IEOR 4404 Simulation **Fall 2024**

Homework 6

Instructor: Henry Lam

Problem 1 For each of the following integrals:

(a)

$$\int_0^1 \exp\{e^x\}dx$$

(b)

$$\int_{-2}^2 e^{x+x^2}dx$$

(c)

$$\int_0^\infty x(1+x^2)^{-2}dx$$

(d)

$$\int_0^\infty \int_0^{x^2} e^{-(x+y)}\sin(xy)dydx$$

do the following: Write a pseudo-code to generate a simulation run that is appropriate for numerically approximating the integral. Then implement 1000 simulation runs on a computer. Give a point estimate and a 95% confidence interval for the value of the integral.

Problem 2 Let $U \sim Unif(0,1)$. By running 1000 simulation runs, give point estimates of $Cov(U, e^U)$ and $Corr(U, \sqrt{1-U^2})$.

Alexandre
Duhamel
afd2153
MSOR

HW6 : Simulation

Problem 1:

a) We want to evaluate $\int_0^1 e^{e^x} dx$.

$I = \int_a^b f(x)\,dx \approx \frac{b-a}{N} \sum_{i=1}^{N} f(U_i)$

Pseudo-code:
- Set $N = 1000$
- For $i = 1$ to N:
  - Generate $U[i] \sim \text{Uniform}(0,1)$
  - Compute $f(U[i]) = \exp(\exp(U[i]))$
- Compute the estimate as the mean of $f(U)$
- Compute the standard deviation of $f(U)$
- Compute 95% confidence interval: CI = estimate ± 1.96 (stdev(f(U))/sqrt(N))
- Return the estimate and CI

```python
def f(x):
    return np.exp(np.exp(x))

def monte_carlo_integration(N):
    U = np.random.uniform(0, 1, N)

    f_values = f(U)

    estimate = np.mean(f_values)

    std_dev = np.std(f_values)
    standard_error = std_dev / np.sqrt(N)

    confidence_interval = (estimate - 1.96 * standard_error, estimate + 1.96 * standard_error)

    return (estimate, confidence_interval)

N = 1000
estimate, confidence_interval = monte_carlo_integration(N)
print(f"Estimated Integral: {estimate}")
print(f"95% Confidence Interval: {confidence_interval}")
```

```
Estimated Integral: 6.148455444365251
95% Confidence Interval: (5.950545330341265, 6.346365558389236)
```

I was curious to see how well it performed. I tested the computed value vs the "actual" value (scipy's computation of the integral).
Using $N = 1000$ largely does the job to estimate the integral.

