Assignment 4

David Fleischer
MACF 402 - Mathematical & Computational Finance II

November 25, 2015

1 **Question 1:** Crude-Monte Carlo and Antithetic Variables

Given a European put option written on an underlying asset such that $S_0 = 100, K = 40, r = 0.0175, \sigma = 0.4$, and 1 year to maturity, we have the theoretical Black-Scholes price

$$V_P^{BS}(S, K, r, \sigma, t, T) = \$0.0816128$$

Performing simulations of the underlying asset's lognormal dynamics we are also able to determine an estimated price for the European price within a desired confidence level (95% two-tailed confidence). Table 1 shows the results of the simulation of N random variates for a crude estimator and $N/2$ random variates generated for an antithetic estimator. Figures 1, 2, and 3 visualizes this information, showing the convergence towards the Black-Scholes prices for each increase in N. We note the consistent superiority of the antithetic estimator over the crude counterpart: The antithetic estimator shows greater estimator efficiency for a sufficiently large number of simulations.

Table 1: Results of crude and antithetic estimation for N and $N/2$ random variates generated.

N Sim.	Est.	CI High	Mean	CI Low	CI Width	Var.	Time	Eff.
1e+02	Crude	0.4841250	0.2017650	-0.0805959	0.5647209	2.075370	0.0001	0.4818418
1e+03	Crude	0.1383060	0.0891728	0.0400395	0.0982665	0.628405	0.0006	1.5913304
1e+04	Crude	0.0972041	0.0814618	0.0657195	0.0314846	0.645096	0.0051	1.5501569
1e+05	Crude	0.0893403	0.0841762	0.0790121	0.0103282	0.694182	0.0499	1.4405444
1e+06	Crude	0.0837244	0.0821020	0.0804796	0.0032448	0.685173	0.4911	1.4594854
1e+07	Crude	0.0823945	0.0818853	0.0813761	0.0010184	0.674906	4.8813	1.4816878
1e+08	Crude	0.0818029	0.0816420	0.0814812	0.0003217	0.673345	48.8796	1.4851228
1e+02	Antithetic	0.2542560	0.0984075	-0.0574408	0.3116968	0.316127	0.0000	3.1632856
1e+03	Antithetic	0.1735840	0.1052780	0.0369724	0.1366116	0.607258	0.0003	1.6467465
1e+04	Antithetic	0.0995359	0.0828844	0.0662328	0.0333031	0.360883	0.0031	2.7709812
1e+05	Antithetic	0.0909729	0.0857646	0.0805564	0.0104165	0.353057	0.0288	2.8324038
1e+06	Antithetic	0.0841459	0.0825204	0.0808948	0.0032511	0.343917	0.2714	2.9076783
1e+07	Antithetic	0.0833184	0.0828047	0.0822910	0.0010274	0.343442	2.7730	2.9116998
1e+08	Antithetic	0.0832796	0.0831166	0.0829537	0.0003259	0.345542	27.7890	2.8940042

