Computer Exercise 5: Variance Reduction Methods

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
from scipy.optimize import root
from utils import describe_sample
```

In this exercise we will approximate the value of the integral $\int_0^1 \mathrm{e}^x$ with different sampling methods and compare their efficiency. To compare the true value of the integral is:

$$\int_0^1 \mathrm{e}^x \, dx = \mathrm{e} - 1 \approx 1.718281828459045$$

Part 1 - Integral Estimation by Crude Monte Carlo

Using the Crude Monte Carlo we method we exploit the fact that the integral of a function $f(x)$ over a domain $[a, b]$ can be estimated by the average of the function values at N random points x_i in the domain $[a, b]$. So we sample points from the uniform distribution between 0 and 1, tranforms those points using the exponential function and then calculate the average of the function values at those points.

In [ ]:
```
num_samples = 100

# Crude Monte Carlo
U = np.random.uniform(0, 1, num_samples)
X = np.exp(U)

describe_sample(X, title="Method: Crude Monte Carlo")
```

```
──────────────────────────────────────────
       >>> SAMPLE STATISTICS <<<
       Method: Crude Monte Carlo
──────────────────────────────────────────
Sample size: 100
Mean: 1.7075
Standard deviation: 0.5281
95% confidence interval: [1.6027 1.8123]
──────────────────────────────────────────
```

Above we have reported the mean value of our samples, which represents the approximated value of the integral. Along with it we have reported the standard deviation and confidence interval of the mean value. For now we can see that the approximated value of the integral is a bit lower than the actual value. As we compute the integral with more methods, the tigthness of the confidence interval will be a good indicator of the accuracy of the method.

Part 2 - Integral Estimation by Antithetic Variables

To estimate the integral by using antithetic variables we transform the uniform samples with the following formula:

$$Y_i = \frac{1}{2}\left(e^{U_i} + e^{1-U_i}\right)$$

In [ ]:
```
Y = (np.exp(U) + np.exp(1 - U)) / 2

describe_sample(Y, title="Method: Antithetic Variates")
```

```
──────────────────────────────────────────
       >>> SAMPLE STATISTICS <<<
      Method: Antithetic Variates
──────────────────────────────────────────
Sample size: 100
Mean: 1.7310
Standard deviation: 0.0663
95% confidence interval: [1.7178 1.7441]
──────────────────────────────────────────
```

With this method we see that the approximation of the integral is closer to the true value of the integral. We also see that the standard deviation has almost dropped with a factor 10. So using antithetic variables has improved the accuracy of the estimation compared to the crude Monte Carlo Method.

Analytically, as seen on the slides, there should be a variance reduction of 98% between these first two methods, however we only saw a variance reduction of 88%.

Part 3 - Integral Estimation by Control Variable

For the next method, we consider to sets of samples. U which is our uniformly distributed samples, and $X = e^U$.

To estimate the integral by using control variables we use the following formula:

$$Z_i = X + c\,(U - \mu_U)$$

where $c = \frac{-\mathrm{Cov}(X,U)}{\mathrm{Var}(U)}$ and $\mu_U = \frac{1}{2}$ is the mean of the uniform samples. We then use Z to estimate the integral.

In [ ]:
```
c = -np.cov(X,U)[0][1]/np.var(U)
Z = X + c*(U - 0.5)

describe_sample(Z, title="Method: Control Variates")
```

```
--------------------------------------------
        >>> SAMPLE STATISTICS <<<
        Method: Control Variates
--------------------------------------------
Sample size: 100
Mean: 1.7300
Standard deviation: 0.0656
95% confidence interval: [1.717  1.7431]
--------------------------------------------
```

With this method we see that the approximation of the integral is slightly closer to the true value of the integral. The standard deviation has the same size, so we do not get a more confident estimate of the integral compared to the antithetic variables method.

Part 4 - Integral Estimation by Stratified Sampling

With stratified sampling, we need to generate new numbers from the uniform distribution, where until now we have been able to use the same samples to approximate the integral. We generate a matrix of 10×100 uniform numbers. To do stratified sampling we then use the following formula:

$$W_i = \frac{1}{10}\sum_{j=1}^{10} e^{\frac{U_{ij}}{10}} e^{\frac{(j-1)}{10}}$$

So from our matrix we generate 100 stratified samples by using 10 uniform samples for each.

In [ ]:
```
strata_number = 10

