8,
17             edgecolor='white', label='prey')
18
19    # Labels
20    plt.title(title)
21    plt.xlabel(xlabel_string)
22    ylim = plt.ylim()
23
24    plt.ylim(0, ylim[1])
25
26    # PREY
27    print("\n PREY STATISTICS")
28
29    # Mean 95% CI
30    m = np.mean(data1)
31    m_approx = round(m, 2)
32    print('Sample mean:', m_approx)
33
34    t = sts.sem(data1)
35    t_approx = round(t, 2)
36
37    print('95% confidence interval of population mean:', [ m_approx -
38    1.96*t_approx, m_approx + 1.96*t_approx])
39
40    plt.fill_between([m_approx - 1.96*t_approx, m_approx +
41    1.96*t_approx], 0, ylim[1], color = 'blue', alpha = 0.3, label = '95%
42    confidence interval of mean of prey', zorder=2)
43
44    # Add a line for the mean
45    plt.axvline(m_approx, color = 'blue', linestyle='--', label = 'mean

```

```

42
43     print("🐾 PREDATOR STATISTICS")
44
45     # Mean 95% CI
46     m = np.mean(data2)
47     m_approx = round(m, 2)
48     print('Sample mean:', m_approx)
49
50     t = sts.sem(data2)
51     t_approx = round(t, 2)
52
53     print('95% confidence interval of population mean:', [ m_approx -
54     1.96*t_approx, m_approx + 1.96*t_approx])
55
56     # Add the confidence interval of the mean
57     plt.fill_between([m_approx - 1.96*t_approx, m_approx +
58     1.96*t_approx], 0, ylim[1], color = 'red', alpha = 0.3, label = '95%
59     confidence interval of mean of predators', zorder=1)
60
61     # Add a line for the mean
62     plt.axvline(m_approx, color = 'red', linestyle='--', label = 'mean
63     predator')
64
65     plt.legend()
66     plt.show()
67
68 import scipy.stats as sts
69
70 class PredatorPreySimulation_Graphs(PredatorPreySimulation):
71     # Modified this to remove the fig cna dconfig, otherwise the
72     # initialzie method returns a series of empty plots.
73     def initialize(self):
74         self.state.fill(self.empty)
75         # Place random prey and predators with the requested probability.
76         for i in range(self.size):
77             for j in range(self.size):
78                 if random.random() < self.prey_density:
79                     self.state[i][j] = self.prey
80                 elif random.random() < self.predator_density:
81                     self.state[i][j] = self.predator
82                 else:
83                     self.state[i][j] = self.empty

```

```

79         self.changed = True # Track whether the state changed between
the
80                               # previous time step and the current one
81
82
83 parameters = [{'predator_death_rate':0, 'prey_death_rate':0,
'predator_birth_rate':1, 'prey_density':0, 'predator_density':0},
84               {'predator_death_rate':0, 'prey_death_rate':0,
'predator_birth_rate':1, 'prey_density':0.01, 'predator_density':0},
85               {'predator_death_rate':0, 'prey_death_rate':0,
'predator_birth_rate':1, 'prey_density':0, 'predator_density':1},
86               {'predator_death_rate':0.04, 'prey_death_rate':0.005,
'predator_birth_rate':0.08, 'prey_density':0.2, 'predator_density':0.2}]
87
88 iterations = 100
89
90 # For each parameter configuration I chose earlier.
91 for parameter_config in parameters:
92
93     # Store the data to plot.
94     prey_population_data = np.zeros(iterations)
95     predator_population_data = np.zeros(iterations)
96
97     # Get many data points
98     for i in range(iterations):
99         sim =
PredatorPreySimulation_Graphs(predator_death_rate=parameter_config['preda
tor_death_rate'],
100     prey_death_rate=parameter_config['prey_death_rate'],
101     predator_birth_rate=parameter_config['predator_birth_rate'],
102     prey_density=parameter_config['prey_density'],
103     predator_density=parameter_config['predator_density'])
104         sim.initialize()
105
106         # Update the simulation so that it converges.
107         for j in range(50):
108             sim.update()
109
110         # Get data for the population
111         prey_population_data[i] = np.sum(sim.state == sim.prey)/(35**2)

```

```

112     predator_population_data[i] = np.sum(sim.state ==
113     sim.predator)/(35**2)
114
115     plot_hist(data1 = prey_population_data, data2 =
116     predator_population_data, percentile_string = 'of the count of the prey
117     population:', title = 'Distribution over prey population', units = '',
118     xlabel_string = 'Proportion of prey/predator populations vs grid size
119     (%)', bins = 10)

```

Run Code

Out [90]

🐘 PREY STATISTICS

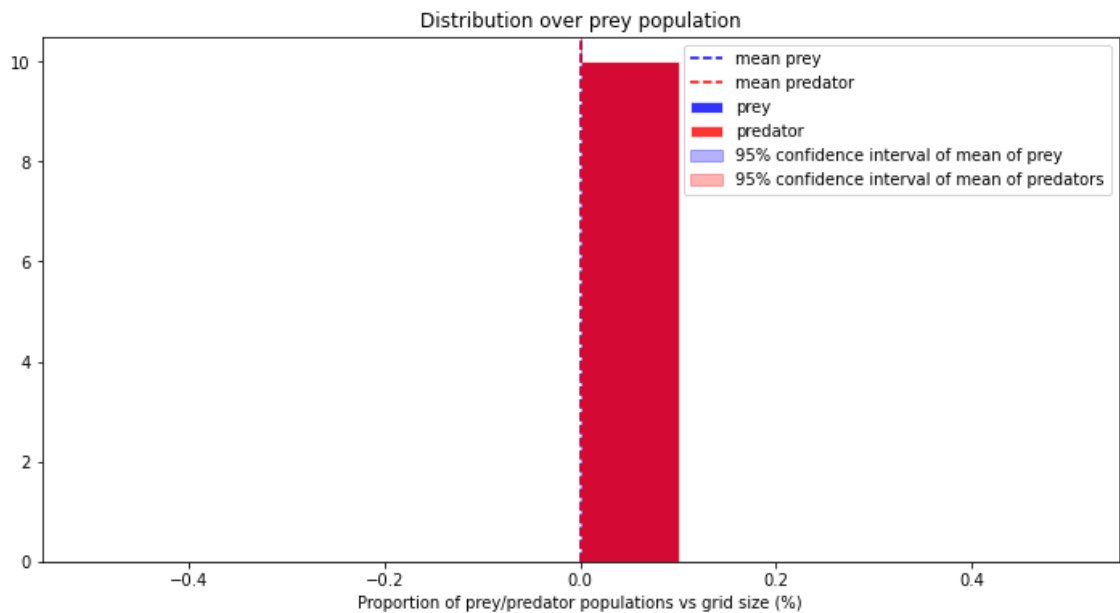
Sample mean: 0.0

95% confidence interval of population mean: [0.0, 0.0]

🐱 PREDATOR STATISTICS

Sample mean: 0.0

95% confidence interval of population mean: [0.0, 0.0]



🐘 PREY STATISTICS

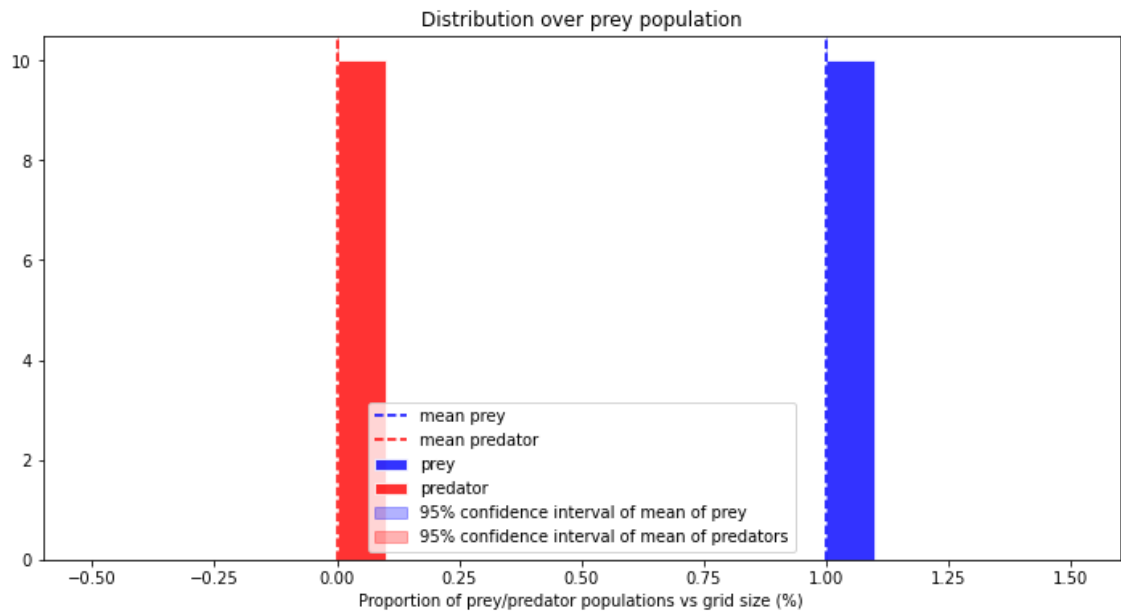
Sample mean: 1.0

95% confidence interval of population mean: [1.0, 1.0]

🐱 PREDATOR STATISTICS

Sample mean: 0.0

95% confidence interval of population mean: [0.0, 0.0]



#### 🐭 PREY STATISTICS

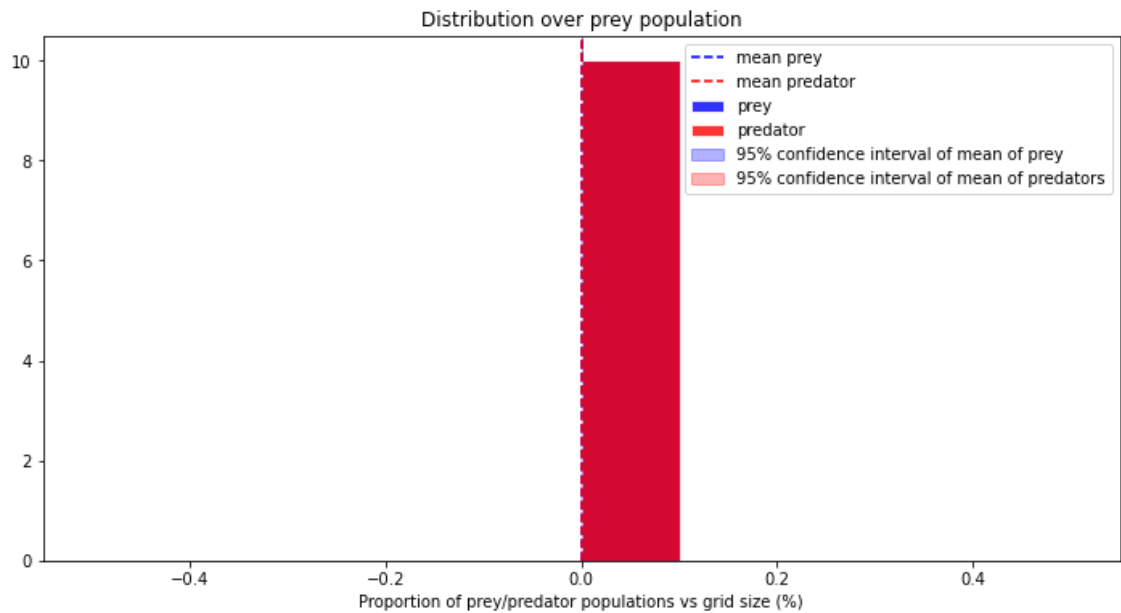
Sample mean: 0.0

95% confidence interval of population mean: [0.0, 0.0]

#### 🐱 PREDATOR STATISTICS

Sample mean: 0.0

95% confidence interval of population mean: [0.0, 0.0]



#### 🐭 PREY STATISTICS

Sample mean: 0.56

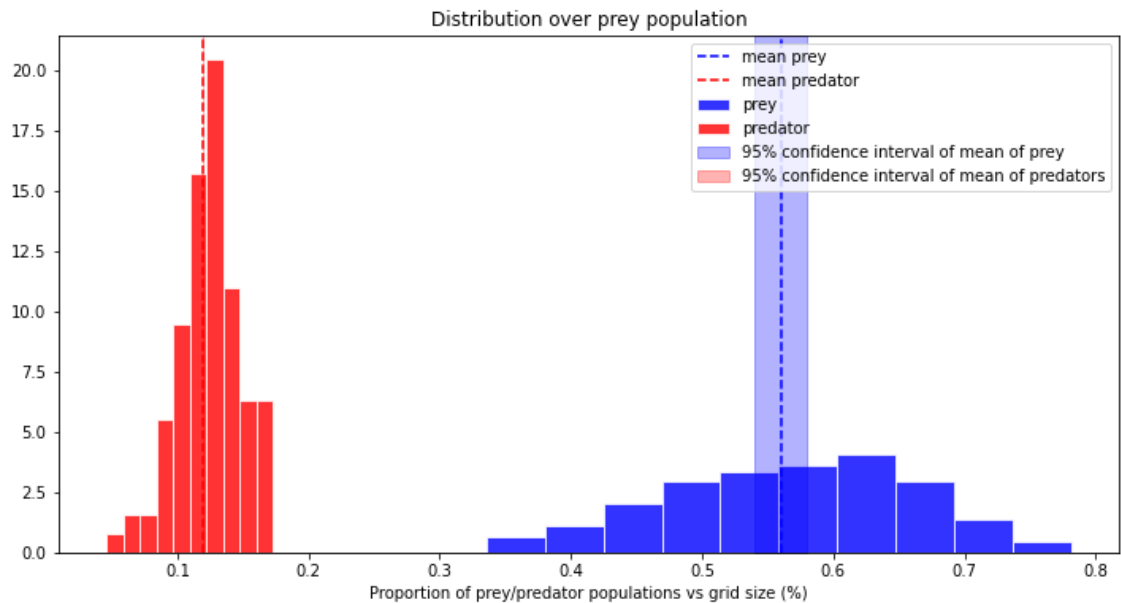
95% confidence interval of population mean: [0.5404000000000001, 0.5796]

#### 🐱 PREDATOR STATISTICS

Sample mean: 0.12

95% confidence interval of population mean: [0.12, 0.12]





Code Cell 9 of 37