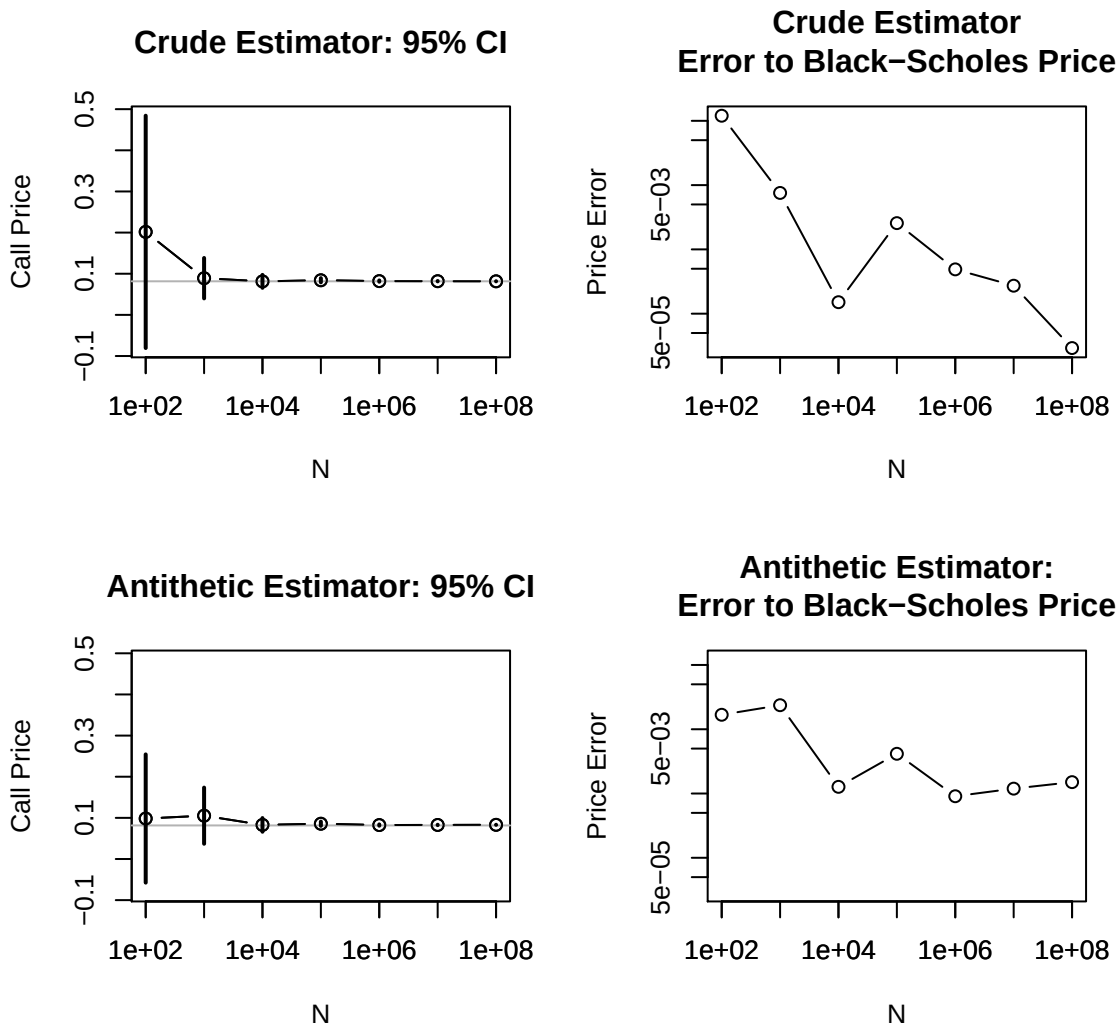

Figure 1: (Top) Crude simulation of a European put option written on the asset described above. (Bottom) Antithetic simulation of the same European put option. The true (Black-Scholes) is plotted in gray. Note the convergence towards the Black-Scholes price and reduction in variance using the antithetic estimator over the crude estimator.

