Stochastic Simulations: Homework 1

Nathan Simonis

November 24, 2020

Abstract

The objective of this homework assignment is to apply and implement quasi monte carlo methods in the valuation of financial derivatives.

Methods

Crude Monte Carlo (CMC)

The objective of this method is to estimate the mean μ of a random variable Z that depends on a stochastic process S (Nobile, 2020). Using the CMC method, this value can be estimated by generating N i.i.d random points $Z^{(1)}, ..., Z^{(N)}$ and then computing:

$$\hat{\mu}_N = \frac{1}{N} \sum_{i=1}^{N} Z^{(i)}$$

Starting from the central limit theorem, it is possible to give a confidence interval to our estimate given by:

$$\mathbb{I}_{\alpha,N} = \left[\hat{\mu}_N - c_{1-\alpha/2} \frac{\sigma}{\sqrt{N}}, \hat{\mu}_N + c_{1-\alpha/2} \frac{\sigma}{\sqrt{N}} \right]$$

Quasi Monte Carlo (QMC)

As the error of the above method has a convergence of order $\mathcal{O}\left(N^{-\frac{1}{2}}\right)$. The QMC method seeks to reduce the latter by ensuring that the process that governs Z is a set of quasi-random points $\mathcal{P}_N$ (Kroese et al., 2011). As this does not allow us to compute an error term, we apply a random shift (RQMC) to $\mathcal{P}_N$. We do this by forming the shifted point sets

$$\mathcal{P}_N^{(i)} = (\mathcal{P}_N + Z_i) \bmod 1, i = 1, ..., \kappa$$

where $Z^{(1)}, ..., Z^{(\kappa)}$ are random vectors.

Our estimated mean then becomes:

$$\hat{\mu}_{\text{QMC}}^{(i)} = \sum_{u \in \mathcal{P}_N^{(i)}} \psi(u), i = 1, ..., \kappa$$

thus $\hat{\mu}$ becomes

$$\hat{\mu}_{\text{QMC}} = \frac{1}{\kappa}\sum_{i=1}^{\kappa}\hat{\mu}_{\text{QMC}}^{(i)}$$

and the confidence interval:

$$\mathbb{I}_\alpha = \left[\hat{\mu}_{\text{QMC}} - c_{1-\alpha/2}\frac{\sigma}{\sqrt{\kappa}}, \hat{\mu}_{\text{QMC}} + c_{1-\alpha/2}\frac{\sigma}{\sqrt{\kappa}}\right]$$

Asian call option

The payoff of such an option can be described as:

$$\Psi_j(S_{t_0}, S_{t_1}, \ldots, S_{t_m}) = P_j\left(\frac{1}{m}\sum_{i=1}^{m} S_{t_i} - K\right), j = 1, 2$$

a.) $P_1(\zeta) = \max(\zeta, 0)$

Using the formulas described in Methods, we can compute the mean and the error of $P_1(\zeta)$. In order for our results to be reproducible, we set a seed of 999 for the pseudo-random number generation and set $N = 10000, \alpha = 5\%, \kappa = 20$. The results obtained for monte carlo and randomized quasi monte carlo are shown in the table. It can be seen that the standard error is smaller for the second method.

	$\hat{\mu}$	$\mathbb{I}_\alpha$	SE
CMC	0.4527	+-0.0236	0.0120
RQMC	0.4813	+-0.0211	0.0108

We then repeat the same process for several values of N. This allows us to see the evolution of the standard error as a function of N (left) and the evolution of the mean as a function of N (right).

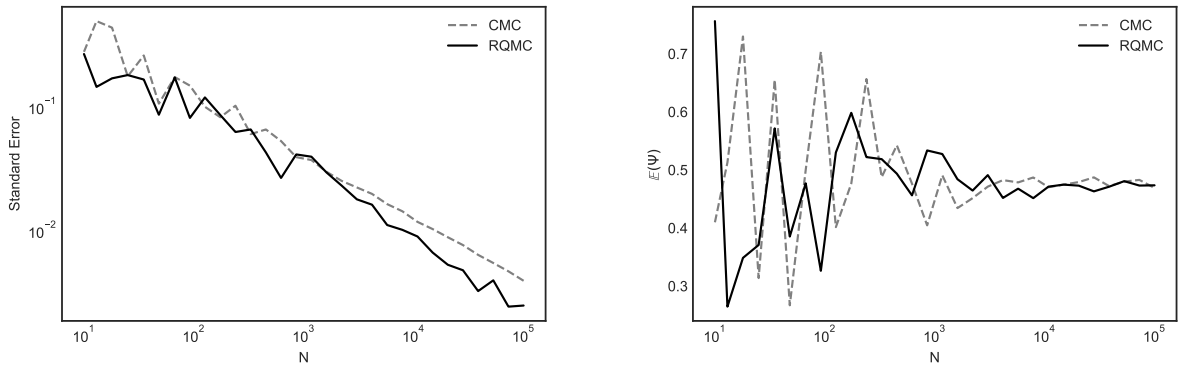

Figure 1: Standard Error and $\hat{\mu}$ against N

It can be seen that the standard error goes down a little faster with the QMC method. I would seem that the convergence rate is thus faster. The QMC therefore allows us to improve the CMC method.

b.) $P_2(\zeta) = \frac{\log\left(1+e^{\beta\zeta}\right)}{\beta}$, $\beta > 0$.

