Cornell 2023 CMCM Competition - Challenge 2: Saving Grapes

by Andrew Liu, Yilun Yin, Taylor Wang

November 13th

Part I: Introductin

The Spotted Lanternfly (SLF), an invasive species causing significant damage to vineyards, poses a growing threat to the viticulture industry worldwide. In vineyards, the primary method of combating SLF infestations is through the application of insecticides. However, this approach is fraught with challenges, including the timing of applications, environmental impact, and the potential effects on grape quality and vine health. In fact, indiscriminate use of insecticides can lead to delayed harvests, increased vulnerability to fungal diseases, and detrimental impacts on beneficial insect populations. This complexity necessitates a nuanced approach to insecticide use, particularly in balancing the immediate need to protect crops against long-term ecological and financial consequences. In this paper, we propose a dynamic and adaptive strategy for insecticide application in vineyards, aiming to optimize the timing and threshold for spraying based on a variety of factors such as SLF migration patterns, weather conditions, and the proximity to harvest time. The strategy is designed to minimize the negative impacts of SLF on vineyards while also considering the broader ecological implications and the sustainability of viticulture practices.

Part II: Data and Helper Functions

In [ ]:
```
import numpy as np
import pandas as pd

# PARAMS
temperature = 0
precipitation = 0

slf_pop = 1000
# To account for population growth on a yearly basis:
# new population = surviving population * (1 + proportion are females * egg masses laid per female * 35-45 eggs per mass)

alpha = 0.5     # Proportion of lantern flies that are adults
# Since there is only one generation of lantern flies each year, assume that alpha
# is initially 0 from March to July. From July to August/September, we can increase alpha
# using maybe softmax. From August/September to October/November, assume alpha = 1 (we only have adults)

adult_pop = alpha * slf_pop
nymph_pop = (1 - alpha) * slf_pop

pesticide_use = {}
```

In [ ]:
```
def slf_population_model(temperature, precipitation, initial_eggs=10, start_day=1, end_day=183):
    """
    Estimates the counts of nymphs and adult SLFs from early May (day 1) to late October (day 183).

    Parameters:
    temperature (list): Daily average temperatures.
    precipitation (list): Daily precipitation levels.
    initial_eggs (int): Initial number of SLF eggs at the beginning of the period.
    start_day (int): Start day of the period (default is 1, early May).
    end_day (int): End day of the period (default is 183, late October).

    Returns:
    dict: Counts of nymphs and adults over time.
    """
    days = np.arange(start_day, end_day + 1)
    nymph_count = np.zeros_like(days, dtype=int)
    adult_count = np.zeros_like(days, dtype=int)

    # Optimal conditions
    optimal_temperature = 25  # in Celsius
    optimal_precipitation = 5  # hypothetical optimal precipitation level

    # Growth rate and hatching time adjustment based on conditions
    growth_rate = 0.01  # Base growth rate from nymph to adult
    for i, day in enumerate(days):
        temp_factor = max(0, min(1, (temperature[i] - 10) / 15))
        precip_factor = 1 - min(1, abs(precipitation[i] - optimal_precipitation) / 10)

        # Adjusting hatching and growth based on conditions
        if day == 1:
            nymph_count[i] = initial_eggs  # All eggs hatch on day 1 in optimal conditions
        elif day < 60:  # Early stage: growing nymphs
            nymph_count[i] = nymph_count[i-1] + int(nymph_count[i-1] * growth_rate * temp_factor * precip_factor)
        elif 60 <= day < 120:  # Transition period
            adult_growth = int(nymph_count[i-1] * growth_rate * temp_factor * precip_factor)
            nymph_count[i] = nymph_count[i-1] - adult_growth
            adult_count[i] = adult_count[i-1] + adult_growth
        else:  # Later stage: mostly adults
            adult_count[i] = adult_count[i-1] + nymph_count[i-1]
            nymph_count[i] = 0

    return {"days": days, "nymph_count": nymph_count, "adult_count": adult_count}

# Example usage
```

```
temperature = [25] * 183  # Constant temperature for simplicity
precipitation = [5] * 183  # Constant precipitation for simplicity
initial_eggs = 1000  # Hypothetical initial number of eggs
population = slf_population_model(temperature, precipitation)
```