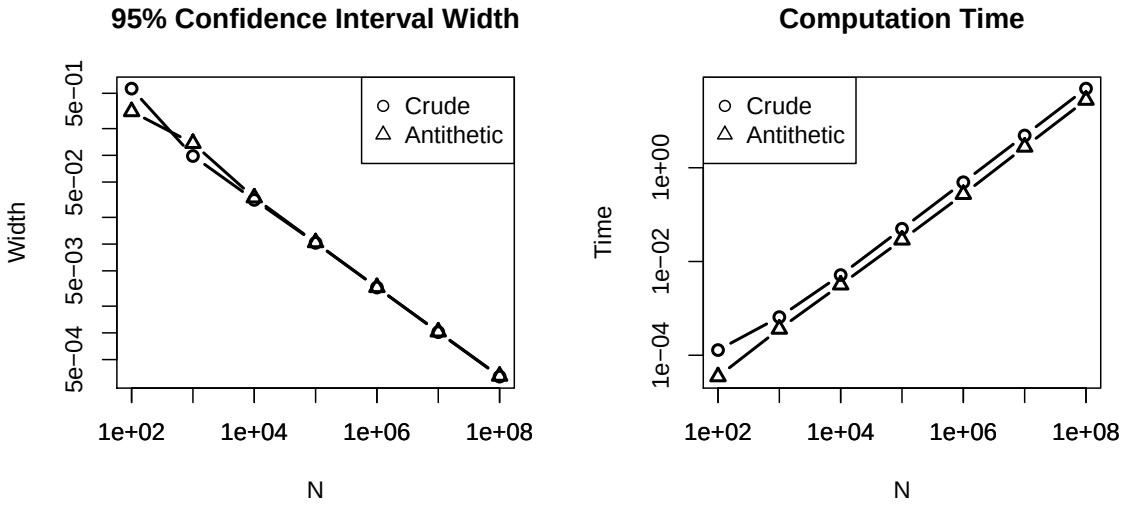

Figure 2: European put option: (Left) Confidence intervals tighten for both crude and antithetic estimators at approximately the same (linear rate). (Right) Computation time increases linearly as a function of N for both crude and antithetic estimators.

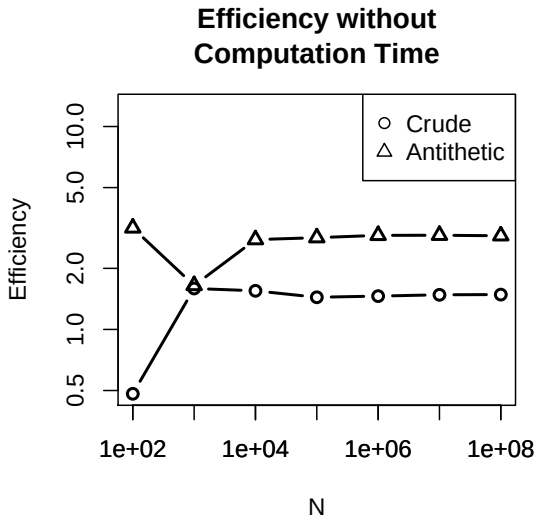

Figure 3: European put option: The antithetic estimator is persistently more efficient than the crude estimator for N sufficiently large.

2 Question 2: Geometric Asian Options

2.1 Problem 2 (a)

Show that

$$\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0 = \sum_{i=1}^{n} \frac{i}{n}\left[(r - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t} Z_{n-i+1}\right]$$
$$= \frac{(n+1)}{2}(r - \frac{1}{2}\sigma^2)\Delta t + \sum_{i=1}^{n} \sigma_i Z_i$$

where $\{Z_i\}_{i=1}^n$ are i.i.d. standard normal random variables and $\sigma_i = \frac{i}{n}\sigma\sqrt{\Delta t}$.

Solution 2 (a). We have that

$$\begin{aligned} S_{t_1} &= S_0 e^{(\cdots)} \\ S_{t_2} = S_{t_1} e^{(\cdots)} &= S_0 e^{(\cdots)} e^{(\cdots)} \\ S_{t_3} = S_{t_2} e^{(\cdots)} &= S_0 e^{(\cdots)} e^{(\cdots)} e^{(\cdots)} \\ &\vdots \\ S_{t_n} &= S_0 \prod_{i=1}^{n} e^{(\cdots)} \end{aligned}$$

We recognize that the number of exponential terms is $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. Hence

$$\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}} = \left[S_0^n \prod_{i=1}^{\frac{n(n+1)}{2}} e^{\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t} Z_i}\right]^{\frac{1}{n}}$$
$$= S_0 \exp\left[\frac{1}{n} \sum_{i=1}^{\frac{n(n+1)}{2}} \left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t} Z_i\right]$$