We repeat the same process as for the previous payoff. Several values are used for $\beta = [1, 2, 5, 10, 20]$.

β	$\hat{\mu}$MC	ciMC	seMC	$\hat{\mu}$QMC	ciQMC	seQMC
1	0.6490	0.0241	0.0123	0.6559	0.0212	0.0108
2	0.5160	0.0242	0.0124	0.5378	0.0143	0.0073
5	0.5013	0.0251	0.0128	0.4660	0.0159	0.0081
10	0.4800	0.0250	0.0128	0.4661	0.0131	0.0067
20	0.4738	0.0244	0.0125	0.4734	0.0114	0.0058

As this payoff function is a smooth approximation of the initial payoff, we can see that the more β increases, the closer $\hat{\mu}$ gets to the initial undiscounted option price. There is also a lower standard error when using the QMC method.

As in the previous subsection, we plot the evolution of the standard error and the estimate $\hat{\mu}$ as a function of N for each β. Once again, a faster convergence can be observed for the QMC, regardless of the β value.

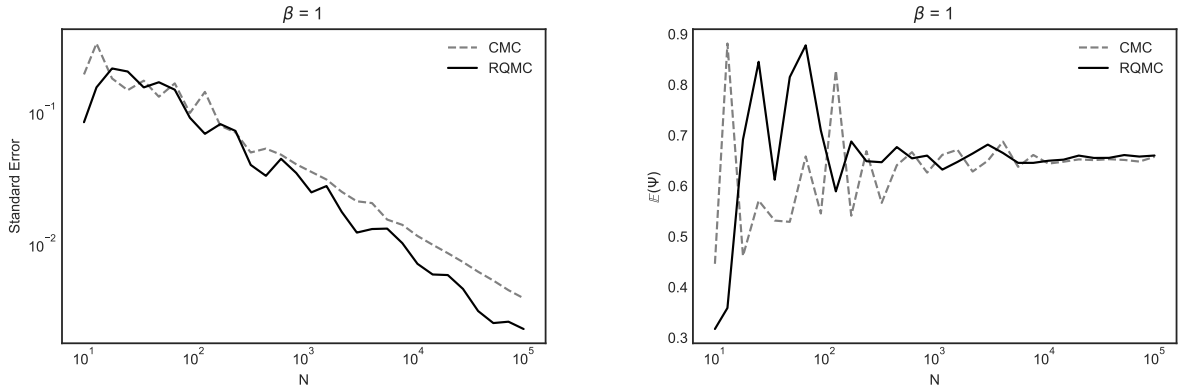

Figure 2: Standard Error and $\hat{\mu}$ against N for $\beta = 1$

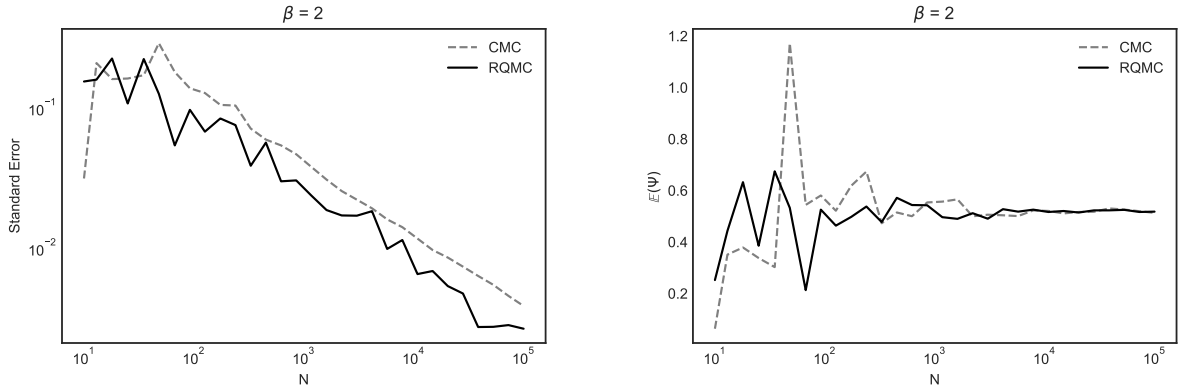

Figure 3: Standard Error and $\hat{\mu}$ against N for $\beta = 2$

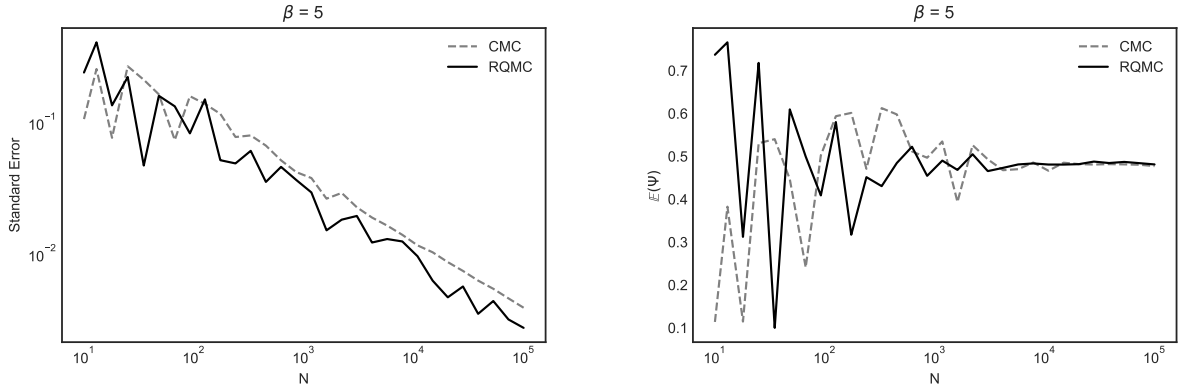

Figure 4: Standard Error and $\hat{\mu}$ against N for $\beta = 5$

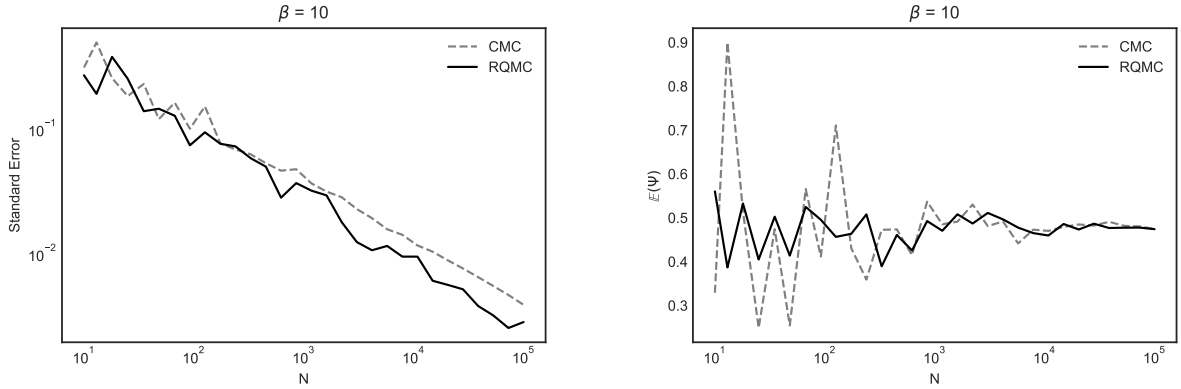

Figure 5: Standard Error and $\hat{\mu}$ against N for $\beta = 10$

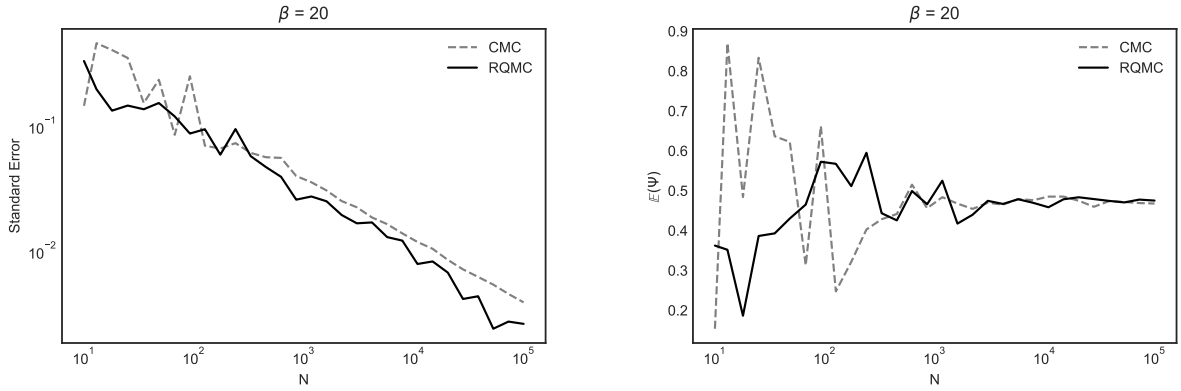

Figure 6: Standard Error and $\hat{\mu}$ against N for $\beta = 20$

Asian binary option

By repeating the same procedure as in the previous exercise, the same conclusions are reached.

The payoff function of such an options contract can be described as:

$$\tilde{\Psi}_j(S_{t_0}, S_{t_1}, \ldots, S_{t_m}) = P_j \left(\frac{1}{m} \sum_{i=1}^{m} S_{t_i} - K \right), j = 1, 2$$