U2 = np.random.uniform(0, 1, size=(num_samples, strata_number))
W = np.mean(np.exp((U2 + np.arange(strata_number))/strata_number), axis=1)

describe_sample(W, title="Method: Stratified Sampling")
```

```
--------------------------------------------
        >>> SAMPLE STATISTICS <<<
       Method: Stratified Sampling
--------------------------------------------
Sample size: 100
Mean: 1.7201
Standard deviation: 0.0166
95% confidence interval: [1.7168 1.7234]
--------------------------------------------
```

With stratified sampling we see that the approximation of the integral is closer to the true value of the integral. The standard deviation has also dropped with a factor 5, so we get a more confident estimate of the integral compared to the control variable method.

Part 5 - Variance Reduction in Blocking System with Control Variates

Using control variates, we wish to see if we can reduce the variance of the estimator from the Poisson arrival process considered in computer exercise 4.

In [ ]:
```python
def BS_control(
        arrival_times,
        service_times,
        control_variate,
        control_mu,
        num_service_units=10,
    ):

    num_samples, num_customers = arrival_times.shape

    # Sample loop
    out = np.zeros(num_samples)
    for i in range(num_samples):
        # Initialize the state of the system
        service_units_occupied = np.zeros(num_service_units)
        blocked_customers = np.zeros(num_customers)

        # Main loop
        for j in range(num_customers):

            # Update the state of the system
            service_units_occupied = np.maximum(0, service_units_occupied - arrival_times[i, j])

            # 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_unit = np.argmin(service_units_occupied)
                service_units_occupied[service_unit] = service_times[i, j]
            else:
                # Block the customer
                blocked_customers[j] = 1

        # Control variates
        X = blocked_customers
        Y = control_variate[i]
        mu_y = control_mu

        c = -np.cov(X,Y)[0][1]/np.var(Y)
        Z = X + c*(Y - mu_y)

        out[i] = np.mean(Z)

    return out
```

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

In [ ]:
```python
# Generate the arrival and service times
A = np.random.exponential(arrival_intensity, (num_samples, num_customers))
S = np.random.exponential(mean_service_time, (num_samples, num_customers))

# Simulate the blocking probability
theta_hats = BS_control(A, S, A, arrival_intensity)
describe_sample(theta_hats, title="Blocking Probability: Control Variates")
```

```
--------------------------------------------------
       >>> SAMPLE STATISTICS <<<
 Blocking Probability: Control Variates
--------------------------------------------------
Sample size: 10
Mean: 0.1169
Standard deviation: 0.0062
95% confidence interval: [0.1124 0.1213]
--------------------------------------------------
```

Part 6 - Effect of Using Common Random Variates

In [ ]:
```python
# Simulation Parameters
num_samples = 100
num_customers = 10000
m = 10
mean_service_time = 8

# Possion parameters
arrival_intensity = 1
```

```
# Hyperexponential parameters (from exercise 4) mean=1
p1 = 0.8; p2 = 0.2
lam1 = 0.8333; lam2 = 5.0
```

Using Distinct Random Variates

In [ ]:
```
# Generate distinct random variates for arrival and service times
A_poisson = np.random.exponential(arrival_intensity, (num_samples, num_customers))
A_hyper = np.random.choice([np.random.exponential(1/lam1), np.random.exponential(1/lam2)], p=[p1, p2], size=(num_samples, num_custo
S_poisson = np.random.exponential(mean_service_time, (num_samples, num_customers))
S_hyper = np.random.exponential(mean_service_time, (num_samples, num_customers))

# Simulate the difference in blocking probability
theta_hats_poisson = BS_control(A_poisson, S_poisson, A_poisson, arrival_intensity)
theta_hats_hyperexponential = BS_control(A_hyper, S_hyper, A_hyper, arrival_intensity)
delta_theta_hats = theta_hats_poisson - theta_hats_hyperexponential
describe_sample(delta_theta_hats, title="Distinct Variates")
```

```
--------------------------------------------
        >>> SAMPLE STATISTICS <<<
            Distinct Variates
--------------------------------------------
Sample size: 100
Mean: 0.1180
Standard deviation: 0.0053
95% confidence interval: [0.1169 0.1191]
--------------------------------------------
```

Using Common Random Variates

In [ ]:
```
# Generate common random variates for arrival and service times
U1 = np.random.uniform(0, 1, (num_samples, num_customers))
U2 = np.random.uniform(0, 1, (num_samples, num_customers))
A_poisson = -np.log(U1)/arrival_intensity
A_hyper = np.where(U2 < p1, -np.log(U1)/lam1, -np.log(U1)/lam2)
S = np.random.exponential(mean_service_time, (num_samples, num_customers))