So,

$$\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) = \log S_0 + \frac{1}{n} \sum_{i=1}^{\frac{n(n+1)}{2}} \left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t} Z_i$$
$$= \log S_0 + \frac{1}{n}\frac{n(n+1)}{2}\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \frac{1}{n} \sum_{i=1}^{\frac{n(n+1)}{2}} \sigma\sqrt{\Delta t} Z_i$$
$$= \log S_0 + \frac{(n+1)}{2}\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \frac{1}{n} \sum_{i=1}^{\frac{n(n+1)}{2}} \sigma\sqrt{\Delta t} Z_i$$

From properties of the normal distribution we note that

$$\begin{aligned}\sum_{i=1}^{\frac{n(n+1)}{2}} \sigma\sqrt{\Delta t}Z_i &\sim N\left(0, \sum_{i=1}^{\frac{n(n+1)}{2}} \sigma^2\Delta t\right)\\ &\sim N\left(0, \frac{n(n+1)}{2}\sigma^2\Delta t\right)\\ &\sim N\left(0, \sum_{i=1}^{n} i\sigma^2\Delta t\right)\\ &\sim \sum_{i=1}^{n} i\sigma\sqrt{\Delta t}Z_i\end{aligned}$$

Thus,

$$\begin{aligned}\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) &= \log S_0 + \frac{(n+1)}{2}\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \frac{1}{n}\sum_{i=1}^{\frac{n(n+1)}{2}} \sigma\sqrt{\Delta t}Z_i\\ &= \log S_0 + \frac{(n+1)}{2}\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \frac{1}{n}\sum_{i=1}^{n} i\sigma\sqrt{\Delta t}Z_i\end{aligned}$$

Letting $\sigma_i := \frac{i}{n}\sigma\sqrt{\Delta t}$ we have

$$\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0 = \frac{(n+1)}{2}\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sum_{i=1}^{n}\sigma_i Z_i$$

as desired.

2.2 Problem 2 (b)

Show that $\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0$ has normal distribution with mean

$$(r - \frac{1}{2}\sigma^2)\frac{(n+1)}{2n}T$$

and variance

$$\frac{\sigma^2(n+1)(2n+1)}{6n^2}T$$

Solution 2 (b). We have that

$$\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0 = \frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\Delta t + \sum_{i=0}^{n}\sigma_i Z_i$$

Thus

$$\begin{aligned}\mathbb{E}\left[\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0\right] &= \mathbb{E}\left[\frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\Delta t + \sum_{i=0}^{n}\sigma_i Z_i\right]\\ &= \frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\Delta t + \sum_{i=0}^{n}\sigma_i\mathbb{E}[Z_i] \quad \text{(by linearity)}\end{aligned}$$

but $Z_i \sim N(0,1) \implies \mathbb{E}[Z_i] = 0$, hence

$$\mathbb{E}\left[\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0\right] = \frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\Delta t$$

We have that $n\Delta t = T \iff \Delta t = \frac{T}{n}$, thus

$$\begin{aligned}\mathbb{E}\left[\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0\right] &= \frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\frac{T}{n} \\ &= \Big(r - \frac{1}{2}\sigma^2\Big)\frac{(n+1)}{2n}T\end{aligned}$$

as desired. Now, for the variance, we use the identity $\mathrm{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}^2[X]$. Computing these terms we get

$$\mathbb{E}^2[X] = \mathbb{E}^2\left[\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0\right] = \left(\Big(r - \frac{1}{2}\sigma^2\Big)\frac{(n+1)}{2n}T\right)^2$$