a.) $\tilde{\Psi}_1(\zeta) = 20 \times \mathbb{I}_{\{\zeta > 0\}}$

	$\hat{\mu}$	CI	SE
CMC	4.704	0.1663	0.0848
RQMC	4.812	0.1272	0.0649

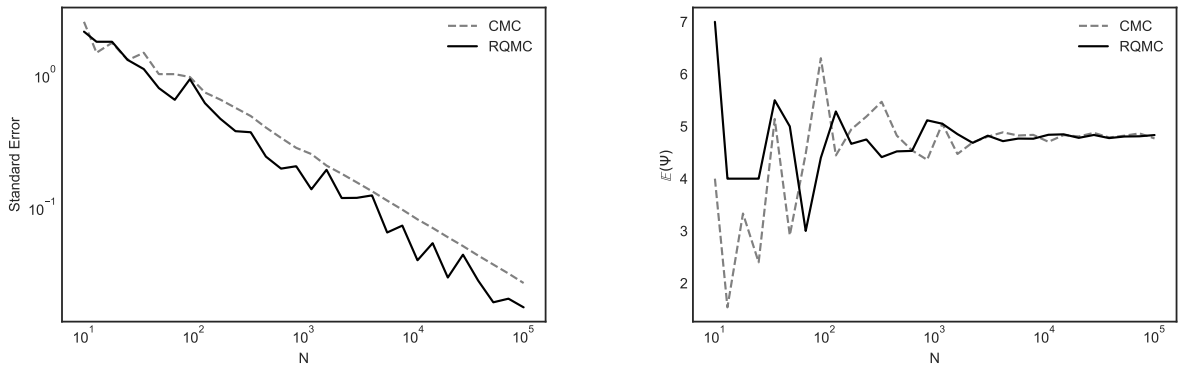

Figure 7: Standard Error and $\hat{\mu}$ against N

b.) $\tilde{\Psi}_2(\zeta) = 20\,(1 + exp(-2\gamma\zeta))^{-1}$

This time we vary γ, it takes the same values of $/beta$ in the previous exercise. We arrive at the same findings.

γ	$\hat{\mu}$MC	ciMC	seMC	$\hat{\mu}$QMC	ciQMC	seQMC
1	5.0689	0.1432	0.0730	5.1008	0.0591	0.0302
2	4.8637	0.1549	0.0790	4.9308	0.1026	0.0524
5	4.9509	0.1639	0.0836	4.8248	0.1016	0.0518
10	4.8708	0.1656	0.0845	4.7584	0.1098	0.0560
20	4.7692	0.1659	0.0846	4.7916	0.0938	0.0479

