Installation

Run the cell below once at the beginning of each session to install the se-lib library. The `from selib import *` statement will import all its functions and classes to be called with no alias.

```
!pip install -q se-lib

from selib import *
```

HW 6 Problem 1 Simulation

Monte Carlo Analysis for Battle Simulation

ChatGPT

```
from selib import *
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st

number_runs = 1000

# vary manpower_buildup_parameter and estimated_total_effort in model definition

end_times = []  # initialize output data list
remaining_blue = [] # initialize output data list
remaining_red = [] # initialize output data list

for run in range(number_runs):
  # increase end time to allow simulation to run until one of the troops be eliminated
  init_sd_model ( start =0, stop =50 , dt = .2, stop_when="blue <= 0 or red <= 0")
  add_stock ("blue", 1000, outflows=["blue_attrition"])
  # if then else to stop attrition when zero troops and prevent negative troop
  add_flow ("blue_attrition", "if_then_else (blue > 0 , red * red_lethality , 0)")
  # add randomness to blue force lethality in uniform distribution between 0.7 and 0.9 for e
  add_auxiliary ("blue_lethality", "random_uniform(0.7,0.9)" )
  add_stock ("red", 1000, outflows =["red_attrition"])
  # if then else to stop attrition when zero troops and prevent negative troop
  add_flow ("red_attrition", "if_then_else (red > 0 , blue * blue_lethality , 0)")
  # add randomness to red force lethality in uniform distribution between 0.7 and 0.9 for ea
  add_auxiliary ("red_lethality", "random_uniform(0.8,1)" )
  sim_results = run_model (verbose=False)
  end_times.append(get_stop_time())
  remaining_blue.append(sim_results["blue"].iloc[-1])
  remaining_red.append(sim_results["red"].iloc[-1])

def confidence_interval(data, confidence=0.95):
    #Calculates the confidence interval for the mean of a dataset.
    mean = np.mean(data)
    sem = st.sem(data)  # Standard error of the mean
    interval = st.t.interval(confidence, len(data) - 1, loc=mean, scale=sem)
    return interval

# Calculate confidence intervals
end_time_ci = confidence_interval(end_times)
remaining_blue_ci = confidence_interval(remaining_blue)
remaining_red_ci = confidence_interval(remaining_red)

#     Output
# --- Output ---
print('Battle Result:')
for i, time in enumerate(end_times):
    print(f'  Run {i+1}: end {time} Remaining Blue Units {remaining_blue[i]:.2f} Remaining R

print("\n--- Confidence Intervals (95%) ---")
print(f"Mean Battle Duration: {np.mean(end_times):.2f}, CI: ({end_time_ci[0]:.2f}, {end_time
print(f"Mean Remaining Blue Units: {np.mean(remaining_blue):.2f}, CI: ({remaining_blue_ci[0]
print(f"Mean Remaining Red Units: {np.mean(remaining_red):.2f}, CI: ({remaining_red_ci[0]:.2

# --- Plotting ---
# End Times Histogram
fig, axis = plt.subplots()
axis.hist(end_times)
axis.set(title='End Times PDF', xlabel='End Time', ylabel='Frequency')
plt.show()  # Display the figure


# Remaining Blue Histogram
fig, axis = plt.subplots()
axis.hist(remaining_blue)
axis.set(title='Remaining Blue PDF', xlabel='Blue Force Remain', ylabel='Frequency')
plt.show()  # Display the figure


# Remaining Red Histogram
fig, axis = plt.subplots()
axis.hist(remaining_red)
axis.set(title='Remaining Red PDF', xlabel='Red Force Remain', ylabel='Frequency')
plt.show()  # Display the figure


# --- Model Diagram ---
draw_model_diagram()
```

