SAVING THE JUNE SUCKER: ECOLOGICAL MODELING INSIGHTS FROM UTAH LAKE

SOPHIE HARMAN, MICHELLE MUMFORD, KAYSHA O'BRIEN, SABRINA SMITH

Abstract. Utah Lake has undergone significant changes in fish populations over recent decades, with the June sucker, a critically endangered species, experiencing near-extinction and subsequent restocking efforts. In this paper, we explore the dynamics of the June sucker population under the effects of predation, competition with non-native species, and hybridization. We develop a comprehensive model that integrates elements from single-species logistic growth, predator-prey dynamics, interspecies competition, and the effects of human intervention. Simulations demonstrate how interactions between these factors influence population trends over time and provide insights into the effectiveness of conservation measures. Our findings highlight the importance of adaptive management in preserving biodiversity within ecosystems.

1. Background/Motivation

The June Sucker (*Chasmistes liorus*) is a fish species native to Utah and found only in Utah Lake and its tributaries. Over the past century, its population has been in steep decline due to urban and industrial growth, introduction of nonnative fish species into the Utah Lake watershed, drought, and other ecological and human factors. Due to this population decline, the United States Fish and Wildlife Service classified the June Sucker as an endangered species in April 1986. In 1999, a group of organizations joined together in efforts to save the June Sucker. These efforts included a breeding and stocking program, the harvesting of nonnative carp, and environmental restoration. This had a positive effect on the population and in 2020 the June Sucker was downlisted from "endangered" to "threatened"[6].

There is little published work concerning the modeling of the June Sucker population. The main existing model is a closed-capture model that aims to estimate the total June Sucker population based on tagging and capturing fish [4]. This model primarily focuses on population estimation rather than exploring the underlying ecological dynamics affecting the population. Our approach differs significantly from the existing closed-capture model. We aim to develop a more comprehensive Ordinary Differential Equation (ODE) based model that estimates the population size and captures the interactions between various ecological factors influencing the June Sucker. The June Sucker is an important population and is known as an *indicator species* [3],

Date: December 7, 2024.

meaning that the health of the June Sucker population can be used to assess the health of the Utah Lake ecosystem as a whole. Therefore it is valuable to predict future population levels of the June Sucker. If the recovery program has a high level of success, it could be implemented for other endangered populations to bring them back to historic levels.

1.1. **Recreating the Yellowstone Effect for the June Sucker.** Both the June Sucker and Yellowstone wolves are species that play crucial roles in their respective ecosystems. While the June Sucker is an indicator species whose health reflects the overall health of Utah Lake's ecosystem, wolves are apex predators whose reintroduction also has profound effects on their environment.

Historically, wolves in Yellowstone were systematically hunted and eradicated by 1926, primarily to protect the elk population [8]. However, their absence triggered an ecological imbalance. Elk numbers surged, leading to overgrazing of berry-producing shrubs, a critical resource for many of the ecosystem's species [5]. This overgrazing disrupted the ecosystem, causing a chain reaction of negative effects—a phenomenon known as a trophic cascade. In 1995, wolves were reintroduced to Yellowstone, sparking an extraordinary ecological recovery [7]. Predation by wolves helped regulate elk populations, allowing vegetation to recover. This, in turn, supported a resurgence of diverse species and even altered river courses [1].

Similar to how wolves regulate elk populations, we investigate how introducing or managing predator populations affects June Suckers and non-native species. Understanding how non-native species and hybridization impact June Suckers is akin to studying how wolves influence other species in Yellowstone. Evaluating the effectiveness of stocking programs and habitat restoration in supporting June Sucker populations parallels human efforts to support wolf populations in Yellowstone. By running simulations, we aim to predict the long-term outcomes of various conservation strategies, much like studies on the Yellowstone ecosystem.

2. Objectives

Our goal is to develop an ODE-based model that captures the essential dynamics influencing the June Sucker population, including:

- Intrinsic growth and carrying capacity
- Competition with non-native species
- Predation by predators
- Hybridization between June Suckers and Utah Suckers
- Human interventions such as stocking and habitat restoration

3. Preliminary Models

When exploring effective models for Yellowstone, we found the Lotka-Volterra model to be widely used and consistently yielding strong results.

Below, we present several variations of the Lotka-Volterra model, each focusing on a specific aspect of ecosystem dynamics.

Symbol	**Description**
$J(t)$	June sucker population at time t
$P(t)$	Predator population at time t
$N(t)$	Competing/Non-native Species population at time t
K	Carrying capacity
r	Growth rate
a	Predation rate
b	Reproduction rate of predators per prey consumed
m	Mortality rate of predator
h	Harvesting rate
c	Conservation effort
α	Competition (impact of predators on June Suckers)
β	Competition (impact of June Suckers on predators)

Table 1. Variables and Parameters for the Model.

Logistic Growth

$$\frac{dS}{dt} = rJ\left(1 - \frac{J}{K}\right)$$

Predator-Prey

$$\frac{dJ}{dt} = rJ\left(1 - \frac{J}{K}\right) - aJP$$

$$\frac{dN}{dt} = bJP - mP$$

Competition

$$\frac{dJ}{dt} = r_J J\left(1 - \frac{J + \alpha P}{K_J}\right)$$

$$\frac{dC}{dt} = r_P P\left(1 - \frac{P + \beta J}{K_P}\right)$$

Human Interaction

$$\frac{dJ}{dt} = rJ\left(1 - \frac{J}{K_P}\right) - hJ + c$$

4. Model Excluding Hybridization

Each of the Lotka-Volterra variants above is relatively simple and, on its own, fails to fully capture the complex dynamics of an ecosystem. To address this, we develop an aggregate model that integrates these variants, enabling us to account for population growth, predator-prey relations, competition, and human interaction simultaneously (see Figure 1).

$$
\begin{aligned}
\frac{dJ}{dt} &= r_J J(1 - \frac{J}{K}) - \alpha JN - \beta JP + H \\
\frac{dN}{dt} &= r_N N(1 - \frac{N}{K_N}) - \delta NJ \\
\frac{dP}{dt} &= s_P(J + N)P - m_P P \\
K(t) &= K_0 + \gamma + H_J(t)
\end{aligned}
$$

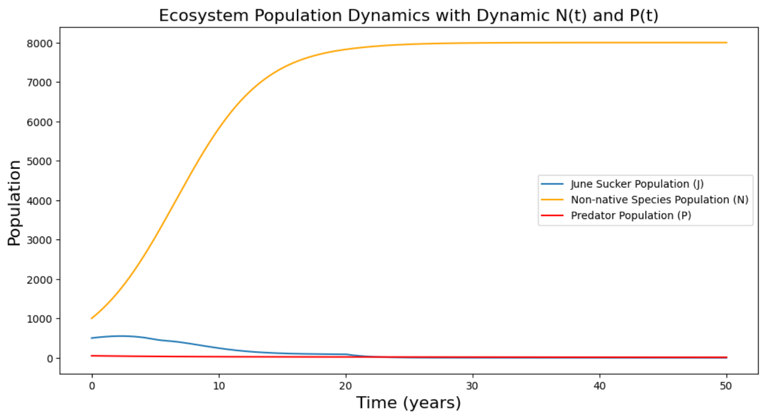


FIGURE 1. Population dynamics of the model excluding hybridization. Initial populations are as follows: June Suckers (500), non-native species (1000), and predators (50)

We identified the following equilibrium points for the model:

- Equilibrium Population of June Suckers: 0.00
- Equilibrium Population of Non-native Species: 8000.00
- Equilibrium Population of Predators: 0.00

These results indicate that June Suckers and predators die off, while non-native species eventually dominate 2. This model fails to accurately capture the ecosystem dynamics and the human intervention efforts. It does not account for hybridization of the June Sucker species, a significant factor that influences the population. It also does not account for the harvesting of non-native species.

5. Model Incorporating Hybridization

In our research, we found that the June Sucker can breed with another native fish species called the Utah Sucker, creating a fertile hybrid. This hybrid is generally not considered part of the June Sucker population, but this breeding does affect the population dynamics. Hybridization not only reduces the number of pure June Suckers but also introduces fertile hybrids that compete for the same resources. To account for this we adapted the previous model to include a hybridization coefficient. This updated model also adds human intervention factors for removal of non-native species and hybrids.

5.1. **Model Assumptions.**

- Species Selection: We focus on one non-native species as the primary competitor to the June Sucker to simplify the model while capturing the essential dynamics of competition and hybridization.
- Parameter Adjustment: Parameters are estimates based on ecological data from similar scenarios to explore the sensitivity of the model to different factors, such as hybridization rate and predation coefficients, and may not accurately represent the true values.
- Hybridization Occurs: We assume that hybridization between June Suckers and non-native species happens at a significant rate and produces fertile offspring that impact the ecosystem.
- Closed Ecosystem: The model assumes no immigration or emigration, focusing solely on the internal dynamics of Utah Lake.
- Shared Resources: All fish populations share the same habitat and resources.
- Hybrid Fertility: Hybrids are fertile and can reproduce
- Competition: Species compete for resources, affecting each other's growth rates
- Hybridization: Occurs at a rate proportional to the product JN
- Predation: Predators feed on all fish species
- Human Intervention: Actions such as stocking and habitat restoration affect populations and carrying capacity
- Non-overlapping Populations: The model assumes that non-native species and predators are separate groups with no overlap. Non-native species are classified as competitors but not predators of the June Sucker.

Symbol	**Description**	**Value**
α, δ	Competition coefficients	1.0
γ	Human intervention on carrying capacity	0.5
K_0	Carrying capacity	10000 fish
m_P	Mortality rate of predators	0.1 (per year)
r_J, r_N, r_H	Growth rate	0.2, 0.3, 0.25 (per year)
$\beta_J, \beta_N, \beta_H$	Predation coefficients	5×10^{-5} (per fish per year)
s_H	Hybridization coefficient	1×10^{-5} (per fish per year)
s_P	Predation success rate	1×10^{-5} (per fish per year)
H_J	Stocking	50 (fish per year)
H_N	Removal of non-native species	-10 (fish per year)
H_H	Removal of hybrids	-5 (fish per year)

TABLE 2. Hybridization Model Parameters.

5.2. **Model Equations.**
J(t): June Sucker Population Dynamics

$$\frac{dJ}{dt} = \underbrace{r_J J \left(1 - \frac{J + c_{JN}N + c_{JH}H}{K(t)}\right)}_{\text{Logistic Growth with Competition}} - \underbrace{\beta_J JP}_{\text{Predation}} - \underbrace{s_H JN}_{\text{Hybridization Loss}} + \underbrace{H_J(t)}_{\text{Stocking}}$$

N(t): Non-native Species Population Dynamics

$$\frac{dN}{dt} = r_N N(1 - \frac{N + c_{NJ}J + c_{NH}H}{K(t)}) - \beta_N NP - s_H JN + H_N(t)$$

H(t): Hybrid Population Dynamics

$$\frac{dH}{dt} = s_H JN + r_H H(1 - \frac{H + c_{HJ}J + c_{HN}N}{K(t)}) - \beta_H HP + H_H(t)$$

P(t): Predator Population Dynamics

$$\frac{dP}{dt} = s_P(J + N + H)P - m_P P$$

K(t): Carrying Capacity Dynamics

$$K(t) = K_0 + \gamma H_J(t)$$

5.3. **Justification of Model Structure.** Logistic growth reflects limited resources and environmental capacity, the competition terms capture the impact of inter-species competition, the predations terms represent loss due to predators, the hybridization terms account for the creation of hybrids and loss of pure species, and human intervention incorporates direct actions affecting populations. The coefficient values were selected based on a review of similar Lotka-Volterra models and their corresponding parameter settings.

6. Results

To analyze the performance of our model, we visualize population dynamics, perform sensitivity analysis, and find the equilibria. From our results, our goal is to gain a deeper understanding of how the factors incorporated into our model influence the June Sucker population in Utah Lake, ultimately informing and improving conservation efforts.