and

$$\begin{aligned}\mathbb{E}[X^2] &= \mathbb{E}\left[\left(\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0\right)^2\right] = \mathbb{E}\left[\left(\frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\Delta t + \sum_{i=0}^{n}\sigma_i Z_i\right)^2\right] \\ &= \mathbb{E}\left[\left(\frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\Delta t\right)^2 + \frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\Delta t\sum_{i=0}^{n}\sigma_i Z_i + \left(\sum_{i=0}^{n}\sigma_i Z_i\right)^2\right] \\ &= \left(\frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\Delta t\right)^2 + \frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\Delta t\sum_{i=0}^{n}\sigma_i\mathbb{E}\big[Z_i\big] + \mathbb{E}\left[\left(\sum_{i=0}^{n}\sigma_i Z_i\right)^2\right] \\ &= \left(\frac{(n+1)}{2}\Big(r - \frac{1}{2}\sigma^2\Big)\Delta t\right)^2 + \mathbb{E}\left[\left(\sum_{i=0}^{n}\sigma_i Z_i\right)^2\right]\end{aligned}$$

Thus, we see that the $\left(\frac{(n+1)}{2}\big(r - \frac{1}{2}\sigma^2\big)\Delta t\right)^2$ terms cancel and are left with

$$\begin{aligned}\mathrm{Var}\left[\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0\right] &= \mathbb{E}\left[\left(\sum_{i=0}^{n}\sigma_i Z_i\right)^2\right] \\ &= \mathbb{E}\left[\left(\sum_{i=0}^{n} X_i\right)^2\right] \quad \text{for } X_i \sim N(0,\sigma_i^2)\end{aligned}$$

To compute this term we use the result that

$$\sum_{i=1}^{n} X_i \sim N\left(0, \sum_{i=1}^{n}\sigma_i^2\right)$$

and for $X \sim N(0,\sigma^2)$ we also have the result that

$$\mathbb{E}[X^n] = \begin{cases} 0 & n \text{ odd} \\ 1 \cdot 3 \cdot \cdots \cdot (n-3) \cdot (n-1) \cdot \sigma^{\frac{n}{2}} & n \text{ even} \end{cases}$$

Thus,

$$\begin{aligned}
\mathrm{Var}\left[\log\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}}\right) - \log S_0\right] = \mathbb{E}\left[\left(\sum_{i=1}^{n} X_i\right)^2\right] &= \sum_{i=1}^{n} \sigma_i^2 \\
&= \left(\sum_{i=1}^{n} \frac{i}{n}\sigma\sqrt{\Delta t}\right)^2 \\
&= \sigma^2 \Delta t \sum_{i=1}^{n} \frac{i^2}{n^2} \\
&= \sigma^2 \Delta t \frac{n(n+1)(2n+1)}{6n^2} \\
&= \frac{\sigma^2(n+1)(2n+1)}{6n^2} T
\end{aligned}$$

as desired.

2.3 Problem 2 (c)

Show that the time-zero price of the time-T expiry geometric average Asian call option with strike price K is

$$C_0^{GA} = e^{-rT}\Big(S_0 e^{\hat{\mu}T}\Phi(\hat{d}_1) - K\Phi(\hat{d}_2)\Big)$$

where

$$\begin{aligned}
\hat{\mu} &= \Big(r - \frac{1}{2}\sigma^2\Big)\frac{(n+1)}{2n} + \frac{1}{2}\hat{\sigma}^2, \quad \hat{\sigma}^2 = \frac{\sigma^2(n+1)(2n+1)}{6n^2} \\
\hat{d}_1 &= \frac{\log\frac{S_0}{K} + (\hat{\mu} + \frac{1}{2}\hat{\sigma}^2)T}{\hat{\sigma}\sqrt{T}} \\
\hat{d}_2 &= \hat{d}_1 - \hat{\sigma}\sqrt{T}
\end{aligned}$$

