```python
import numpy as np
import matplotlib.pyplot as plt
from simulator import MM1QueueSimulator, confidence_interval
import argparse
from scipy.interpolate import interp1d

#parameters
arrival_rate = 1.0  #λ
service_rate = 2.0  #μ
simulation_time = 700.0  #seconds
replications = 50

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("--warmup", type=float, default=0.0, help=f"Time of the simulation (default simulation_time / 3)")
parser.add_argument("--replications", type=int, default=replications, help=f"Number of independent replications we do (default {replications})")
args = parser.parse_args()

arrival_rate = args.arrival_rate
service_rate = args.service_rate
if(service_rate <= arrival_rate):
    print("Cannot continue: system is not stable! Must have μ > λ!")
    exit(0)
simulation_time = args.simulation_time
replications = args.replications
warmup = args.warmup
if args.warmup == 0.0:
    warmup = simulation_time / 3
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

############################APPLYING ADDITIONAL OUTPUT ANALYSIS: INDEPENDENT REPLICATIONS AND WARMUP ELIMINATION#############################

#We do independent replications
empirical_averages = []
empirical_averages_warmupped = []

all_times = []
all_cumulative_means = []
all_times_now = []
all_cumulative_means_now = []

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

    empirical_avg = simulator.average_packets_in_system() #average #packets in the system
    empirical_averages.append(empirical_avg)
    empirical_avg = simulator.average_packets_in_system(warmup) #average #packets in the system
    empirical_averages_warmupped.append(empirical_avg)

    if warmup != 0.0: #if we specify a warmup, we eliminate first packets
        filtered_data = [(t, count) for t, count in simulator.packets_in_system if t >= warmup]

        #we compute relative times, starting from 0
        start_time = filtered_data[0][0]
        times, packets = zip(*[(t - start_time, count) for (_, (t, count)) in enumerate(filtered_data, 1)])

    times_now, packets_now = zip(*simulator.packets_in_system)

    times_now = np.array(times_now)           #times_now without eliminating warmup
    times = np.array(times)                   #times eliminating warmup
    packets_now = np.array(packets_now)       #packets_now without eliminating warmup
    packets = np.array(packets)               #packets eliminating warmup

    total_time = 0
    total_weighted_packets = 0
    cumulative_mean = []
    for i in range(len(times)-1):
        now = times[i]
        actual_packets = packets[i]
        after = times[i+1]

        delta = after-now
        total_time += delta
        total_weighted_packets += actual_packets * delta
        cumulative_mean.append(total_weighted_packets / total_time)

    cumulative_mean.append(cumulative_mean[-1]) #to avoid mismatch in length between means and times that would bring to problems in interpolation phase

    #same but without warmup
    total_time = 0
    total_weighted_packets = 0
    cumulative_mean_now = []
    for i in range(len(times_now)-1):
        now = times_now[i]
        actual_packets = packets_now[i]
        after = times_now[i+1]

        delta = after-now
        total_time += delta
        total_weighted_packets += actual_packets * delta
        cumulative_mean_now.append(total_weighted_packets / total_time)

    cumulative_mean_now.append(cumulative_mean_now[-1]) #to avoid mismatch in length between means and times that would bring to problems in interpolation phas

    all_times.append(times)
    all_cumulative_means.append(cumulative_mean)

    all_times_now.append(times_now)
    all_cumulative_means_now.append(cumulative_mean_now)


#compute time grid
max_time = max([t[-1] if len(t) > 0 else 0 for t in all_times_now])
```

```
time_grid = np.linspace(0, max_time, num=500)

interpolated_curves = []
for times, means in zip(all_times, all_cumulative_means):
    f = interp1d(times, means, kind="nearest", fill_value="extrapolate")
    interp_curve = f(time_grid)
    interpolated_curves.append(interp_curve)

interpolated_curves = np.array(interpolated_curves)
mean_curve = np.mean(interpolated_curves, axis=0)
warmup_std = np.nanstd(interpolated_curves)

interpolated_curves_now = []
for times, means in zip(all_times_now, all_cumulative_means_now):
    f = interp1d(times, means, kind="nearest", fill_value="extrapolate")
    interp_curve = f(time_grid)
    interpolated_curves_now.append(interp_curve)

interpolated_curves_now = np.array(interpolated_curves_now)
mean_curve_now = np.mean(interpolated_curves_now, axis=0)
no_warmup_std = np.std(interpolated_curves_now)

#Results
rho = arrival_rate / service_rate
theoretical_avg = rho / (1 - rho)
print(f"\nTheoretical average number of packets in system: {theoretical_avg:.4f}\n")

print(f"Empirical average number of packets in system: {np.mean(empirical_averages):.4f} (standard deviation {np.std(empirical_averages):.4f})")
lower_ci, upper_ci = confidence_interval(empirical_averages)
print(f"Confidence Interval: [{lower_ci:.4f}, {upper_ci:.4f}]\n")

print(f"[Warmup {warmup:.4f}]Empirical average number of packets in system: {np.mean(empirical_averages_warmupped):.4f} (standard deviation {np.std(empirical_a
lower_ci, upper_ci = confidence_interval(empirical_averages_warmupped)
print(f"Confidence Interval: [{lower_ci:.4f}, {upper_ci:.4f}]\n")

print(f"About IR...\nNO WARMUP STD DEV: {no_warmup_std:.4f}\nWARMUP STD DEV: {warmup_std:.4f}")

#Plotting
plt.figure(figsize=(12, 6))
plt.plot(time_grid, mean_curve_now, label='Without warmup', color='red')
plt.plot(time_grid, mean_curve, label=f'With warmup {warmup:.2f} / {simulation_time:.0f}', color='blue')  #an horizontal line at the end because we have less d
plt.axhline(y=arrival_rate / (service_rate - arrival_rate), linestyle='--', color='black', label=f'Theorical value: {theoretical_avg:.3f}')
plt.xlabel("Time")
plt.ylabel("Average number of packets in the system")
plt.title("Cumulated average number of packets in the system (independent replications)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```