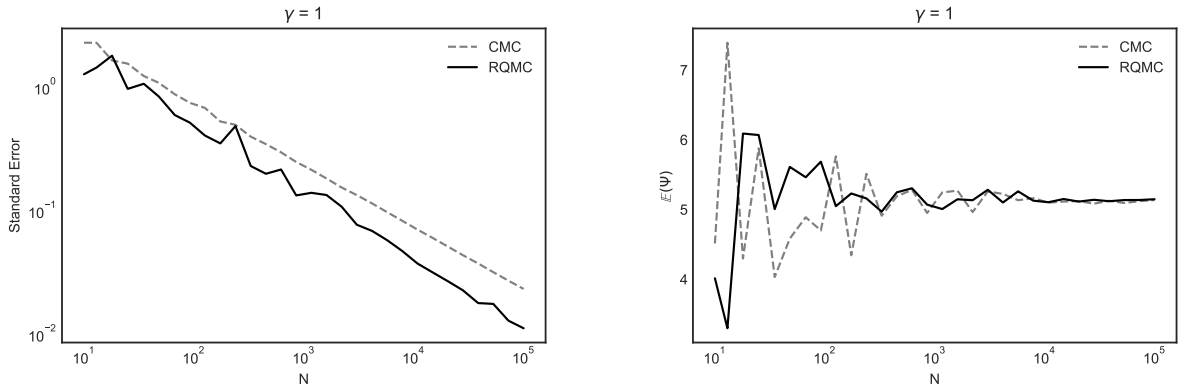

Figure 8: Standard Error and $\hat{\mu}$ against N for γ=1

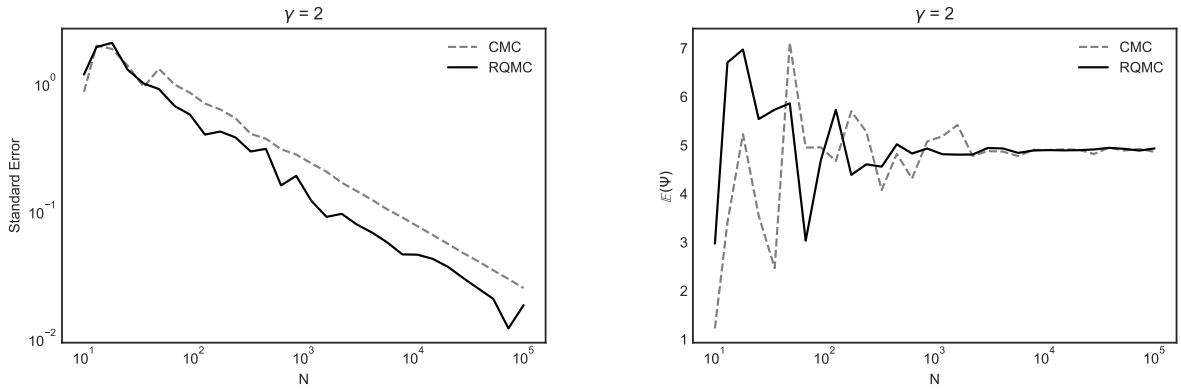

Figure 9: Standard Error and $\hat{\mu}$ against N for γ=2

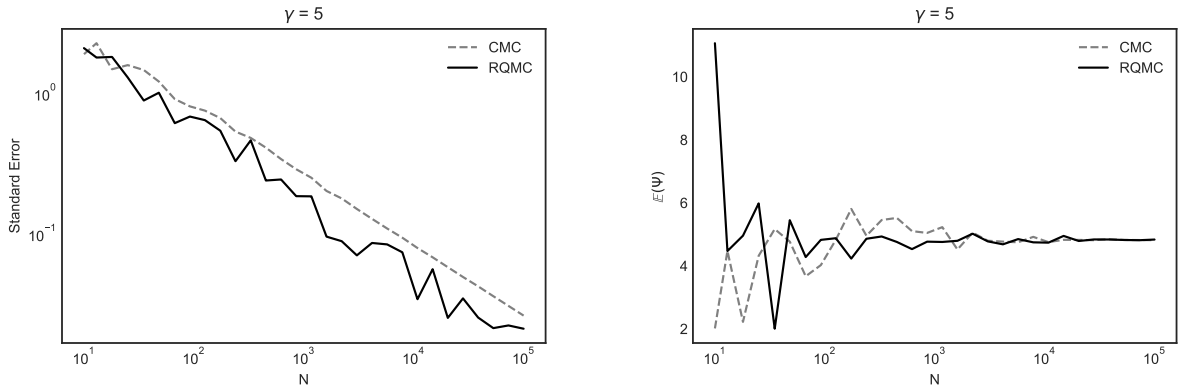

Figure 10: Standard Error and $\hat{\mu}$ against N for γ=5

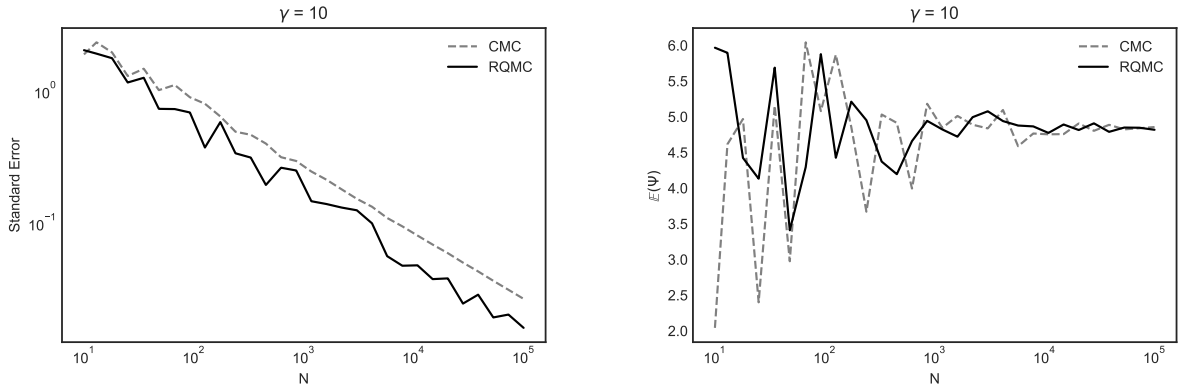

Figure 11: Standard Error and $\hat{\mu}$ against N for γ=10

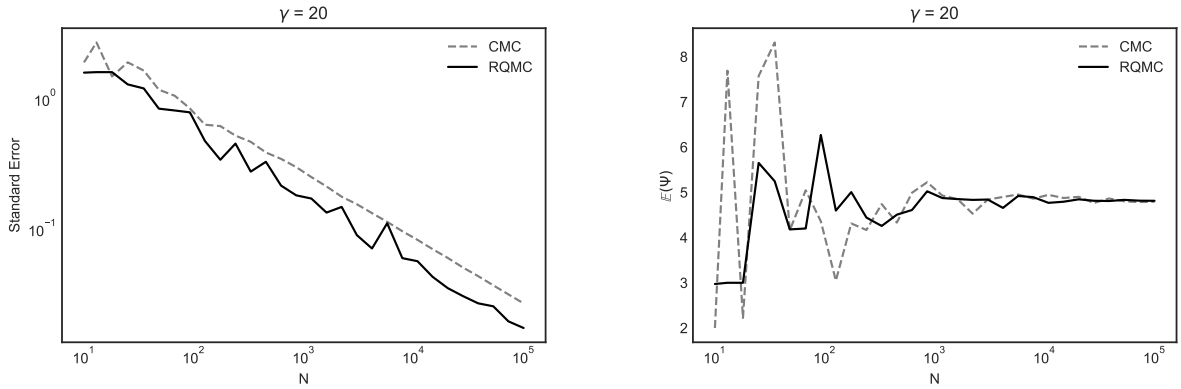

Figure 12: Standard Error and $\hat{\mu}$ against N for γ=20

References

Kroese, D. P., Taimre, T., & Botev, Z. (2011). Handbook of monte carlo methods.
Nobile, F. (2020). Lecture notes : Math-414. *EPFL*.