# Simulate the difference in blocking probability
theta_hats_poisson = BS_control(A_poisson, S, U1, 0.5)
theta_hats_hyperexponential = BS_control(A_hyper, S, U1, 0.5)
delta_theta_hats = theta_hats_poisson - theta_hats_hyperexponential
describe_sample(delta_theta_hats, title="Common Variates")
```

```
--------------------------------------------
        >>> SAMPLE STATISTICS <<<
             Common Variates
--------------------------------------------
Sample size: 100
Mean: -0.0171
Standard deviation: 0.0037
95% confidence interval: [-0.0179 -0.0164]
--------------------------------------------
```

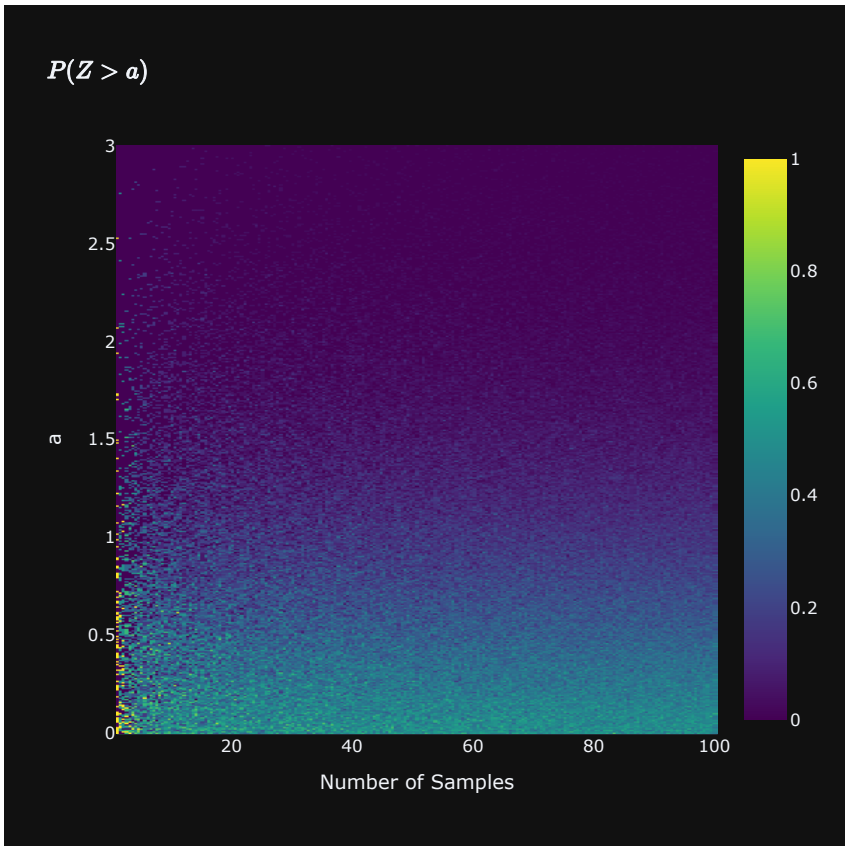
Part 7 - Probability Estimation

In this part we will revisit the crude Monte Carlo method to estimate the probability that $Z > a$ where $Z \sim \mathcal{N}(0, 1)$. We will do this by generating a set of standard normal random variables and then estimate the probability that $Z > a$ by directly considering only the samples from Z where $Z_i > a$.

In [ ]:
```
sigma=1
num_samples=1000
a = 2
Z = np.random.normal(0, sigma, num_samples)
I = Z > a

describe_sample(I, title="Crude Monte Carlo")

as_ = np.linspace(0, 3, 1000)
N = np.linspace(1, 100, 1000, dtype=int)

p = np.zeros((len(as_), len(N)))
for i, a in enumerate(as_):
    for j, n in enumerate(N):
        Z = np.random.normal(0, sigma, n)
        p[i,j] = np.mean(Z > a)

# Plot heat map
fig = go.Figure(data=go.Heatmap(z=p, x=N, y=as_, colorscale='Viridis'))
```

```
fig.update_layout(title=r"Crude Monte Carlo: Estimated $P(Z>a)$", xaxis_title="Number of Samples", yaxis_title="a", width=600, heigl
fig.show()
```