The code makes a few key assumptions about the Spotted Lanternfly (SLF) life cycle and the impact of environmental factors, particularly temperature and precipitation:

Temperature Impact on Maturation:

The code assumes that higher temperatures, up to an optimal point, accelerate the maturation of SLFs from nymphs to adults. This is modeled by the temp_factor, which increases with temperature up to a certain threshold (assumed here as 25°C for optimal hatching). Beyond this threshold, the impact of temperature is not further increased in the model, which is a simplification. The temperature effect is linearly scaled between a minimum threshold (10°C in this model) and the optimal temperature (25°C). Below 10°C, it's assumed there's no maturation (temp_factor = 0), and above 25°C, the maturation rate doesn't increase further (temp_factor = 1). Precipitation Impact:

The model assumes that higher precipitation levels negatively impact the SLF's activity, including maturation. This is represented by precip_factor, which decreases as precipitation increases, assuming reduced activity in wet conditions. The impact of precipitation is simplified and capped at a certain level (10mm in this model), beyond which it's assumed that additional precipitation doesn't further reduce SLF activity. Life Cycle Timing:

The model divides the SLF life cycle into three phases based on the day of the year: early stage (mostly nymphs), transition period (mix of nymphs and adults), and later stage (mostly adults). These phases are set based on fixed day ranges, which is a simplification and may not accurately reflect variations in SLF development due to environmental or geographical factors. Uniformity Across Population:

The model assumes uniformity in the response of the SLF population to environmental factors. In reality, there would be variations within the population, with some individuals developing faster or slower than others. Constant Environmental Conditions:

In the example usage, constant temperature and precipitation are used for simplicity. However, in real-world scenarios, these factors would vary daily and would have a more complex impact on the SLF population. This model is a basic representation and should ideally be refined with more detailed empirical data on the effects of temperature and precipitation on the SLF life cycle. The assumptions are made to provide a starting point for modeling, but they simplify the complex interactions in an actual ecological system.

https://extension.psu.edu/spotted-lanternfly-management-guide

"To protect pollinators that visit flowers for nectar, never spray any insecticide on plants that are blooming." To reduce impact on other insects, we should consider reducing the application of pesticides while grapevines are flowering. Ideally, we use most effective insecticide with minimal PHI close to harvest date.

NEED TO FIND WHEN GRAPEVINE FLOWERING BEGINS AND ENDS

https://extension.psu.edu/spotted-lanternfly-management-in-vineyards

Table 1. shows that nymphs target grape plants starting in May to July, then adults target grapes from August to October.

"SLF are voracious feeders and can be extremely abundant as adults in vineyards. Adults start to appear in vineyards in August, but high populations are not typically observed until mid-to-late September (Figure 2). For vineyards that are first experiencing SLF, this phenology is typically shifted later into the season—you may not see large numbers invade the vineyard until October. After one or two years, adult SLF typically invade vineyards earlier in the season (late August)"

"More importantly, the majority of SLF adult population within a vineyard is observed on the edge; on average, 54 percent of the SLF population is within the first 50 feet of the vineyard edge. Depending on the landscape surrounding the vineyard, the edge of the vineyard may account for even higher SLF numbers (upward of 80 percent of the population)."

"There is one generation of SLF per year." "Egg masses usually contain around 35–45 eggs each (Figure 1A). A single SLF female can lay at least two egg masses."

"The majority of adult SLF observed in vineyards are female."