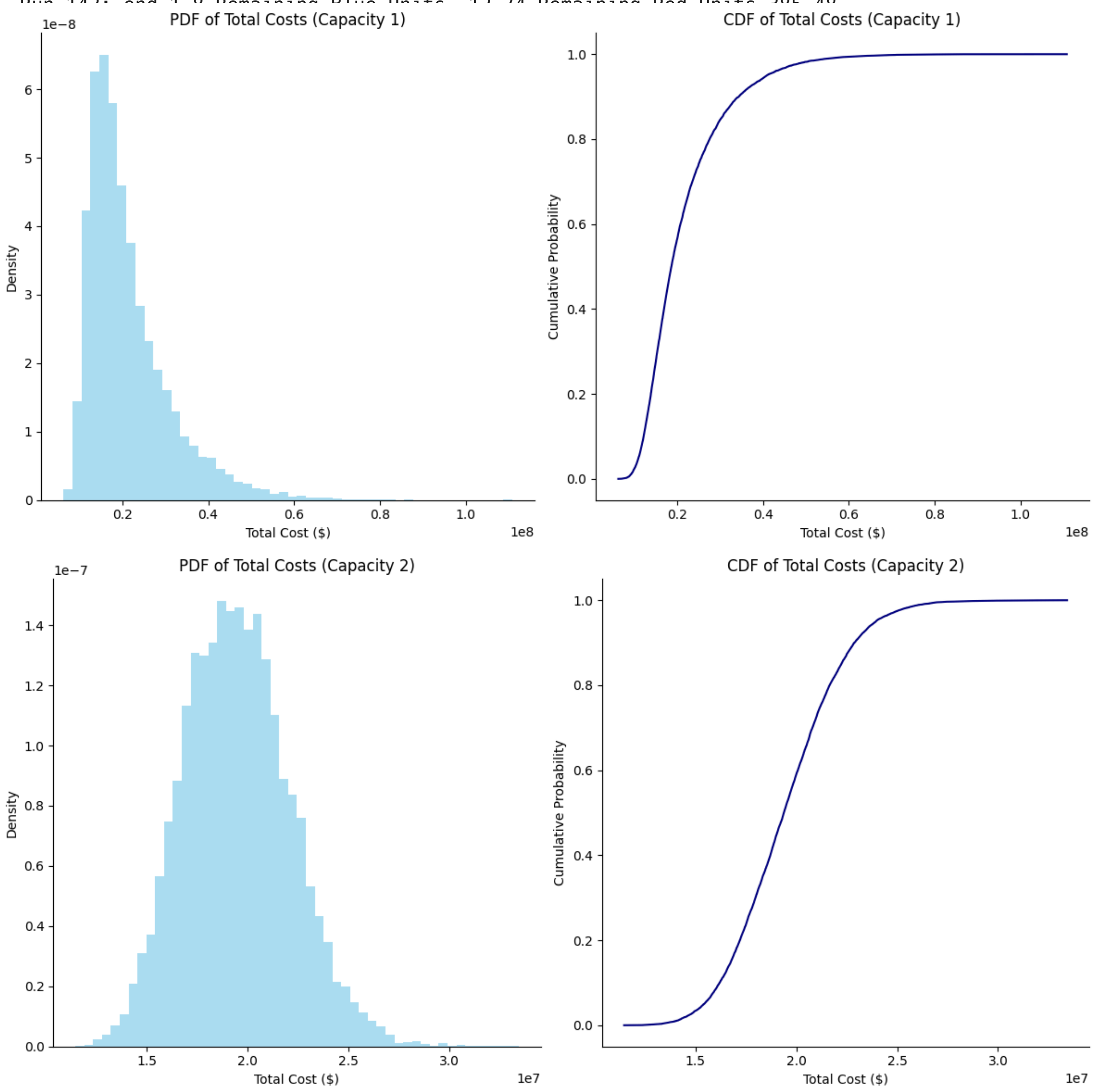
```
Battle Result:
  Run 1: end 2.0 Remaining Blue Units -16.60 Remaining Red Units 324.51
  Run 2: end 2.2 Remaining Blue Units -38.09 Remaining Red Units 295.91
  Run 3: end 2.2 Remaining Blue Units -49.52 Remaining Red Units 288.07
  Run 4: end 2.0 Remaining Blue Units -51.52 Remaining Red Units 335.77
  Run 5: end 2.4 Remaining Blue Units -16.85 Remaining Red Units 218.11
  Run 6: end 2.0 Remaining Blue Units -27.60 Remaining Red Units 328.80
  Run 7: end 2.2 Remaining Blue Units -28.67 Remaining Red Units 294.79
  Run 8: end 2.0 Remaining Blue Units -53.41 Remaining Red Units 350.83
  Run 9: end 2.2 Remaining Blue Units -2.32 Remaining Red Units 268.88
  Run 10: end 2.8 Remaining Blue Units -3.02 Remaining Red Units 148.72
  Run 11: end 2.0 Remaining Blue Units -55.15 Remaining Red Units 345.77
  Run 12: end 2.4 Remaining Blue Units -15.11 Remaining Red Units 231.73
  Run 13: end 2.2 Remaining Blue Units -46.96 Remaining Red Units 299.73
  Run 14: end 2.2 Remaining Blue Units -3.98 Remaining Red Units 255.52
  Run 15: end 2.0 Remaining Blue Units -48.87 Remaining Red Units 341.18
  Run 16: end 1.8 Remaining Blue Units -7.57 Remaining Red Units 368.22
  Run 17: end 2.0 Remaining Blue Units -31.95 Remaining Red Units 329.95
  Run 18: end 2.2 Remaining Blue Units -11.46 Remaining Red Units 248.32
  Run 19: end 2.4 Remaining Blue Units -28.06 Remaining Red Units 246.07
  Run 20: end 2.2 Remaining Blue Units -14.70 Remaining Red Units 267.00
  Run 21: end 2.4 Remaining Blue Units -22.02 Remaining Red Units 240.74
  Run 22: end 2.4 Remaining Blue Units -4.93 Remaining Red Units 214.81
  Run 23: end 2.0 Remaining Blue Units -23.45 Remaining Red Units 324.32
  Run 24: end 2.4 Remaining Blue Units -9.47 Remaining Red Units 207.17
  Run 25: end 2.6 Remaining Blue Units -5.18 Remaining Red Units 177.68
  Run 26: end 2.2 Remaining Blue Units -39.68 Remaining Red Units 292.82
  Run 27: end 2.0 Remaining Blue Units -31.40 Remaining Red Units 324.22
  Run 28: end 2.4 Remaining Blue Units -28.46 Remaining Red Units 232.45
  Run 29: end 2.0 Remaining Blue Units -61.53 Remaining Red Units 361.26
  Run 30: end 2.2 Remaining Blue Units -0.57 Remaining Red Units 240.49
  Run 31: end 2.0 Remaining Blue Units -26.11 Remaining Red Units 334.99
```