6.1. **Visualizing Population Dynamics.** The simulation (see Figure 2) illustrates the population dynamics of June suckers, non-native species, hybrids, and predators over a span of 50 years. The model that incorporates hybridization produces more realistic population dynamics compared to the model that does not account for hybridization. Key Population Trends:

- June Sucker Population: Increases during human intervention (years 5-20) but declines thereafter due to competition, predation, and hybridization
- Non-native Species Population: Rapidly increases, then stabilizes

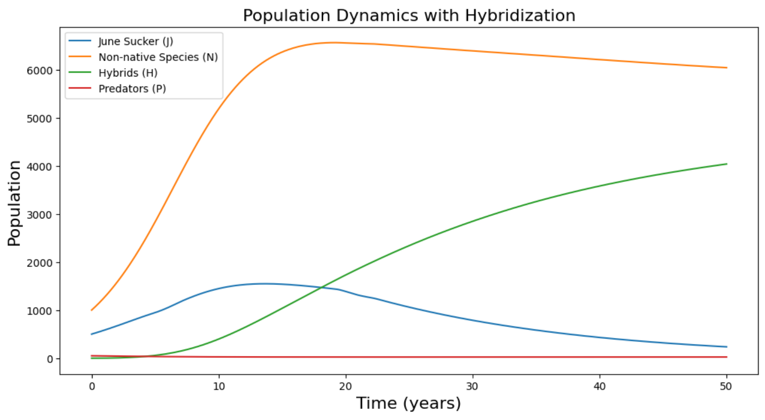


FIGURE 2. Population dynamics of the model including hybridization. Initial populations are as follows: June Suckers (500), non-native species (1000), predators (50), and hybrids (0). Note that while the predator population approaches 0, closer inspection in the code shows it does not reach it within the 50-year simulation period. However, projections beyond this period indicate a decline towards extinction, which may suggest a limitation of the model.

- Hybrid Population: Grows steadily
- Predator Population: While the predator population does not die out within the 50-year simulation period, extending the simulation beyond this timeframe reveals a gradual decline towards extinction.

This decline can be attributed to several factors. The unchecked growth of non-native species causes a significant imbalance in the ecosystem, placing considerable stress on the June Sucker population. This claim is supported by the assertion that "to reverse the negative impact of carp on a shallow lake, such as Utah Lake, the carp population must be reduced by 75%" [2]. In addition to competition with non-native species, June Suckers face pressure from hybrid species, as hybridization reduces the prevalence of the pure June Sucker population. Evidence of the June Sucker population's decline highlights the serious concern for its survival.

6.2. **Sensitivity Analysis.** We varied the impact hybridization coefficient s_H to observe its effect on the June Sucker population (see Figure 3). A higher s_H leads to a significant decline—and in some cases, the extinction—of the June Sucker population due to increased hybridization.

Similarly, we varied the growth rate of hybrids r_H to see its effect on the hybrid population (see Figure 4). Higher r_H results in a larger hybrid population which further suppresses the June Sucker population through competition.

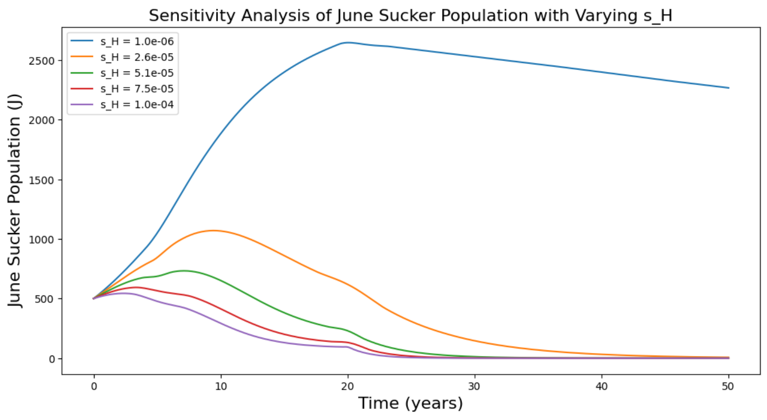


FIGURE 3. Sensitivity analysis of the June Sucker population with different hybridization coefficients.

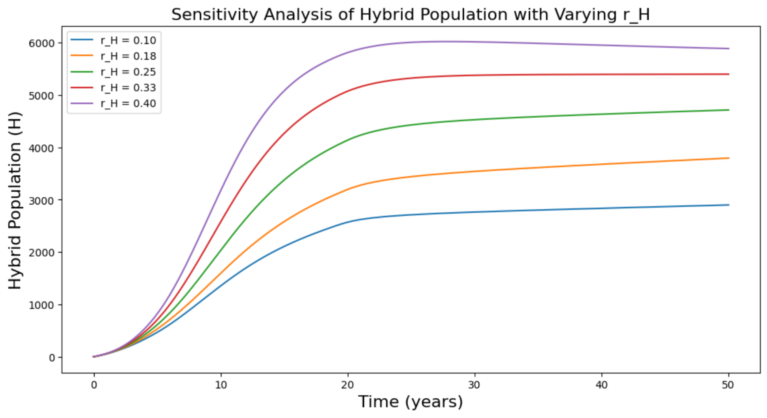


FIGURE 4. Sensitivity analysis of the June Sucker population with different hybridization growth rate.

We identified the following equilibrium points for the model incorporating hybridization (See Section 5.2 for model equations):

- Equilibrium Population of June Suckers: -245.62
- Equilibrium Population of Non-native Species: 32.44
- Equilibrium Population of Hybrids: 19.98
- Equilibrium Population of Predators: 0.00

The negative equilibrium value for June Suckers (see Figure 5) suggests that their population will eventually die out. This outcome reflects the overdominance of hybridization and competition, rendering the June Sucker population unsustainable under equilibrium conditions. In contrast, non-native species and hybrids stabilize at small but positive population levels, potentially outcompeting the June Sucker. The absence of predators removes a key regulatory factor for both non-native species and hybrids, likely contributing to their persistence. The sustained presence of hybrids highlights the adverse effects of hybridization on the survival of the June Sucker.

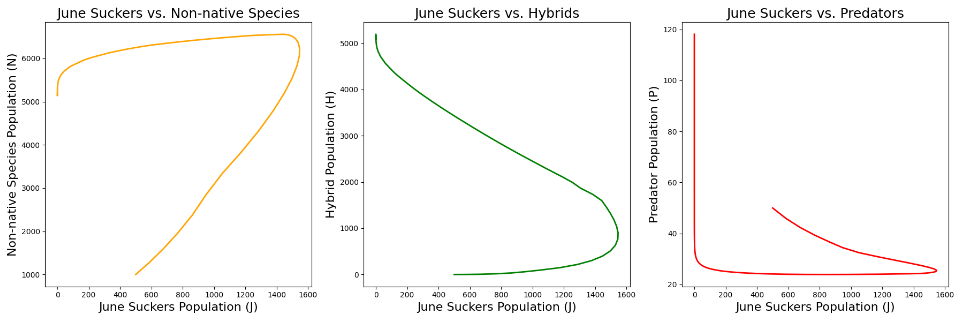


FIGURE 5. Phase plots of June Suckers versus non-native species, hybrids, and predators, respectively. The relationship between June Suckers and non-native species indicates that neither population stabilizes independently. The interaction between June Suckers and hybrids exhibits a predominantly inverse relationship. The interaction between June Suckers and predators also highlights dynamics that prevent independent stabilization of either population.

By creating a model that accurately reflects June Sucker population dynamics, our hope was to find a set of parameter values that bring the threatened species to more stable population numbers. Since the June Suckers decline despite human intervention, we have not found appropriate parameter values to achieve the "Yellowstone Effect". In future work, we would like to continue to explore parameter values to see if human intervention efforts can be optimized for a more promising outcome.

7. Analysis/Conclusions

In this paper, we developed a model to predict changes in the June Sucker population over time, incorporating factors like breeding, stocking, predation, and hybridization. Our analysis shows that hybridization dominates once stocking ends, leading to a decline in the June Sucker population. The model captures key population dynamics but struggles to produce stable predictions when parameters are adjusted to realistic values, highlighting the need for refinement.

Our model has several limitations. For example, it assumes that stocking ends abruptly rather than gradually, limiting its ability to capture more realistic scenarios. Additionally, the initial population values do not accurately reflect true population numbers. Ecosystems are incredibly complex and cannot be fully represented by a mathematical model. These limitations highlight oversimplified assumptions and potentially overlooked ecological factors.

In addition to addressing model limitations, future work should focus on refining parameters, incorporating additional complexities like environmental changes and genetic diversity, and exploring phenomena like bifurcations. Another area for investigation is the potential for physical separation of June Suckers and hybrids to mitigate the effects of hybridization. Such improvements could enhance predictive power and provide actionable insights for June Sucker revitalization, as well as for populations facing similar challenges. Although we were unsuccessful in finding optimal parameter values for the June Sucker's survival, we remain hopeful that future efforts, incorporating these strategies, will yield better results.

References

[1] Farquhar, Brodie. “Wolves Have Changed Yellowstone’s Ecosystem since the ’90s.” Yellowstone National Park, 22 June 2023, www.yellowstonepark.com/things-to-do/wildlife/wolf-reintroduction-changes-ecosystem/.

[2] Fiscal Highlights - June, `le.utah.gov/lfa/LFADocs.jsp?month=6&pubid=4809`. Accessed 6 Dec. 2024.

[3] June Sucker Recovery Implementation Program (n.d.). Meet the June Sucker. https://www.junesuckerrecovery.org/meet-the-june-sucker

[4] Landom, K., Conner, M., Ecology Center and Watershed Sciences Department, Utah State University, and Department of Wildland Resources, Utah State University. (2020). Annual assessment of June Sucker spawning population abundance and survival in Utah Lake 2008 – 2019. In Annual Report Submitted to the June Sucker Recovery Implementation Program.

[5] Patent, Dorothy H., Dan Hartman, and Cassie Hartman. When the wolves returned : restoring nature’s balance in Yellowstone. New York: Walker Distributed to the trade by Macmillan, 2008. Print.

[6] Szuszwalak, J. (2020, December 31). June sucker has moved from endangered to threatened: U.S. Fish and Wildlife Service. FWS.gov. https://www.fws.gov/press-release/2020 12/june-sucker-has-moved-endangered-threatened

[7] Smith, D.W., and Ferguson,G.(2005). Decade of the wolf: Returning the wild to Yellowstone. Guilfod, CT: Lyons Press.

[8] United States. National Park Service. (n.d.). Wolves in Yellowstone. Retrieved November 17, 2024, from https://www.nps.gov/yell/learn/nature/wolves.htm.

Numerical Simulations and Analysis

Consolidated script with different iterations of the ODE model. We perform numerical simulations to analyze the impact of the June sucker's reintroduction on the Utah Lake ecosystem, both with and without hybridization.

Objectives:

- Simulate the June sucker population dynamics over a specified time period.
- Analyze the effects of human intervention, competition, predation, and hybridization on the population.
- Perform sensitivity analyses on key parameters.
- Provide code for replication and further exploration

```python
# Import necessary libraries
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
```

1. Model Implementation without Hybridization

First, we implement the ODE model for the June sucker population without considering hybridization. We will simulate scenarios with and without human intervention and analyze the effects of competition and predation.