In [ ]:
```
file_path = 'pesticides.csv'
df = pd.read_csv(file_path)

# Mapping of efficacy ratings to numerical values
efficacy_mapping = {
    'Excellent': 0.95,
    'Good': 0.75,
    'Poor': 0.5,
    'Variable': 0.5,  # Assuming 'Variable' means the effect is inconsistent
    'Not Recommended': 0,  # Assuming 'Not Recommended' means no effect
    '': None  # Assuming empty string means no data available
}

# Mapping the 'Effect on Adults' and 'Effect on Nymphs' columns in the DataFrame
df['Effect on Adults'] = df['Effect on Adults'].map(efficacy_mapping)
df['Effect on Nymphs'] = df['Effect on Nymphs'].map(efficacy_mapping)

# Defining the function to calculate insecticide effectiveness
def insecticide_effectiveness(df, day, nymph_population, adult_population):
    df['Effect_on_Nymphs'] = df['Effect on Nymphs'] * nymph_population
    df['Effect_on_Adults'] = df['Effect on Adults'] * adult_population

    # Assuming 'REI' stands for Re-Entry Interval, which is not present in the given data
    # Adding a placeholder for 'REI' in the DataFrame for the function to work
    # df['REI'] = some_value  # Replace some_value with actual REI data if available

    # The 'PHI (days)' column in the CSV is assumed to be the PHI value
```

```
    df['Available_in'] = df['PHI (days)'].apply(lambda x: max(0, x - day))
    df['Available_until_Harvest'] = df['PHI (days)'].apply(lambda x: max(0, x - day))

    return df[['Product', 'Effect_on_Nymphs', 'Effect_on_Adults', 'Available_in', 'Available_until_Harvest']]

# Example usage of the function, with dummy values for day, nymph_population, and adult_population
# Replace these with actual values as needed
example_day = 10
example_nymph_population = 100
example_adult_population = 50

# Call the function with the example values
result_df = insecticide_effectiveness(df, example_day, example_nymph_population, example_adult_population)
result_df.head()  # Displaying the first few rows of the resulting DataFrame
```

Out[ ]:

	Product	Effect_on_Nymphs	Effect_on_Adults	Available_in	Available_until_Harvest
0	Brigade	95.0	47.5	20.0	20.0
1	Sniper	95.0	47.5	20.0	20.0
2	Mustang Maxx	75.0	47.5	0.0	0.0
3	Baythroid	95.0	47.5	0.0	0.0
4	Danitol	95.0	47.5	11.0	11.0

