Comparative Analysis of Solar Panel Systems versus Grid Systems for Residential Energy Provision

Shaya Engelman

2024-07-15

## Abstract

This report presents a comprehensive simulation study comparing the cost-effectiveness of solar panel systems and grid-powered systems for residential energy provision over a 20-year period. The simulation accounts for various factors such as financing methods, interest rates, degradation rates, and net metering. The results indicate that solar panel systems are generally more cost-effective than grid-powered systems, even when financed. Tax credits and net metering significantly impact the cost-effectiveness of solar panel systems.

## Introduction

The increasing cost of electricity and the growing concern for environmental sustainability have led many homeowners to consider alternative energy sources such as solar panels. This study aims to compare the long-term cost-effectiveness of solar panel systems versus traditional grid-powered systems for residential energy provision. The simulation spans a 20-year period, incorporating various parameters such as interest rates, system degradation rates, and available tax credits.

## Methodology

### Data Collection

Electricity rates by month were verified using data from the New York State Energy Research and Development Authority (NYSERDA). This source provided reliable and accurate monthly average electricity rates for residential use. The cost per kilowatt (kW) for solar panels was referenced from EnergySage, a platform that offers up-to-date cost estimates for solar panel installations in New York. Sunlight hours by month were obtained from FootprintHero, which provides precise peak sunlight hours for various locations. These sources ensured that the data used in the simulation was both current and accurate.

### Simulation Model

#### Constants for Solar Panel System

The cost per kilowatt for the solar panel system was set at $3,150, and the simulation period was defined as 20 years. The degradation rate of the solar panels was assumed to be 1% per year, reflecting the gradual decrease in efficiency over time. A plot of various degradation rates is included as well.