```
In [83] 1 # your code her  
        2
```

Run Code

Code Cell 10 of 37

```
In [ ] 1 # your code here
```

Run Code

Question 4 of 11

(c) Run your simulation for vanishing prey death rate, predator death rate, and predator birth rate, and produce a new plot to illustrate the time evolution of the prey and predator population in this instance. Interpret the results you obtained.

Vanishing prey death rate, predator death rate, and predator birth rate means that I will set all of their values to equal 0.

We know that no prey or predator will die randomly and that predators will NOT reproduce whenever they eat prey.

Let's look at the plot below. We can see that both populations start at a similar proportion of 0.2 which is expected as this is the given density for both prey and predators.

After that, the predators die off as they are not able to reproduce and converge to a value close to 0 - about 1% of the grid.

On the other hand, the prey begin reproducing and converge to a value close to 1 - about 94% of the grid.

This makes sense because the predators that are far away from the prey initially die off. On the other hand there must be a few predators close to the prey that continue living. They eat a bit of the prey at every step but they cannot reproduce. They also never die off since their death rate is 0.

Code Cell 11 of 37

```
In [217] 1 # Set the number of steps the simulation runs for (each step is one day)
          2 steps = 100
          3 prey_population_data = np.zeros((steps,))
          4 predator_population_data = np.zeros((steps,))
          5 total_population_data = np.zeros((steps,))
          6
          7 sim = PredatorPreySimulation(predator_death_rate=0, prey_death_rate=0,
          8 predator_birth_rate=0, prey_density=0.2, predator_density=0.2)
          9 sim.initialize()
          10
          11 prey_count = np.sum(sim.state == sim.prey)
          12 predator_count = np.sum(sim.state == sim.predator)
          13 prey_population_data[0] = prey_count/(35**2)
          14 predator_population_data[0] = predator_count/(35**2)
          15 total_population_data[0] = prey_count+predator_count/(35**2)
```

```

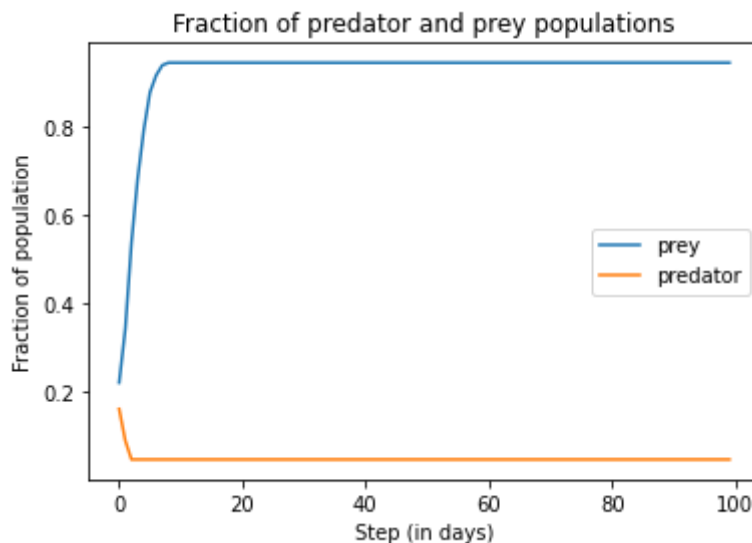
17
18 for i in range(1, steps):
19     sim.update()
20     prey_count = np.sum(sim.state == sim.prey)
21     predator_count = np.sum(sim.state == sim.predator)
22     prey_population_data[i] = prey_count/(35**2)
23     predator_population_data[i] = predator_count/(35**2)
24     total_population_data[i] = (prey_count+predator_count)/(35**2)
25
26
27 # Plot experimental results
28
29 plt.title('Fraction of predator and prey populations')
30 plt.xlabel('Step (in days)')
31 plt.ylabel('Fraction of population')
32 plt.plot(list(range(steps)), prey_population_data, label='prey')
33 plt.plot(list(range(steps)), predator_population_data, label='predator')
34 plt.legend()
35

```

Run Code

Out [217]

<matplotlib.legend.Legend at 0x7f71e81753a0>



Code Cell 12 of 37

```
In [218] 1 print("Prey converges to a fraction of:",prey_population_data[-1])
          2 print("Predator converges to a fraction
of:",predator_population_data[-1])
```

Run Code

```
Out [218] Prey converges to a fraction of: 0.9436734693877551
Predator converges to a fraction of: 0.045714285714285714
```

Code Cell 13 of 37

```
In [219] 1 sim = PredatorPreySimulation(predator_death_rate=0, prey_death_rate=0,
predator_birth_rate=0, prey_density=0.2, predator_density=0.2)
          2 make_animation(sim, total_frames=100, steps_per_frame=1)
```

Run Code

