Computer Exercise 4: Discrete Event Simulation

In [ ]:
```
# Plotting
import plotly.graph_objects as go
import plotly.express as px
import plotly.subplots as sp
import plotly.io as pio
pio.renderers.default = "notebook+pdf"
pio.templates.default = "plotly_dark"

# Utilities
import numpy as np
import pandas as pd
import math
from scipy.stats import t
```

Part 1 - Poisson Arrivals

Function for Discrete Event Simulation

In [ ]:
```
def blocking_system_simulation(
        num_service_units,
        num_customers,
        arrival_sample_fun,
        service_time_sample_fun,
        num_samples=None
    ):


    def single_sample():
        # Initialize the state of the system
        service_units_occupied = np.zeros(num_service_units)
        blocked_customers = 0

        # Main loop
        for _ in range(num_customers):

            # Sample arrival of a new customer
            arrival = arrival_sample_fun()

            # Update the state of the system
            service_units_occupied = np.maximum(0, service_units_occupied - arrival)

            # Check if a service unit is available
            if any(service_units_occupied == 0):
                # Sample the service time and assign the customer to the first available service unit
                service_time = service_time_sample_fun()
                service_unit = np.argmin(service_units_occupied)
                service_units_occupied[service_unit] = service_time
            else:
                # Block the customer
                blocked_customers += 1

        return blocked_customers/num_customers

    if num_samples is None:
        return single_sample()
    else:
        return np.array([single_sample() for _ in range(num_samples)])


def simulation_stats(theta_hats, alpha, verbose=False):
    n = len(theta_hats)
    theta_bar = np.mean(theta_hats)
    S = np.sqrt((np.sum(theta_hats**2) - n*theta_bar**2)/(n-1))
    CI = theta_bar + S/np.sqrt(n) * t.ppf([alpha/2, 1-alpha/2], n-1)

    if verbose:
        print(f"Simulated blocking probability: {np.round(theta_bar, 4)}")
        print(f"Standard deviation: {np.round(S, 4)}")
        print(f"{(1-alpha)*100}% confidence interval: {np.round(CI, 4)}")
```

```python
    return np.array([theta_bar, theta_bar-CI[0], CI[1]-theta_bar])


def analytic_block_prob(lam, s, m):
    A = lam * s
    return A**m/math.factorial(m)/np.sum([A**i/math.factorial(i) for i in range(m+1)])
```

Run Simulation

In [ ]:
```python
num_samples = 10
num_customers = 10000
m = 10
mean_service_time = 8
arrival_intensity = 1
alpha = 0.05

# Samplers
arrival_time_sampler = lambda: np.random.exponential(arrival_intensity)
service_time_sampler = lambda: np.random.exponential(mean_service_time)

# Compute the analytical blocking probability
B = analytic_block_prob(arrival_intensity, mean_service_time, m)
print(f"Analytical blocking probability: {np.round(B, 4)}")

# Simulate the blocking probability
theta_hats = blocking_system_simulation(m, num_customers, arrival_time_sampler, service_time_sampler, num_samples)
stats_exponential = simulation_stats(theta_hats, alpha, verbose=True)
```

```
Analytical blocking probability: 0.1217
Simulated blocking probability: 0.1215
Standard deviation: 0.0072
95.0% confidence interval: [0.1164 0.1267]
```

Part 2 - Renewal Arrivals

We will now consider renewal arrivals.

(a) Erlang Inter Arrival Times

In [ ]:
```python
# Erlang sampler
k = 1
arrival_time_sampler = lambda: np.random.gamma(k, 1/arrival_intensity)

# Simulate the blocking probability
theta_hats = blocking_system_simulation(m, num_customers, arrival_time_sampler, service_time_sampler, num_samples)
stats_erlang = simulation_stats(theta_hats, alpha, verbose=True)
```

```
Simulated blocking probability: 0.1211
Standard deviation: 0.0061
95.0% confidence interval: [0.1168 0.1255]
```