#### Constants for Grid System

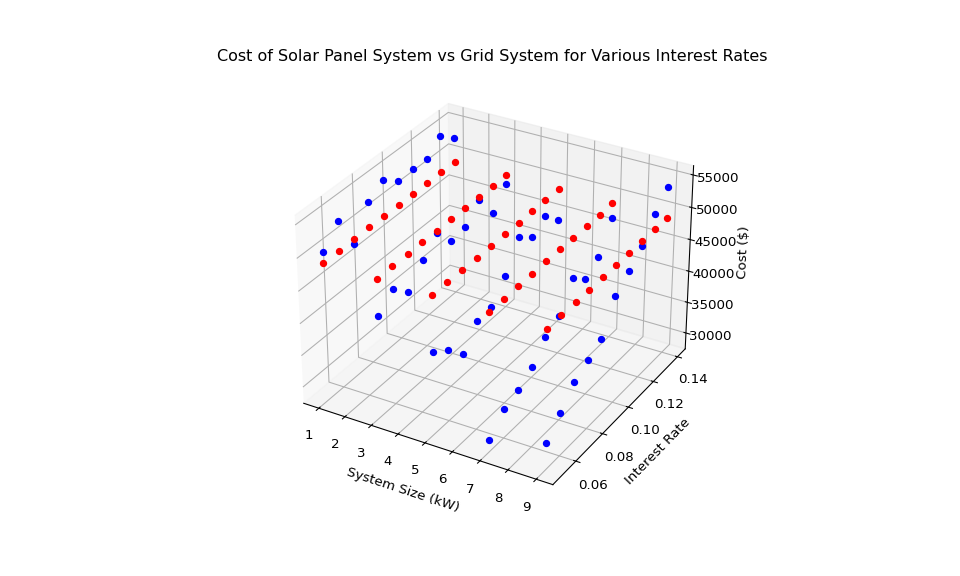
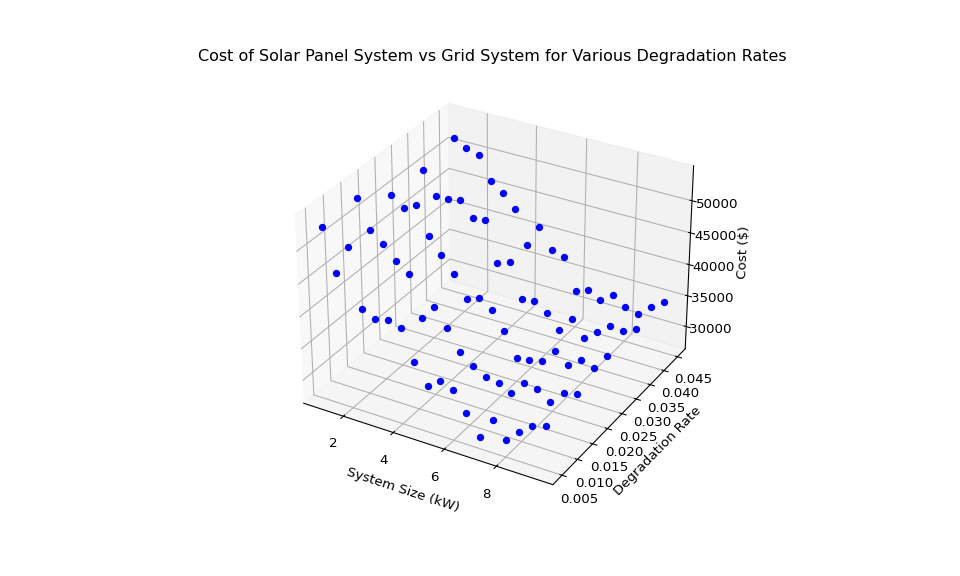
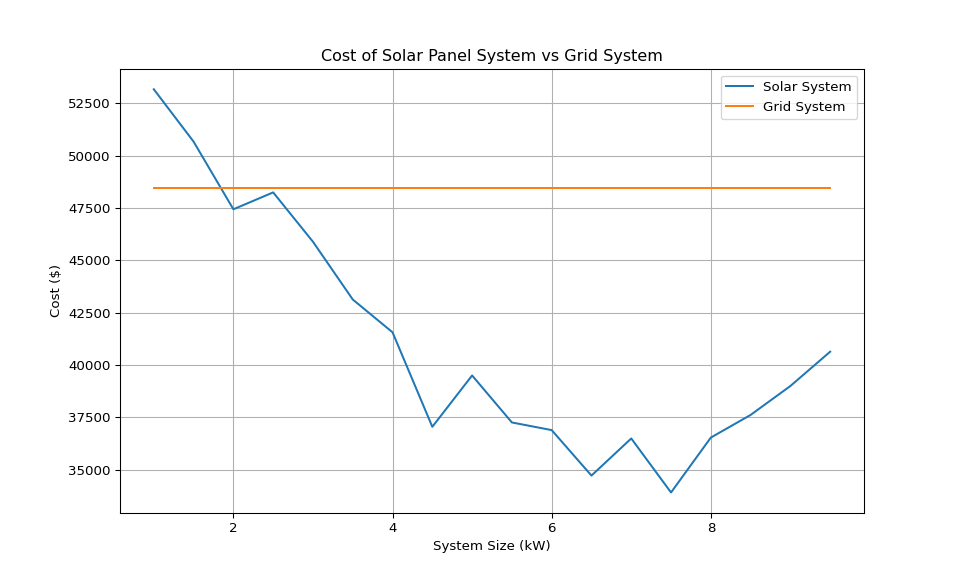
The average monthly electricity usage for a typical household was set at 877 kWh, with a standard deviation of 50 kWh to account for variability. The daily usage standard deviation was set at 100 kWh, reflecting the daily fluctuations in energy consumption.

#### Energy System Classes

The simulation model included two main classes: the EnergySystem class and its subclasses, SolarPanelSystem and GridSystem. The EnergySystem class was responsible for simulating energy usage and costs for different energy systems and storing monthly demand variability and peak demand statistics. The SolarPanelSystem class, inheriting from EnergySystem, calculated initial installation costs, including federal and state tax credits, and simulated monthly energy generation based on sunlight hours and system degradation. It also handled excess energy generation through net metering options, such as payout, reset, or rollover. The GridSystem class, also inheriting from EnergySystem, simulated monthly energy usage costs based on grid rates.