```
Out [219] 96%|██████████| 96/100 [00:18<00:01, 2.95it/s]
... STREAM TRUNCATED
```

<Figure size 432x288 with 0 Axes>

1.00

0:00 / 0:10



(d) Explicitly compare your simulation results with the results obtained from [this](#) simulation.

Normal    ▾    **B**   *I*   U   🔒   ”   ‹ ›    $\frac{1}{2}$     $\frac{2}{3}$     $\frac{3}{4}$     $\frac{4}{5}$    A

### Comparison with c):

I set the 3 parameters to equal 0. I initialized the grid for the Reset menu by clicking on Random -- this leads to a random configuration of predators and prey.

The I clicked the button on the top to load the 'Next 25 generations' 4 times to get to step 100.

I repeated this process multiple times to make sure the behavior always converges in the same way.

The results from my simulation are very similar to the results from this simulation.

In this given simulation:

- During the first few steps the predators eat the prey that is already there.
- Then they run out of prey so most of them die off. Only a few (mostly individual cells) survive - the ones positioned next to a cluster of prey.
- The prey then begins reproducing rapidly until it fills the rest of the grid. The predators stay the same since they cannot reproduce and they cannot die off.

We can observe the population proportions from the phase space graph. This is one example: 26 predators and 1127 prey.

In terms of proportions this means that the prey is  $1127/35^2 = 0.92$  and the predators are  $26/35^2 = 0.02$  of the 35x35 grid. This is very close to the empirical results that I obtained as well.

This was comparing the graph. Now I will also compare the grids. In the grid from the given simulation on the website we can see that there can be certain structures of predators (i.e. 2 predators with a space in the middle) which will prevent a prey to reproduce there. This means there can be a few empty cells. Now look at code cell 14. We can see similar structures. The black cells are the predators, the white are empty, and the grey cells are prey. We can see exactly those formations of 2 predators with a space in the middle that remains empty or other structures like predators in a square.

So both the mathematical interpretation in terms of proportion of populations and the practical practical interpretation in terms of the grid, are similar between my implementation and the implementation on the website.

Initially there was a "mistake" in my results:

One difference in my results was when we look at the prey population. There was a slight oscillation up and down. This isn't supposed to happen. The system is supposed to converge to a state that does not change after some point.

The slight oscillation might be because the very few predators eat the some prey cells. BUT the prey cells are supposed to reproduce on the same step which means they would fill up that space back.

I eventually solved this BUT it all depends on the interpretation of the rules. How does the prey reproduce? Do we consider the new empty spaces left after the predator just ate or not? This is in rule 2 of my code starting at line 84 in the main code in code cell 1.

**Other than than I also compared my simulation with all the edge cases I considered above and some other:**

Edge case 1: the grid begins only with empty cells. Both in my simulation and in the online simulation the grid remain empty throughout.

Edge case 2: the grid begins only with prey. In my edge case I set the prey death rate to 0 so no prey dies. I set the same parameters for the online simulation. In both cases we observe the same trend: the whole grid fills up with prey. I also tried setting up a slightly higher parameter for the prey death rate (0.2) and in both simulations the prey takes up most of the grid but not entirely.

Edge case 3: the grid begins only with predators. In both simulations they all die out and the grid remains empty.

These were all good signs that the update rules operate as expected.

I also compared some other regimes.

Code cell 5: predator\_death\_rate=0, prey\_death\_rate=0, predator\_birth\_rate=1, prey\_density=0.6, predator\_density=0.4

In both simulations eventually the grid became empty because the predators reproduce each time and consume all the prey so they're left with no food and die off.

Code cell 7: predator\_death\_rate=0.04, prey\_death\_rate=0.005, predator\_birth\_rate=0.08, prey\_density=0.2, predator\_density=0.2

An interesting regime: predators have enough prey to eat until the prey population declines. Predators start declining (not enough food) while prey start reproducing more (more empty space). Which means the predators have enough food again. This repeats and keeps

forming different patterns. The predators kind of follow the prey around to eat it. Both simulations exhibit the same pattern.

I also tested some other special cases. In the online simulation there's a special pattern called 'Chase' where a triangle widget of prey and some predators move diagonally from top right to bottom left. In code cell 15 I wanted to see if my simulation leads to the same pattern. After setting up the parameters and initializing the initial state, you can watch the simulation which produces exactly the same trend.

Overall after comparing some edge cases including cases that form patterns I would say that both simulations act similarly. After testing it, believe my implementation works as expected.

Code Cell 14 of 37

```
In [220] 1 sim = PredatorPreySimulation(predator_death_rate=0, prey_death_rate=0,
    2 predator_birth_rate=0, prey_density=0.2, predator_density=0.2)
    3 make_animation(sim, total_frames=100, steps_per_frame=1)
```

Run Code

Out [220] 95%|██████████| 95/100 [00:18<00:01, 2.87it/s]  
... STREAM TRUNCATED

<Figure size 432x288 with 0 Axes>

1.00

0:00 / 0:10



Code Cell 15 of 37

```
In [7] 1 # your code here
2 sim = PredatorPreySimulation(predator_death_rate=0, prey_death_rate=0,
3 predator_birth_rate=1, prey_density=0, predator_density=0)
4
5 def make_animation_edge(sim, total_frames, steps_per_frame=1,
6 interval=100):
7     """
8     This function creates a visualization for the behavior of the
9     predator prey model.
10
11     Attributes
12     -----
13     sim: instance of a class
14         The simulation to be visualized
15     total_frames: int
16         The number of frames we want to visualize
17     steps_per_frame: int
18         The number of steps (updates of the state of the simulation) per
19         frame
20     interval: int
21         Delay between frames in milliseconds
22     """
23     # Import libraries
24     from matplotlib.animation import FuncAnimation
25     from IPython.display import HTML
26     from tqdm import tqdm
27
28     def update(frame_number):
29         """ This function generates and updates the state of the
30         simulation and plot"""
31         for _ in range(steps_per_frame):
32             sim.update()
33             progress_bar.update(1)
34         return [sim.observe()]
35
36     sim.initialize()
37     empty = 0
38     sim.state.fill(empty) # initialize the simulation
```



```

34     sim.state[11][10] = 1
35     sim.state[11][11] = 1
36     sim.state[12][10] = 1
37     sim.state[12][11] = 1
38     sim.state[12][12] = 1
39     sim.state[12][13] = 1
40     sim.state[13][10] = 1
41     sim.state[13][11] = 1
42     sim.state[13][12] = 1
43     sim.state[13][13] = 1
44     sim.state[13][14] = 1
45     sim.state[9][11] = 2
46     sim.state[9][12] = 2
47     sim.state[10][11] = 2
48     sim.state[10][12] = 2
49     sim.state[10][13] = 2
50     sim.state[11][12] = 2
51     sim.state[11][13] = 2
52     sim.state[11][14] = 2
53     sim.state[12][13] = 2
54     sim.state[12][14] = 2
55     progress_bar = tqdm(total=total_frames) # include a progress tracker
56     # Make animation
57     animation = FuncAnimation(
58         sim.figure, update, init_func=lambda: [], frames=total_frames,
59         interval=interval)
59     output = HTML(animation.to_html5_video())
60     sim.figure.clf()
61     return output
62
63 make_animation_edge(sim, total_frames=50, steps_per_frame=1)

```

Run Code

Out [7]

```

0%|          | 0/50 [00:27<?, ?it/s]

6%|█         | 3/50 [00:00<00:02, 23.43it/s]
12%|██        | 6/50 [00:00<00:01, 23.13it/s]
18%|███       | 9/50 [00:00<00:01, 21.23it/s]

```

26%	████		13/50	[00:00<00:02, 17.91it/s]
30%	████		15/50	[00:00<00:02, 16.27it/s]
34%	████		17/50	[00:00<00:02, 14.97it/s]
38%	████		19/50	[00:01<00:02, 13.82it/s]
42%	████		21/50	[00:01<00:02, 12.60it/s]
46%	████		23/50	[00:01<00:02, 11.41it/s]
50%	████		25/50	[00:01<00:02, 10.78it/s]
54%	████		27/50	[00:02<00:02, 9.85it/s]
58%	████		29/50	[00:02<00:02, 9.44it/s]
60%	████		30/50	[00:02<00:02, 8.87it/s]
62%	████		31/50	[00:02<00:02, 8.60it/s]
64%	████		32/50	[00:02<00:02, 7.94it/s]
66%	████		33/50	[00:02<00:02, 7.48it/s]
68%	████		34/50	[00:02<00:02, 7.39it/s]
70%	████		35/50	[00:03<00:02, 7.29it/s]
72%	████		36/50	[00:03<00:01, 7.17it/s]
74%	████		37/50	[00:03<00:01, 7.07it/s]
76%	████		38/50	[00:03<00:01, 6.91it/s]
78%	████		39/50	[00:03<00:01, 6.75it/s]
80%	████		40/50	[00:03<00:01, 6.54it/s]
82%	████		41/50	[00:04<00:01, 6.33it/s]
84%	████		42/50	[00:04<00:01, 6.27it/s]
86%	████		43/50	[00:04<00:01, 6.17it/s]
88%	████		44/50	[00:04<00:01, 5.67it/s]
90%	████		45/50	[00:04<00:00, 5.50it/s]
92%	████		46/50	[00:04<00:00, 5.53it/s]

<Figure size 432x288 with 0 Axes>

1.00

0:00 / 0:05

Code Cell 16 of 37

In [ ]