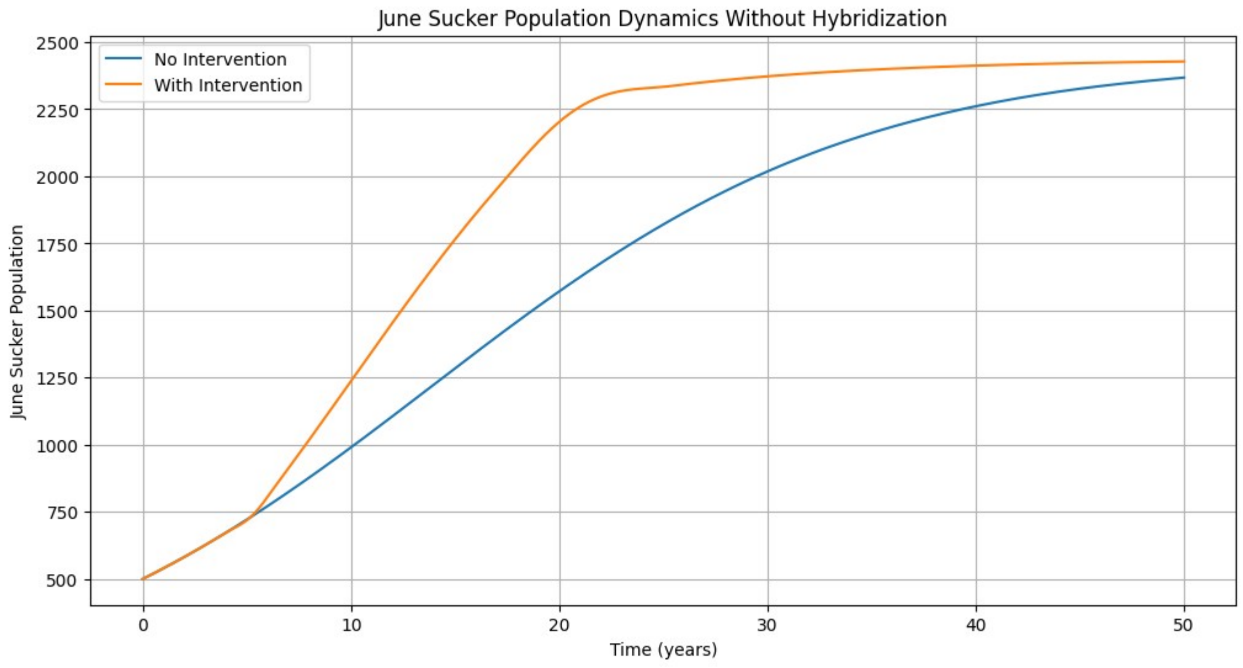
Parameters and Initial Conditions

- Intrinsic Growth Rate (r_J): 0.2 per year
- Baseline Carrying Capacity (K_0): 5,000 fish
- Competition Coefficient (alpha): 1e-4 per fish per year
- Predation Coefficient (beta): 5e-5 per fish per year
- Effectiveness of Human Intervention (gamma): 0.5 increase in carrying capacity per unit of H(t)
- Non-native Species Population (N): 1,000 fish
- Predator Population (P): 50 individuals
- Human Intervention Function (H(t)): Variable
- Initial June Sucker Population (J_0): 500 fish

```python
# Define the model parameters
r_J = 0.2           # Intrinsic growth rate of June sucker
K_0 = 5000          # Baseline carrying capacity
alpha = 1e-4        # Competition coefficient
beta = 5e-5         # Predation coefficient
gamma = 0.5         # Effectiveness of human intervention
N = 1000            # Non-native species population
P = 50              # Predator population
```

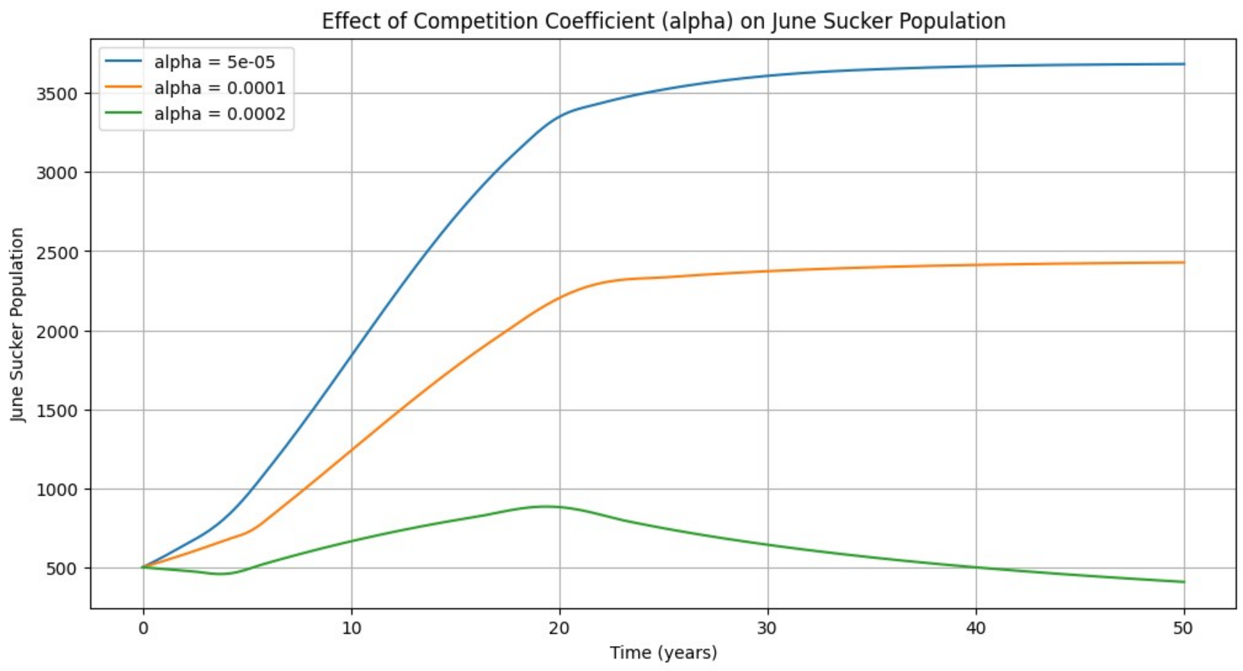
```
# Define the human intervention function H(t)
def H(t):
    # Scenario: Constant stocking and habitat restoration between year
5 and 20
    if 5 <= t <= 20:
        return 50    # Stocking rate of 50 fish per year
    else:
        return 0

# Define the carrying capacity function K(t)
def K(t):
    return K_0 + gamma * H(t)

# Define the ODE system for June sucker population
def june_sucker_model(t, J):
    dJdt = r_J * J * (1 - J / K(t)) - alpha * J * N - beta * J * P +
H(t)
    return dJdt

# Set the time span for the simulation
t_span = (0, 50)   # Simulate from year 0 to year 50
t_eval = np.linspace(t_span[0], t_span[1], 500)  # Evaluation times

# Set the initial conditions
J_0 = 500  # Initial June sucker population

# Solve the ODE without human intervention
# First, set H(t) to zero for no intervention
def H_no_intervention(t):
    return 0

# Redefine K(t) for no intervention
def K_no_intervention(t):
    return K_0 + gamma * H_no_intervention(t)

def june_sucker_model_no_intervention(t, J):
    dJdt = r_J * J * (1 - J / K_no_intervention(t)) - alpha * J * N -
beta * J * P + H_no_intervention(t)
    return dJdt

sol_no_intervention = solve_ivp(
    june_sucker_model_no_intervention, t_span, [J_0], t_eval=t_eval,
method='RK45'
)

# Solve the ODE with human intervention
sol_with_intervention = solve_ivp(
    june_sucker_model, t_span, [J_0], t_eval=t_eval, method='RK45'
)
```

```
# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(sol_no_intervention.t, sol_no_intervention.y[0], label='No
Intervention')
plt.plot(sol_with_intervention.t, sol_with_intervention.y[0],
label='With Intervention')
plt.title('June Sucker Population Dynamics Without Hybridization')
plt.xlabel('Time (years)')
plt.ylabel('June Sucker Population')
plt.legend()
plt.grid(True)
plt.show()
```



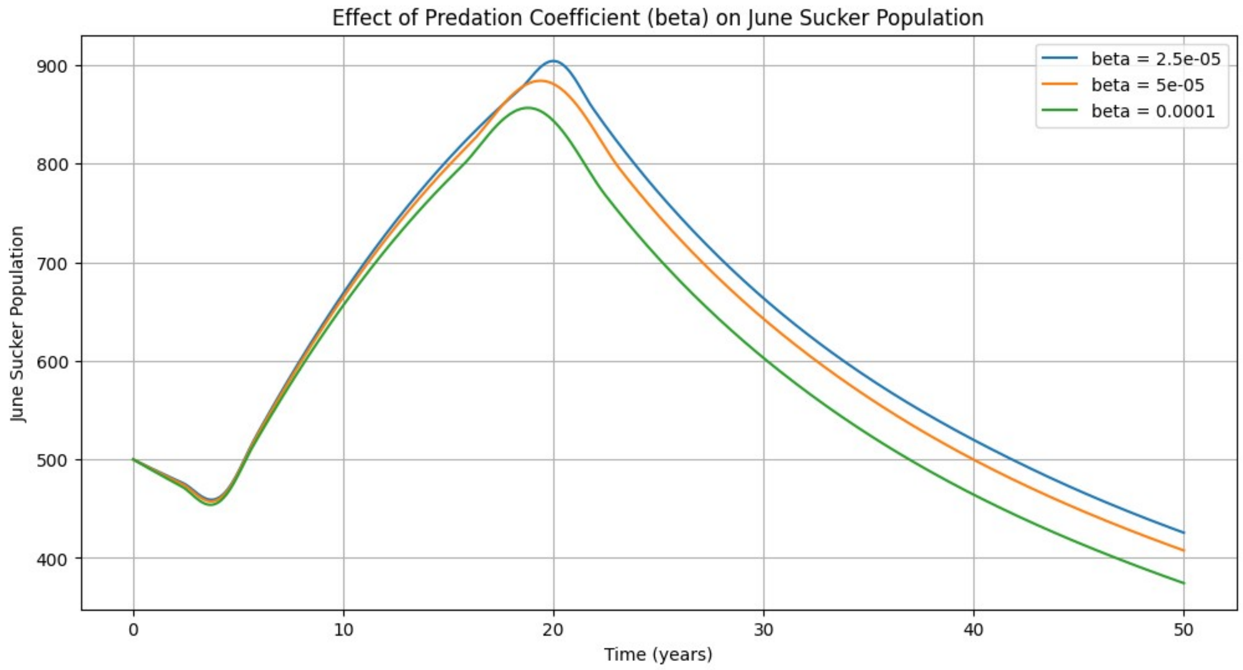
Sensitivity Analysis: Varying the Competition Coefficient (alpha)

```
# Test different values of alpha
alpha_values = [5e-5, 1e-4, 2e-4]
plt.figure(figsize=(12, 6))

for alpha in alpha_values:
    # Redefine the ODE model with the new alpha
    def june_sucker_model_alpha(t, J):
        dJdt = r_J * J * (1 - J / K(t)) - alpha * J * N - beta * J * P
+ H(t)
        return dJdt

    sol = solve_ivp(
        june_sucker_model_alpha, t_span, [J_0], t_eval=t_eval,
method='RK45'
    )
    plt.plot(sol.t, sol.y[0], label=f'alpha = {alpha}')

plt.title('Effect of Competition Coefficient (alpha) on June Sucker
Population')
plt.xlabel('Time (years)')
plt.ylabel('June Sucker Population')
plt.legend()
plt.grid(True)
plt.show()
```



Sensitivity Analysis: Varying the Predation Coefficient (beta)