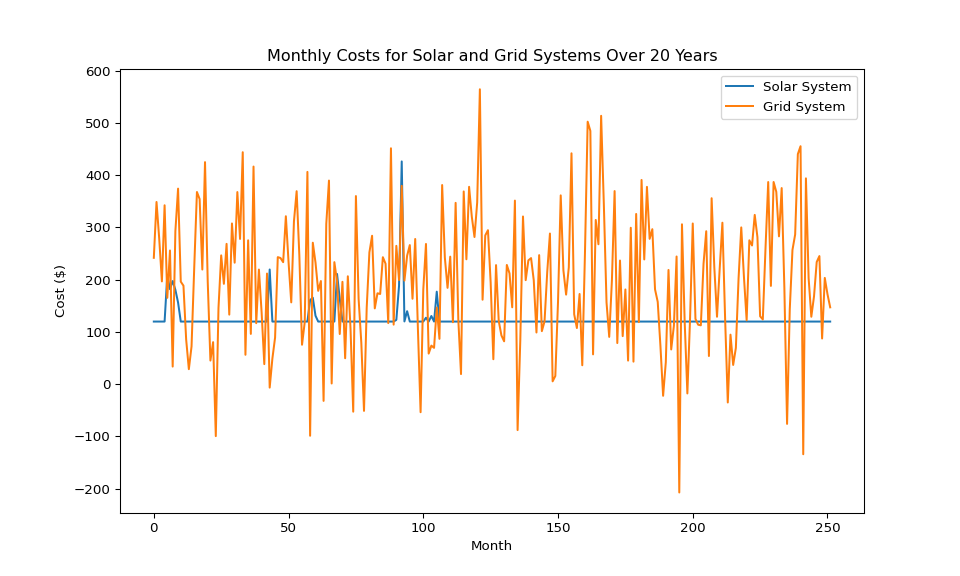
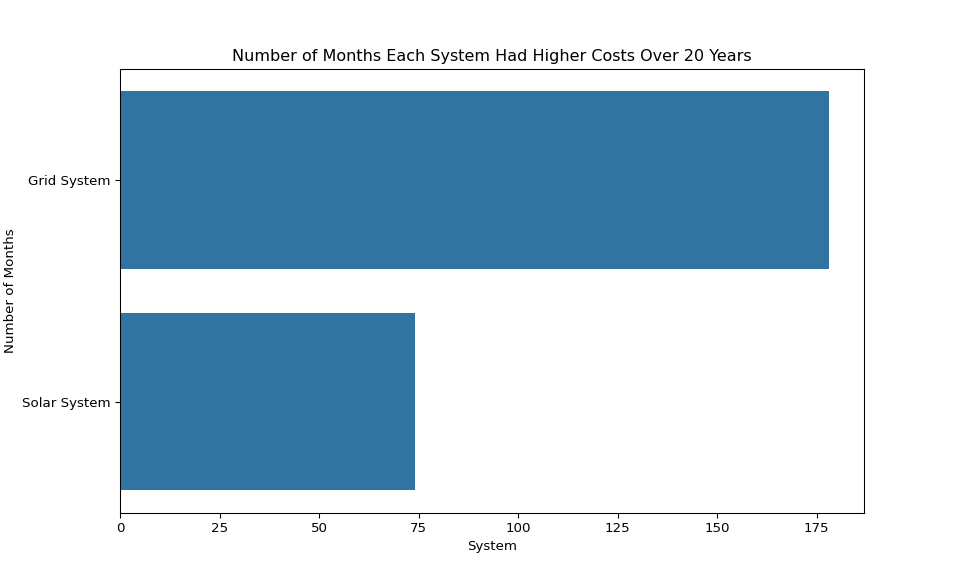
### Parameter Sweep

The simulation included a parameter sweep over system sizes ranging from 1 kW to 10 kW in increments of 0.5 kW. Interest rates varied from 5% to 15% in increments of 1%, and degradation rates ranged from 1% to 10% in increments of 1%. This comprehensive sweep ensured that the study considered a wide range of possible scenarios and their impact on cost-effectiveness.



### Monthly Costs

Monthly costs of a grid system were calculated and compared to the monthly cost of a solar system based on the optimal amount of KWs found in the previous parameter sweep and current interest rates of 9%. It was found that the majority of individual months had a larger cost if using a grid system. However, a not-insignificant amount of months resulted in a solar system having a larger cost. This highlights that while it is almost certainly cheaper to have a solar system in the long run, any given individual monthly cost is no guarantee to be cheaper.

## Validation and Verification

### Code Reviews and Debugging

To ensure the correctness of the implementation, thorough code reviews and debugging sessions were conducted. Peer-reviewed code helped identify and correct potential errors or inefficiencies, enhancing the overall reliability and performance of the simulation model.