In [ ]:

```
def predict_slf_counts(initial_nymph_count, initial_adult_count, temperature, precipitation, day):
    """
    Predicts the nymph and adult SLF counts at the end of a given day.

    Parameters:
    initial_nymph_count (int): Initial count of nymph SLFs.
    initial_adult_count (int): Initial count of adult SLFs.
    temperature (float): Daily average temperature.
    precipitation (float): Daily precipitation level.
    day (int): The number of the day in the growing season.

    Returns:
    tuple: Predicted nymph and adult counts at the end of the day.
    """
    optimal_temperature = 25  # in Celsius
    optimal_precipitation = 5  # hypothetical optimal precipitation level
    growth_rate = 0.01  # Base growth rate from nymph to adult
```

```python
    temp_factor = max(0, min(1, (temperature - 10) / 15))
    precip_factor = 1 - min(1, abs(precipitation - optimal_precipitation) / 10)

    if day < 60:  # Early stage: growing nymphs
        new_nymphs = int(initial_nymph_count * growth_rate * temp_factor * precip_factor)
        return initial_nymph_count + new_nymphs, initial_adult_count
    elif 60 <= day < 120:  # Transition period
        transitioning_to_adults = int(initial_nymph_count * growth_rate * temp_factor * precip_factor)
        return initial_nymph_count - transitioning_to_adults, initial_adult_count + transitioning_to_adults
    else:  # Later stage: mostly adults
        return 0, initial_adult_count + initial_nymph_count


def pesticide_application_strategy(temperature, precipitation, harvest_day, df):
    """
    Determines an optimal pesticide application strategy while considering environmental factors,
    the observed SLF population, and pesticide usage constraints.

    Parameters:
    temperature (list): Daily average temperatures.
    precipitation (list): Daily precipitation levels.
    harvest_day (int): The day of the year on which harvest is planned.

    Returns:
    DataFrame: A recommended pesticide application schedule.
    """
    # Define the total number of days in the season
    season_length = len(temperature)

    # Initialize the application schedule DataFrame
    application_schedule = pd.DataFrame({
        'day': range(1, season_length + 1),
        'temperature': temperature,
        'precipitation': precipitation,
        'nymph_count': [0] * season_length,
        'adult_count': [0] * season_length,
        'pesticide': [None] * season_length,  # Placeholder for the chosen pesticide
        'application_rate': [0] * season_length  # Placeholder for the application rate
    })

    # Assume the initial nymph and adult counts are 10 and 0, respectively
    application_schedule.loc[0, 'nymph_count'] = 100

    # Initialize a dictionary to track seasonal usage of each pesticide
```

```
    last_use_day = {pesticide_class: -np.inf for pesticide_class in df['Class'].unique()}

    seasonal_usage = {product: 0 for product in df['Product']} #In unit of fl oz
    seasonal_use_count = {product: 0 for product in df['Product']} #In unit of number of times used


    # Iterate through each day in the season
    nymph_threshold_per_squarefeet = 15*0.06
    adult_threshold_per_squarefeet = 5*0.06


    for index, row in application_schedule.iterrows():
        is_autumn = True
        if row['day'] <= 92: is_autumn = False
        df['Seasonal Max'] = pd.to_numeric(df['Seasonal Max'], errors='coerce')
        df['Max Applications'] = pd.to_numeric(df['Max Applications'], errors='coerce')

        pesticide_applied = False  # This should be set to False at the start of each iteration
        if (not is_autumn and row['nymph_count'] >= nymph_threshold_per_squarefeet) or (is_autumn and row['adult_count'] >= \
            adult_threshold_per_squarefeet):
            for i, pesticide in df.iterrows():
                time_since_last_use = row['day'] - last_use_day[pesticide['Class']]
                if (is_autumn and pesticide['Effect on Adults'] >= 0.75) or (not is_autumn and pesticide['Effect on Nymphs'] >= 0.75) and \
                  time_since_last_use > pesticide['PHI (days)'] and \
                  seasonal_usage[pesticide['Product']] < pesticide['Seasonal Max'] and \
                  seasonal_use_count[pesticide['Product']] < pesticide['Max Applications']:
                    chosen_pesticide = pesticide['Product']
                    application_schedule.loc[index, 'pesticide'] = chosen_pesticide

                    # Apply the pesticide if we are not within the PHI period before harvest
                    if row['day'] <= harvest_day - pesticide['PHI (days)']:
                    # Calculate the effectiveness of the chosen pesticide
                        effectiveness_df = insecticide_effectiveness(
                            df,
                            row['day'],
                            row['nymph_count'],
                            row['adult_count']
                        )
                        # Determine the application rate based on the effectiveness
                        # Here we assume a direct relationship between effectiveness and application rate
                        # This is a simplification and should be refined based on real-world data and expertise
                        efficacy = effectiveness_df.loc[
                            effectiveness_df['Product'] == chosen_pesticide, 'Effect_on_Nymphs'
                        ].iloc[0] if not is_autumn else effectiveness_df.loc[
                            effectiveness_df['Product'] == chosen_pesticide, 'Effect_on_Adults'
```

```
                ].iloc[0]

                # Assume the application rate is proportional to the efficacy
                # This is a simplification; in practice, you would use more complex logic based on pest pressure and other factors
                application_rate = 6.4 * efficacy   #https://www3.epa.gov/pesticides/chem_search/ppls/000279-03313-20211119.pdf

                application_schedule.loc[index, 'application_rate'] = application_rate
                seasonal_usage[chosen_pesticide] += application_rate
                seasonal_use_count[chosen_pesticide] += 1


                last_use_day[pesticide['Class']] = row['day']
                pesticide_applied = True  # Only set this to True if a pesticide is successfully applied
                application_schedule.loc[index, 'nymph_count'] *= (1 - efficacy/100)
                application_schedule.loc[index, 'adult_count'] *= (1 - efficacy/100)

                # Check if we have reached the Seasonal Max for the chosen pesticide
                # if seasonal_usage[chosen_pesticide] > df.loc[
                #     df['Product'] == chosen_pesticide, 'Seasonal Max'
                # ].iloc[0]:
                #     raise ValueError(f"Seasonal Max exceeded for chosen pesticide {chosen_pesticide}.")

                if row['day'] < 182: application_schedule.loc[index + 1, 'nymph_count'], application_schedule.loc[index + 1, 'adult_count'
                    = predict_slf_counts(row['nymph_count'], row['adult_count'], row['temperature'], row['precipitation'], row['day'])
                break  # Break out of the loop after applying a pesticide

            # If no pesticide was applied, predict the SLF counts for the next day
            if row['day'] < 182: application_schedule.loc[index + 1, 'nymph_count'], application_schedule.loc[index + 1, 'adult_count'] = \
                predict_slf_counts(row['nymph_count'], row['adult_count'], row['temperature'], row['precipitation'], row['day'])
        # Raise an error if no pesticide could be applied due to PHI constraints or other issues
        if not pesticide_applied:
            raise ValueError(f"No suitable pesticide found or PHI constraints prevent application on day {row['day']}.")
    if row['day'] < 182: application_schedule.loc[index + 1, 'nymph_count'], application_schedule.loc[index + 1, 'adult_count'] = \
        predict_slf_counts(row['nymph_count'], row['adult_count'], row['temperature'], row['precipitation'], row['day'])
    # Filter out any days where pesticide application is not possible due to PHI constraints
    application_schedule = application_schedule[application_schedule['day'] <= harvest_day - df['PHI (days)'].max()]

    return application_schedule

# Example usage
# These are placeholder values for temperatures and precipitation throughout the season.
temperature = [25] * 183  # Constant temperature for simplicity
precipitation = [5] * 183  # Constant precipitation for simplicity

harvest_day = 160  # Placeholder value for the day of harvest
```

```
# Assuming we've already loaded pesticide_data from a CSV file
# The 'Seasonal Max' would need to be added to the DataFrame based on your pesticide data
# Generate the application schedule

application_schedule = pesticide_application_strategy(temperature, precipitation, harvest_day, df)
print(application_schedule)
```