Appendix A Python Code

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st
import pandas as pd
import sobol_new as sn
from pathlib import Path as P
import utils

np.random.seed(999)

plt.style.use("seaborn-white")

# ============================================================================
# Parameters
# ============================================================================

m = 256 # Number of steps
r = 0.5 # Risk-free rate
sigma = 0.3 # Volatility
T = 2 # Time to maturity
S0 = 5 # Initial stock price
K = 10 # Strike

t = np.linspace(0,T,m) # Time grid

# Simulations
n_simulations = 10000

# Confidence intervals
alpha = 0.05
crit_val = st.t.ppf(1-alpha/2, n_simulations-1)

kappa = 20 # K splits for RQMC

# ============================================================================
# Compute \mathbb{E}(\Psi) for P_1
# ============================================================================
```

```
# MC
muMC, stdMC, seMC = utils.runSimMC(m, r, sigma, t, T, S0, n_simulations,
                                   K, utils.asian, par=None, payoff="p1")

# QMC

muQMC, stdQMC, seQMC = utils.runSimQMC(m, r, sigma, t, T, S0, n_simulations, K,
                                       kappa, utils.asian, par=None, payoff="p1")

res_df = pd.DataFrame({"mu":[muMC, muQMC],
                       "CI":[crit_val*seMC, crit_val*seQMC],
                       "SE":[seMC, seQMC]})

res_df.index = ["CMC", "RQMC"]
print(res_df)
print(res_df.round(4).to_latex())

# # ================================================================================
# # Compute \mathbb{E}(\Psi) for P_2 and different values of \beta
# # ================================================================================

BETA = [1, 2, 5, 10, 20]

res_df = {"muMC":[],"ciMC":[], "seMC":[],
          "muQMC":[],"ciQMC":[], "seQMC":[]}

for i in range(len(BETA)):

    muMC, stdMC, seMC = utils.runSimMC(m, r, sigma, t, T, S0, n_simulations,
                                       K, utils.asian, par=BETA[i], payoff="p2")

    muQMC, stdQMC, seQMC = utils.runSimQMC(m, r, sigma, t, T, S0, n_simulations, K,
                                           kappa, utils.asian, par=BETA[i], payoff="p2")

    ciMC = crit_val*seMC
    ciQMC = crit_val*seQMC

    res_df["muMC"].append(muMC)
    res_df["ciMC"].append(ciMC)
    res_df["seMC"].append(seMC)

    res_df["muQMC"].append(muQMC)
    res_df["ciQMC"].append(ciQMC)
    res_df["seQMC"].append(seQMC)
```

```
res_df = pd.DataFrame(res_df)
res_df.index = BETA
print(res_df)
print(res_df.round(4).to_latex())

# ============================================================================
# Generate plots for £\Psi_1£
# ============================================================================

N = np.logspace(1, 5, 30).astype(np.int)

res_df = {"muMC":[], "seMC":[],
          "muQMC":[], "seQMC":[]}

for n in N:
    muMC, stdMC, seMC = utils.runSimMC(m, r, sigma, t, T, S0, n,
                               K, utils.asian, par=None, payoff="p1")

    muQMC, stdQMC, seQMC = utils.runSimQMC(m, r, sigma, t, T, S0, n, K,
                                    kappa, utils.asian, par=None, payoff="p1")

    res_df["muMC"].append(muMC)
    res_df["seMC"].append(seMC)
    res_df["muQMC"].append(muQMC)
    res_df["seQMC"].append(seQMC)

res_df = pd.DataFrame(res_df)

# Error as function of N

plt.plot(N, res_df["seMC"], label="CMC", linestyle="--", color="grey")
plt.plot(N, res_df["seQMC"], label="RQMC", color="black")
plt.legend()
plt.xscale("log")
plt.yscale("log")
plt.xlabel("N")
plt.ylabel("Standard Error")
plt.savefig("figures/Ex1/Ex1_SEvsN_psi1.pdf")
plt.show()

# mu as function of N

plt.plot(N, res_df["muMC"], label="CMC", linestyle="--", color="grey")
plt.plot(N, res_df["muQMC"], label="RQMC", color="black")
plt.legend()
```

```
plt.xscale("log")
plt.xlabel("N")
plt.ylabel(r"$\mathbb{E}(\Psi)$")
plt.savefig("figures/Ex1/Ex1_MUvsN_psi1.pdf")
plt.show()

# ============================================================================
# Generate plots for £\Psi_2£
# ============================================================================
BETA = [1, 2, 5, 10, 20]
N = np.logspace(1, 5, 30).astype(np.int)

for i in range(len(BETA)): #Loop \beta

    res_df = {"muMC":[], "seMC":[],
          "muQMC":[], "seQMC":[]}
    for n in N: #Loop N
        muMC, stdMC, seMC = utils.runSimMC(m, r, sigma, t, T, S0, n,
                                K, utils.asian, par=BETA[i], payoff="p2")

        muQMC, stdQMC, seQMC = utils.runSimQMC(m, r, sigma, t, T, S0, n, K,
                                        kappa, utils.asian, par=BETA[i], payoff="p2")

        res_df["muMC"].append(muMC)
        res_df["seMC"].append(seMC)
        res_df["muQMC"].append(muQMC)
        res_df["seQMC"].append(seQMC)

    res_df = pd.DataFrame(res_df)