HW 6 Problem 2 Simulation

a. 1 & 2 Maintenance Facility Simulation

```python
from selib import *
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st

#Monte Carlo Runs
number_runs = 10000

#Cost
cost_per_day = 250000
cost_per_wait_day = 500000

def run_simulation(capacity):
    end_times = []
    all_wait_times = []
    all_service_times = []
    total_costs = []

    for run in range(number_runs):
        init_de_model()

        add_source(name='incoming_ships',
                   entity_name="Ship",
                   num_entities=20,
                   connections={'maintenance facility': .5, 'no maintenance': .5},
                   interarrival_time='1')

        add_server(name='maintenance facility',
                   connections={'serviced ships': 1},
                   service_time='np.random.uniform(0.5, 3)',
                   capacity=capacity)

        add_terminate('serviced ships')
        add_terminate('no maintenance')

        model_data, entity_data = run_model(verbose=False)

        wait_times = model_data['maintenance facility']['waiting_times']
        service_times = model_data['maintenance facility']['service_times']

        end_time = max(entity['departure'] for entity in entity_data.values())

        all_wait_times.extend(wait_times)
        all_service_times.extend(service_times)
        end_times.append(end_time)

        # Calculate Total Cost
        total_cost = cost_per_day * capacity * end_time + cost_per_wait_day * (sum(wait_tim
        total_costs.append(total_cost)

    return total_costs


total_costs_capacity_1 = run_simulation(capacity=1)
total_costs_capacity_2 = run_simulation(capacity=2)



def plot_pdf_cdf(data, title):
    # PDF
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.hist(data, bins=50, density=True, alpha=0.7, color='skyblue')
    plt.title(f'PDF of {title}')
    plt.xlabel('Total Cost ($)')
    plt.ylabel('Density')

    # CDF
    plt.subplot(1, 2, 2)
    sorted_data = np.sort(data)
    yvals = np.arange(len(sorted_data)) / float(len(sorted_data) - 1)
    plt.plot(sorted_data, yvals, color='navy')
    plt.title(f'CDF of {title}')
    plt.xlabel('Total Cost ($)')
    plt.ylabel('Cumulative Probability')

    plt.tight_layout()
    plt.show()
#PDF & CDF plot for 1 & 2 maintenance facility
plot_pdf_cdf(total_costs_capacity_1, 'Total Costs (Capacity 1)')
plot_pdf_cdf(total_costs_capacity_2, 'Total Costs (Capacity 2)')
```

```
Run 122: end 2.2 Remaining Blue Units -49.04 Remaining Red Units 304.18
Run 123: end 2.2 Remaining Blue Units -12.79 Remaining Red Units 259.96
Run 124: end 2.2 Remaining Blue Units -16.07 Remaining Red Units 259.64
Run 125: end 2.2 Remaining Blue Units -24.43 Remaining Red Units 271.43
Run 126: end 2.2 Remaining Blue Units -41.28 Remaining Red Units 302.00
Run 127: end 2.2 Remaining Blue Units -22.72 Remaining Red Units 269.22
Run 128: end 2.0 Remaining Blue Units -13.57 Remaining Red Units 330.32
Run 129: end 2.0 Remaining Blue Units -32.85 Remaining Red Units 329.82
Run 130: end 3.0 Remaining Blue Units -12.93 Remaining Red Units 131.90
Run 131: end 2.6 Remaining Blue Units -19.22 Remaining Red Units 194.42
Run 132: end 2.2 Remaining Blue Units -38.34 Remaining Red Units 284.36
Run 133: end 2.0 Remaining Blue Units -46.68 Remaining Red Units 333.09
Run 134: end 2.0 Remaining Blue Units -56.60 Remaining Red Units 334.61
Run 135: end 1.8 Remaining Blue Units -16.04 Remaining Red Units 379.57
Run 136: end 2.2 Remaining Blue Units -4.86 Remaining Red Units 262.72
Run 137: end 2.0 Remaining Blue Units -7.96 Remaining Red Units 316.68
Run 138: end 2.0 Remaining Blue Units -36.56 Remaining Red Units 319.38
Run 139: end 2.2 Remaining Blue Units -17.71 Remaining Red Units 276.61
Run 140: end 2.0 Remaining Blue Units -49.98 Remaining Red Units 346.73
Run 141: end 2.0 Remaining Blue Units -3.52 Remaining Red Units 300.99
Run 142: end 1.8 Remaining Blue Units -12.74 Remaining Red Units 385.48
```



```
Run 188: end 2.2 Remaining Blue Units -34.33 Remaining Red Units 279.78
Run 189: end 2.2 Remaining Blue Units -5.82 Remaining Red Units 244.51
Run 190: end 2.4 Remaining Blue Units -15.73 Remaining Red Units 234.02
Run 191: end 2.2 Remaining Blue Units -3.54 Remaining Red Units 246.85
Run 192: end 2.8 Remaining Blue Units -10.67 Remaining Red Units 148.91
Run 193: end 2.2 Remaining Blue Units -18.29 Remaining Red Units 273.32
Run 194: end 2.0 Remaining Blue Units -34.36 Remaining Red Units 328.72
Run 195: end 2.2 Remaining Blue Units -0.93 Remaining Red Units 245.18
Run 196: end 2.4 Remaining Blue Units -28.71 Remaining Red Units 228.36
Run 197: end 2.0 Remaining Blue Units -35.25 Remaining Red Units 345.72
Run 198: end 2.2 Remaining Blue Units -39.08 Remaining Red Units 278.25
Run 199: end 2.0 Remaining Blue Units -8.53 Remaining Red Units 318.85
Run 200: end 2.2 Remaining Blue Units -45.26 Remaining Red Units 294.42
Run 201: end 2.0 Remaining Blue Units -4.61 Remaining Red Units 329.72
Run 202: end 2.0 Remaining Blue Units -52.67 Remaining Red Units 345.08
```