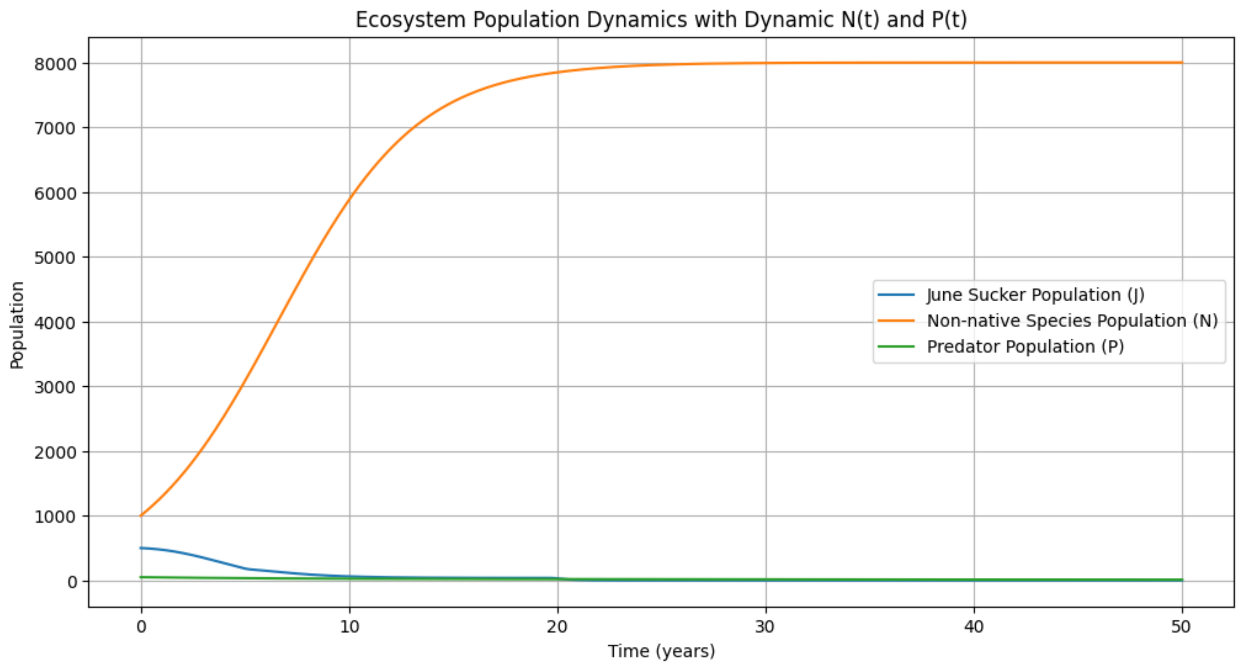
```
# Test different values of beta
beta_values = [2.5e-5, 5e-5, 1e-4]
plt.figure(figsize=(12, 6))

for beta in beta_values:
    # Redefine the ODE model with the new beta
    def june_sucker_model_beta(t, J):
        dJdt = r_J * J * (1 - J / K(t)) - alpha * J * N - beta * J * P
+ H(t)
        return dJdt

    sol = solve_ivp(
        june_sucker_model_beta, t_span, [J_0], t_eval=t_eval,
method='RK45'
    )
    plt.plot(sol.t, sol.y[0], label=f'beta = {beta}')

plt.title('Effect of Predation Coefficient (beta) on June Sucker
Population')
plt.xlabel('Time (years)')
plt.ylabel('June Sucker Population')
plt.legend()
plt.grid(True)
plt.show()
```



2. Model Implementation with Dynamic N(t) and P(t)

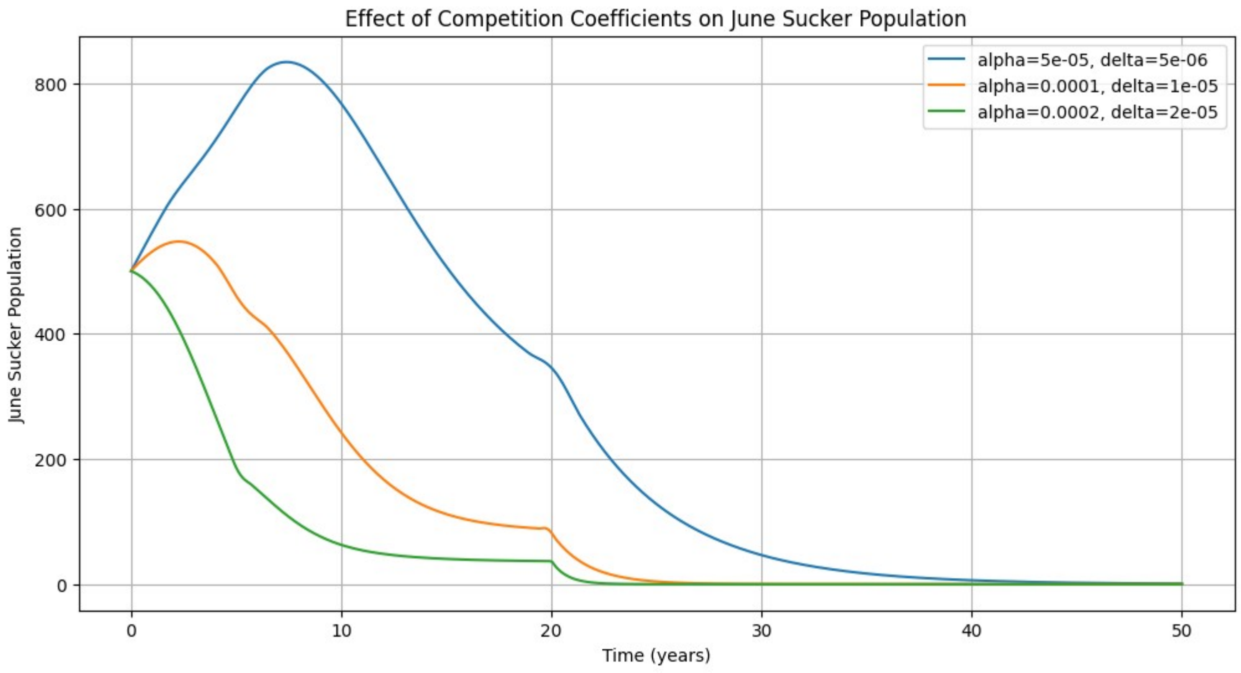
Next, we extend the model to include dynamic populations of non-native species (N) and predators (P). This allows us to analyze the interactions between these species and the June sucker over time.

```
# Define additional model parameters
# Non-native species parameters
r_N = 0.3            # Intrinsic growth rate of non-native species
K_N = 8000           # Carrying capacity for non-native species
delta = 1e-5         # Competition coefficient (impact of J on N)

# Predator parameters
s = 1e-5             # Predation success rate
m = 0.1              # Mortality rate of predators

# Redefine the ODE system including N(t) and P(t)
def ecosystem_model(t, y):
    J, N, P = y  # Unpack the variables
    # June sucker population dynamics
    dJdt = r_J * J * (1 - J / K(t)) - alpha * J * N - beta * J * P +
H(t)
    # Non-native species population dynamics
    dNdt = r_N * N * (1 - N / K_N) - delta * N * J
    # Predator population dynamics
    dPdt = s * (J + N) * P - m * P
    return [dJdt, dNdt, dPdt]

# Set initial conditions for N and P
N_0 = 1000  # Initial non-native species population
P_0 = 50    # Initial predator population

# Solve the ODE system
sol = solve_ivp(
    ecosystem_model, t_span, [J_0, N_0, P_0], t_eval=t_eval,
method='RK45'
)

# Extract solutions
J_t = sol.y[0]
N_t = sol.y[1]
P_t = sol.y[2]

# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(sol.t, J_t, label='June Sucker Population (J)')
plt.plot(sol.t, N_t, label='Non-native Species Population (N)')
plt.plot(sol.t, P_t, label='Predator Population (P)')
plt.title('Ecosystem Population Dynamics with Dynamic N(t) and P(t)')
plt.xlabel('Time (years)')
plt.ylabel('Population')
plt.legend()
plt.grid(True)
plt.show()
```



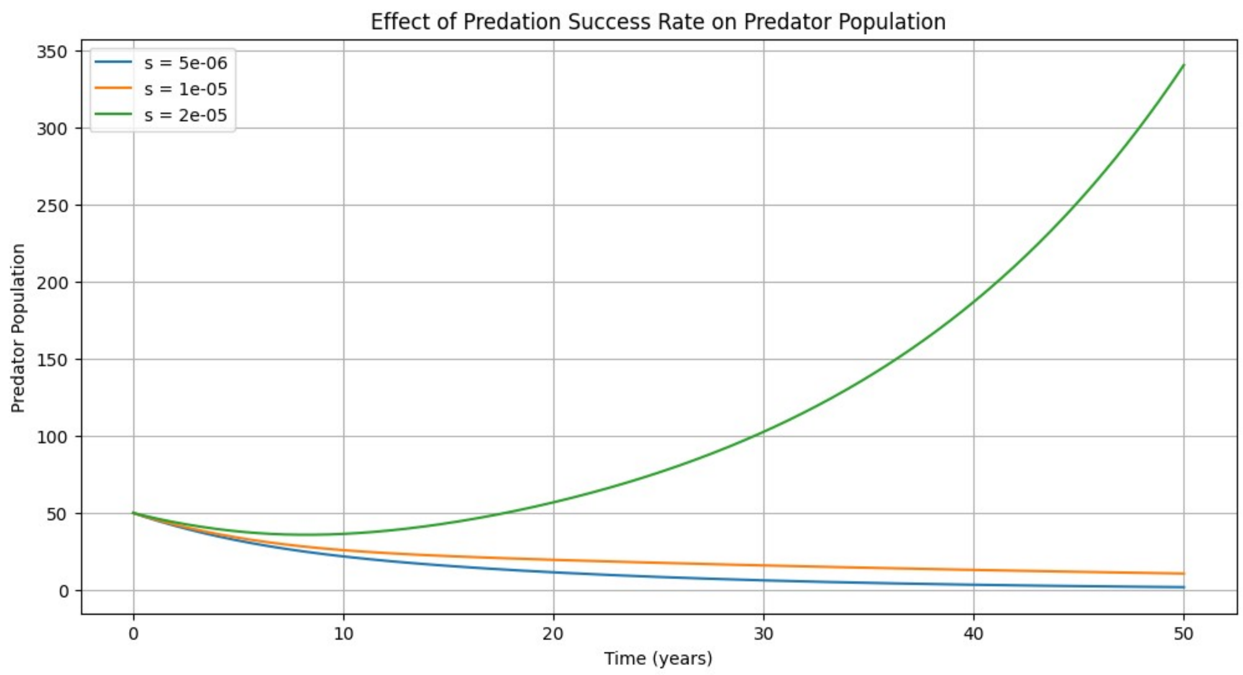
Sensitivity Analysis: Varying Competition Coefficients (alpha and delta)

```
# Test different values of alpha and delta
alpha_values = [5e-5, 1e-4, 2e-4]
delta_values = [5e-6, 1e-5, 2e-5]

plt.figure(figsize=(12, 6))

for alpha, delta in zip(alpha_values, delta_values):
    def ecosystem_model_competition(t, y):
        J, N, P = y
        dJdt = r_J * J * (1 - J / K(t)) - alpha * J * N - beta * J * P
+ H(t)
        dNdt = r_N * N * (1 - N / K_N) - delta * N * J
        dPdt = s * (J + N) * P - m * P
        return [dJdt, dNdt, dPdt]

    sol = solve_ivp(
        ecosystem_model_competition, t_span, [J_0, N_0, P_0],
t_eval=t_eval, method='RK45'
    )

    plt.plot(sol.t, sol.y[0], label=f'alpha={alpha}, delta={delta}')

plt.title('Effect of Competition Coefficients on June Sucker
Population')
plt.xlabel('Time (years)')
plt.ylabel('June Sucker Population')
plt.legend()
plt.grid(True)
plt.show()
```



Sensitivity Analysis: Varying Predation Success Rate (s)

```
# Test different values of s
s_values = [5e-6, 1e-5, 2e-5]

plt.figure(figsize=(12, 6))

for s in s_values:
    def ecosystem_model_predation(t, y):
        J, N, P = y
        dJdt = r_J * J * (1 - J / K(t)) - alpha * J * N - beta * J * P
+ H(t)
        dNdt = r_N * N * (1 - N / K_N) - delta * N * J
        dPdt = s * (J + N) * P - m * P
        return [dJdt, dNdt, dPdt]

    sol = solve_ivp(
        ecosystem_model_predation, t_span, [J_0, N_0, P_0],
t_eval=t_eval, method='RK45'
    )

    plt.plot(sol.t, sol.y[2], label=f's = {s}')

plt.title('Effect of Predation Success Rate on Predator Population')
plt.xlabel('Time (years)')
plt.ylabel('Predator Population')
plt.legend()
plt.grid(True)
plt.show()
```

