Homework 6

```
!pip install se-lib --quiet
```

Show hidden output

ChatGPT

Problem 1: Continuous Battle Simulator Using Lanchester's Law for Aimed Fire

```
# @title Problem 1: Continuous Battle Simulator Using Lanchester's Law for Aimed Fire

import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as st

Battle_End_Time_list = []
Y_Troop_list = []
X_Troop_list = []

a_list = []
b_list = []
x_list = []
y_list = []
dx_list = []

# Start Monte Carlo Loop Here
for run in range(1000):

  # troop sizes
  x0, y0 = 1000, 1000

  #simulation time step size
  dt = .05

  # initializations
  time, x, y = 0, x0, y0

  # integration of troop levels using Euler's method
  while x > 0 and y > 0: # run until a troop is eliminated

    # Set a and b from uniform distribution for each pass
    a = np.random.uniform(.7, .9)
    b = np.random.uniform(.8, 1)

    # Save values of a and b to ensure we are getting values as expected
    a_list.append(a)
    b_list.append(b)

    #Set the first value in the list
    x_list.append(x)
    y_list.append(y)
    dx_list.append(time)

    time += dt
    # attrition rates
    dx = -b * y
    dy = -a * x

    Remaining_X_Troops = x
    Remaining_Y_Troops = y

    # troop levels
    x += dx * dt
    y += dy * dt

  Battle_End_Time_list.append(time)
  Y_Troop_list.append(Remaining_Y_Troops)
  X_Troop_list.append(Remaining_X_Troops)

#print(f'Battle End Time = {Battle_End_Time}')
#print(f'Remaining X Troops = {Remaining_X_Troops}')
#print(f'Remaining Y Troops = {Remaining_Y_Troops}')

# End Monte Carlo Loop here


data = Battle_End_Time_list
print(f"Battle End Time mean 95% CI  = ", st.norm.interval(confidence=0.95, loc=np.mean(data), scale=(np.std(data) / np.sqrt(np.s

data = X_Troop_list
print(f"X Troop End Strength mean 95% CI  = ", st.norm.interval(confidence=0.95, loc=np.mean(data), scale=(np.std(data) / np.sqrt

data = Y_Troop_list
```

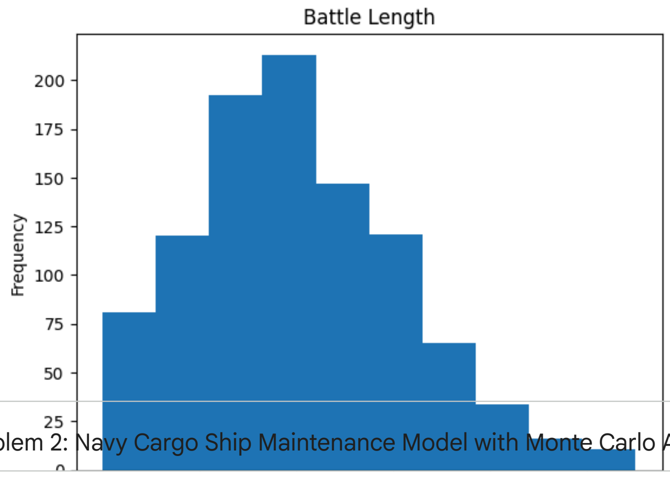
```
print(f"Y Troop End Strength mean 95% CI  = ", st.norm.interval(confidence=0.95, loc=np.mean(data), scale=(np.std(data) / np.sqrt

print(f"Battle End Time= ", Battle_End_Time_list)
print(f"X_Troop_End_Strength= ", X_Troop_list)
print(f"Y_Troop_End_Strength= ", Y_Troop_list)

# PDF
#plt.hist(Battle_End_Time_list)

fig, ax = plt.subplots()
plt.hist(Battle_End_Time_list)
fig
ax.set(title=f'Battle Length',
         ylabel='Frequency',
         xlabel='Battle Length (Time units)',
         )
#
fig, ax = plt.subplots()
plt.hist(X_Troop_list)
fig
ax.set(title=f'X Troop End Strength',
         ylabel='Frequency',
         xlabel='X Troop End Size',
         )
#
fig, ax = plt.subplots()
plt.hist(Y_Troop_list)
fig
ax.set(title=f'Y Troop End Strength',
         ylabel='Frequency',
         xlabel='Y Troop End Size',
         )
```

```
Battle End Time mean 95% CI  =  (np.float64(2.1028351045780314), np.float64(2.115364895421969))
X Troop End Strength mean 95% CI  =  (np.float64(6.970117669046947), np.float64(7.503973466895823))
Y Troop End Strength mean 95% CI  =  (np.float64(318.4332467199466), np.float64(321.79033839032377))
Battle End Time=  [2.1500000000000004, 2.25, 2.0500000000000007, 2.2, 2.000000000000001, 1.900000000000001, 1.950000000000001, 2.25, 2
X_Troop_End_Strength=  [0.28349574158225543, 7.09499939337198, 4.236814744074442, 0.6228322620176581, 0.5662724434665023, 0.8897316375
Y_Troop_End_Strength=  [309.79725390594416, 279.68344192869165, 331.4510570518068, 306.5326948015231, 353.2121024529546, 382.056943443
[Text(0.5, 1.0, 'Y Troop End Strength'),
 Text(0, 0.5, 'Frequency'),
 Text(0.5, 0, 'Y Troop End Size')]
```