```
──────────────────────────────────────────
        >>> SAMPLE STATISTICS <<<
            Crude Monte Carlo
──────────────────────────────────────────
Sample size: 1000
Mean: 0.0180
Standard deviation: 0.1330
95% confidence interval: [0.0097 0.0263]
──────────────────────────────────────────
```



First we estimated the probability of $Z > a$ with $a = 2$ and 1000 samples. Here we find that $p(Z > a) \approx 0.0250$. The standard deviation of this result is 0.1526 which is quite high. We can see that the confidence interval is quite wide, so we are not very confident in our estimate.

Part 8 - Integral Estimation by Importance Sampling

In this part we will go back to considering a new method for approximating the integral $\int_0^1 \mathrm{e}^x$. This time we will use importance sampling. In importance sampling we start by using samples from the uniform distribution U, we then use the inversion method to generate samples from the exponential distribution $Y = \frac{-\log(U)}{\lambda}$. The importance sampling estimator is then given by:

$$\theta = \mathbb{E}\left(\frac{h(Y)f(Y)}{g(y)}\right)$$

As given by the exercise, we set $g(y) = \lambda \mathrm{e}^{-\lambda y}$. We set $h(y) = \mathrm{e}^y$ as this is the integral we wish to approximate, and $f(y) = \mathbf{1}_{[0,1]}(y)$ as we are only interested in the integral over the interval $[0, 1]$.

Now the final step before approximating θ is to find out what value of λ to use. We will do this by deriving λ from the condition that the variance of the estimator is minimized.

Determining variance minimizing λ by analytical derivation

Computing the analytic variance $\mathbb{V}\left[\frac{h(X)f(X)}{g(X)}\right]$. The functions $f(X)$, $g(X)$, and $h(X)$ are given as,

$$g(X) = \lambda e^{-\lambda X}, \quad f(X) = \mathbf{1}_{0 \le X \le 1}, \quad h(X) = e^X.$$

Using the definition of variance for a random variable,

$$\mathbb{V}\left[\frac{h(X)f(X)}{g(X)}\right] = \mathbb{E}\left[\left(\frac{h(X)f(X)}{g(X)}\right)^2\right] - \left(\mathbb{E}\left[\frac{h(X)f(X)}{g(X)}\right]\right)^2.$$

These expectations can be computed as follows,

$$\mathbb{E}\left[\frac{h(X)f(X)}{g(X)}\right] = \int_0^1 \frac{h(X)f(X)}{g(X)} g(X)dX = \int_0^1 h(X)dX = \int_0^1 e^X dX = e - 1$$

$$\mathbb{E}\left[\left(\frac{h(X)f(X)}{g(X)}\right)^2\right] = \int_0^1 \left(\frac{h(X)f(X)}{g(X)}\right)^2 g(X)dX = \int_0^1 \frac{h(X)^2}{g(X)}dX = \int_0^1 \frac{e^X}{\lambda e^{-\lambda X}}dX = \frac{1}{\lambda}\int_0^1 e^{(1+\lambda)X}dX = \frac{e^{1+\lambda}-1}{\lambda(1+\lambda)}.$$

Inserting into the variance formula,

$$\mathbb{V}\left[\frac{h(X)f(X)}{g(X)}\right] = \frac{e^{1+\lambda}-1}{\lambda(1+\lambda)} - (e-1)^2.$$

Differentiating the variance with respect to λ, setting it equal to zero and solving for λ yields,

$$\frac{d}{d\lambda}\left(\mathbb{V}\left[\frac{h(X)f(X)}{g(X)}\right]\right) = \frac{(\lambda^2-2)e^{2+\lambda}+2(\lambda+1)}{\lambda^2(2+\lambda)^2} = 0 \quad \Rightarrow \quad \lambda^* = 1.3548.$$

Note: The second derivative of the variance with respect to λ evaluated at λ^* is positive, thus λ^* is a minimizer.

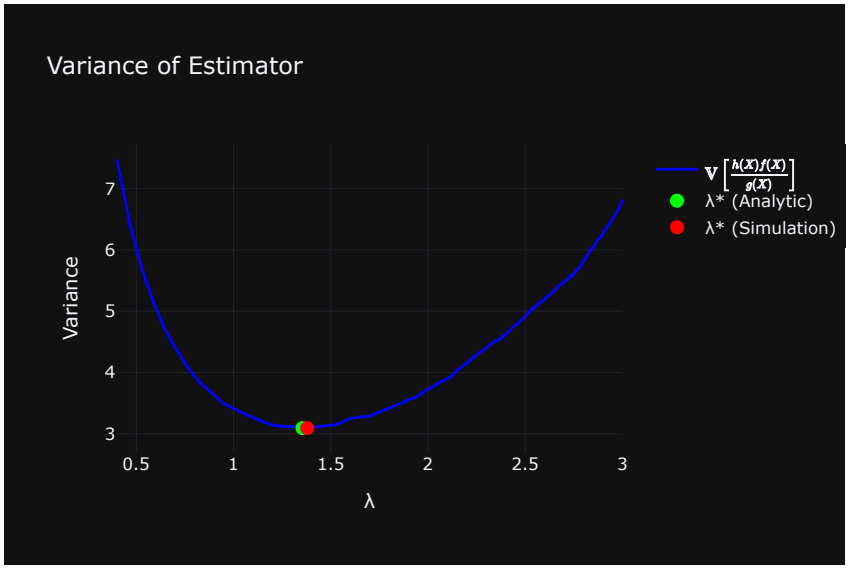
In [ ]:
```
lam_analytic = root(lambda λ: ((λ**2 - 2)*np.exp(2 + λ) + 2 * (λ + 1)) / (λ**2*(2 + λ)**2), 1).x[0]
print(f"λ* = {lam_analytic:.4f}")
```