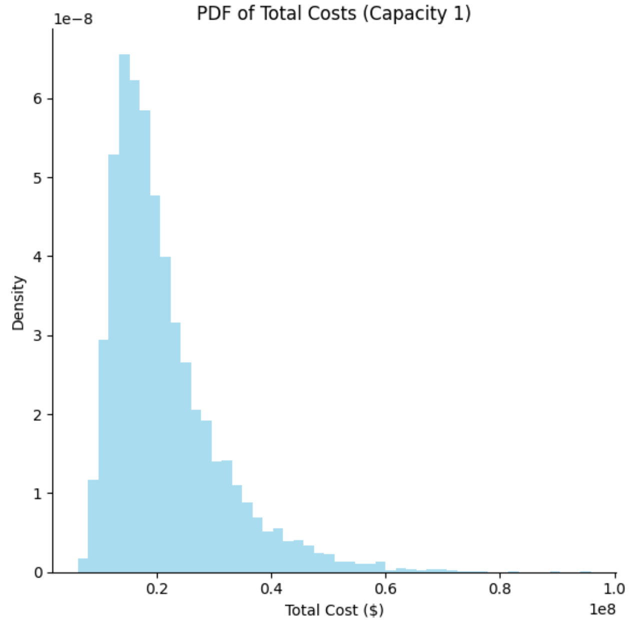
b. 1 & 2 Maintenance Facility Comparison

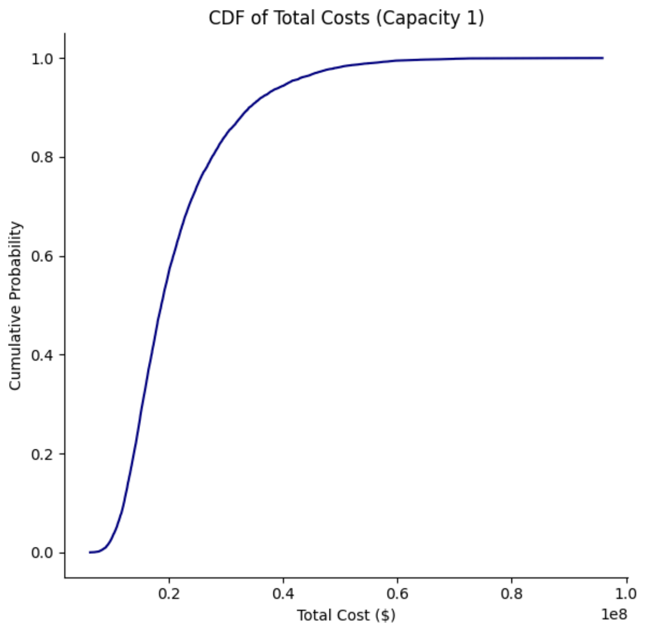
```python
from selib import *
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st

# --- Simulation Parameters ---
number_runs = 10000