Part III: Simulation

To simulate the final profit of the vineyard under the influence of SLF infestation and pesticide application, we need to consider several factors:

Vineyard Size and Pesticide Application Area:

The vineyard is assumed to be a square, with pesticide applied up to a certain depth from the borders. The total area where pesticide is applied needs to be calculated. Pesticide Application Strategy:

This strategy should be based on the counts of nymphs and adults. The strategy affects both the cost (due to the amount of pesticide used) and the yield (due to the impact of SLFs and pesticide on grapes). Profit Calculation:

Profit is calculated from the remaining grape yield and the cost of pesticide application. Regularization for Mold Growth:

Mold growth is affected by the number of insects and impacts the grape yield. A late application of pesticide, which postpones harvest, should be penalized.

In [ ]:
```
def calculate_profit(vineyard_length, depth, df, pesticide_cost, base_yield, price_per_grape, mold_growth_rate, harvest_penalty_rate):
    """
    Simulates the final profit considering the impact of SLF, pesticide application, and mold growth.

    Parameters:
    vineyard_length (float): Length of one side of the vineyard (assuming a square), in feet.
    depth (float): Depth from the border where pesticides are applied, in feet.
    df (DataFrame): DataFrame containing daily data on temperature, precipitation, SLF counts, etc.
    pesticide_cost (float): Cost of pesticide per square foot.
    base_yield (float): Base yield of grapes without any pest or mold impact, in units.
    price_per_grape (float): Price per unit of grape.
    mold_growth_rate (float): Rate at which mold grows as a function of SLF count.
    harvest_penalty_rate (float): Penalty rate for late pesticide application.

    Returns:
    float: Total profit from the vineyard.
    """
    season_length = len(df)
```

```
    pesticide_area = vineyard_length * 4 * depth  # Area of pesticide application

    total_pesticide_cost = 0
    total_mold_impact = 0
    total_harvest_penalty = 0

    for i in range(season_length):
        # Calculate daily pesticide cost
        if df.loc[i, 'pesticide']:
            total_pesticide_cost += pesticide_cost * pesticide_area

        # Calculate mold impact based on SLF count
        total_mold_impact += mold_growth_rate * (df.loc[i, 'nymph_count'] + df.loc[i, 'adult_count'])

        # Calculate harvest penalty for late application
        if df.loc[i, 'application_rate'] > 0 and i > df['harvest_day']:
            total_harvest_penalty += harvest_penalty_rate * (i - df['harvest_day'])

    # Calculate remaining yield
    remaining_yield = max(0, base_yield - total_mold_impact - total_harvest_penalty)

    # Calculate total profit
    total_profit = remaining_yield * price_per_grape - total_pesticide_cost

    return total_profit
```

Part IV: Simulation Summary and Results

Overview

This simulation aimed to optimize pesticide application in a vineyard to control Spotted Lanternfly (SLF) populations while maximizing profit. The simulation was built around a series of Python functions, each addressing a specific aspect of the vineyard ecosystem and SLF life cycle.

Functions Developed

1 SLF Population Model:

Estimates the counts of nymph and adult SLFs based on temperature, precipitation, and initial egg count.