(b) Hyper Exponential Arrival Times

In [ ]:
```python
# Hyperexponential sampler
p1 = 0.8; p2 = 0.2
lam1 = 0.8333; lam2 = 5.0

arrival_time_sampler = lambda: np.random.choice([np.random.exponential(1/lam1), np.random.exponential(1/lam2)], p=[p1, p2])

# Simulate the blocking probability
theta_hats = blocking_system_simulation(m, num_customers, arrival_time_sampler, service_time_sampler, num_samples)
stats_hyperexponential = simulation_stats(theta_hats, alpha, verbose=True)
```

```
Simulated blocking probability: 0.1378
Standard deviation: 0.0056
95.0% confidence interval: [0.1338 0.1418]
```

In [ ]:
```python
# Collect results from varying arrival times
results_vary_arrival_times = np.array([stats_exponential, stats_erlang, stats_hyperexponential])
df_vary_arrival_times = pd.DataFrame(results_vary_arrival_times,
    columns=["Blocking Probability", "Lower Bound", "Upper Bound"])
df_vary_arrival_times["Distribution"] = ["Exponential", "Erlang", "Hyperexponential"]
```

Part 3 - Service Time Distribution

In [ ]:
```
arrival_time_sampler = lambda: np.random.exponential(1/arrival_intensity)
```

(a) Constant Service Time

In [ ]:
```
# Constant "sampler"
service_time_sampler = lambda: mean_service_time

# Simulate the blocking probability
theta_hats = blocking_system_simulation(m, num_customers, arrival_time_sampler, service_time_sampler, num_samples)
stats_const = simulation_stats(theta_hats, alpha, verbose=True)
```

```
Simulated blocking probability: 0.1236
Standard deviation: 0.0047
95.0% confidence interval: [0.1202 0.1269]
```

In [ ]:

In [ ]:
```
A = np.array([np.mean(np.array([service_time_sampler() for _ in range(10000)])) for _ in range(100)])

A.mean()
```

Out[ ]:
```
8.0
```

(b) Pareto Service Time

In [ ]:
```
# Pareto sampler

# k = 1.05
k = 1.05
beta = 8*(k-1)/k # Ensure that the mean is 8
service_time_sampler = lambda: beta*(np.random.uniform())**(-1/k)
theta_hats = blocking_system_simulation(m, num_customers, arrival_time_sampler, service_time_sampler, num_samples)
stats_pareto105 = simulation_stats(theta_hats, alpha)

# k = 2.05
k = 2.05
beta = 8*(k-1)/k # Ensure that the mean is 8
service_time_sampler = lambda: beta*(np.random.uniform())**(-1/k)
theta_hats = blocking_system_simulation(m, num_customers, arrival_time_sampler, service_time_sampler, num_samples)
stats_pareto205 = simulation_stats(theta_hats, alpha)
```

As k increases, the Pareto distribution skews more to the right, resulting in decreasing service times which in turn allows for more customers to be served.

(c) Gamma and Chi-Square Service Time

In [ ]:
```
# Gamma sampler
shape = 4; scale = 2 # Has mean shape*scale = 8
service_time_sampler = lambda: np.random.gamma(scale, shape)

# Simulate the blocking probability
theta_hats = blocking_system_simulation(m, num_customers, arrival_time_sampler, service_time_sampler, num_samples)
stats_cauchy = simulation_stats(theta_hats, alpha, verbose=True)
```

```
Simulated blocking probability: 0.1229
Standard deviation: 0.0042
95.0% confidence interval: [0.1199 0.1259]
```