```

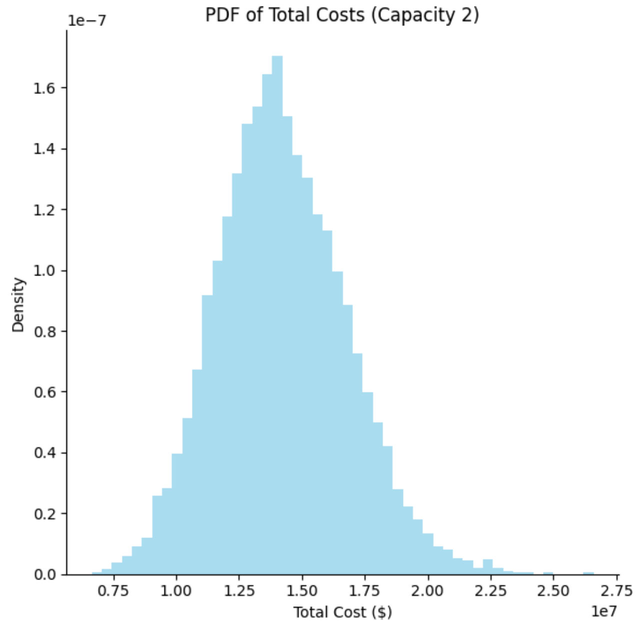
```python
# --- Simulation End Time ---
simulation_end_time = 20  # days

# --- Cost Parameters ---
cost_per_day = 250000
cost_per_wait_day = 500000

# --- Confidence Level ---
confidence_level = 0.90
alpha = 1 - confidence_level

# --- Function to Run Simulation for a Given Capacity ---
def run_simulation(capacity):
    end_times = []
    all_wait_times = []
    all_service_times = []
    total_costs = []

    for run in range(number_runs):
        init_de_model()

        add_source(name='incoming_ships',
                   entity_name="Ship",
                   num_entities=20,
                   connections={'maintenance facility': .5, 'no maintenance': .5},
                   interarrival_time='1')

        add_server(name='maintenance facility',
                   connections={'serviced ships': 1},
                   service_time='np.random.uniform(0.5, 3)',
                   capacity=capacity)

        add_terminate('serviced ships')
        add_terminate('no maintenance')

        model_data, entity_data = run_model(verbose=False)

        wait_times = model_data['maintenance facility']['waiting_times']
        service_times = model_data['maintenance facility']['service_times']

        end_time = max(entity['departure'] for entity in entity_data.values())

        all_wait_times.extend(wait_times)
        all_service_times.extend(service_times)
        if end_time <= simulation_end_time:
          end_times.append(end_time)

        # Calculate Total Cost
        total_cost = cost_per_day * end_time + cost_per_wait_day * (sum(wait_times) + sum(
        total_costs.append(total_cost)

    return total_costs


# --- Run Simulations for Capacities 1 and 2 ---
total_costs_capacity_1 = run_simulation(capacity=1)
total_costs_capacity_2 = run_simulation(capacity=2)

# --- Plotting Functions ---
def plot_pdf_cdf(data, title):
    # PDF
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.hist(data, bins=50, density=True, alpha=0.7, color='skyblue')
    plt.title(f'PDF of {title}')
    plt.xlabel('Total Cost ($)')
    plt.ylabel('Density')

    # CDF
    plt.subplot(1, 2, 2)
    sorted_data = np.sort(data)
    yvals = np.arange(len(sorted_data)) / float(len(sorted_data) - 1)
    plt.plot(sorted_data, yvals, color='navy')
    plt.title(f'CDF of {title}')
    plt.xlabel('Total Cost ($)')
    plt.ylabel('Cumulative Probability')

    plt.tight_layout()
    plt.show()

# --- Generate and Save Plots ---
plot_pdf_cdf(total_costs_capacity_1, 'Total Costs (Capacity 1)')
plot_pdf_cdf(total_costs_capacity_2, 'Total Costs (Capacity 2)')

# --- Hypothesis Testing ---

# Calculate sample statistics
mean_1 = np.mean(total_costs_capacity_1)
mean_2 = np.mean(total_costs_capacity_2)
std_1 = np.std(total_costs_capacity_1)
std_2 = np.std(total_costs_capacity_2)
n_1 = len(total_costs_capacity_1)
n_2 = len(total_costs_capacity_2)

# Calculate the pooled standard error
pooled_standard_error = np.sqrt((std_1**2 / n_1) + (std_2**2 / n_2))

# Calculate the Z test statistic
z_statistic = (mean_1 - mean_2) / pooled_standard_error

# Calculate the critical Z value for a two-tailed test
z_critical = st.norm.ppf(1 - alpha / 2)

# Calculate the confidence interval
margin_of_error = z_critical * pooled_standard_error
confidence_interval = (mean_1 - mean_2 - margin_of_error, mean_1 - mean_2 + margin_of_erro

# Perform the hypothesis test
p_value = 2 * (1 - st.norm.cdf(abs(z_statistic)))  # Two-tailed test

# Output the results
print("\n--- Hypothesis Testing ---")
print(f"90% Confidence Interval for the difference between means: ({confidence_interval[0]
print(f"Z Statistic: {z_statistic:.2f}")
print(f"P-value: {p_value}")

# Determine if the null hypothesis is rejected
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference between the means
else:
    print("Fail to reject the null hypothesis: There is no significant difference between
```