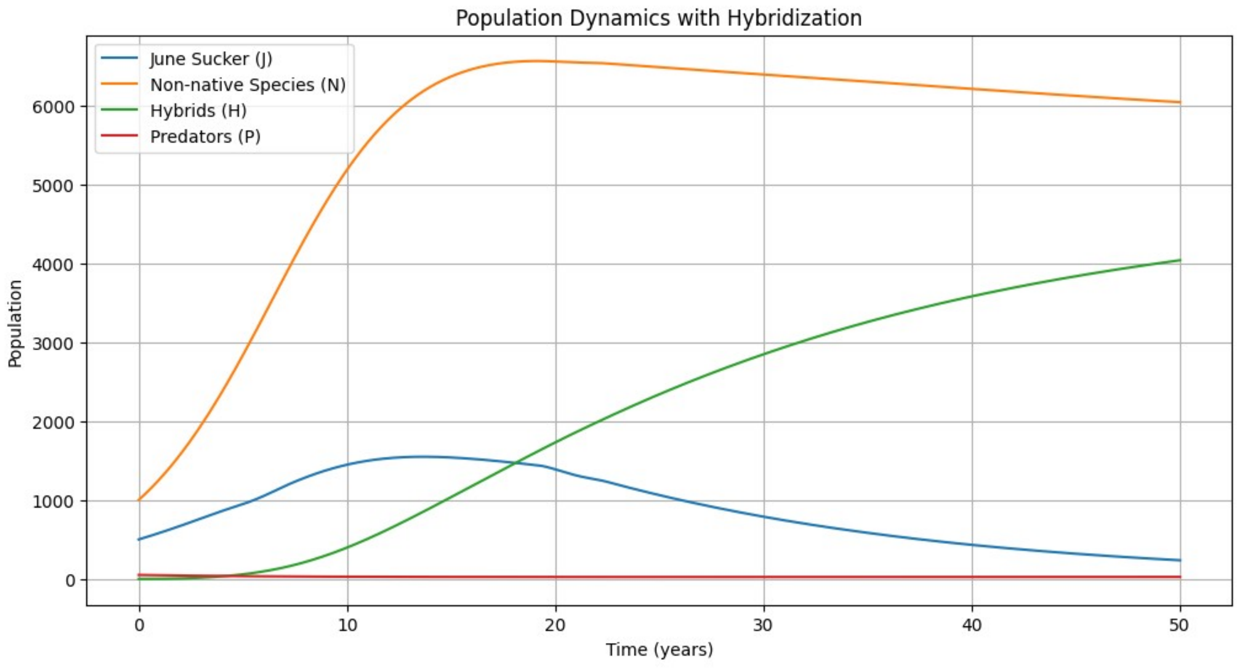


3. Model Implementation with Hybridization

Finally, we incorporate hybridization into the model, adding a hybrid population (H) and updating the equations accordingly. We also perform sensitivity analyses on the hybridization coefficient (s_H) and the growth rate of hybrids (r_H).

```
# Updated model parameters including hybrids
r_J = 0.2
r_N = 0.3
r_H = 0.25
K_0 = 10000
c_JN = c_NJ = 1.0
c_JH = c_NH = c_HJ = c_HN = 0.9
beta_J = beta_N = beta_H = 5e-5
s_H = 1e-5
s_P = 1e-5
m_P = 0.1
gamma = 0.5

# Human intervention functions (updated)
def H_J(t):
    return 50 if 5 <= t <= 20 else 0

def H_N(t):
```

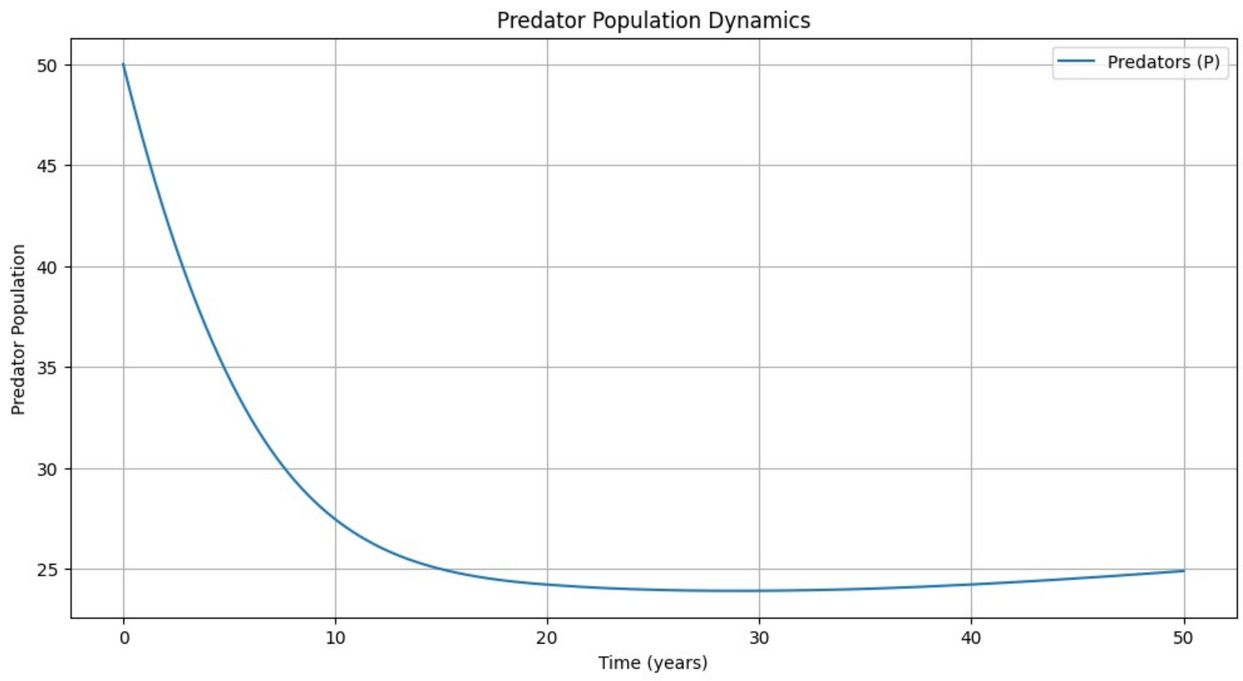
```python
    return -10

def H_H(t):
    return -5

# Carrying capacity function (updated)
def K(t):
    return K_0 + gamma * H_J(t)

# ODE system including hybrids
def ecosystem_model_hybrid(t, y):
    J, N, H, P = y
    dJdt = (
        r_J * J * (1 - (J + c_JN * N + c_JH * H) / K(t))
        - beta_J * J * P
        - s_H * J * N
        + H_J(t)
    )
    dNdt = (
        r_N * N * (1 - (N + c_NJ * J + c_NH * H) / K(t))
        - beta_N * N * P
        - s_H * J * N
        + H_N(t)
    )
    dHdt = (
        s_H * J * N
        + r_H * H * (1 - (H + c_HJ * J + c_HN * N) / K(t))
        - beta_H * H * P
        + H_H(t)
    )
    dPdt = s_P * (J + N + H) * P - m_P * P
    return [dJdt, dNdt, dHdt, dPdt]

# Time span and initial conditions
t_span = (0, 50)
t_eval = np.linspace(*t_span, 500)
initial_conditions = [500, 1000, 0, 50]  # J_0, N_0, H_0, P_0

# Solving the ODE system
solution = solve_ivp(ecosystem_model_hybrid, t_span,
initial_conditions, t_eval=t_eval)

# Extracting results
J_t, N_t, H_t, P_t = solution.y

# Plotting the results
plt.figure(figsize=(12, 6))
plt.plot(solution.t, J_t, label='June Sucker (J)')
plt.plot(solution.t, N_t, label='Non-native Species (N)')
plt.plot(solution.t, H_t, label='Hybrids (H)')
plt.plot(solution.t, P_t, label='Predators (P)')
plt.title('Population Dynamics with Hybridization')
plt.xlabel('Time (years)')
plt.ylabel('Population')
plt.legend()
plt.grid(True)
plt.show()
```



Checking if Predator Population Ever Reaches Zero

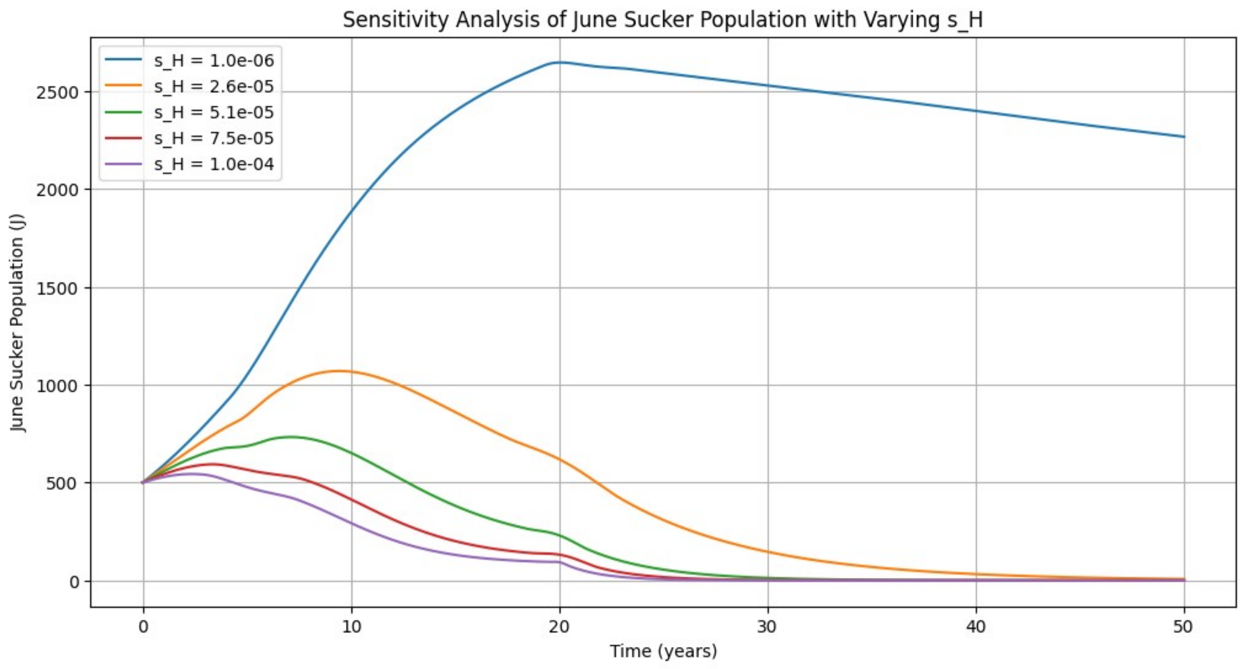
```
# Define a small threshold for extinction
threshold = 1e-6

# Find indices where predator population is zero or near zero
extinction_indices = np.where(P_t <= threshold)[0]

if extinction_indices.size > 0:
    # Get the corresponding times
    extinction_times = solution.t[extinction_indices]
    print(f"The predator population reaches zero or near zero at
times: {extinction_times}")
else:
    print("The predator population never reaches zero during the
simulation.")

# Plot the predator population
plt.figure(figsize=(12, 6))
plt.plot(solution.t, P_t, label='Predators (P)')
plt.title('Predator Population Dynamics')
plt.xlabel('Time (years)')
plt.ylabel('Predator Population')
plt.legend()
plt.grid(True)
plt.show()
```

```
The predator population never reaches zero during the simulation.
```



Sensitivity Analysis: Impact of Hybridization Coefficient (s_H)