### Proprietary Solar Calculator Validation

A proprietary solar calculator from a family member who sells solar systems was used to validate the simulation results. The numbers generated by the simulation matched those from the proprietary calculator, ensuring consistency and accuracy. This external validation added an extra layer of confidence in the simulation’s findings. Similar findings can likely be found by utilizing free quotes offered by solar panel sales companies. They often use such proprietary calculators.

### Data Sources

Data sources were carefully chosen and verified for accuracy. Electricity rates by month were verified from NYSERDA, ensuring reliable monthly average electricity rates for residential use. The cost per kilowatt for solar panels was referenced from EnergySage, which provides up-to-date cost estimates for solar panel installations in New York. Sunlight hours by month were obtained from FootprintHero, which offers precise peak sunlight hours for various locations. These sources ensured the accuracy and reliability of the data used in the simulation.

### Model Verification

The simulation model underwent thorough verification to ensure its accuracy and reliability. Code reviews and debugging were conducted to identify and fix any issues in the code. Unit testing was implemented to test individual functions and ensure they performed as expected.

### Simulation Validation

Simulation validation was conducted through a comparison with historical data and sensitivity analysis. Historical data comparison involved comparing simulation results with historical energy usage and cost data, ensuring the model’s accuracy. Sensitivity analysis assessed the impact of varying key parameters, such as interest rates and degradation rates, on the simulation results. Preliminary results were shared with peers and experts for feedback, further validating the simulation’s accuracy.

## Results

### Cost Comparison

The simulation results indicate that solar panel systems are generally more cost-effective than grid-powered systems over a 20-year period. Even when financing the system, it requires an interest rate of around 15% for the costs to exceed those of a grid-based system. Buying the system outright lowers the cost even more significantly.

Buying a system outright is by far the cheapest long-term option, however, it requires a significant financial outlay that a grid system does not. Financing a system is the most similar pay structure to a grid system and still comes out significantly cheaper in almost all scenarios. As mentioned, individual monthly costs are not guaranteed to be lower, but the total cost of a 20 year system would likely be much lower.

### Impact of Tax Credits and Net Metering

The analysis shows that major tax credits by the federal government and the New York government significantly reduce the initial cost of solar panel systems. Changes in these tax policies could affect the cost-effectiveness of solar systems. Similarly, the simulation would have to be ran again for different municipalities with varying tax credits. In this simulation, the available tax credits lowered the cost of most solar systems by more than half. This cannot be overstated as a driving reason behind the financial benefit of using solar power.

Net metering is a billing arrangement that allows homeowners and businesses that generate their own electricity with their solar system to deliver unused energy back into the grid. This process provides a significant degree of flexibility, as it allows any excess energy produced by the solar system to be credited back to the homeowner’s account.

In essence, when your solar panels produce more electricity than you use in your home, that excess energy is sent back into the power grid. The power utility then credits your account for the excess power contributed. These credits can be used to offset the costs of power drawn from the grid when your system is not producing enough electricity, such as during the night or on heavily overcast days.

Without net metering, managing a solar system becomes more complex and potentially more expensive. If a homeowner were unable to feed excess power back into the grid, they would need to rely solely on the energy produced by their solar panels at the time of production. This would require either a larger array of panels to ensure sufficient power during periods of lower sunlight, or a battery storage system to store excess power for use when needed. Both of these options would substantially increase the initial and maintenance costs of the system.

This simulation assumed net-metering was available, with varying methods of dealing with yearly excess. Somewhere where that isn’t an option, the simulation would have to be run again using the increased costs.

## Conclusion

This study demonstrates that solar panel systems are generally more cost-effective than grid-powered systems for residential energy provision over a 20-year period. The findings underscore the significant role of tax credits and net metering in enhancing the affordability of solar energy. Future work should include continuous validation with new data and scenarios to maintain accuracy and relevance.

## References

1. New York State Energy Research and Development Authority (NYSERDA). (n.d.). [Electricity Prices](https://www.nyserda.ny.gov/Researchers-and-Policymakers/Electricity-Prices).
2. EnergySage. (n.d.). [How much do solar panels cost in New York?](https://www.energysage.com/solar-panels/ny/).
3. FootprintHero. (n.d.). [Solar Insolation Data](https://www.footprinthero.com/).
4. U.S. Department of Energy. (n.d.). [Net Metering](https://www.energy.gov/eere/solar/net-metering).