    # Error as function of N

    plt.plot(N, res_df["seMC"], label="CMC", linestyle="--", color="grey")
    plt.plot(N, res_df["seQMC"], label="RQMC", color="black")
    plt.legend()
    plt.xscale("log")
    plt.yscale("log")
    plt.xlabel("N")
    plt.ylabel("Standard Error")
    plt.title(r"$\beta$"+f" = {BETA[i]}")
    plt.savefig(f"figures/Ex1/Ex1_SEvsN_psi2_beta{BETA[i]}.pdf")
    plt.show()

    # mu as function of N
```

```
    plt.plot(N, res_df["muMC"], label="CMC", linestyle="--", color="grey")
    plt.plot(N, res_df["muQMC"], label="RQMC", color="black")
    plt.legend()
    plt.xscale("log")
    plt.xlabel("N")
    plt.ylabel(r"$\mathbb{E}(\Psi)$")
    plt.title(r"$\beta$"+f" = {BETA[i]}")
    plt.savefig(f"figures/Ex1/Ex1_MUvsN_psi2_beta{BETA[i]}.pdf")
    plt.show()
```

Listing 1: Tables and plots for Asian call option.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st
import pandas as pd
import sobol_new as sn
from pathlib import Path as P
import utils

np.random.seed(999)

plt.style.use("seaborn-white")

# ============================================================================
# Parameters
# ============================================================================

m = 256 # Number of steps
r = 0.5 # Risk-free rate
sigma = 0.3 # Volatility
T = 2 # Time to maturity
S0 = 5 # Initial stock price
K = 10 # Strike

t = np.linspace(0,T,m) # Time grid

# Simulations
n_simulations = 10000

# Confidence intervals
alpha = 0.05
crit_val = st.t.ppf(1-alpha/2, n_simulations-1)

kappa = 20 # K splits for RQMC
```

```
# ============================================================================
# Compute \mathbb{E}(\Psi) for P_1
# ============================================================================
# MC
muMC, stdMC, seMC = utils.runSimMC(m, r, sigma, t, T, S0, n_simulations,
                                   K, utils.binary_asian, par=None, payoff="p1")

# QMC

muQMC, stdQMC, seQMC = utils.runSimQMC(m, r, sigma, t, T, S0, n_simulations, K,
                                       kappa, utils.binary_asian, par=None, payoff="p1")

res_df = pd.DataFrame({"mu":[muMC, muQMC],
                       "CI":[crit_val*seMC, crit_val*seQMC],
                       "SE":[seMC, seQMC]})

res_df.index = ["CMC", "RQMC"]
print(res_df)
print(res_df.round(4).to_latex())

# # ============================================================================
# # Compute \mathbb{E}(\Psi) for P_2 and different values of \gamma
# # ============================================================================

GAMMA = [1, 2, 5, 10, 20]

res_df = {"muMC":[],"ciMC":[], "seMC":[],
          "muQMC":[],"ciQMC":[], "seQMC":[]}

for i in range(len(GAMMA)):

    muMC, stdMC, seMC = utils.runSimMC(m, r, sigma, t, T, S0, n_simulations,
                                       K, utils.binary_asian, par=GAMMA[i], payoff="p2")

    muQMC, stdQMC, seQMC = utils.runSimQMC(m, r, sigma, t, T, S0, n_simulations, K,
                                           kappa, utils.binary_asian, par=GAMMA[i], payoff="p2")

    ciMC = crit_val*seMC
    ciQMC = crit_val*seQMC

    res_df["muMC"].append(muMC)
    res_df["ciMC"].append(ciMC)
    res_df["seMC"].append(seMC)
```

```
    res_df["muQMC"].append(muQMC)
    res_df["ciQMC"].append(ciQMC)
    res_df["seQMC"].append(seQMC)

res_df = pd.DataFrame(res_df)
res_df.index = GAMMA
print(res_df)
print(res_df.round(4).to_latex())

# ============================================================================
# Generate plots for £\Psi_1£
# ============================================================================

N = np.logspace(1, 5, 30).astype(np.int)

res_df = {"muMC":[], "seMC":[],
          "muQMC":[], "seQMC":[]}

for n in N:
    muMC, stdMC, seMC = utils.runSimMC(m, r, sigma, t, T, S0, n,
                                K, utils.binary_asian, par=None, payoff="p1")

    muQMC, stdQMC, seQMC = utils.runSimQMC(m, r, sigma, t, T, S0, n, K,
                                    kappa, utils.binary_asian, par=None, payoff="p1")

    res_df["muMC"].append(muMC)
    res_df["seMC"].append(seMC)
    res_df["muQMC"].append(muQMC)
    res_df["seQMC"].append(seQMC)

res_df = pd.DataFrame(res_df)

# Error as function of N

plt.plot(N, res_df["seMC"], label="CMC", linestyle="--", color="grey")
plt.plot(N, res_df["seQMC"], label="RQMC", color="black")
plt.legend()
plt.xscale("log")
plt.yscale("log")
plt.xlabel("N")
plt.ylabel("Standard Error")
plt.savefig("figures/Ex2/Ex2_SEvsN_psi1.pdf")
plt.show()

# mu as function of N
```

```
plt.plot(N, res_df["muMC"], label="CMC", linestyle="--", color="grey")
plt.plot(N, res_df["muQMC"], label="RQMC", color="black")
plt.legend()
plt.xscale("log")
plt.xlabel("N")
plt.ylabel(r"$\mathbb{E}(\Psi)$")
plt.savefig("figures/Ex2/Ex2_MUvsN_psi1.pdf")
plt.show()

# ============================================================================
# Generate plots for £\Psi_2£
# ============================================================================
GAMMA = [1, 2, 5, 10, 20]
N = np.logspace(1, 5, 30).astype(np.int)

for i in range(len(GAMMA)): #Loop \GAMMA

    res_df = {"muMC":[], "seMC":[],
          "muQMC":[], "seQMC":[]}
    for n in N: #Loop N
        muMC, stdMC, seMC = utils.runSimMC(m, r, sigma, t, T, S0, n,
                                K, utils.binary_asian, par=GAMMA[i], payoff="p2")

        muQMC, stdQMC, seQMC = utils.runSimQMC(m, r, sigma, t, T, S0, n, K,
                                        kappa, utils.binary_asian, par=GAMMA[i], payoff="p

        res_df["muMC"].append(muMC)
        res_df["seMC"].append(seMC)
        res_df["muQMC"].append(muQMC)
        res_df["seQMC"].append(seQMC)

    res_df = pd.DataFrame(res_df)

    # Error as function of N

    plt.plot(N, res_df["seMC"], label="CMC", linestyle="--", color="grey")
    plt.plot(N, res_df["seQMC"], label="RQMC", color="black")
    plt.legend()
    plt.xscale("log")
    plt.yscale("log")
    plt.xlabel("N")
    plt.ylabel("Standard Error")
    plt.title(r"$\gamma$"+f" = {GAMMA[i]}")
    plt.savefig(f"figures/Ex2/Ex2_SEvsN_psi2_gamma{GAMMA[i]}.pdf")
```

```
    plt.show()

    # mu as function of N

    plt.plot(N, res_df["muMC"], label="CMC", linestyle="--", color="grey")
    plt.plot(N, res_df["muQMC"], label="RQMC", color="black")
    plt.legend()
    plt.xscale("log")
    plt.xlabel("N")
    plt.ylabel(r"$\mathbb{E}(\Psi)$")
    plt.title(r"$\gamma$"+f" = {GAMMA[i]}")
    plt.savefig(f"figures/Ex2/Ex2_MUvsN_psi2_gamma{GAMMA[i]}.pdf")
    plt.show()
```