```
1 # your code here
```

Run Code

Question 6 of 11

(e) The Lotka-Volterra equations are a set of mathematical equations that describe the dynamics of a predator-prey system. These equations can be used to predict the population growth and decline of both the predator and prey populations over time.

$$\frac{dp(t)}{dt} = \alpha p(t) - \beta p(t) P(t) \quad \text{Eq. (1)}$$

$$\frac{dP(t)}{dt} = \delta p(t) P(t) - \gamma P(t) \quad \text{Eq. (2)}$$

where  $p$  is the prey population,  $P$  is the predator population, and  $\{\alpha, \beta, \gamma, \delta\}$  is a set of (positive) interaction parameters, all constant in time.

Offer an intuitive explanation for why these equations are a sensible parametrization of the predator-prey system. Explain what are the variables, update rules, and any underlying assumptions important for this analytical model to be a good starting point for describing this system.

Normal  **B** *I* U  ” ‹/›     A

The mathematical representation of the system here is represented with two differential equations. We can see that they represent the predator and prey populations and the interaction between them as a function of time which makes sense since I did a similar thing for my implementation. Each step is actually one time unit (i.e. a day seems like appropriate unit).

The variables are:

- $t$ : independent variable representing the time (in some time unit like days)
- $p$ : the prey population as a function of time
- $P$ : the predator population as a function of time
- $\{\alpha, \beta, \gamma, \delta\}$  is a set of (positive) interaction parameters

- $\beta$ : the coefficient of the predator population, representing the predator's negative effects on prey population growth
- $\delta$ : the conversion rate of prey to predators, represents predators eating prey and reproducing in its place
- $\gamma$ : mortality death rate of predators

The update rules are represented by the two differential equations are explained in depth below. They describe the dynamics of the system in terms of how the two populations (prey and predators) change through time as they interact through prey and predator growth.

Equation 1: represents the development of the prey population over time

- The prey population grows at a rate of  $\alpha$  and is proportional to its current size. This is isolated and independent of predators - in the absence of them.
- However we also need to consider how predators affect the prey. The prey population shrinks at a rate of  $\beta$  (similar to the random death rate of prey that we have in the simulation) and is also affected by the current number of predators (negative growth). The random deaths represent natural mortality.

Equation 2: represents the development of the predator population over time

- The predator population grows at a rate of  $\delta$  (similar to the birth rate parameter for the predators to reproduce in our simulation) considering the current prey and predator populations. This term represents the chance of the predators to reproduce as they eat prey - growth is proportional to the rate at which predators consume prey.
- On the other hand, the predator population shrinks with a rate of  $\gamma$  (similar to the random death rate in our simulation). This is a natural rate of mortality.

Interestingly, something that these equations do not account for given the assignment prompt, specifically equation 2, is the fact that predators die if there are not enough prey. I am not familiar with differential equations but I believe that the second term could be modified so that it is also proportional to the prey population (i.e.  $\gamma * P(t) * p(t)$ ).

The equations of the mathematical model make several assumptions:

- Predator and prey interact randomly.
- The predators have the prey as a food source. However, it assumes that the prey has an infinite supply of food and other resources as well.
- The predators also just need the prey as a food source and no other resources - i.e. water.
- The set of (positive) interaction parameters are constant throughout the whole time. In reality they may change if they are impacted by other factors like a natural disaster or pollution that would increase the natural mortality rate for example.
- Predators can only reproduce in a rate proportional to the prey population (first term

These assumptions are relatively realistic and help us abstract unnecessary details. If we wish to study their relationship with food source or other resources or how a natural disaster or pollution might impact them, then that would be a different model with a different explanatory challenge and power. So for our purposes this is a good mathematical model.

Question 7 of 11

(f) Use a numerical method such as Euler's to solve the Lotka-Volterra equations and study the system's behavior. For this, choose appropriate values for the parameters  $\alpha, \beta, \gamma, \delta$  in the Lotka-Volterra equations, as well as initial population values for the predator and prey populations. You should include that information in the text box provided below. You can use a range of

- a. 0.5 to 0.8 for  $\alpha$ ,
- b. 0.0001 to 0.001 for  $\beta$ ,
- c. 0.2 to 0.5 for  $\gamma$  and
- d. 0.0001 to 0.001 for  $\delta$  as a starting point, but keep in mind that the actual parameter values may fall outside of these ranges depending on the specific system being modeled.

Code cells are provided for you to apply the numerical methods to solve the system's dynamics.

Normal        **B**    *I*    U                                A

I chose Euler's method to solve the Lotka-Volterra equations.

There are several parameter pairs I chose:

1) Case 1

- $x_{\text{initial}}=0, x_{\text{final}}=100, h=0.01, \alpha=0.8, \beta=0.0001, \gamma=0.5, \delta=0.0001,$   
 $\text{initial\_prey}=20, \text{initial\_predators}=20$
- In this case, there is an oscillating pattern. The predator and prey populations grow exponentially and peak at a similar level. Then they shrink exponentially and converge to values close to 0. At some point they rebound and this pattern repeats.

2) Case 2

- $x_{\text{initial}}=0, x_{\text{final}}=1000, h=0.01, \alpha=0.5, \beta=0.0005, \gamma=0.3, \delta=0.0001,$   
 $\text{initial\_prey}=20, \text{initial\_predators}=20$
- In this case, there is also an oscillation pattern but it's different.
  - The prey and predator populations grow rapidly but the prey reaches a higher peak. After that the predators eat most prey so both populations shrink rapidly again.
  - This pattern continues many times in short intervals of time but every time the peaks each of the two populations become higher and higher. So overall the pattern looks like the peaks are growing exponentially.

- It makes sense compared to the previous configuration because the natural death of the predators is lower so they bounce back faster. Also, beta, the coefficient of the predator population, representing the predator's negative effects on prey population growth, is lower so the prey population is able to reach a higher peak every time.

### 3) Case 3

- $x_{\text{initial}}=0$ ,  $x_{\text{final}}=50$ ,  $h=0.01$ ,  $\alpha=1$ ,  $\beta=0.000$ ,  $\gamma=1$ ,  $\delta=0.000$ ,  $\text{initial\_prey}=10$ ,  $\text{initial\_predators}=10$
- In this case, the predator population completely dies off while the prey population keeps growing endlessly.
- This makes sense (and it was the outcome I tried to cause). I wanted to see a pattern that wasn't oscillating. The growth rate of the prey is 1 (guaranteed) while the negative rate is 0 so the prey only keeps growing. On the other hand, it's the opposite for the predators. Gamma the death rate is guaranteed while the reproduction chance is 0. So all predators quickly die.

Code Cell 17 of 37

```
In [27] 1 # Code adapted from CS51 Assignment 2
2 def euler_method(x_initial=0, x_final=20, h=0.1, alpha=0.8, beta=0.0001,
3   gamma=0.5, delta=0.0001, initial_prey=20, initial_predators=20):
4
5     Parameters
6     -----
7     x_initial: int
8         Start time.
9     x_final: int
10        End time.
11     h: float
12        Step size.
13     alpha: float
14        Parameter: the reproduction rate of the prey population in the
15        absence of predators.
16     beta: float
17        Parameter: the coefficient of the predator population,
18        representing the predator's negative effects on prey population growth.
19     gamma: float
20        Parameter: mortality death rate of predators
21     delta: float
22        Parameter: the conversion rate of prey to predators, represents
```

```

21     initial_prey: int
22         Number of prey in the initial configuration.
23     initial_predator: int
24         Number of predators in the initial configuration.
25
26     Returns
27     -----
28     None (prints a plot)
29     """
30
31     # Set the step size and parameter values (values taken from table).
32     steps_number = int((x_final - x_initial)/h + 1) # number of steps
33
34     # Variables:
35     t = np.linspace(x_initial, x_final, steps_number) # storing t values
36     p = np.zeros(steps_number) # for storing prey population values
37     P = np.zeros(steps_number) # for storing predator population values
38
39     # Set initial condition.
40     P[0] = initial_predators
41     p[0] = initial_prey
42
43     # Define functions for the two differential equations.
44     def dpdt(t, p, P):
45         """ A function that describes the first differential equation
46         update rule for the Lotka-Volterra model dp/dt """
47         return alpha*p - beta*p*P
48
49     def dPdt(t, p, P):
50         """ A function that describes the second differential equation
51         update rule for the Lotka-Volterra model dP/dt """
52         return delta*p*P - gamma*P
53
54     # Iterate for the number of steps (iterative nature of Euler's
55     method).
56     for i in range(steps_number-1):
57         p[i+1] = p[i] + h*dpdt(t[i], p[i], P[i])
58         P[i+1] = P[i] + h*dPdt(t[i], p[i], P[i])
59
60     # Create the plot.
61     plt.plot(t,p,linewidth=2,label='p(t): prey', color = 'purple')

```

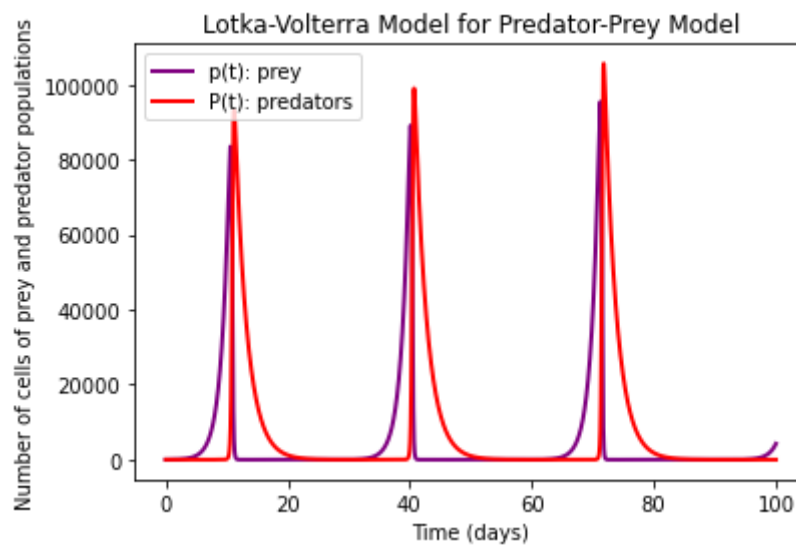
```

60
61 plt.title('Lotka-Volterra Model for Predator-Prey Model')
62 plt.xlabel('Time (days)')
63 plt.ylabel('Number of cells of prey and predator populations')
64 plt.legend(loc='best')
65 plt.show()
66
67 euler_method(h=0.01, x_final=100)

```

Run Code

Out [27]



Code Cell 18 of 37