## Appendix

### Code Implementation

import numpy as np  
import matplotlib.pyplot as plt  
import heapq  
  
# Constants for Solar Panel System  
COST\_PER\_KW = 3150  
NUM\_YEARS = 20  
  
# Constants for Grid System  
sunlight\_hours = {  
 'jan': 2.08, 'feb': 3.06, 'mar': 3.9, 'apr': 5.05,  
 'may': 6.02, 'jun': 6.35, 'jul': 6.34, 'aug': 5.52,  
 'sep': 4.52, 'oct': 3.31, 'nov': 2.27, 'dec': 1.64  
}  
  
days\_in\_month = {  
 'jan': 31, 'feb': 28, 'mar': 31, 'apr': 30,  
 'may': 31, 'jun': 30, 'jul': 31, 'aug': 31,  
 'sep': 30, 'oct': 31, 'nov': 30, 'dec': 31  
}  
  
MEAN\_USAGE = 877  
STD\_DEV\_MONTHLY = 50  
NUM\_MONTHS = 12  
STD\_DEV\_DAILY = 100  
degradation\_rate = 0.01  
  
monthly\_usage = np.random.normal(MEAN\_USAGE, STD\_DEV\_MONTHLY, NUM\_MONTHS)  
monthly\_usage = np.sort(monthly\_usage)[::-1]  
  
months = {  
 'jan': 7, 'feb': 8, 'mar': 10, 'apr': 11,  
 'may': 4, 'jun': 3, 'jul': 1, 'aug': 2,  
 'sep': 5, 'oct': 12, 'nov': 9, 'dec': 6  
}  
  
sorted\_months = sorted(months, key=months.get)  
  
monthly\_usage = dict(zip(sorted\_months, monthly\_usage))  
  
rates = {  
 'jan': 0.2357, 'feb': 0.236, 'mar': 0.2118, 'apr': 0.1981,  
 'may': 0.2047, 'jun': 0.2163, 'jul': 0.2227, 'aug': 0.2233,  
 'sep': 0.2223, 'oct': 0.227, 'nov': 0.2272, 'dec': 0.2252  
}  
  
  
class EnergySystem:  
 def \_\_init\_\_(self, rates, monthly\_usage, std\_dev\_daily):  
 self.rates = rates  
 self.monthly\_usage = monthly\_usage  
 self.std\_dev\_daily = std\_dev\_daily  
 self.cost\_of\_system = 0  
 self.cumulative\_costs = [self.cost\_of\_system]  
 self.monthly\_demand\_variability = {}  
 self.monthly\_peak\_demand = {}  
  
 def process\_event(self, event):  
 month = event['month']  
 self.simulate\_month(month)  
  
 def simulate\_energy\_used(self, month):  
 daily\_usages = np.random.normal(self.monthly\_usage[month] / days\_in\_month[month], self.std\_dev\_daily, days\_in\_month[month])  
 month\_used = np.sum(daily\_usages)  
 return daily\_usages, month\_used  
  
 def simulate\_month(self, month):  
 raise NotImplementedError("Subclasses should implement this!")  
   
 def store\_demand\_stats(self, month, daily\_usages):  
 self.monthly\_demand\_variability[month] = np.std(daily\_usages)  
 self.monthly\_peak\_demand[month] = np.max(daily\_usages)  
  
class SolarPanelSystem(EnergySystem):  
 def \_\_init\_\_(self, kw\_of\_system, payment\_method, annual\_interest\_rate, degradation\_rate):  
 super().\_\_init\_\_(rates, monthly\_usage, STD\_DEV\_DAILY)  
 self.kw\_of\_system = kw\_of\_system  
 self.degradation\_rate = degradation\_rate  
 self.cost\_per\_kw = COST\_PER\_KW  
 self.cost\_of\_installation = self.kw\_of\_system \* self.cost\_per\_kw  
 self.fed\_tax\_credit = 0.30 \* self.cost\_of\_installation  
 self.ny\_tax\_credit = 5000 if self.cost\_of\_installation \* 0.25 > 5000 else self.cost\_of\_installation \* 0.25  
 self.initial\_cost = self.cost\_of\_installation - self.fed\_tax\_credit - self.ny\_tax\_credit  
 self.payment\_method = payment\_method  
 self.coned\_cbc\_charge = 1.09 \* self.kw\_of\_system  
 self.coned\_cust\_charge = 18  
 self.excess = 0  
  
