```python
import numpy as np
from simulator import MM1QueueSimulator, confidence_interval
import argparse

#parameters default
arrival_rate = 1.0  #λ
service_rate = 2.0  #μ
simulation_time = 250  #seconds
replications = 5
warmup = simulation_time / 3

parser = argparse.ArgumentParser(description="M/M/1 Queue Simulation")
parser.add_argument("--arrival_rate", type=float, default=arrival_rate, help=f"Arrival rate (λ, default {arrival_rate})")
parser.add_argument("--service_rate", type=float, default=service_rate, help=f"Service rate (μ, default {service_rate})")
parser.add_argument("--simulation_time", type=float, default=simulation_time, help=f"Time of the simulation (default {simulation_time})")
parser.add_argument("--replications", type=int, default=replications, help=f"Number of independent replications we do (default {replications})")
parser.add_argument("--warmup", type=float, default=warmup, help=f"Time of the simulation (default simulation_time / 3)")
args = parser.parse_args()

arrival_rate = args.arrival_rate
service_rate = args.service_rate
if(service_rate <= arrival_rate):
    print("Cannot continue! Must have μ > λ!")
    exit(0)
simulation_time = args.simulation_time
replications = args.replications
warmup = simulation_time / 3
warmup = args.warmup

print(f"You are running the simulator with an arrival_rate of {arrival_rate}, a service_rate of {service_rate} and a simulation_time of {simulation_time} (warm

emp_X = []
emp_Y = []

averages_stratium = []
queue_lengths = {}

strata_times = {}  # key: qlen, value: list of traversal time of those packets arrived when in the system we had 'key' packets
strata_counts = {} # key: qlen, value: number of packets arrived when in the system we had 'key' packets
                   #it contains our ni, for qlen (i) = 0 packets in queue, qlen (i) = 1 packet in queue, ...

for j in range(replications):
    simulator = MM1QueueSimulator(arrival_rate=arrival_rate, service_rate=service_rate, simulation_time=simulation_time)
    simulator.simulate()

    X = simulator.average_time_in_system(warmup)
    Y = simulator.average_packets_in_queue(warmup)
    emp_X.append(X)
    emp_Y.append(Y)

    if warmup != 0.0:
        filtered_data = [el for el in simulator.time_in_system if el[2] >= warmup]
    else:
        filtered_data = [el for el in simulator.time_in_system]

    for time_in_sys, qlen, _ in filtered_data:
        if qlen not in strata_times:
            strata_times[qlen] = []
            strata_counts[qlen] = 0
        strata_times[qlen].append(time_in_sys)
        strata_counts[qlen] += 1


rho = arrival_rate / service_rate
E_Y = (rho ** 2) / (1 - rho) #average packets in queue
print(f"Queue length: empirical is {np.mean(emp_Y)}, theoretical is {E_Y}\n")

print(f"Theoretical {1/(service_rate-arrival_rate)}")

X = np.array(emp_X)
print(f"Naive: {np.mean(X)} (variance: {np.var(X)})")

#CONTROL VARIATES
Y = np.array(emp_Y)

cov_XY = np.cov(X, Y, ddof=1)[0, 1] #it returns [[var(X), cov(X, Y)],[cov(Y, X), var(Y)]] so we select the proper stuff
#ddof = 1 to divide for n-1 instead of n
var_Y = np.var(Y, ddof=1)
c_star = - cov_XY / var_Y

X_cv = X + c_star * (Y - E_Y)
print(f"CV: {np.mean(X_cv)} (variance: {np.var(X_cv)})  [variance reduction: {100 * (np.var(X) - np.var(X_cv))/np.var(X):.2f}%] [c* {c_star}] ")

#POST STRATIUM
total_packets = sum(strata_counts.values())
post_strat_mean = 0
post_strat_var = 0

post_strat_mean_theory = 0
post_strat_var_theory = 0

def k_packets_ahead(k, rho=(arrival_rate / service_rate)):
    return (1-rho) * (rho**k)

for qlen in strata_times:
    ####Empirical pi
    mean_stratum = np.mean(strata_times[qlen])      #Xi bar -> E[X|Y = yi]
    weight = strata_counts[qlen] / total_packets    #pi
    post_strat_mean += mean_stratum * weight

    var_stratum = np.var(strata_times[qlen], ddof=1) if len(strata_times[qlen]) > 1 else 0                          #Var(X|Y=yi)
    post_strat_var += (weight ** 2) * (var_stratum / strata_counts[qlen]) if strata_counts[qlen] > 0 else 0         #pi^2 * Var(X|Y = yi) / ni

    ####Theoretical pi
    pi_theory = k_packets_ahead(qlen)
    post_strat_mean_theory += mean_stratum * pi_theory
    post_strat_var_theory += (pi_theory ** 2) * (var_stratum / strata_counts[qlen]) if strata_counts[qlen] > 0 else 0

    #print(f"[{qlen}] Theory = {pi_theory:.3f}, Empirical = {weight:.3f}") #just to visualize the differences

print(f"Post-stratified estimator using empirical pi: {post_strat_mean} (variance: {post_strat_var}) [variance reduction: {100* (np.var(X) - post_strat_var) / 
print(f"Post-stratified estimator using theoretical pi: {post_strat_mean_theory} (variance: {post_strat_var_theory}) [variance reduction: {100* (np.var(X) - po
```