```

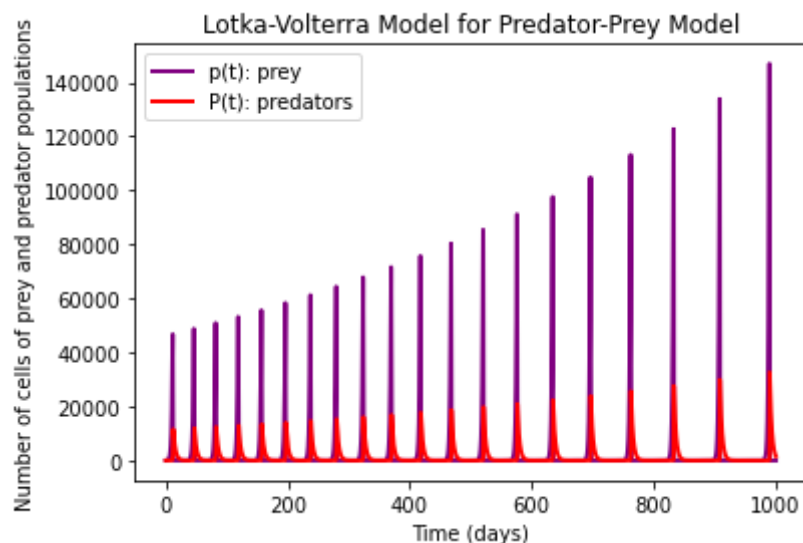
In [24] 1 # Interesting regime when the populations oscilate but also grow
        2 euler_method(x_initial=0, x_final=1000, h=0.01, alpha=0.5, beta=0.0005,
          gamma=0.3 , delta=0.0001, initial_preay=20, initial_predators=20)

```

Run Code



Out [24]

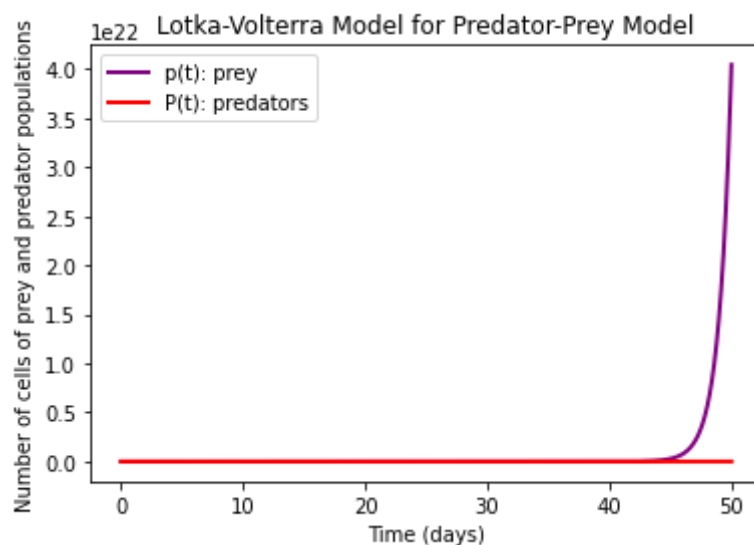


Code Cell 19 of 37

```
In [39] 1 # Case 3
        2 # Wanted to make sure the pattern doesn't oscillate when it shouldn't.
        3 euler_method(x_initial=0, x_final=50, h=0.01, alpha=1, beta=0.000,
          gamma=1, delta=0.000, initial_pre=10, initial_predators=10)
```

Run Code

Out [39]



Code Cell 20 of 37

```
In [39] 1 # your code here
```

Run Code

Code Cell 21 of 37

```
In [ ] 1 # your code here
```

Run Code

Question 8 of 11

**(g)** [Optional Challenge] Interpret the empirical results you obtained in light of this theoretical modeling. Make sure to make a thorough comparison between the expected theoretical results and the simulation results you obtained. Are there specific regimes where the theoretical approximation is particularly good, or bad? Explain your reasoning.

Normal        **B**    *I*    U                                A

Code Cell 22 of 37

```
In [ ] 1 # your code here
```

Run Code

Code Cell 23 of 37

```
In [ ] 1 # your code here
```

Run Code

Code Cell 24 of 37

```
In [ ] 1 # your code here
```

Run Code

Code Cell 25 of 37

```
In [ ] 1 # your code here
```

Run Code

```
In [ ] 1 # your code here
```

[Run Code](#)

## Problem 3: Simulating Dice Rolls

- LOs: #EmpiricalAnalysis, #PythonImplementation, and #CodeReadability
- **Optional LO:** #TheoreticalAnalysis

(Adapted from Problem 3 Shonkwiler p.43)

(a) Simulate 1,000 rolls and 10,000 rolls (separately) of a pair of dice in Python 🎲🎲 and, for each roll, record the following information:

- The sum of the two dice,
- The individual pair, e.g. (1, 1), (1, 2), ..., (6, 6). Please note that the individual pair does distinguish between (a, b) and (b, a)--they are **not** regarded as the same.

```
In [2] 1 import random
        2 import numpy as np
        3 import scipy.stats as sts
        4 import matplotlib.pyplot as plt
        5 random.seed(10)
        6
        7 def simulate_dice(rolls, num_dice):
        8     """
        9     A function that implements die rolls.
       10
       11     Parameters
       12     -----
       13     rolls: int
       14         The number of rolls
```

```

16         The number of dice we roll on each of the rolls
17
18     Returns
19     -----
20     sum_data: list
21         A list of the sum of the dice at each roll
22     pars_data: list of tuples
23         A list of the pars of dices rolled at each roll
24     """
25     # Initialize arrays to store the data
26     pairs_data = []
27     sum_data = []
28
29     # Go over each roll
30     for trial in range(rolls):
31         # Store the values of the dice rolled on this roll.
32         current_dice = []
33         # For each die depending on the number
34         for die in range(num_dice):
35             die = random.randrange(1,7)
36             current_dice.append(die)
37
38         # Store the relevant stats.
39         pairs_data.append(tuple(current_dice))
40         sum_data.append(sum(current_dice))
41
42     return sum_data, pairs_data

```

Run Code

Code Cell 28 of 37

```

In [3] 1 simulation_case1 = simulate_dice(1000, 2) # 1000 rolls
        2 simulation_case2 = simulate_dice(10_000, 2) #10,000 rolls
        3

```

Code Cell 29 of 37

```

In [133] 1 # b) i) a histogram of the results for the sum of the dice for both 1,000
          2 and 10,000 rolls; include the mean and the 95% confidence intervals for
          3 both the distribution and the mean in the corresponding histograms. If
          4 you increased the number of simulations, which confidence intervals would
          5 become narrower if any?
          6
          7 def plot_hist(data, percentile_string, title, units, xlabel_string,
          8 bins):
          9     """ A function nthat is used to create histograms including the mean,
          10     95% confidence interval, and 99th percentile
          11
          12     data: list
          13         The data to be plotted
          14     title: str
          15         The title of the histogram
          16     units: str
          17         The units of the variable
          18     xlabel_string: str
          19         The label for the x-axis
          20     """
          21     # 99 percentile
          22     perc99 = np.percentile(data, 99)
          23     print('99th percentile', percentile_string, perc99, units)
          24
          25     # Mean 95% CI
          26     m = np.mean(data)
          27     m_approx = round(m, 2)
          28     print('Sample mean:', m_approx)
          29
          30     t = sts.sem(data)
          31     t_approx = round(t, 2)
          32
          33     print('95% confidence interval of population mean:', [ m_approx -
          34     1.96*t_approx, m_approx + 1.96*t_approx])
          35
          36     # Create the figure
          37     plt.figure(figsize=(12, 6))
          38     plt.hist(data, bins=bins, density=True, color = 'black', alpha =1,
          39     edgecolor = 'white')

```

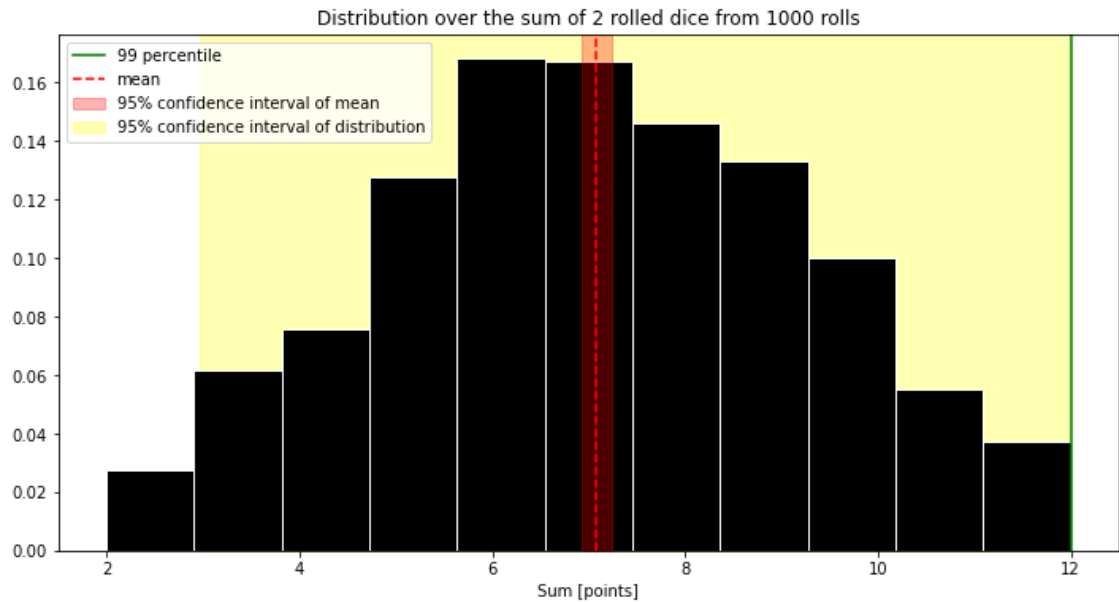
```