 if self.payment\_method == 'financed':  
 self.monthly\_payment = self.calculate\_monthly\_payment(self.initial\_cost, annual\_interest\_rate, NUM\_YEARS \* 12)  
 self.cost\_of\_system = 0  
 elif self.payment\_method == 'outright':  
 self.cost\_of\_system = self.initial\_cost  
 else:  
 raise ValueError('Invalid payment method. Please choose "financed" or "outright".')  
  
 self.cumulative\_costs = [self.cost\_of\_system]  
  
 def process\_event(self, event):  
 month = event['month']  
 self.simulate\_month(month)  
  
 def simulate(self, num\_years, excess\_handling):  
 # Initialize event calendar with the first month  
 event\_calendar = [(i, {'month': month}) for i, month in enumerate(sorted\_months)]  
 heapq.heapify(event\_calendar) # Turn the list into a heap  
   
 # Event processing loop  
 while event\_calendar:  
 # Get the next event  
 time, event = heapq.heappop(event\_calendar)  
   
 # Process the event  
 self.process\_event(event)  
   
 # Generate new event (next year's month)  
 if time < num\_years \* 12:  
 new\_event = (time + 12, {'month': event['month']})  
 heapq.heappush(event\_calendar, new\_event)  
   
 # Handle yearly excess  
 if event['month'] == 'mar':  
 self.handle\_yearly\_excess(excess\_handling)  
   
 # Collect statistics (e.g., cumulative costs)  
 self.cumulative\_costs.append(self.cost\_of\_system)  
   
 # Return cumulative costs at the end of the simulation  
 return self.cumulative\_costs  
  
 def calculate\_monthly\_payment(self, principal, annual\_interest\_rate, loan\_term\_in\_months):  
 """Calculate the monthly payment for a loan."""  
 monthly\_interest\_rate = annual\_interest\_rate / 12  
 monthly\_payment = principal \* (monthly\_interest\_rate \* (1 + monthly\_interest\_rate)\*\*loan\_term\_in\_months) / ((1 + monthly\_interest\_rate)\*\*loan\_term\_in\_months - 1)  
 return monthly\_payment  
  
 def simulate\_month\_generated(self, month):  
 hours = np.random.normal(sunlight\_hours[month], size=days\_in\_month[month])  
 month\_generated = np.sum(hours) \* self.kw\_of\_system \* (1 - self.degradation\_rate)\*\*(NUM\_YEARS/12)  
 return month\_generated  
   
 def simulate\_month(self, month):  
 month\_generated = self.simulate\_month\_generated(month)  
 daily\_usages, month\_used = self.simulate\_energy\_used(month)  
   
 if month\_generated > month\_used:  
 excess = month\_generated - month\_used  
 self.excess += excess  
 else:  
 deficit = month\_used - month\_generated  
 if self.excess >= deficit:  
 self.excess -= deficit  
 deficit = 0  
 else:  
 deficit -= self.excess  
 self.excess = 0  
 cost = deficit \* rates[month]  
 self.cost\_of\_system += cost  
  
 self.cost\_of\_system += self.coned\_cbc\_charge  
 self.cost\_of\_system += self.coned\_cust\_charge  
  
 if self.payment\_method == 'financed':  
 self.cost\_of\_system += self.monthly\_payment  
  
 # Store variability and peak demand for the month  
 self.store\_demand\_stats(month, daily\_usages)  
  
 def handle\_yearly\_excess(self, excess\_handling):  
 if excess\_handling == 'payout':  
 self.cost\_of\_system -= self.excess \* 0.04  
 self.excess = 0  
 elif excess\_handling == 'reset':  
 self.excess = 0  
 elif excess\_handling == 'rollover':  
 pass  
 else:  
 raise ValueError('Invalid excess handling method. Please choose "payout", "reset", or "rollover".')  
  