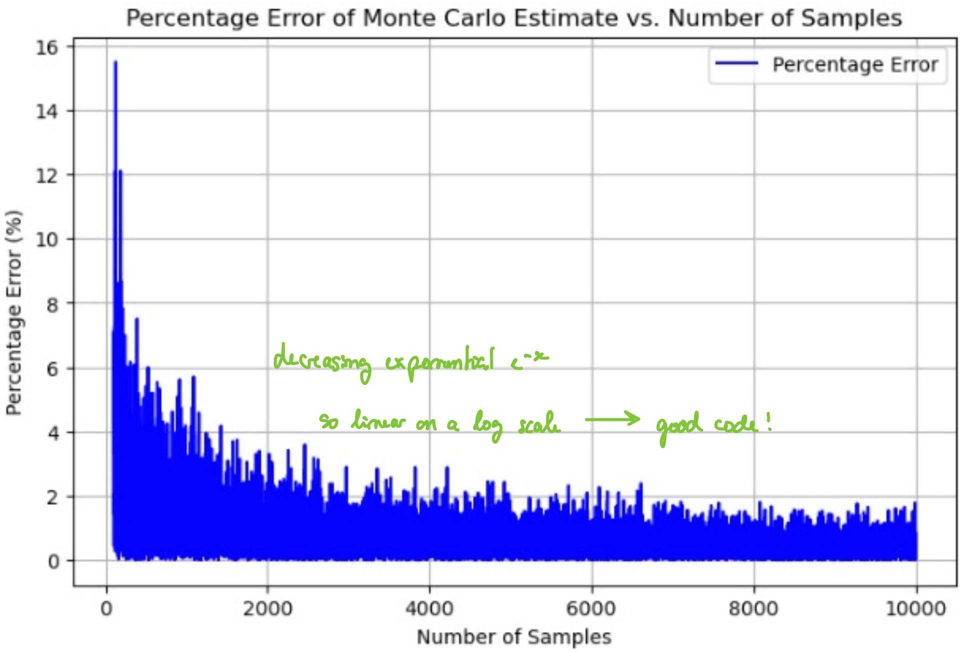
```
# Compute the actual integral using scipy's quad function
actual_integral, _ = quad(f, 0, 1)

sample_sizes = np.arange(100, 10001, 1)

errors = []

for N in sample_sizes:
    monte_carlo_estimate = monte_carlo_integration(N)[0]
    # Calculate the percentage error
    error = abs((monte_carlo_estimate - actual_integral) / actual_integral) * 100
    errors.append(error)

plt.figure(figsize=(8, 5))
plt.plot(sample_sizes, errors, label="Percentage Error", color='blue')
plt.title("Percentage Error of Monte Carlo Estimate vs. Number of Samples")
plt.xlabel("Number of Samples")
plt.ylabel("Percentage Error (%)")
plt.grid(True)
plt.legend()
plt.show()
```



b) We want to evaluate $\int_{-2}^{2} e^{x+x^2}\,dx$

Pseudo-code: Set $N = 1000$

For $i = 1$ to N: Generate $U[i] \sim \text{Uniform}(-2, 2)$

Compute $f(U[i]) = \exp(U[i] + U[i]**2)$

Compute the estimate as the mean of $f(U)$

Compute the standard deviation of $f(U)$

Compute 95% confidence interval: $CI = \text{estimate} \pm 1.96\,(\text{stdev}(f(U))/\text{sqrt}(N))$

Return the estimate and CI

```python
def f(x):
    return np.exp(x + x**2)

def monte_carlo_integration_b(N):
    sum_f = 0
    f_values = []

    for i in range(N):
        U = np.random.uniform(0, 1)
        X = 4 * U - 2
        fx = f(X)
        f_values.append(fx)
        sum_f += fx

    estimate = 4 * sum_f / N

    std_dev = np.std(f_values)
    standard_error = std_dev / np.sqrt(N)

    confidence_interval = (estimate - 1.96 * standard_error, estimate + 1.96 * standard_error)
    return estimate, confidence_interval

N = 1000

estimate_b, confidence_interval_b = monte_carlo_integration_b(N)

actual_integral_b, _ = quad(f, -2, 2)

print(f"Monte Carlo Estimated Integral: {estimate_b}")
print(f"Actual value of the Integral (from scipy): {actual_integral_b}")
print(f"95% Confidence Interval: {confidence_interval_b}")
```

```
Monte Carlo Estimated Integral: 89.42806312552695
Actual value of the Integral (from scipy): 93.16275329244199
95% Confidence Interval: (85.68882844401834, 93.16729780703555)
```

$$\begin{cases} x = \dfrac{1-u}{u} \\ dx = \dfrac{-1}{u^2}\,du \\ \infty \to 0 \\ 0 \to 1 \end{cases}$$

Simple variable change

c) We want to evaluate $\int_0^\infty \underbrace{x(1+x^2)^{-2}}_{f(x)}\,dx = \int_0^1 f\left(\frac{1-u}{u}\right)\frac{1}{u^2}\,du$

Pseudo-code:

- Set $N = 1000$
- For $i = 1$ to N:
  - Generate $U[i] \sim$ Uniform$(0,1)$
  - Compute $g(U[i]) = f[1 - U[i]/U[i]]/U[i]^2$
- Compute the estimate as the mean of $g(U)$
- Compute the standard deviation of $g(U)$
- Compute 95% confidence interval: CI = estimate ± 1.96 (stdev(g(U))/ sqrt(N))
- Return the estimate and CI

```
def f(x):
    return x *(1 + x**2)**(-2)

def transformed_function(u):
    x = (1 - u) / u
    return f(x) / u**2

def monte_carlo_integration_c(N):
    sum_f = 0
    f_values = []

    for i in range(N):
        u = np.random.uniform(0, 1)
        fx = transformed_function(u)
        f_values.append(fx)
        sum_f += fx

    estimate = sum_f / N

    std_dev = np.std(f_values)
    standard_error = std_dev / np.sqrt(N)

    confidence_interval = (estimate - 1.96 * standard_error, estimate + 1.96 * standard_error)

    return (estimate, confidence_interval)

N = 1000

estimate_c, confidence_interval_c = monte_carlo_integration_c(N)

actual_integral_c, _ = quad(f, 0, np.inf)

# Print the results
print(f"Monte Carlo Estimated Integral: {estimate_c}")
print(f"Actual value of the Integral (from scipy): {actual_integral_c}")
print(f"95% Confidence Interval: {confidence_interval_c}")
```

```
Monte Carlo Estimated Integral: 0.5005328375511389
Actual value of the Integral (from scipy): 0.5
95% Confidence Interval: (0.4796077112276244, 0.5214579638746533)
```

d) We want to evaluate $\int_0^\infty \int_0^{x^2} e^{-(x+y)} \sin(xy)\, dy\, dx = \int_0^1 \int_0^{\frac{(1-u)^2}{u^2}} \frac{e^{-\left(\frac{1-u}{u}+y\right)} \sin\left(\frac{1-u}{u} \cdot y\right)}{u^2}\, dy\, du$

$$\begin{cases} x = \frac{1-u}{u} \\ dx = \frac{-1}{u^2} du \\ +\infty \to 0 \quad \text{and } x^2 = \frac{(1-u)^2}{u^2} \\ 0 \to 1 \end{cases}$$

Pseudo-code:

- Set $N = 1000$, outer sum $= 0$
- For $i = 1$ to N_i:
  - Generate $U \sim \text{Uniform}(0,1)$
  - Set $x = 1 - U / U$, inner sum $= 0$
  - For ~~j~~ $i = 1$ to N_i:
    - Generate $U_y \sim \text{Uniform}(0, x^2) = y$
    - Set f : $\exp(-(x+y)) + \sin(x*y) / ((1-U)**2)$
    - inner sum += f
  - outersum += (innersum / N_i) $\times x^2$
- estimate = outersum / N_o
- return estimate   # compute and return CI & estimate as usual

```python
def f(x, y):
    return np.exp(-(x + y)) * np.sin(x * y)

def transformed_f(U, y):
    x = U / (1 - U)
    return f(x, y) * (1 / (1 - U)**2

def monte_carlo_integration_d(N_outer, N_inner):
    sum_f = 0
    f_values = []

    for _ in range(N_outer):
        U = np.random.uniform(0, 1)
        x = U / (1 - U)
        inner_sum = 0
        for _ in range(N_inner):
            Y = np.random.uniform(0, x**2)
            fx = transformed_f(U, Y)
            inner_sum += fx

        inner_average = (x**2) * inner_sum / N_inner
        f_values.append(inner_average)
        sum_f += inner_average

    estimate = sum_f / N_outer

    std_dev = np.std(f_values)
    standard_error = std_dev / np.sqrt(N_outer)

    confidence_interval = (estimate - 1.96 * standard_error, estimate + 1.96 * standard_error)

    return estimate, confidence_interval

N_outer = 1000
N_inner = 1000

estimate_d, confidence_interval_d = monte_carlo_integration_d(N_outer, N_inner)

real_integral, _ = dblquad(lambda y, x: np.exp(-(x + y)) * np.sin(x * y), 0, np.inf, lambda x:

print(f"Monte Carlo Estimated Integral: {estimate_d}")
print(f"95% Confidence Interval: {confidence_interval_d}")
print(f"Actual value of the Integral (from scipy): {real_integral}")
```

```
Monte Carlo Estimated Integral: 0.160858584097772
95% Confidence Interval: (0.1469888607630262, 0.17472830743251783)
Actual value of the Integral (from scipy): 0.16346389681778564
```