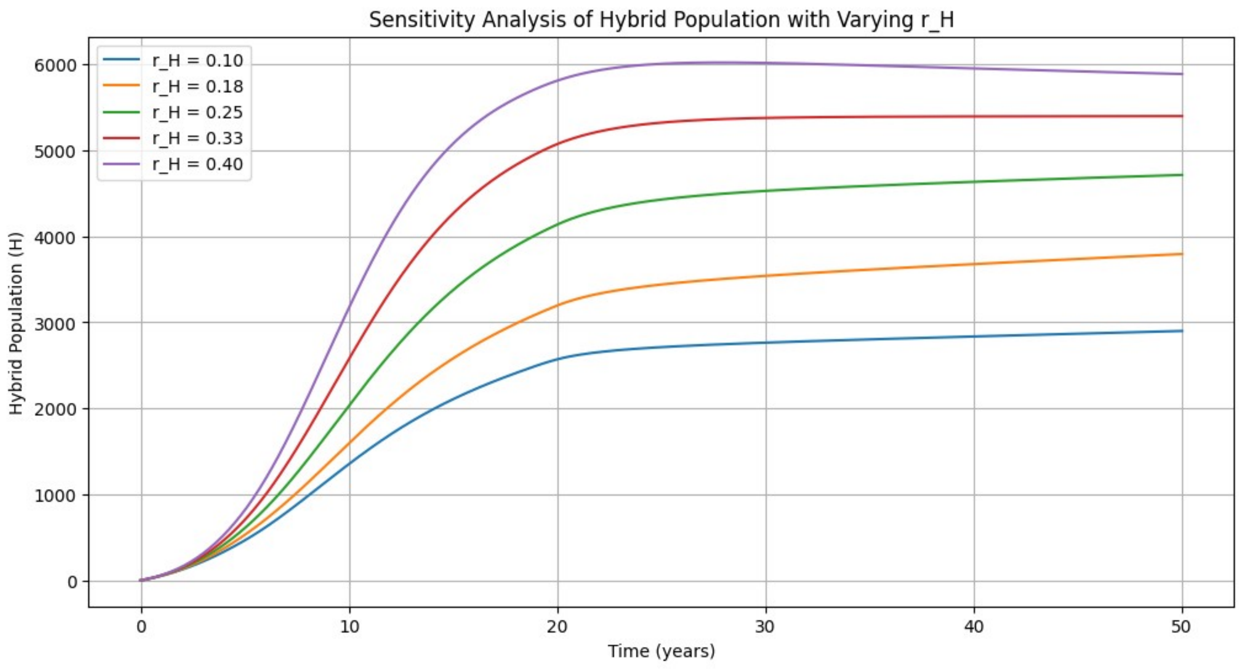
```
# ODE system modified to accept s_H as a parameter
def ecosystem_model_sH(t, y, s_H):
    J, N, H, P = y
    dJdt = (
        r_J * J * (1 - (J + c_JN * N + c_JH * H) / K(t))
        - beta_J * J * P
        - s_H * J * N
        + H_J(t)
    )
    dNdt = (
        r_N * N * (1 - (N + c_NJ * J + c_NH * H) / K(t))
        - beta_N * N * P
        - s_H * J * N
        + H_N(t)
    )
    dHdt = (
        s_H * J * N
        + r_H * H * (1 - (H + c_HJ * J + c_HN * N) / K(t))
        - beta_H * H * P
        + H_H(t)
    )
    dPdt = s_P * (J + N + H) * P - m_P * P
    return [dJdt, dNdt, dHdt, dPdt]

# Range of s_H values for sensitivity analysis
s_H_values = np.linspace(1e-6, 1e-4, 5)

# Prepare to store results
results = []

# Perform simulations for each s_H value
for s_H in s_H_values:
    # Solving the ODE system with the current s_H
    solution = solve_ivp(
        lambda t, y: ecosystem_model_sH(t, y, s_H),
        t_span,
        initial_conditions,
        t_eval=t_eval
    )
    # Store the time and June sucker population
    results.append({
        's_H': s_H,
        'time': solution.t,
        'J_t': solution.y[0]
    })

# Plotting the results
plt.figure(figsize=(12, 6))

for res in results:
    plt.plot(
        res['time'],
        res['J_t'],
        label=f's_H = {res["s_H"]:.1e}'
    )

plt.title('Sensitivity Analysis of June Sucker Population with Varying
s_H')
plt.xlabel('Time (years)')
plt.ylabel('June Sucker Population (J)')
plt.legend()
plt.grid(True)
plt.show()
```



Sensitivity Analysis: Impact of Growth Rate of Hybrids (r_H)

```
# ODE system modified to accept r_H as a parameter
def ecosystem_model_rH(t, y, r_H):
    J, N, H, P = y
    dJdt = (
        r_J * J * (1 - (J + c_JN * N + c_JH * H) / K(t))
        - beta_J * J * P
        - s_H * J * N
        + H_J(t)
    )
    dNdt = (
        r_N * N * (1 - (N + c_NJ * J + c_NH * H) / K(t))
        - beta_N * N * P
        - s_H * J * N
        + H_N(t)
    )
    dHdt = (
        s_H * J * N
        + r_H * H * (1 - (H + c_HJ * J + c_HN * N) / K(t))
        - beta_H * H * P
        + H_H(t)
    )
    dPdt = s_P * (J + N + H) * P - m_P * P
    return [dJdt, dNdt, dHdt, dPdt]

# Range of r_H values for sensitivity analysis
r_H_values = np.linspace(0.1, 0.4, 5)

# Prepare to store results
```

```
results = []

# Perform simulations for each r_H value
for r_H in r_H_values:
    # Solving the ODE system with the current r_H
    solution = solve_ivp(
        lambda t, y: ecosystem_model_rH(t, y, r_H),
        t_span,
        initial_conditions,
        t_eval=t_eval
    )
    # Store the time and hybrid population
    results.append({
        'r_H': r_H,
        'time': solution.t,
        'H_t': solution.y[2]
    })

# Plotting the results
plt.figure(figsize=(12, 6))

for res in results:
    plt.plot(
        res['time'],
        res['H_t'],
        label=f'r_H = {res["r_H"]:.2f}'
    )

plt.title('Sensitivity Analysis of Hybrid Population with Varying 
r_H')
plt.xlabel('Time (years)')
plt.ylabel('Hybrid Population (H)')
plt.legend()
plt.grid(True)
plt.show()

# Maximum hybrid population
for res in results:
    max_H = np.max(res['H_t'])
    print(f"r_H = {res['r_H']:.2f}, Maximum Hybrid Population: 
{max_H:.2f}")
```



```
r_H = 0.10, Maximum Hybrid Population: 2899.89
r_H = 0.18, Maximum Hybrid Population: 3794.32
r_H = 0.25, Maximum Hybrid Population: 4713.88
r_H = 0.33, Maximum Hybrid Population: 5397.52
r_H = 0.40, Maximum Hybrid Population: 6020.80
```

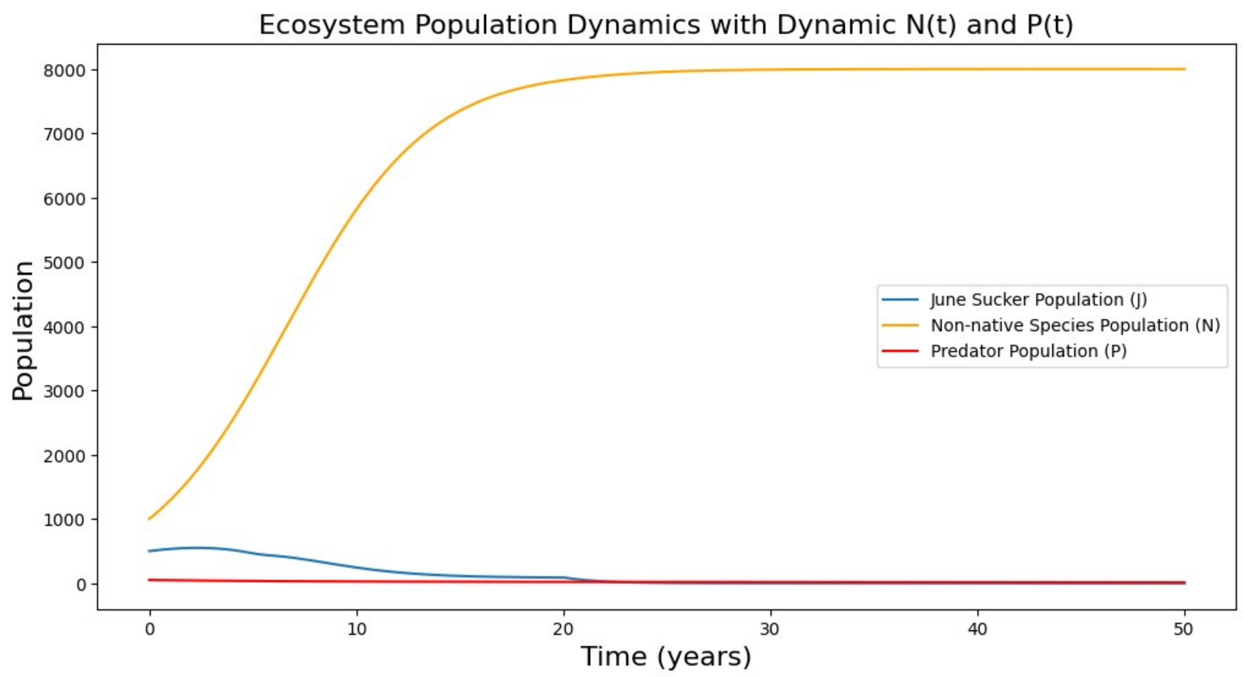
Import libraries and tools

```
from scipy.optimize import fsolve
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
```

No hybridization model

```
# Define the model parameters
r_J = 0.2               # Intrinsic growth rate of June sucker
K_0 = 5000              # Baseline carrying capacity
alpha = 1e-4            # Competition coefficient
beta = 5e-5             # Predation coefficient
gamma = 0.5             # Effectiveness of human intervention
N = 1000                # Non-native species population
P = 50                  # Predator population

# Non-native species parameters
r_N = 0.3               # Intrinsic growth rate of non-native species
K_N = 8000              # Carrying capacity for non-native species
delta = 1e-5            # Competition coefficient (impact of J on N)

# Predator parameters
s = 1e-5                # Predation success rate
m = 0.1                 # Mortality rate of predators

# Set the time span for the simulation
t_span = (0, 50)   # Simulate from year 0 to year 50
t_eval = np.linspace(t_span[0], t_span[1], 500)  # Evaluation times

def H(t):
    # Scenario: Constant stocking and habitat restoration between year
5 and 20
    if 5 <= t <= 20:
        return 50    # Stocking rate of 50 fish per year
    else:
        return 0

# Define the carrying capacity function K(t)
def K(t):
    return K_0 + gamma * H(t)

# Redefine the ODE system including N(t) and P(t)
def ecosystem_model(t, y):
    J, N, P = y  # Unpack the variables
    # June sucker population dynamics
    dJdt = r_J * J * (1 - J / K(t)) - alpha * J * N - beta * J * P +
H(t)
```

```python
    # Non-native species population dynamics
    dNdt = r_N * N * (1 - N / K_N) - delta * N * J
    # Predator population dynamics
    dPdt = s * (J + N) * P - m * P
    return [dJdt, dNdt, dPdt]

# Set the initial conditions
J_0 = 500  # Initial June sucker population

# Set initial conditions for N and P
N_0 = 1000  # Initial non-native species population
P_0 = 50    # Initial predator population

# Solve the ODE system
sol = solve_ivp(
    ecosystem_model, t_span, [J_0, N_0, P_0], t_eval=t_eval,
method='RK45'
)

# Extract solutions
J_t = sol.y[0]
N_t = sol.y[1]
P_t = sol.y[2]

# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(sol.t, J_t, label='June Sucker Population (J)')
plt.plot(sol.t, N_t, color='orange', label='Non-native Species
Population (N)')
plt.plot(sol.t, P_t, color='red', label='Predator Population (P)')
plt.title('Ecosystem Population Dynamics with Dynamic N(t) and P(t)',
fontsize=16)
plt.xlabel('Time (years)', fontsize=16)
plt.ylabel('Population', fontsize=16)
plt.legend()
plt.show()
```