class GridSystem(EnergySystem):  
 def \_\_init\_\_(self, rates, monthly\_usage, std\_dev\_daily):  
 super().\_\_init\_\_(rates, monthly\_usage, std\_dev\_daily)  
 self.cost\_of\_system = 0  
 self.cumulative\_costs = [self.cost\_of\_system]  
  
 def process\_event(self, event):  
 month = event['month']  
 self.simulate\_month(month)  
  
 def simulate(self, num\_years):  
 # Initialize event calendar with the first month  
 event\_calendar = [(i, {'month': month}) for i, month in enumerate(sorted\_months)]  
 heapq.heapify(event\_calendar) # Turn the list into a heap  
  
 # Event processing loop  
 while event\_calendar:  
 # Get the next event  
 time, event = heapq.heappop(event\_calendar)  
  
 # Process the event  
 self.process\_event(event)  
  
 # Generate new event (next year's month)  
 if time < num\_years \* 12:  
 new\_event = (time + 12, {'month': event['month']})  
 heapq.heappush(event\_calendar, new\_event)  
  
 # Collect statistics (e.g., cumulative costs)  
 self.cumulative\_costs.append(self.cost\_of\_system)  
  
 # Return cumulative costs at the end of the simulation  
 return self.cumulative\_costs  
  
 def simulate\_month(self, month):  
 daily\_usages, month\_used = self.simulate\_energy\_used(month)  
 cost = month\_used \* self.rates[month]  
 self.cost\_of\_system += cost  
  
 # Store variability and peak demand for the month  
 self.store\_demand\_stats(month, daily\_usages)  
  
# Initialize SolarPanelSystem with parameters  
system = SolarPanelSystem(8, 'financed', 0.15, degradation\_rate)  
grid\_system = GridSystem(rates, monthly\_usage, STD\_DEV\_DAILY)  
  
# Simulate for the given number of years  
cumulative\_solar\_costs = system.simulate(NUM\_YEARS, 'rollover')  
cumulative\_grid\_costs = grid\_system.simulate(NUM\_YEARS)  
  
# Plot the cumulative costs for comparison  
plt.figure(figsize=(8, 6))  
plt.plot(cumulative\_solar\_costs, label='Solar Panel System')  
plt.plot(cumulative\_grid\_costs, label='Grid System')  
plt.xlabel('Year')  
plt.ylabel('Cumulative Total Cost ($)')  
plt.title('Cumulative Total Cost Comparison')  
plt.legend()  
plt.grid(True)  
plt.show()  
  
# Parameter sweep  
system\_sizes = np.arange(1, 10, 0.5) # System sizes to sweep over  
solar\_costs = [] # List to store the final cost of the solar system for each size  
grid\_costs = [] # List to store the final cost of the grid system for each size  
  
# Create a grid system  
grid\_system = GridSystem(rates, monthly\_usage, STD\_DEV\_DAILY)  
grid\_system.simulate(NUM\_YEARS)  
grid\_cost = grid\_system.cumulative\_costs[-1]  
grid\_costs = [grid\_cost] \* len(system\_sizes) # The cost of the grid system is the same for all system sizes  
  
# Sweep over system sizes  
for size in system\_sizes:  
 # Create a solar panel system  
 solar\_system = SolarPanelSystem(size, 'financed', 0.06, degradation\_rate)  
 solar\_system.simulate(NUM\_YEARS, 'payout')  
 solar\_cost = solar\_system.cumulative\_costs[-1]  
 solar\_costs.append(solar\_cost)  
  
# Plot the results  
plt.figure(figsize=(10, 6))  
plt.plot(system\_sizes, solar\_costs, label='Solar System')  
plt.plot(system\_sizes, grid\_costs, label='Grid System')  
plt.xlabel('System Size (kW)')  
plt.ylabel('Cost ($)')  
plt.title('Cost of Solar Panel System vs Grid System')  
plt.legend()  
plt.grid(True)  
plt.show()  
  
# Parameter sweep  
system\_sizes = np.arange(1, 10, 0.5) # System sizes to sweep over  
degradation\_rates = np.arange(0.005, 0.05, 0.01) # Degradation rates to sweep over  
  
# Create a grid system  
grid\_system = GridSystem(rates, monthly\_usage, STD\_DEV\_DAILY)  
grid\_system.simulate(NUM\_YEARS)  
grid\_cost = grid\_system.cumulative\_costs[-1]  
  