```
λ* = 1.3548
```

Determining variance minimizing λ by simulation

Next we also determine the variance minimizing λ by simulation. We do this by computing the variance of the estimator for different values of λ and then find the value of λ that minimizes the variance.

In [ ]:
```
# Define functions
g = lambda x, λ: λ*np.exp(-λ*x)
f = lambda x: (x > 0)*(x < 1)
h = lambda x: np.exp(x)

# Sample
num_samples = 100000
U = np.random.uniform(0, 1, num_samples)

# Find the optimal lambda by simulation
lambdas = np.linspace(0.4, 3, 1000)
variances = np.zeros_like(lambdas)
for i, λ in enumerate(lambdas):
    Y = -np.log(U)/λ
    theta = h(Y)*f(Y)/g(Y, λ)
    variances[i] = np.var(theta)
lam_sim = lambdas[np.argmin(variances)]

# Plot the variance as a function of lambda
fig = go.Figure(data=go.Scatter(x=lambdas, y=variances, marker=dict(color='Blue'), name=r"$\mathbb{V}\left[\frac{h(X)f(X)}{g(X)}\rig
fig.add_trace(go.Scatter(x=[lam_analytic], y=[np.min(variances)], mode='markers', marker=dict(size=10, color='Lime'), name="λ* (Ana
fig.add_trace(go.Scatter(x=[lam_sim], y=[np.min(variances)], mode='markers', marker=dict(size=10, color='Red'), name="λ* (Simulatio
fig.update_layout(title="Variance of Estimator", xaxis_title="λ", yaxis_title="Variance", width=600, height=400)
fig.show()
```



Get sample statistics

Now we compare the estimates of θ using the two different λ values.

In [ ]:
```
describe_sample(h(Y)*f(Y)/g(Y, lam_sim), title=f"IS: λ={lam_sim:.2f} (Simulated)")
describe_sample(h(Y)*f(Y)/g(Y, lam_analytic), title=f"IS: λ={lam_analytic:.2f} (Analytic)")
```

```
──────────────────────────────────────────────────
        >>> SAMPLE STATISTICS <<<
         IS: λ=1.38 (Simulated)
──────────────────────────────────────────────────
Sample size: 100000
Mean: 1.6142
Standard deviation: 1.2903
95% confidence interval: [1.6062 1.6222]
──────────────────────────────────────────────────
──────────────────────────────────────────────────
        >>> SAMPLE STATISTICS <<<
          IS: λ=1.35 (Analytic)
──────────────────────────────────────────────────
Sample size: 100000
Mean: 1.6253
Standard deviation: 1.2840
95% confidence interval: [1.6173 1.6332]
──────────────────────────────────────────────────
```

From the two approximations of the integral using importance sampling, we see that the estimated value is a bit lower than the true value. We also see that the standard deviation is quite high compared to previous results. So the importance sampling method does not look promising with the given distribution.

Part 9 - Importance Sampling Estimator for Pareto Distribution

The first moment distribution of the Pareto distribution is given in lecture as,

$$g(x) = G_1(x) = 1 - \left(\frac{x}{\beta}\right)^{1-k}.$$

Using the same $f(x)$ and $h(x)$ as in the previous part,

$$f(x) = \mathbf{1}_{0 \le x \le 1}, \quad h(x) = e^x,$$

The importance sampling estimator is given by,

$$\theta = \mathbb{E}\left[\frac{h(Y)f(Y)}{g(Y)}\right] = \mathbb{E}\left[\frac{e^Y}{1-\left(\frac{Y}{\beta}\right)^{1-k}}\right],$$

where $Y \sim Pa(\beta, k)$. For samples close to β this becomes numerically unstable.