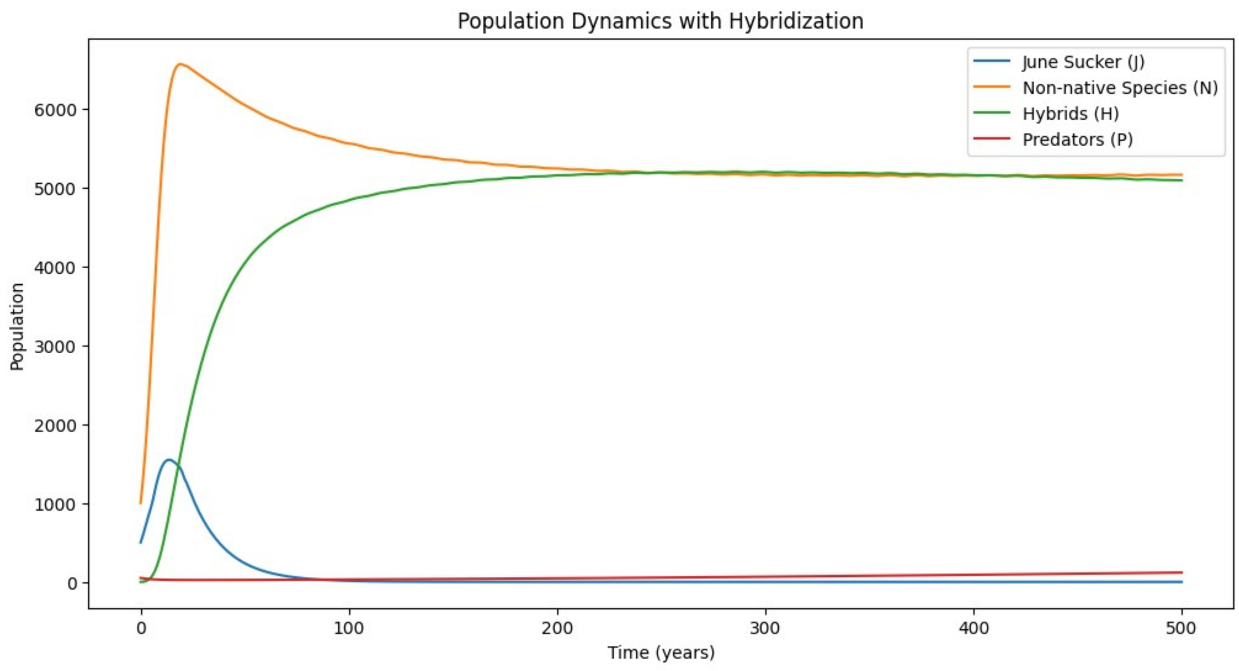


```
# Define the equilibrium equations
def equilibrium_system(vars):
    J, N, P = vars  # Unpack the variables
    # June sucker population dynamics
    dJdt = r_J * J * (1 - J / K(0)) - alpha * J * N - beta * J * P +
H(0)
    # Non-native species population dynamics
    dNdt = r_N * N * (1 - N / K_N) - delta * N * J
    # Predator population dynamics
    dPdt = s * (J + N) * P - m * P
    return [dJdt, dNdt, dPdt]

# Initial guess for the solver
initial_guess = [500, 1000, 50]  # Initial populations of J, N, P

# Solve the system
equilibrium_points = fsolve(equilibrium_system, initial_guess)

# Extract and print the equilibrium points
J_eq, N_eq, P_eq = equilibrium_points
print(f"Equilibrium Population of June Suckers (J): {J_eq:.2f}")
print(f"Equilibrium Population of Non-native Species (N): {N_eq:.2f}")
print(f"Equilibrium Population of Predators (P): {P_eq:.2f}")
```

```
Equilibrium Population of June Suckers (J): 0.00
Equilibrium Population of Non-native Species (N): 0.00
Equilibrium Population of Predators (P): 0.00
```

Final combined model

```python
# Updated model parameters including hybrids
r_J = 0.2
r_N = 0.3
r_H = 0.25
K_0 = 10000
c_JN = c_NJ = 1.0
c_JH = c_NH = c_HJ = c_HN = 0.9
beta_J = beta_N = beta_H = 5e-5
s_H = 1e-5
s_P = 1e-5
m_P = 0.1
gamma = 0.5

# Human intervention functions (updated)
def H_J(t):
    return 50 if 5 <= t <= 20 else 0

def H_N(t):
    return -10

def H_H(t):
    return -5

# Carrying capacity function (updated)
def K(t):
    return K_0 + gamma * H_J(t)

# ODE system including hybrids
def ecosystem_model_hybrid(t, y):
    J, N, H, P = y
    dJdt = (
        r_J * J * (1 - (J + c_JN * N + c_JH * H) / K(t))
        - beta_J * J * P
        - s_H * J * N
        + H_J(t)
    )
    dNdt = (
        r_N * N * (1 - (N + c_NJ * J + c_NH * H) / K(t))
        - beta_N * N * P
        - s_H * J * N
        + H_N(t)
    )
    dHdt = (
        s_H * J * N
        + r_H * H * (1 - (H + c_HJ * J + c_HN * N) / K(t))
        - beta_H * H * P
        + H_H(t)
    )
    dPdt = s_P * (J + N + H) * P - m_P * P
    return [dJdt, dNdt, dHdt, dPdt]

# Time span and initial conditions
t_span = (0, 500) # Adjusted time span to determine when equilibrium
takes place
t_eval = np.linspace(*t_span, 500)
initial_conditions = [500, 1000, 0, 50]  # J_0, N_0, H_0, P_0

# Solving the ODE system
solution = solve_ivp(ecosystem_model_hybrid, t_span,
initial_conditions, t_eval=t_eval)

# Extracting results
J_t, N_t, H_t, P_t = solution.y

# Plotting the results
plt.figure(figsize=(12, 6))
plt.plot(solution.t, J_t, label='June Sucker (J)')
plt.plot(solution.t, N_t, label='Non-native Species (N)')
plt.plot(solution.t, H_t, label='Hybrids (H)')
plt.plot(solution.t, P_t, label='Predators (P)')
plt.title('Population Dynamics with Hybridization')
plt.xlabel('Time (years)')
plt.ylabel('Population')
plt.legend()
plt.show()
```



Calculate equilibrium points of final combined model

```python
# Redefine the system with no time dependence for equilibrium analysis
def equilibrium_system(y, t=0):
    J, N, H, P = y
    dJdt = (
        r_J * J * (1 - (J + c_JN * N + c_JH * H) / K_0)
        - beta_J * J * P
        - s_H * J * N
        + 50 # Assuming constant stocking of June suckers
    )
    dNdt = (
        r_N * N * (1 - (N + c_NJ * J + c_NH * H) / K_0)
        - beta_N * N * P
        - s_H * J * N
        - 10  # Assuming constant removal of non-native species
    )
    dHdt = (
        s_H * J * N
        + r_H * H * (1 - (H + c_HJ * J + c_HN * N) / K_0)
        - beta_H * H * P
        - 5  # Assuming constant removal of hybrids
    )
    dPdt = s_P * (J + N + H) * P - m_P * P
    return [dJdt, dNdt, dHdt, dPdt]

# Solve for equilibrium
J_eq, N_eq, H_eq, P_eq = fsolve(equilibrium_system,
initial_conditions)

# Print the results
print(f"Equilibrium June Sucker Population (J):          {J_eq}")
print(f"Equilibrium Non-native Species Population (N): {N_eq}")
print(f"Equilibrium Hybrid Population (H):               {H_eq}")
print(f"Equilibrium Predator Population (P):             {P_eq}")
```

```
Equilibrium June Sucker Population (J):         -245.60430962500055
Equilibrium Non-native Species Population (N): 32.43469781599387
Equilibrium Hybrid Population (H):              19.97531368896398
Equilibrium Predator Population (P):            3.3645717464827046e-09
```

Calculate when equilibrium points occur

```python
# Equilibrium points
equilibrium_values = [-245.60, 32.43, 19.98, 0]  # Suppose June Sucker
equilibrium is 0 instead of -245.60
tolerance = 1e-2  # Define a tolerance for "close to equilibrium"

# Time points and species populations from the solution
time_points = solution.t
```

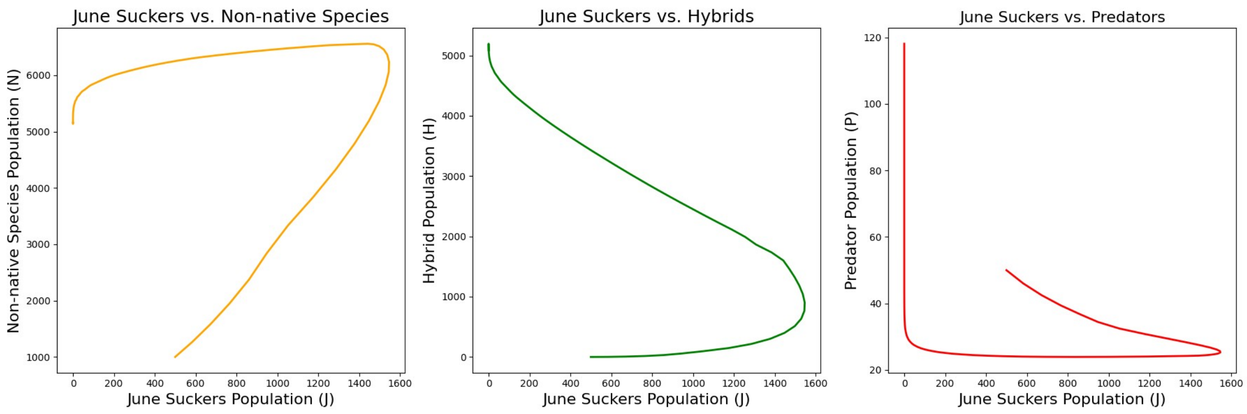
```python
populations = solution.y  # J_t, N_t, H_t, P_t

# Find the time when each species hits equilibrium
def find_equilibrium_time(time_points, population, equilibrium_value,
tolerance):
    for t, pop in zip(time_points, population):
        if abs(pop - equilibrium_value) <= tolerance:
            return t
    return None  # Return None if equilibrium is not reached

equilibrium_times = [
    find_equilibrium_time(time_points, species_pop, eq_value,
tolerance)
    for species_pop, eq_value in zip(populations, equilibrium_values)
]

# Display results
species_names = ["June Sucker (J)", "Non-native Species (N)", "Hybrid
(H)", "Predator (P)"]
for species, eq_time in zip(species_names, equilibrium_times):
    if eq_time is not None:
        print(f"{species} reaches equilibrium at t = {eq_time:.2f}")
    else:
        print(f"{species} does not reach equilibrium within the
simulation time.")
```

```
June Sucker (J) does not reach equilibrium within the simulation time.
Non-native Species (N) does not reach equilibrium within the
simulation time.
Hybrid (H) does not reach equilibrium within the simulation time.
Predator (P) does not reach equilibrium within the simulation time.
```