# Initialize a 3D plot  
fig = plt.figure(figsize=(10, 6))  
ax = fig.add\_subplot(111, projection='3d')  
  
# Sweep over system sizes and degradation rates  
for i, size in enumerate(system\_sizes):  
 for j, degradation\_rate in enumerate(degradation\_rates):  
 # Create a solar panel system  
 solar\_system = SolarPanelSystem(size, 'financed', 0.06, degradation\_rate)  
 solar\_system.simulate(NUM\_YEARS, 'payout')  
 solar\_cost = solar\_system.cumulative\_costs[-1]  
   
 # Plot the result  
 ax.scatter(size, degradation\_rate, solar\_cost, color='b')  
 #ax.scatter(size, degradation\_rate, grid\_cost, color='r')  
  
# Set labels and title  
ax.set\_xlabel('System Size (kW)')  
ax.set\_ylabel('Degradation Rate')  
ax.set\_zlabel('Cost ($)')  
ax.set\_title('Cost of Solar Panel System vs Grid System for Various Degradation Rates')  
  
# Show the plot  
plt.show()  
  
  
# Parameter sweep  
system\_sizes = np.arange(1, 10, 2) # System sizes to sweep over  
interest\_rates = np.arange(0.05, 0.15, 0.01) # Interest rates to sweep over  
  
# Create a grid system  
grid\_system = GridSystem(rates, monthly\_usage, STD\_DEV\_DAILY)  
grid\_system.simulate(NUM\_YEARS)  
grid\_cost = grid\_system.cumulative\_costs[-1]  
  
# Initialize a 3D plot  
fig = plt.figure(figsize=(10, 6))  
ax = fig.add\_subplot(111, projection='3d')  
  
# Sweep over system sizes and interest rates  
for i, size in enumerate(system\_sizes):  
 for j, interest\_rate in enumerate(interest\_rates):  
 # Create a solar panel system  
 solar\_system = SolarPanelSystem(size, 'financed', interest\_rate, degradation\_rate)  
 solar\_system.simulate(NUM\_YEARS, 'payout')  
 solar\_cost = solar\_system.cumulative\_costs[-1]  
   
 # Plot the result  
 ax.scatter(size, interest\_rate, solar\_cost, color='b')  
 ax.scatter(size, interest\_rate, grid\_cost, color='r')  
  
# Set labels and title  
ax.set\_xlabel('System Size (kW)')  
ax.set\_ylabel('Interest Rate')  
ax.set\_zlabel('Cost ($)')  
ax.set\_title('Cost of Solar Panel System vs Grid System for Various Interest Rates')  
  
# Show the plot  
plt.show()  
  
# Instantiate the systems  
solar\_system = SolarPanelSystem(7, 'financed', 0.09, 0.01)  
grid\_system = GridSystem(rates, monthly\_usage, STD\_DEV\_DAILY)  
  
# Simulate the systems  
solar\_costs = solar\_system.simulate(NUM\_YEARS, 'rollover')  
grid\_costs = grid\_system.simulate(NUM\_YEARS)  
  
# Calculate monthly costs  
solar\_monthly\_costs = [solar\_costs[i] - solar\_costs[i-1] for i in range(1, len(solar\_costs))]  
grid\_monthly\_costs = [grid\_costs[i] - grid\_costs[i-1] for i in range(1, len(grid\_costs))]  
  
# Generate the plot  
plt.figure(figsize=(10, 6))  
plt.plot(range((NUM\_YEARS+1)\*12), solar\_monthly\_costs, label='Solar System')  
plt.plot(range((NUM\_YEARS+1)\*12), grid\_monthly\_costs, label='Grid System')  
plt.xlabel('Month')  
plt.ylabel('Cost ($)')  
plt.title('Monthly Costs for Solar and Grid Systems Over 20 Years')  
plt.legend()  
plt.show()  
  
import seaborn as sns  
  
# Determine which system had a higher cost each month  
higher\_cost\_system = ['Solar System' if solar > grid else 'Grid System'   
 for solar, grid in zip(solar\_monthly\_costs, grid\_monthly\_costs)]  
  
# Generate the count plot  
plt.figure(figsize=(10, 6))  
sns.countplot(higher\_cost\_system)  
plt.xlabel('System')  
plt.ylabel('Number of Months')  
plt.title('Number of Months Each System Had Higher Costs Over 20 Years')  
plt.show()