Problem 2:

- We'll generate $U \sim Unif(0,1)$ for $N = 1000$ simulations

Compute $\exp(U)$, sample mean of U and $\exp(U)$ and $\operatorname{cov}(U, e^U) = \frac{1}{N}\sum_{i=1}^{N}(U_i - \bar{U})(e^{U_i} - \overline{e^U})$

for each U

As well as $\operatorname{corr}(U, \sqrt{1-U^2}) = \frac{\operatorname{cov}(U, \sqrt{1-U^2})}{\sigma_U \, \sigma_{\sqrt{1-U^2}}}$ where $\sigma_x = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$

This gives us one value, continue N times to compute the point estimates.

- Here is the code:

```python
n = 1000

U_samples = [random.uniform(0, 1) for _ in range(n)]

e_U_samples = [math.exp(U) for U in U_samples]
sqrt_1_minus_U2_samples = [math.sqrt(1 - U**2) for U in U_samples]

mean_U = sum(U_samples) / n
mean_e_U = sum(e_U_samples) / n
mean_sqrt_1_minus_U2 = sum(sqrt_1_minus_U2_samples) / n

var_U = (sum(U**2 for U in U_samples) / n) - (mean_U**2)
var_e_U = (sum(e_U**2 for e_U in e_U_samples) / n) - (mean_e_U**2)
var_sqrt_1_minus_U2 = (sum(sqrt_1_minus_U2**2 for sqrt_1_minus_U2 in sqrt_1_minus_U2_samples) / n) - (mean_sqrt_1_mi

cov_U_eU = (sum(U_samples[i] * e_U_samples[i] for i in range(n)) / n) - (mean_U * mean_e_U)
cov_U_sqrt_1_minus_U2 = (sum(U_samples[i] * sqrt_1_minus_U2_samples[i] for i in range(n)) / n) - (mean_U * mean_sqrt

corr_U_eU = cov_U_eU / math.sqrt(var_U * var_e_U)
corr_U_sqrt_1_minus_U2 = cov_U_sqrt_1_minus_U2 / math.sqrt(var_U * var_sqrt_1_minus_U2)

print(f"Mean of U = {mean_U}")
print(f"Mean of e^U = {mean_e_U}")
print(f"Mean of sqrt(1 - U^2) = {mean_sqrt_1_minus_U2}")

print(f"Variance of U = {var_U}")
print(f"Variance of e^U = {var_e_U}")
print(f"Variance of sqrt(1 - U^2) = {var_sqrt_1_minus_U2}")

print(f"Cov(U, e^U) = {cov_U_eU}")
print(f"Corr(U, e^U) = {corr_U_eU}")

print(f"Cov(U, sqrt(1 - U^2)) = {cov_U_sqrt_1_minus_U2}")
print(f"Corr(U, sqrt(1 - U^2)) = {corr_U_sqrt_1_minus_U2}")
```

```
Mean of U = 0.503709649152896
Mean of e^U = 1.7248547204467222
Mean of sqrt(1 - U^2) = 0.7829908728326228
Variance of U = 0.08357347476016408
Variance of e^U = 0.2440984164863993
Variance of sqrt(1 - U^2) = 0.049628407650909145
Cov(U, e^U) = 0.14172608451159385
Corr(U, e^U) = 0.992277350278174
Cov(U, sqrt(1 - U^2)) = -0.05970877639846067
Corr(U, sqrt(1 - U^2)) = -0.927126023024029
```

good training haha