Create phase plots

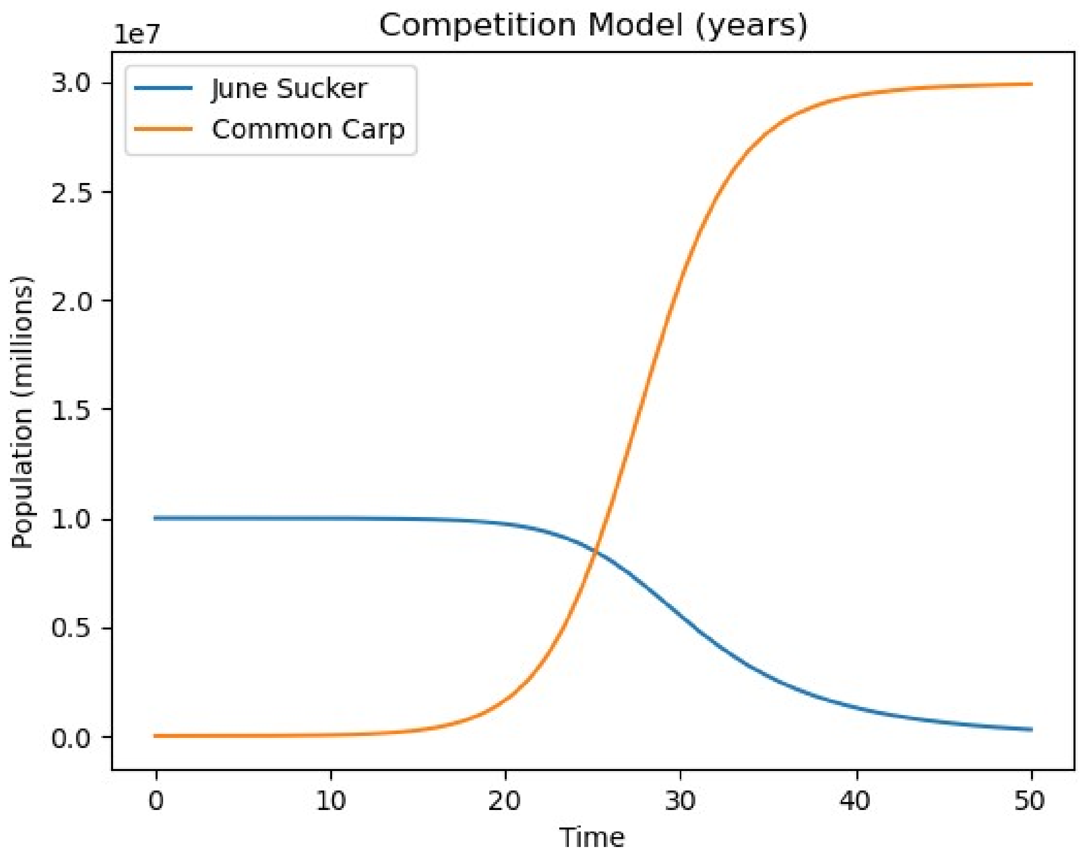
```python
# Create a figure with 3 subplots in one row
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Phase plot 1: June suckers vs. non-native species
axes[0].plot(J_t, N_t, lw=2, color='orange')
axes[0].set_title('June Suckers vs. Non-native Species', fontsize=18)
axes[0].set_xlabel('June Suckers Population (J)', fontsize=16)
axes[0].set_ylabel('Non-native Species Population (N)', fontsize=16)

# Phase plot 2: June suckers vs. hybrids
axes[1].plot(J_t, H_t, lw=2, color='green')
axes[1].set_title('June Suckers vs. Hybrids', fontsize=18)
axes[1].set_xlabel('June Suckers Population (J)', fontsize=16)
axes[1].set_ylabel('Hybrid Population (H)', fontsize=16)

# Phase plot 3: June suckers vs. predators
```

```
axes[2].plot(J_t, P_t, lw=2, color='red')
axes[2].set_title('June Suckers vs. Predators', fontsize=16)
axes[2].set_xlabel('June Suckers Population (J)', fontsize=16)
axes[2].set_ylabel('Predator Population (P)', fontsize=16)

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



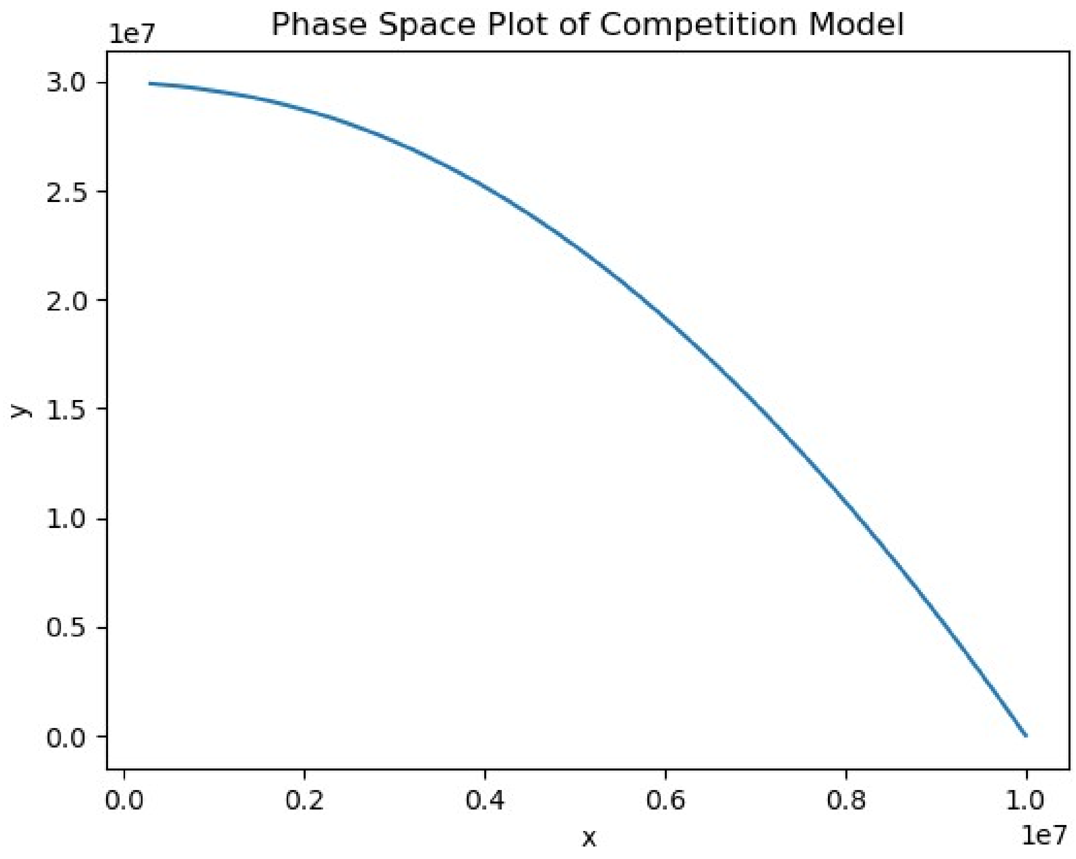
Competition Model

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
from scipy.integrate import solve_ivp

def competition(t, y, alpha, beta, r_s, r_c, k_s, k_c):
    """Compute right hand side of the competition model based on june
sucker and
    common carp populations at the given time.

    Parameters:
        y ((2, ) ndarray): A vector representing june sucker and
common carp populations at time t.
        t (float): Current time.

    Returns:
        (tuple): A tuple corresponding to right hand side of the
competition model.
    """

    S, C = y
    return (r_s*S*(1 - (S + (alpha*C))/k_s), r_c*C*(1 - (C +
(beta*S))/k_c))

t0, tf = 0, 50
t = np.linspace(t0, tf, 200)
t_span = (t0, tf) # Time Domain
y0 = np.array([10000000, 1000]) # Initial Conditions

## Initial Parameters ##

r_s = 0.1 # June Sucker Growth Rate
r_c = 0.4 # Common Carp Growth Rate
k_s = 10000000 # June Sucker Carrying Capacity
k_c = 30000000 # Common Carp Carrying Capacity
alpha = 0.8 # Competition Coefficient: Impact of Common Carp on June
Suckers
beta = 0.2 # Competition Coefficient: Impact of June Suckers on Common
Carp

# Solve the System
sol = solve_ivp(competition, t_span, y0, t_eval=t, args=(alpha, beta,
r_s, r_c, k_s, k_c))

# Plot June Sucker and Common Carp Populations Over Time
plt.plot(sol.t, sol.y[0], label='June Sucker')
plt.plot(sol.t, sol.y[1], label='Common Carp')
plt.xlabel('Time')
plt.ylabel('Population (millions)')
plt.title('Competition Model (years)')
plt.legend()
plt.show()
```



Phase Space Diagram

```
# Phase Space Plot
plt.plot(sol.y[0], sol.y[1])
plt.title('Phase Space Plot of Competition Model')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



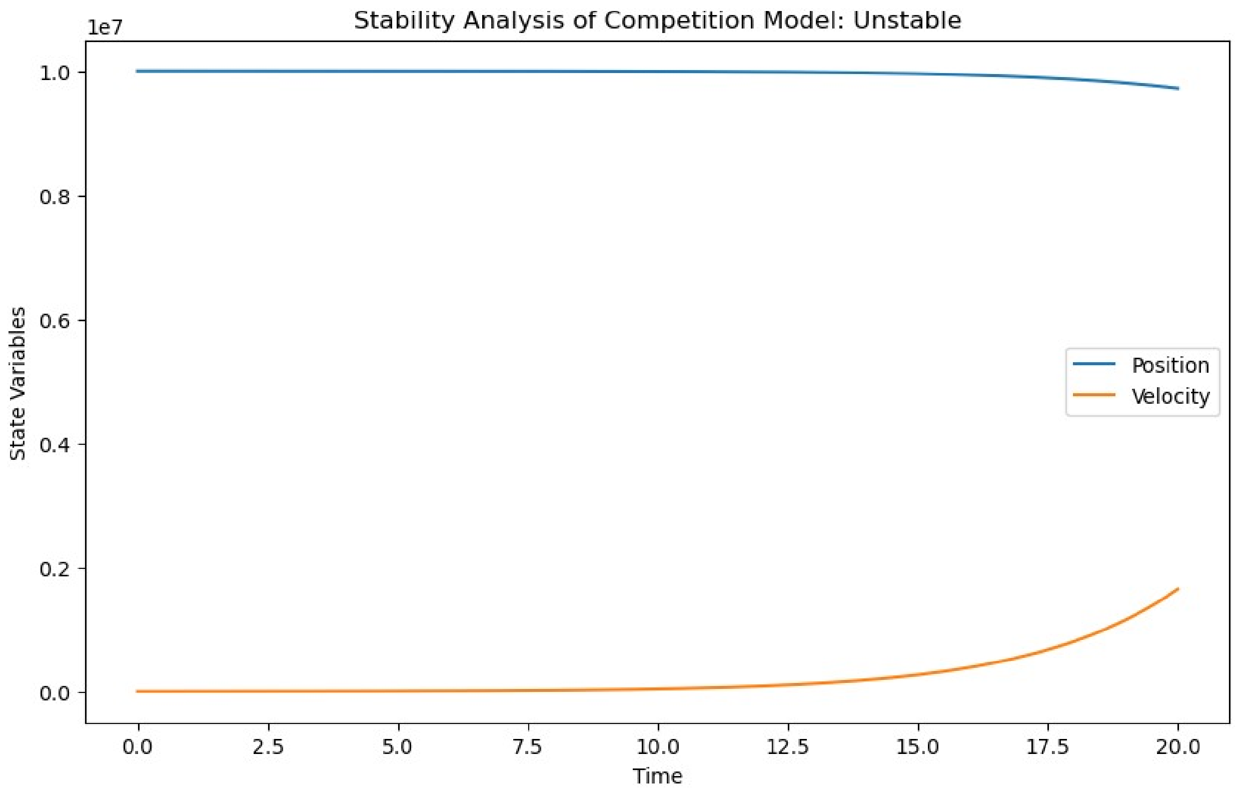
Equilibrium Points

```python
def competitionEQ(y, alpha, beta, r_s, r_c, k_s, k_c):
    """Compute right hand side of the competition model based on june
sucker and
    common carp populations at the given time.

    Parameters:
        y ((2, ) ndarray): A vector representing june sucker and
common carp populations at time t.
        t (float): Current time.

    Returns:
        (tuple): A tuple corresponding to right hand side of the
competition model.
    """

    S, C = y
    return (r_s*S*(1 - (S + (alpha*C))/k_s), r_c*C*(1 - (C +
(beta*S))/k_c))

# Find equilibrium point
initial_guess = np.array([0.1, 0.4])
```

```
equilibrium = fsolve(competitionEQ, initial_guess, args=(alpha, beta,
r_s, r_c, k_s, k_c))
```

Stability Analysis

```
# Jacobian
def jacobian(S, C, alpha, beta, r_s, r_c, k_s, k_c):
    return np.array([[r_s*(-alpha*C + k_s - 2*S) / k_s, -(alpha*r_s*S)
/ k_s],
                     [-(beta*r_c*C) / k_c, r_c*(-beta*S + k_c - 2*C) /
k_c]])

# Stability Analysis
def stability(S, C, alpha, beta, r_s, r_c, k_s, k_c):
    J = jacobian(S, C, alpha, beta, r_s, r_c, k_s, k_c)
    eigenvalues = np.linalg.eigvals(J)
    stability = "Stable" if np.all(np.real(eigenvalues) < 0) else
"Unstable"
    return stability

## Initial Parameters ##
t0, tf = 0, 50
t = np.linspace(t0, tf, 200)
t_span = (t0, tf) # Time Domain
y0 = np.array([10000000, 1000]) # START SLIGHTLY AWAY FROM THE
EQUILIBRIUM!!!

r_s = 0.1 # June Sucker Growth Rate
r_c = 0.4 # Common Carp Growth Rate
k_s = 10000000 # June Sucker Carrying Capacity
k_c = 30000000 # Common Carp Carrying Capacity
alpha = 0.8 # Competition Coefficient: Impact of Common Carp on June
Suckers
beta = 0.2 # Competition Coefficient: Impact of June Suckers on Common
Carp

# Solve the System
sol = solve_ivp(competition, t_span, y0, args=(alpha, beta, r_s, r_c,
k_s, k_c), dense_output=True)

# Evaluate Stability
stable = stability(equilibrium[0], equilibrium[1], alpha, beta, r_s,
r_c, k_s, k_c)

# Generate Points for Plotting
t = np.linspace(0, 20, 200)
y = sol.sol(t)

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(t, y[0], label='Position')
plt.plot(t, y[1], label='Velocity')
plt.title(f'Stability Analysis of Competition Model: {stable}')
plt.xlabel('Time')
plt.ylabel('State Variables')
plt.legend()
plt.show()
```


Stability Analysis of Competition Model: Unstable
1e7
State Variables
Position
Velocity
Time