32
33     # Labels
34     plt.title(title)
35     plt.xlabel(xlabel_string)
36     ylim = plt.ylim()
37
38     plt.ylim(0, ylim[1])
39
40     # Plot the 99 percentile
41     plt.plot([perc99, perc99], [0, ylim[1]], 'g-', label = '99
percentile')
42
43     # Add a line for the mean
44     plt.axvline(m_approx, color = 'red', linestyle='--', label = 'mean')
45
46     # Add the confidence interval of the mean
47     plt.fill_between([m_approx - 1.96*t_approx, m_approx +
1.96*t_approx], 0, ylim[1], color = 'red', alpha = 0.3, label = '95%
confidence interval of mean', zorder=2)
48
49     # Calculate the 95% confidence interval of the distribution
50     confidence_interval_lower, confidence_interval_upper =
np.percentile(data, [2.5, 97.5])
51
52     # Add the confidence interval of the distribution
53     plt.fill_between([confidence_interval_lower,
confidence_interval_upper], 0, ylim[1], color = 'yellow', alpha = 0.3,
label = '95% confidence interval of distribution')
54
55     print('95% confidence interval of the distribution:',
[confidence_interval_lower, confidence_interval_upper])
56
57     plt.legend()
58     plt.show()
59
60
61 plot_hist(data = simulation_case1[0], percentile_string = 'of the sum of
2 rolled dice with 1000 rolls:', title = 'Distribution over the sum of 2
rolled dice from 1000 rolls', units = 'points', xlabel_string = 'Sum
[points]', bins = 11)
62
63 plot_hist(data = simulation_case2[0], percentile_string = 'of the sum of
2 rolled dice with 10,000 rolls:', title = 'Distribution over the sum of
2 rolled dice from 10,000 rolls', units = 'points', xlabel_string = 'Sum
[points]', bins = 11)

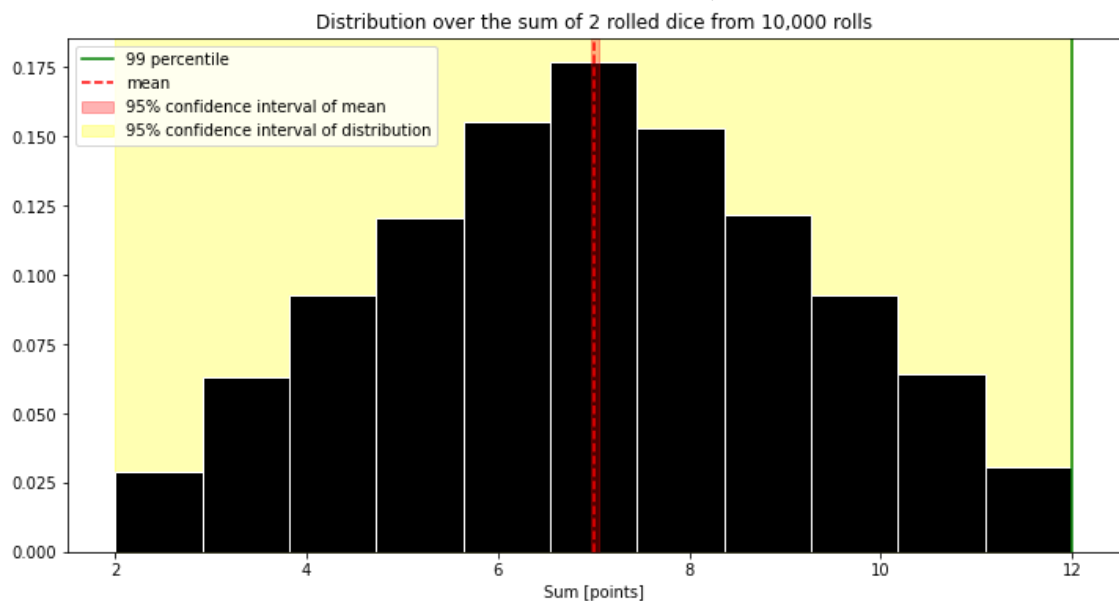
```

## Run Code

Out [133] 99th percentile of the sum of 2 rolled dice with 1000 rolls: 12.0 points  
Sample mean: 7.08  
95% confidence interval of population mean: [6.9232000000000005, 7.2368]  
95% confidence interval of the distribution: [2.9750000000000014, 12.0]



99th percentile of the sum of 2 rolled dice with 10,000 rolls: 12.0 points  
Sample mean: 7.01  
95% confidence interval of population mean: [6.9708, 7.0492]  
95% confidence interval of the distribution: [2.0, 12.0]