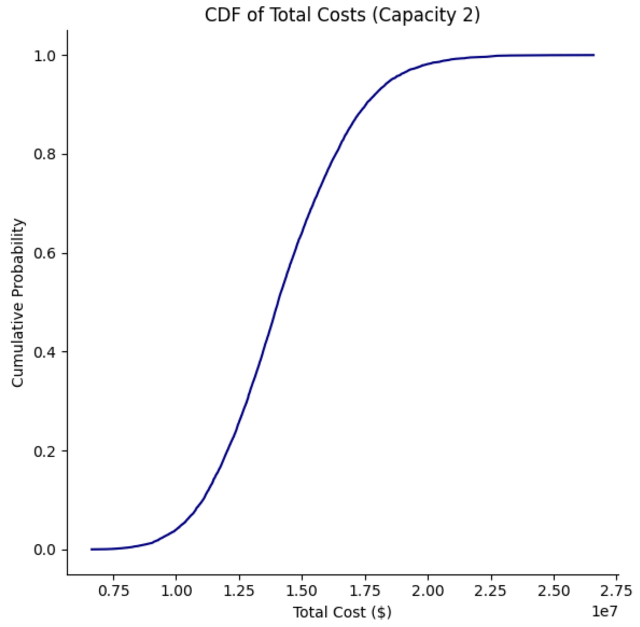
```
Run 346: end 2.8 Remaining Blue Units -22.06 Remaining Red Units 166.84
Run 347: end 2.0 Remaining Blue Units -47.63 Remaining Red Units 323.06
Run 348: end 2.6 Remaining Blue Units -19.24 Remaining Red Units 185.82
Run 349: end 2.2 Remaining Blue Units -39.82 Remaining Red Units 286.40
Run 350: end 2.2 Remaining Blue Units -39.36 Remaining Red Units 269.02
Run 351: end 2.0 Remaining Blue Units -40.04 Remaining Red Units 339.11
Run 352: end 2.4 Remaining Blue Units -12.95 Remaining Red Units 216.00
Run 353: end 1.8 Remaining Blue Units -23.50 Remaining Red Units 379.43
Run 354: end 2.6 Remaining Blue Units -35.53 Remaining Red Units 203.52
Run 355: end 1.8 Remaining Blue Units -16.85 Remaining Red Units 378.09
```

```
  Run 399: end 1.8 Remaining Blue Units -10.85 Remaining Red Units 378.09
```









```
  Run 402: end 2.4 Remaining Blue Units −38.97 Remaining Red Units 251.73
  Run 403: end 2.2 Remaining Blue Units −11.11 Remaining Red Units 266.19
  Run 404: end 2.4 Remaining Blue Units −41.93 Remaining Red Units 256.46
  Run 405: end 2.2 Remaining Blue Units −29.98 Remaining Red Units 267.48
  Run 406: end 2.6 Remaining Blue Units −30.49 Remaining Red Units 198.05
  Run 407: end 2.6 Remaining Blue Units −22.44 Remaining Red Units 186.17
  Run 408: end 2.2 Remaining Blue Units −52.38 Remaining Red Units 296.67
  Run 409: end 2.2 Remaining Blue Units −12.30 Remaining Red Units 264.70
  Run 410: end 2.0 Remaining Blue Units −41.33 Remaining Red Units 351.39
  Run 411: end 2.2 Remaining Blue Units −49.29 Remaining Red Units 294.15
  Run 412: end 2.2 Remaining Blue Units −1.09 Remaining Red Units 244.09
  Run 413: end 2.2 Remaining Blue Units −36.50 Remaining Red Units 276.66
  Run 414: end 2.8 Remaining Blue Units −15.67 Remaining Red Units 158.90
  Run 415: end 2.2 Remaining Blue Units −5.04 Remaining Red Units 242.34
  Run 416: end 2.0 Remaining Blue Units −3.96 Remaining Red Units 302.77
```