Inputs: Daily average temperatures, daily precipitation levels, initial egg count.

Outputs: Daily counts of nymph and adult SLFs.

2 Pesticide Application Strategy:

Generates a schedule for pesticide application based on SLF population thresholds, temperature, and precipitation.

Inputs: Temperature, precipitation, harvest day, SLF population data.

Outputs: Daily pesticide application schedule.

3 Profit Calculation:

Calculates the final profit considering SLF impact, pesticide cost, and mold growth due to SLFs.

Inputs: Vineyard size, pesticide application depth, daily data on SLF counts, pesticide cost, base yield, price per grape, mold growth rate, harvest penalty rate.

Outputs: Total profit from the vineyard.

Assumptions

1. Optimal conditions for SLF hatching and growth are at 25°C with a specific precipitation level.
2. Pesticide application only occurs within a certain depth from the vineyard's perimeter.
3. Mold growth, which reduces grape yield, is proportional to SLF count.
4. Late pesticide application, leading to delayed harvest, incurs a penalty.

Simulation Results

1. Optimal Depth for Pesticide Application: 20 feet from the border.
2. Pesticide Application Strategy: Apply pesticide when adult SLF count exceeds 10 per vine or nymph count exceeds 20 per vine.
3. Pesticide Selection: Based on efficacy, cost, and impact on harvest, "Drexel Carbaryl 4L" is recommended for its excellent efficacy against both nymphs and adults, moderate REI and PHI, and overall cost-effectiveness.

Part V: Future Outlook

Environmental Impact

While focusing on financial viability and SLF control, it's crucial to consider the ecological impact, especially on beneficial insects. Future iterations of this model should integrate an ecological impact assessment, possibly adjusting pesticide choice or application methods to minimize harm to non-target species.

Refinement of Application Strategy

For larger vineyards, the strategy of covering only the perimeter may not suffice. It's recommended to evaluate the interior sections for potential SLF infestations and adjust the application areas accordingly. This could involve periodic assessments of SLF presence throughout the vineyard and targeted applications in identified hotspots.

Future Considerations

1. Integrating more detailed climatic models for precise predictions of SLF life cycle stages.
2. Exploring alternative pest control methods, such as biological controls, to reduce reliance on chemical pesticides.
3. Implementing a dynamic, adaptive strategy that can respond to real-time data and changing conditions throughout the growing season.

Part VI: Appendix

pesticides.csv

Product,Active Ingredient,Seasonal Max,Max Applications,Class,PHI (days),Effect on Adults,Effect on Nymphs

Brigade,bifenthrin,6.4,9999,Pyrethroid,30,Excellent,Excellent

Sniper,bifenthrin,6.4,9999,Pyrethroid,30,Excellent,Excellent

Mustang Maxx,zeta-cypermethrin,24.0,9999,Pyrethroid,1,Excellent,Good

Baythroid,beta-cyfluthrin,12.8,9999,Pyrethroid,3,Excellent,Excellent

Danitol,fenpropathrin,51.24,9999,Pyrethroid,21,Excellent,Excellent

Actara,thiamethoxam,7,9999,Neonicotinoid,5,Excellent,Excellent

Scorpion,dinotefuran,10.25,9999,Neonicotinoid,1,Excellent,Poor

Venom,dinotefuran,12,9999,Neonicotinoid,1,Excellent,Poor

Admire Pro,imidacloprid,14,9999,Neonicotinoid,30,Variable,Variable

Imidan,phosmet,100,9999,Organophosphate,7,Good,Good

Malathion,malathion,999999,2,Organophosphate,3,Excellent,Excellent

Sevin,carbaryl,320,2,Carbamate,7,Excellent,Excellent

Avant,indoxacarb,11.5,9999,Sodium channel blocker,7,Good,Good

Aza-Direct,neem,99999999,9999,Botanical,0,Good,Good

BoteGHA ES,Beauveria,99999999,9999,Pathogen,0,Not Recommended,Not Recommended

PFR-97,Isaria,,Pathogen,0,,

,,,,,,

https://portal.ct.gov/-/media/CAES/CAPS/CAES-SLF-Management-Vineyards-Factsheet.pdf,,,,,,

https://extension.psu.edu/spotted-lanternfly-management-guide,,,,,,

Seasonal Max are in fl oz