```

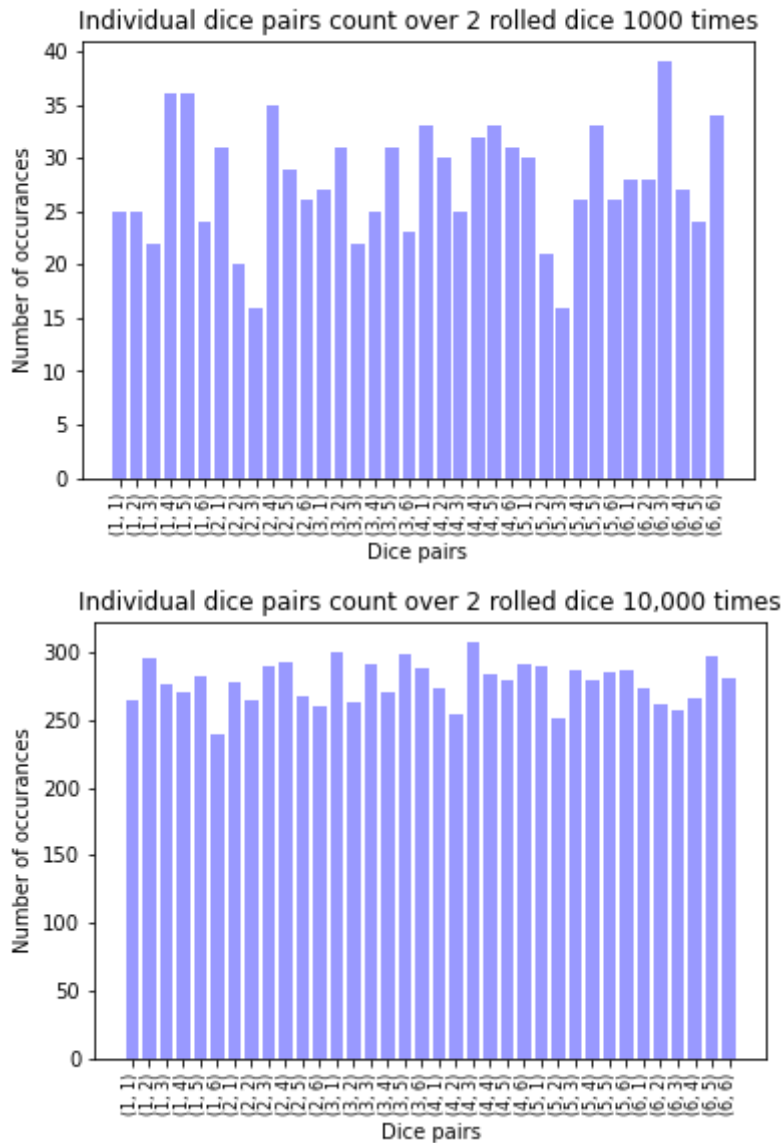
In [134] 1 # b) ii) a bar chart for the results of the individual pairs, and comment
          2 on the results you obtained.
          3
          4 from collections import Counter
          5 import matplotlib
          6
          7 def plot_barchart(data, xlabel, ylabel, title):
          8     """ A function that plots a bar chart"""
          9
          10     # Pairs of dice on each roll
          11     pairs = list(map(str, data.keys()))
          12
          13     # Count of each pair - number of times it occurred
          14     counts = list(data.values())
          15
          16     plt.figure(figsize=(36, 18))
          17
          18     # creating the bar plot
          19     fig, ax = plt.subplots()
          20     ax.bar(pairs, counts, color='blue', alpha = 0.4)
          21     ax.set_xticks(pairs)
          22     plt.xticks(rotation=90, fontsize=8)
          23
          24     plt.xlabel(xlabel)
          25     plt.ylabel(ylabel)
          26     plt.title(title)
          27     plt.show()
          28
          29 plot_barchart(data=Counter(sorted(simulation_case1[1])), xlabel = "Dice
          30 pairs", ylabel = "Number of occurrences", title = "Individual dice pairs
          31 count over 2 rolled dice 1000 times")
          32
          33 plot_barchart(data=Counter(sorted(simulation_case2[1])), xlabel = "Dice
          34 pairs", ylabel = "Number of occurrences", title = "Individual dice pairs
          35 count over 2 rolled dice 10,000 times")
          36
          37 simulation_case1000000 = simulate_dice(1_000_000, 2)
          38
          39 plot_barchart(data=Counter(sorted(simulation_case1000000[1])), xlabel =
          40 "Dice pairs", ylabel = "Number of occurrences", title = "Individual dice
          41 pairs count over 2 rolled dice 1,000,000 times")

```

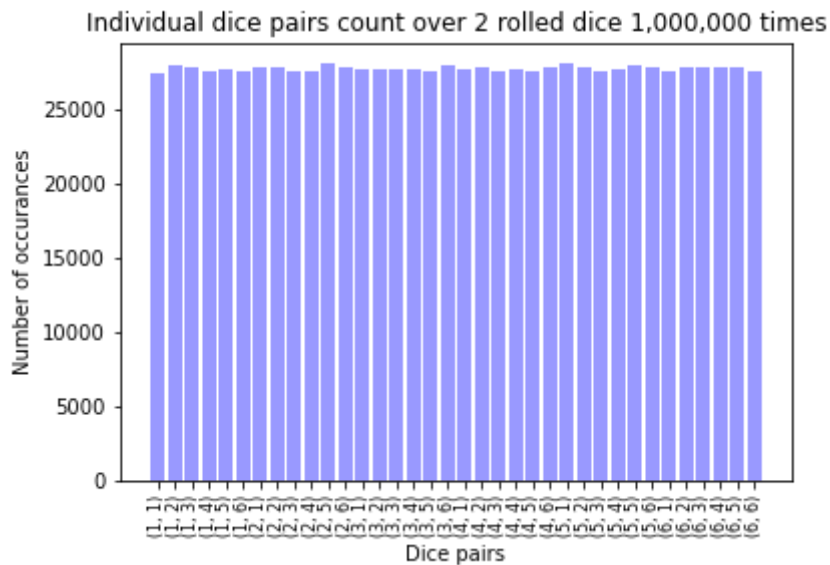
Out [134]

&lt;Figure size 2592x1296 with 0 Axes&gt;

&lt;Figure size 2592x1296 with 0 Axes&gt;



&lt;Figure size 2592x1296 with 0 Axes&gt;



Code Cell 31 of 37

In [97] 1

Run Code

Question 9 of 11

**(b) Create:**

- (i) a histogram of the results for the sum of the dice for both 1,000 and 10,000 rolls; include the mean and the 95% confidence intervals for both the distribution and the mean in the corresponding histograms. If you increased the number of simulations, which confidence intervals would become narrower if any?
  - a. **[Optional challenge]** Derive the theoretical value for the mean of the sum of dice, and compare it with the empirical results.
- (ii) a bar chart for the results of the individual pairs, and comment on the results you obtained.

Include the answers to the questions in the box provided below and use the coding cells to create the histogram and bar chart.

Normal ⌵ **B** *I* U 🔗 ” ‹›  $\frac{1}{2}$   $\frac{1}{3}$   $\frac{1}{4}$   $\frac{1}{5}$  A

i) If we increased the number of simulations, that means that the sample size would increase. We expect that both intervals, the confidence interval of the distribution and the confidence interval of the mean, would become more accurate. This is because as the sample size becomes bigger, the estimations for the parameters get closer to the true ones of the whole population. For the CI of the population mean we should expect a more precise prediction and a narrower CI. However for the CI of the distribution that's not necessarily the case and can depend on factors like the data variability. Let's look at the empirical results we obtained from before:

For 1000 rolls:

99th percentile of the sum of 2 rolled dice with 1000 rolls: 12.0 points

Sample mean: 7.08

95% confidence interval of population mean: [6.9232000000000005, 7.2368]

95% confidence interval of the distribution: [2.9750000000000014, 12.0]

For 10,000 rolls:

99th percentile of the sum of 2 rolled dice with 10,000 rolls: 12.0 points

Sample mean: 7.01

95% confidence interval of population mean: [6.9708, 7.0492]

95% confidence interval of the distribution: [2.0, 12.0]

We can see our expectation holding true for the 95% confidence interval of the population mean as its estimation becomes slightly narrower and converges towards 7 -- the error becomes smaller as we get more data points which is why we get this narrower interval.

For the 95% confidence interval of the distribution, we can see that in fact the interval gets wider instead of narrower. One reason this might be is because 1000 rolls are not always enough for the system to converge to the true behavior and get a distribution that resembles the one of the population. It means that we actually got slightly more data points on the right side of 2 which is why the lower bound became 2.9. However when we add more trials (10,000 rolls) we can see that the 95% confidence interval is wider which is closer to the CI of the true population (my explanation is at the end of the next paragraph).

I also experimented with other values for the random seed.

When I set the random seed equal to 2000 (instead of 10) and we look at the 95% confidence interval of the distribution we can see that in both cases (1000 or 10,000 rolls) the CI [2, 12] which is also the range of the distribution -- the sums can have values from a minimum of 2 (rolling two 1s) to a maximum of 12 (rolling two 6s). In this case 1000 and 10,000 rolls were enough for the simulation to converge with these random values. After that

even if we increase the number of rolls the confidence interval will not become smaller. There can be other reasons for this. For example, the variability of the data is such that the outer two bars 2 and 12 will take up more than 5% of the data every time there are enough trials. This is because from the  $6 \times 6 = 36$  total combinations we can roll (6 options for the first die and 6 more for the second die) to get a sum of 2 or a sum of 12, there are only 2 combinations. This makes it  $2/36 = 5.56\%$  of all options. We can also see that on the histogram in terms of how the two bars on the sides are slightly above 0.025. Therefore the 95% confidence interval truly is  $[2.0, 12.0]$  and it has turned out that 1000 rolls are enough to converge to this conclusion.

In the first case when the seed was equal to 10 the issue is probably that in that random sample the values were slightly skewed to the right so the lower limit increased by a bit.

### [Optional challenge]:

Let's use the properties of the discrete uniform distribution to solve this problem. I referred to this page for the formulas: [https://en.wikipedia.org/wiki/Discrete\\_uniform\\_distribution](https://en.wikipedia.org/wiki/Discrete_uniform_distribution)

Each roll of an individual die can be perceived as a random variable. An individual die follows a discrete uniform distribution. The formula for the mean is  $(a+b)/2$  which is  $(1+6)/2 = 3.5$

We can sum up the two uniform distribution because of a property called linearity of expectations that allowed us to break down a more complex random variable (expected sum of 2 dice) into simpler and smaller components of two individual dice.

Therefore for the mean of the sum of two dice we have  $(3.5+3.5) = 7$

When comparing with the empirical results we can see that they are very close together. This makes sense because our samples are relatively big and the problem is relatively simple so it was able to converge to the true statistics of the population. We can also see that the sample of 10,000 rolls is perfectly aligned with the theoretical analysis while the one with 1000 rolls was 0.1 away from the mean sum.


ii)

There are  $6 \times 6 = 36$  total possible pairs of dice we can roll (6 options for the first die and 6 more for the second die). We know that the dice are not biased meaning that each die has an equal probability  $1/6$  to land on any number. Therefore our expectation is that the distribution of all pairs should be a uniform distribution with all pairs occurring a similar number of times.

Let's look at the empirical results. Some things to be looking out for are whether any of the pairs appeared significantly more times than some other pairs. This would make it

In the bar chart with 1000 rolls we expect each pair to occur  $1000/36 = 27$  times. However we can see that the distribution has some peaks like those at (1,1) and (3,3) that appear more than 35 times.

In the bar chart with 10,000 rolls the distribution more closely resembles a uniform one but still not exactly. We can't conclude anything from these observations. It doesn't mean that the dice were biased. It can simply be attributed to randomness. This is why I increased the sample size and also plotted a bar chart with 1,000,000 rolls which we can see represents almost a perfect uniform distribution.

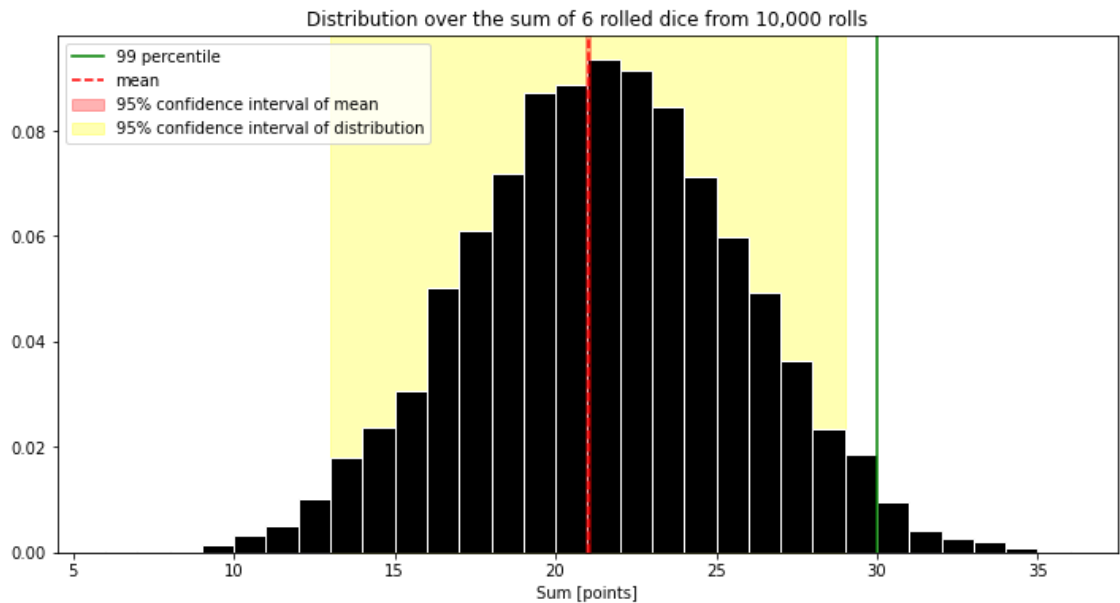
(c) Repeat the above steps (except for question 2b) for rolling six dice, , this time simulating 10,000 rolls (please note that rolling the six dice once each corresponds to 1 roll).

Code Cell 32 of 37