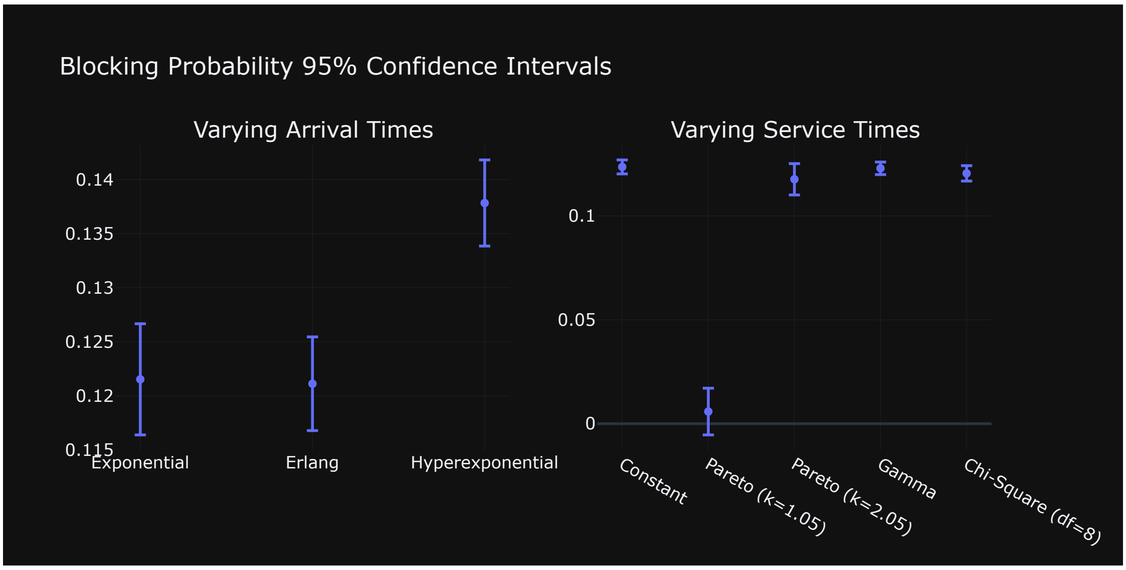
In [ ]:
```
# Chi-Square sampler
df = 8 # Has mean df = 8
service_time_sampler = lambda: np.random.chisquare(df)

# Simulate the blocking probability
theta_hats = blocking_system_simulation(m, num_customers, arrival_time_sampler, service_time_sampler, num_samples)
stats_chi2 = simulation_stats(theta_hats, alpha, verbose=True)
```

```
Simulated blocking probability: 0.1205
Standard deviation: 0.0052
95.0% confidence interval: [0.1168 0.1242]
```

In [ ]:
```
# Collect results for varying service times
results_vary_service_times = np.array([stats_const, stats_pareto105, stats_pareto205, stats_cauchy, stats_chi2])
df_vary_service_times = pd.DataFrame(results_vary_service_times,
    columns=["Blocking Probability", "Lower Bound", "Upper Bound"])
df_vary_service_times["Distribution"] = ["Constant", "Pareto (k=1.05)", "Pareto (k=2.05)", "Gamma", "Chi-Square (df=8)"]
```

Part 4 - Confidence Intervals

In [ ]:
```
fig = sp.make_subplots(rows=1, cols=2, subplot_titles=["Varying Arrival Times", "Varying Service Times"])
scatter1 = px.scatter(df_vary_arrival_times, x="Distribution", y="Blocking Probability", error_y="Upper Bound", error_y_minus="Lowe
scatter2 = px.scatter(df_vary_service_times, x="Distribution", y="Blocking Probability", error_y="Upper Bound", error_y_minus="Lowe
fig.add_trace(scatter1.data[0], row=1, col=1)
fig.add_trace(scatter2.data[0], row=1, col=2)
fig.update_layout(title="Blocking Probability 95% Confidence Intervals", width=800, height=400)
fig.show()
```



When varying the arrival time distribution, we observe that the confidence intervals for the blocking probability have similar width. This consistency suggests that the uncertainty associated with the blocking probability remains unaffected by whether the arrivals are more dispersed or concentrated. However, it is important to note that the blocking probability itself is influenced by the arrival time distribution.

Also when varying the service time distribution, but maintaining the mean, we observe that the confidence intervals for the blocking probability have similar widths, except for the case of the Pareto distribution with $k = 1.05$. This suggests that the distribution of the blocking probability might depend on the statistical moments of the service time distribution rather than the distribution itself. However, the Pareto distribution with $k = 1.05$ is an exception to this observation, which can be attributed to the fact that the pareto has no mean when $k \leq 1$, thus the mean of samples with $k = 1.05$ is highly unstable and seem to generally be smaller than the theoretical mean. This results in a lower blocking probability as an empirically smaller mean service time implies that more customers can be served.