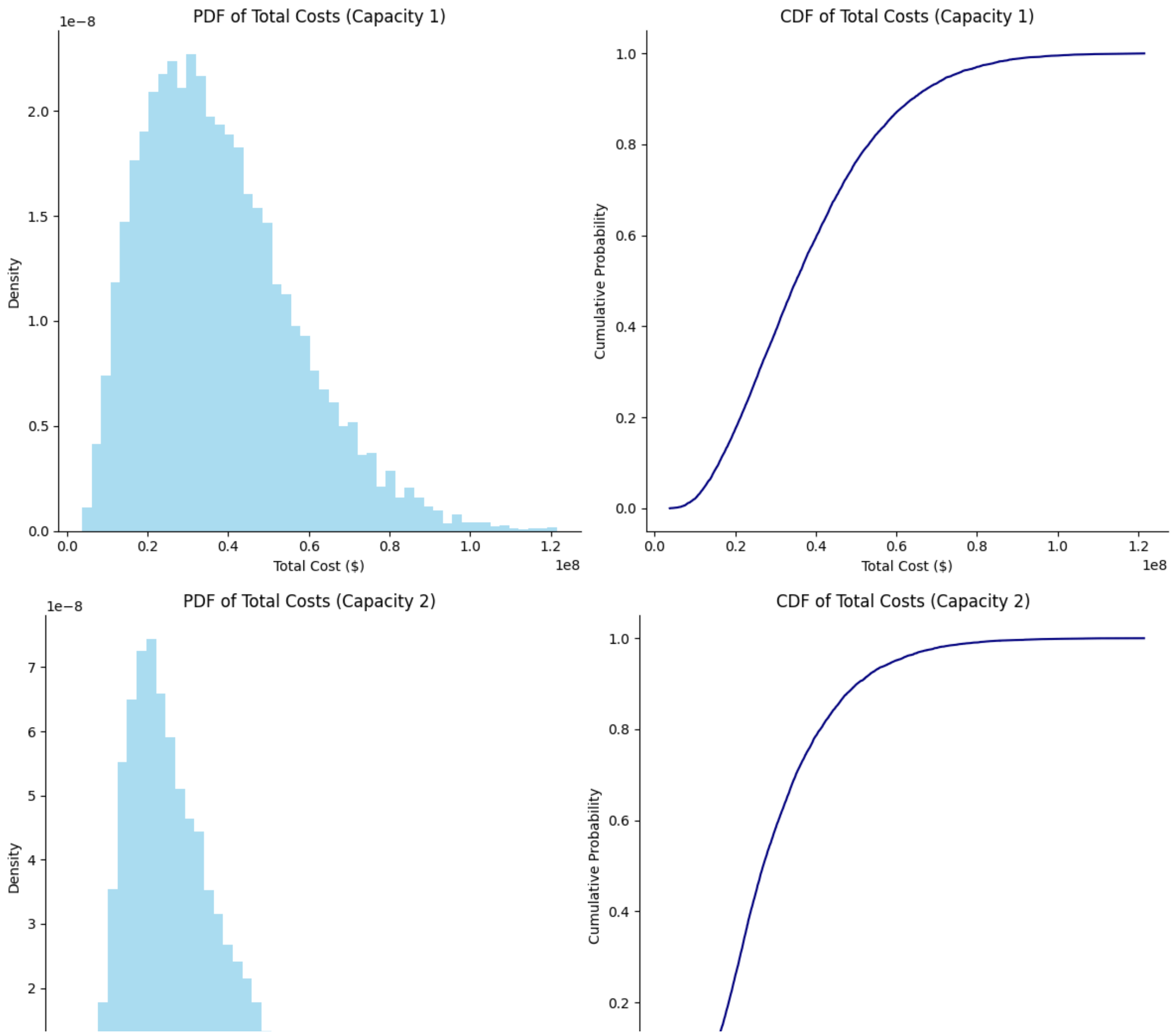
```
--- Hypothesis Testing ---
90% Confidence Interval for the difference between means: (7027220.82, 7957887.39)
Z Statistic: 71.56
P-value: 0.0
Reject the null hypothesis: There is a significant difference between the means.
```

c. 1 & 2 Maintenance Facility Comparison with Exponential arrival time

```
from selib import *
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st

# --- Simulation Parameters ---
number_runs = 10000