```
In [135] 1 # This is very volatile (the distribution looks different every time we
          2 run the code unless we fix the random seed) so below I also plotted the
          3 histogram with 1 million rolls to observe the CLT better.
          4
          5 simulation_case3 = simulate_dice(10_000, 6)
          6
          7 # The minimum sum of six dice is 6*1 and maximum sum is 6*6 = 36 which is
          8 why I chose these values for the bins.
          9
          10 plot1 = plot_hist(data = simulation_case3[0], percentile_string = 'of the
              sum of 2 rolled dice with 10,000 rolls:', title = 'Distribution over the
              sum of 6 rolled dice from 10,000 rolls', units = 'points', xlabel_string
              = 'Sum [points]', bins = list(range(6, 37)))
          11
          12 # 1 million rolls
          13 simulation_case4 = simulate_dice(1_000_000, 6)
          14
          15 plot1 = plot_hist(data = simulation_case4[0], percentile_string = 'of the
              sum of 2 rolled dice with 1,000,000 rolls:', title = 'Distribution over
              the sum of 6 rolled dice from 1,000,000 rolls', units = 'points',
              xlabel_string = 'Sum [points]', bins = list(range(6, 37)))
```

Run Code

```
Out [135] 99th percentile of the sum of 2 rolled dice with 10,000 rolls: 30.0 points
          Sample mean: 21.02
          95% confidence interval of population mean: [20.9416. 21.098399999999998]
```

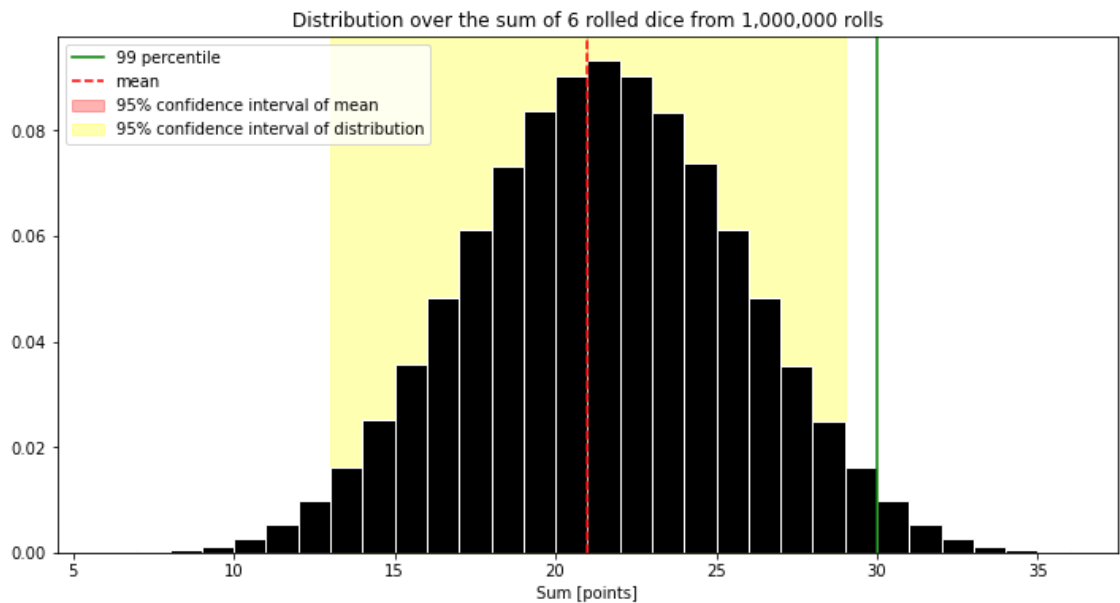


99th percentile of the sum of 2 rolled dice with 1,000,000 rolls: 30.0 points

Sample mean: 21.0

95% confidence interval of population mean: [21.0, 21.0]

95% confidence interval of the distribution: [13.0, 29.0]



Code Cell 33 of 37

In [135] 1

Run Code

Code Cell 34 of 37

In [135] 1

Run Code

Code Cell 35 of 37

In [ ] 1 # your code here

Run Code

Code Cell 36 of 37

In [ ] 1 # your code here

Run Code

Question 10 of 11

**(d)** Compare the histograms for 2 dice and 6 dice when you performed 10,000 rolls and critically comment on any similarities or differences. Were the results expected? Explain.

Normal  **B** *I* U  ” ‹›     A

We can see that in both cases they approximately resemble the normal distribution. They both have a peak that is at the center of their range. The histogram for 2 dice is more closely follows the normal distribution compared to the histogram for 6 dice. This is most likely because in the histogram of 2 dice we have 11 options for the sum and we are more likely to get an even sample compared to the 31 options for the 6 dice. (Putting this in brackets to indicate a side note: this is why I also plotted a histogram with 1 million rolls for 6 dice and



center of the range). A big difference between the two histograms is the range: [2, 12] for 2 dice and [6, 36] for 6 dice which I already mentioned can be explained by the bigger number of possible outcomes.

Let's also examine the relevant stats to uncover some differences:

2 dice:

99th percentile of the sum of 2 rolled dice with 10,000 rolls: 12.0 points

Sample mean: 7.01

95% confidence interval of population mean: [6.9708, 7.0492]

95% confidence interval of the distribution: [2.0, 12.0]

6 dice:

99th percentile of the sum of 2 rolled dice with 10,000 rolls: 30.0 points

Sample mean: 21.02

95% confidence interval of population mean: [20.9416, 21.098399999999998]

95% confidence interval of the distribution: [13.0, 29.0]

The theoretical value for the mean of 6 dice is:  $3.5 \cdot 6 = 21$

Some insights that we gain are that:

- I mentioned that the range for the distribution for 6 dice is larger which we can see is also reflected in the CI for the - distribution. It has a larger 95% confidence interval of the distribution: [13.0, 29.0] compared to [2.0, 12.0].
- Another difference is in the sample means: we can see that the the 6 dice have a much higher value. This is expected and it matches the theoretical value for the mean as well. Intuitively it is of course going to be higher since we some 4 additional dice every time (at least a minimum of 4 additional points).
- The 99th percentiles are also different and as expected the one for 6 dice is much higher. Another interesting observation is that the 99th percentile for 2 dice matches the upper bound of the range for 2 dice but we cannot say the same for the case of 6 dice.
  - In the case of 2 dice the probability of getting a sum of 12 is  $1/36 = 2.78\%$  -- therefore on the right side of the distribution we have more than 1 percent that fall into this category which is why the 99th percentile matches it.
  - In the case of 6 dice the probability of getting a sum of 36 is  $1/6^6$  which is 0.0021%. Therefore the 99th percentile for the sum is much lower than 36 (the upper bound of the range) and it makes sense that it is about 30 points. We can keep performing similar calculations to determine that.
- There is a slight difference in the 95% confidence interval of population mean. The

0.08). It's a very small difference. It can probably be attributed to the fact that the 2 dice histogram is more tightly centered around the mean and it also has a smaller range so it leads to a more accurate prediction even if slightly.

Question 11 of 11

**(e)** Discuss any patterns or trends observed in the results and how they relate to the Central Limit Theorem.

Here's a code cell in case you need it.

Normal  **B** *I* U        A

On a broader perspective the CLT says that despite the underlying distribution of the variable, if there is a large number of i.i.d. random variables then their sum will always follow an approximately normal distribution. In our case we have a die and its number at each roll as the random variable. We examine the sum of the values of several dice at each roll. As mentioned in the previous question we can see that both histograms for 2 and 6 dice for 10,000 rolls approximately follow a normal distribution with a peak in the middle of the range. This is exactly what the CLT states (predicts) and what we expected to happen so the results are consistent with the theorem.

CLT here is expressed through the idea of: the more simulations the smaller the error. If we get the log of the error, the x and y values are both increasing exponentially so dividing them by each other gives a linear relationship. The error scales in this way:  $\text{error} \sim 1/\sqrt{\text{total number of simulation}}$ . As we saw in Session 10 it's some sort of linear decay when we plot how the error evolves with the total number of simulations. So we know that the error converges very fast to 0 as we increase the number of simulations. Please refer to the plot on the left below which portrays this in the context of the dice simulation with 6 dice and the error for the expected value of the sum.

If applied the log to both sides (log error, log # of simulations):  
We use the coefficient 1/2:  $\text{Log error (y)} \sim -\frac{1}{2} * \text{log \# simulations (x)}$  so it's some sort of linear decay (please refer to the plot on the right below which portrays this in the context of the dice).

We can also see the effect of the CLT in our results. The confidence intervals are narrower (the error is smaller) when the number of simulations is bigger.

```

In [16] 1 # Code adapted from Session 10 Monty Hall breakout workbook
        2
        3 import matplotlib.pyplot as plt
        4
        5 def get_absolute_errors(scenario='standard', true_value = 21,
        6                       required_experiments=100000, plotting = True):
        7     absolute_errors = []
        8
        9     experiments = 0
       10     successes = 0
       11     while experiments < required_experiments:
       12         current_sum = np.mean(simulate_dice(experiments, 6)[0])
       13         experiments += 1
       14         absolute_errors.append(abs(true_value - current_sum))
       15     if plotting:
       16         plt.figure(figsize = (14, 4))
       17         ## Standard Plot
       18         plt.subplot(1, 2, 1)
       19         plt.plot(range(1, experiments + 1), absolute_errors, '.', alpha =
0.2, )
       20         plt.xlabel('Number of simulations')
       21         plt.ylabel('Absolute error')
       22         plt.title('Error for 6 dice and EV of 21')
       23         ## Log-log scale plot
       24         plt.subplot(1, 2, 2)
       25         plt.loglog(range(1, experiments + 1), absolute_errors, '.', alpha
= 0.2)
       26         plt.xlabel('Log(number of simulations)')
       27         plt.ylabel('Log(absolute error)')
       28         plt.title('Error for 6 dice and EV of 21')
       29         plt.show()
       30     return None
       31
       32 # Number of times to run the experiment
       33 get_absolute_errors(required_experiments = 1000)

```

[Run Code](#)

Out [16]