Problem 2: Navy Cargo Ship Maintenance Model with Monte Carlo Approach (adapted from HW 4)

```
# @title Problem 2: Navy Cargo Ship Maintenance Model with Monte Carlo Approach (adapted from HW 4)

from selib import *
import matplotlib.pyplot as plt
import random as random



#For Number_Maintenance_Facilities in range(2):
for Number_Facilities in range(2):
  Number_Maintenance_Facilities = Number_Facilities + 1 #since indexing starts at zero

  # initialize an empty list
  Total_Cost_list = []
  Ave_Wait_Time_list = []
  #Number_Maintenance_Facilities = 1
  Number_of_Monte_Carlo_Runs = 10000

  init_de_model()

  add_source('Incoming_Ships',
           entity_name="Ship",
           num_entities = 20,
           connections={'Facility': 0.5, 'Maintenance_Not_Needed': 0.5},
          interarrival_time='np.random.exponential(10)')
          #interarrival_time='np.random.uniform(1, 1)')

  add_server(name='Facility',
           connections={'Served_Ships': 1},
           service_time='np.random.uniform(0.5, 3.0)', #service time is in days
           capacity = Number_Maintenance_Facilities)

  add_terminate('Served_Ships')
  add_terminate('Maintenance_Not_Needed')

  draw_model()

  for run in range(Number_of_Monte_Carlo_Runs):
    model_data, entity_data = run_model()

    Resource_Utilization = model_data['Facility']['resource_utilization']
    Resource_Busy_Time = model_data['Facility']['resource_busy_time']

    Total_Operation_time = Resource_Busy_Time/Resource_Utilization
    Facility_Operational_Cost = 250000 * Total_Operation_time * Number_Maintenance_Facilities  #need to include the number of ma
    Total_service_time = sum(model_data['Facility']['service_times'])
    Total_waiting_time = sum(model_data['Facility']['waiting_times'])
    Total_waiting_and_service_time = Total_service_time +Total_waiting_time
    Opportunity_Cost = 500000 * Total_waiting_and_service_time
    Total_Cost = Facility_Operational_Cost + Opportunity_Cost
    Number_of_Ships_Served = len(model_data['Facility']['waiting_times'])
    Ave_Waiting_Time = Total_waiting_time/Number_of_Ships_Served

    Total_Cost_list.append(Total_Cost)
    Ave_Wait_Time_list.append(Ave_Waiting_Time)

  # PDF
```

```python
  #plt.hist(Total_Cost_list)

  binNumber = np.linspace(min(Total_Cost_list), max(Total_Cost_list), 11) # create 4 bins each 1 units wide with 8 bin edges
  ## PDF
  fig, axis = plt.subplots()
  axis.hist(Total_Cost_list, color='blue', bins=binNumber)
  axis.set(xlabel = 'Total Cost', ylabel='Frequency', title='PDF of Total Cost')
  # sort data
  sorted_data = np.sort(Total_Cost_list)
  # empirical cdf
  # cumulative count array of evenly spaced values length of the sorted data [0, 1, 2, ...] normalized 0-1
  cum_probabilities = np.arange(len(sorted_data))/float(len(sorted_data)-1)
  # CDF
  fig, axis = plt.subplots()
  axis.set(title=f'CDF of Total Cost',
         xlabel='Total Cost',
         ylabel='Cumulative Probability',
         #xticks=range(10),
         yticks=np.linspace(0, 1.0, 11)
         )
  plt.grid(True)
  axis.plot(sorted_data, cum_probabilities)

  #Calculate the confidence intervals

  if Number_Maintenance_Facilities == 1:
    Sample_mean_Capacity_1 = np.mean(Total_Cost_list)
    StandardDeviation_Capacity_1 = np.std(Total_Cost_list)
    Total_Cost_list_Facility_1 = Total_Cost_list
  if Number_Maintenance_Facilities == 2:
    Sample_mean_Capacity_2 = np.mean(Total_Cost_list)
    StandardDeviation_Capacity_2 = np.std(Total_Cost_list)
    Total_Cost_list_Facility_2 = Total_Cost_list

#end of loop through facility capacity, 1 or 2

SE = np.sqrt(((StandardDeviation_Capacity_2*StandardDeviation_Capacity_2)/len(Total_Cost_list)) + ((StandardDeviation_Capacity_1

#Critical Value
Z_alpha_over_2 = 1.64 # for 90% CI

Lower_Bound = (Sample_mean_Capacity_2 - Sample_mean_Capacity_1) - Z_alpha_over_2 * SE
Upper_Bound =  (Sample_mean_Capacity_2 - Sample_mean_Capacity_1) + Z_alpha_over_2 * SE

print('Sample_mean_Capacity_2 = ', Sample_mean_Capacity_2)
print('Sample_mean_Capacity_1 = ', Sample_mean_Capacity_1)

print('Total_Cost_list_Facility_2 = ', Total_Cost_list_Facility_2)
print('Total_Cost_list_Facility_1 = ', Total_Cost_list_Facility_1)
print('Confidence Interval = ', Lower_Bound, Upper_Bound)

#Hypothesis Test
# 2-sided test
H0 = "HO: Difference in means = 0 " #no significant difference between means
HA = "HA: Difference in means is significantly different" #ie. difference in means <> 0
alpha = .1 # Significant level(α)
alpha_over_2 = alpha / 2

#Assuming normal distributions
Test_Statistic = (Sample_mean_Capacity_2 - Sample_mean_Capacity_1) / SE

#We will reject the null hypothesis if the test statistic falls within the rejection region.
if(Test_Statistic >  abs(Z_alpha_over_2)):
  print(f"\033[91mReject null hypothesis {H0} and accept {HA}\033[0m\n")
else:
   print(f"Fail to reject null hypothesis {H0}\n")

```