Listing 2: Tables and plot for Binary Asian option.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st
import pandas as pd
import sobol_new as sn

def simGBM(m, r, sigma, t, T, S0, n_simulations, dW):
    """Generate path for stock price"""
    dt = T/m
    W = np.cumsum(dW*np.sqrt(dt), axis=1)
    t = np.broadcast_to(t, shape=(n_simulations,m))
    X = (r-0.5*sigma**2)*t + sigma*W
    S = S0 * np.exp(X)
    S[:,0] = S0
    return S

def runSimMC(m, r, sigma, t, T, S0, n_simulations,
             K, func, par=None, payoff="p1"):
    """Run simulations for crude MC"""
    dW = np.random.normal(size=(n_simulations, m))
    S = simGBM(m, r, sigma, t, T, S0, n_simulations, dW)
    zeta = np.mean(S, axis=1) - K
    psi = func(zeta, par, payoff)

    muMC = np.mean(psi)
    stdMC = np.std(psi)
    seMC = stdMC/np.sqrt(n_simulations)

    return muMC, stdMC, seMC
```

```
def runSimQMC(m, r, sigma, t, T, S0, n_simulations,
              K, kappa, func, par=None, payoff="p1"):
    """Run simulations for Randomly shifted QMC"""
    n_simulations = int(np.ceil(n_simulations / kappa))
    sobol_seq = sn.generate_points(n_simulations, m)
    MU = np.zeros(kappa)

    for i in range(kappa):
        U = np.random.uniform(size=(m,))
        U = np.broadcast_to(U, shape=(n_simulations, m))

        shifted_seq = np.add(sobol_seq, U) % 1

        dW = st.norm.ppf(shifted_seq)

        S = simGBM(m, r, sigma, t, T, S0, n_simulations, dW)
        zeta = np.mean(S, axis=1) - K
        psi = func(zeta, par, payoff)
        MU[i] = np.mean(psi)

    muQMC = np.mean(MU)
    stdQMC = np.std(MU)
    seQMC = stdQMC / np.sqrt(kappa)

    return muQMC, stdQMC, seQMC


def asian(zeta, beta = None, payoff = "p1"):
    """Payoff functions for asian options"""
    if payoff == "p1":
        return np.maximum(zeta, 0)
    elif payoff == "p2":
        return np.log(1+np.exp(beta*zeta))/beta

def binary_asian(zeta, gamma = None, payoff = "p1"):
    """Payoff functions for binary options"""
    if payoff == "p1":
        return 20 * (1.0*(zeta>0))
    elif payoff == "p2":
        return 20*(1+np.exp(-2*gamma*zeta))**(-1)
```

Listing 3: Helper functions for CMC and QMC simulations.