```

```
# --- Simulation End Time ---
simulation_end_time = 20  # days

# --- Cost Parameters ---
cost_per_day = 250000
cost_per_wait_day = 500000

# --- Confidence Level ---
confidence_level = 0.90
alpha = 1 - confidence_level

# --- Function to Run Simulation for a Given Capacity ---
def run_simulation(capacity):
    end_times = []
    all_wait_times = []
    all_service_times = []
    total_costs = []

    for run in range(number_runs):
        init_de_model()

        add_source(name='incoming_ships',
                   entity_name="Ship",
                   num_entities=20,
                   connections={'maintenance facility': .5, 'no maintenance': .5},
                   interarrival_time='np.random.exponential(scale=10/24)')

        add_server(name='maintenance facility',
                   connections={'serviced ships': 1},
                   service_time='np.random.uniform(0.5, 3)',
                   capacity=capacity)

        add_terminate('serviced ships')
        add_terminate('no maintenance')

        model_data, entity_data = run_model(verbose=False)

        wait_times = model_data['maintenance facility']['waiting_times']
        service_times = model_data['maintenance facility']['service_times']

        end_time = max(entity['departure'] for entity in entity_data.values())

        all_wait_times.extend(wait_times)
        all_service_times.extend(service_times)
        if end_time <= simulation_end_time:
          end_times.append(end_time)

        # Calculate Total Cost
        total_cost = cost_per_day * end_time + cost_per_wait_day * (sum(wait_times) + sum(
        total_costs.append(total_cost)

    return total_costs


# --- Run Simulations for Capacities 1 and 2 ---
total_costs_capacity_1 = run_simulation(capacity=1)
total_costs_capacity_2 = run_simulation(capacity=2)

# --- Plotting Functions ---
def plot_pdf_cdf(data, title):
    # PDF
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.hist(data, bins=50, density=True, alpha=0.7, color='skyblue')
    plt.title(f'PDF of {title}')
    plt.xlabel('Total Cost ($)')
    plt.ylabel('Density')

    # CDF
    plt.subplot(1, 2, 2)
    sorted_data = np.sort(data)
    yvals = np.arange(len(sorted_data)) / float(len(sorted_data) - 1)
    plt.plot(sorted_data, yvals, color='navy')
    plt.title(f'CDF of {title}')
    plt.xlabel('Total Cost ($)')
    plt.ylabel('Cumulative Probability')

    plt.tight_layout()
    plt.show()

# --- Generate and Save Plots ---
plot_pdf_cdf(total_costs_capacity_1, 'Total Costs (Capacity 1)')
plot_pdf_cdf(total_costs_capacity_2, 'Total Costs (Capacity 2)')

# --- Hypothesis Testing ---

# Calculate sample statistics
mean_1 = np.mean(total_costs_capacity_1)
mean_2 = np.mean(total_costs_capacity_2)
std_1 = np.std(total_costs_capacity_1)
std_2 = np.std(total_costs_capacity_2)
n_1 = len(total_costs_capacity_1)
n_2 = len(total_costs_capacity_2)

# Calculate the pooled standard error
pooled_standard_error = np.sqrt((std_1**2 / n_1) + (std_2**2 / n_2))

# Calculate the Z test statistic
z_statistic = (mean_1 - mean_2) / pooled_standard_error

# Calculate the critical Z value for a two-tailed test
z_critical = st.norm.ppf(1 - alpha / 2)

# Calculate the confidence interval
margin_of_error = z_critical * pooled_standard_error
confidence_interval = (mean_1 - mean_2 - margin_of_error, mean_1 - mean_2 + margin_of_errc

# Perform the hypothesis test
p_value = 2 * (1 - st.norm.cdf(abs(z_statistic)))  # Two-tailed test

# Output the results
print("\n--- Hypothesis Testing ---")
print(f"90% Confidence Interval for the difference between means: ({confidence_interval[0]
print(f"Z Statistic: {z_statistic:.2f}")
print(f"P-value: {p_value}")

# Determine if the null hypothesis is rejected
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference between the means
else:
    print("Fail to reject the null hypothesis: There is no significant difference between
```

```
Run 560: end 2.0 Remaining Blue Units -41.04 Remaining Red Units 324.57
Run 561: end 2.0 Remaining Blue Units -46.91 Remaining Red Units 359.82
Run 562: end 2.2 Remaining Blue Units -36.60 Remaining Red Units 289.67
Run 563: end 2.2 Remaining Blue Units -10.64 Remaining Red Units 252.33
Run 564: end 2.2 Remaining Blue Units -25.21 Remaining Red Units 277.40
Run 565: end 2.0 Remaining Blue Units -41.24 Remaining Red Units 344.16
Run 566: end 3.0 Remaining Blue Units -20.96 Remaining Red Units 142.21
Run 567: end 2.0 Remaining Blue Units -35.81 Remaining Red Units 342.56
Run 568: end 2.2 Remaining Blue Units -8.54 Remaining Red Units 261.08
Run 569: end 2.0 Remaining Blue Units -37.97 Remaining Red Units 345.28
```


PDF of Total Costs (Capacity 1)
CDF of Total Costs (Capacity 1)
Density
Cumulative Probability
Total Cost ($)
1e−8
1e8
PDF of Total Costs (Capacity 2)
CDF of Total Costs (Capacity 2)