Solution 2 (c). We note that

$$\begin{aligned}
\frac{S_{t_i}}{S_{t_{i-1}}} &= \frac{S_{t_{i-1}} e^{(r-\frac{1}{2}\sigma^2)\Delta t + \sigma(B_{t_{i-1}} - B_{t_{i-2}})}}{S_{t_{i-1}}} \\
&= e^{(r-\frac{1}{2}\sigma^2)\Delta t + \sigma(B_{t_i} - B_{t_{i-1}})} \\
\implies \log\frac{S_{t_i}}{S_{t_{i-1}}} &= \left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\left(B_{t_i} - B_{t_{i-1}}\right) \\
\implies \log\frac{S_{t_i}}{S_{t_{i-1}}} &\sim N\left(\left(r - \frac{1}{2}\sigma^2\right)\Delta t, \sigma^2\Delta t\right)
\end{aligned}$$

Thus

$$\begin{aligned}
\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}} &= \left[S_0^n \frac{S_{t_1}^n}{S_0^n}\frac{S_{t_2}^{n-1}}{S_{t_1}^{n-1}} \cdots \frac{S_{t_{n-1}}^2}{S_{t_{n-2}}^2}\frac{S_{t_n}}{S_{t_{n-1}}}\right]^{\frac{1}{n}} \\
&= S_0\left[\frac{S_{t_1}^n}{S_0^n}\frac{S_{t_2}^{n-1}}{S_{t_1}^{n-1}} \cdots \frac{S_{t_{n-1}}^2}{S_{t_{n-2}}^2}\frac{S_{t_n}}{S_{t_{n-1}}}\right]^{\frac{1}{n}}
\end{aligned}$$

Letting $X_i = \log \frac{S_{t_i}}{S_{t_{i-1}}}$ we have $\left[\left(\frac{S_{t_i}}{S_{t_{i-1}}}\right)^k\right]^{\frac{1}{n}} = \exp\left[\left(\frac{k}{n}\right) X_i\right]$, so

$$\begin{aligned}\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}} &= S_0 \exp\left[\left(\frac{n}{n}\right) X_1 + \left(\frac{n-1}{n}\right) X_2 + \cdots + \left(\frac{2}{n}\right) X_{n-1} + \left(\frac{1}{n}\right) X_n\right] \\ &= S_0 \exp\left[\frac{1}{n}\sum_{k=1}^{n}(n-k+1)X_k\right]\end{aligned}$$

But

$$\begin{aligned}\sum_{k=1}^{n}(n-k+1)X_k &= n^2\sum_{k=1}^{n} X_k - \frac{n(n+1)}{2}\sum_{k=1}^{n} X_k + n\sum_{k=1}^{n} X_k \\ &= \left[n - \frac{(n+1)}{2} + 1\right] n\sum_{k=1}^{n} X_k \\ &= \left[\frac{2n-(n+1)+2}{2}\right] n\sum_{k=1}^{n} X_k \\ &= \frac{n(n+1)}{2}\sum_{k=1}^{n} X_k \\ &= \sum_{k=1}^{n} kX_k\end{aligned}$$

Thus,

$$\begin{aligned}\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}} &= S_0 \exp\left[\frac{1}{n}\sum_{k=1}^{n}(n-k+1)X_k\right] \\ &= S_0 \exp\left[\frac{1}{n}\sum_{k=1}^{n} kX_k\right]\end{aligned}$$

Recalling that $X_i \sim N\left(\left(r - \frac{1}{2}\sigma^2\right)\Delta t, \sigma^2\Delta t\right)$ we use the result that, for $W_i \sim N(\mu_i, \sigma_i^2)$ and $c_i \in \mathbb{R}$,

$$\sum_{i=1}^{n} c_i W_i \sim N\left(\sum_{i=1}^{n} c_i\mu_i, \sum_{i=1}^{n} c_i^2\sigma_i^2\right)$$

Hence

$$\begin{aligned}\frac{1}{n}\sum_{k=1}^{n} kX_k &\sim N\left(\frac{1}{n}\left(r - \frac{1}{2}\sigma^2\right)\Delta t\sum_{k=1}^{n} k, \frac{1}{n^2}\sigma^2\Delta t\sum_{k=1}^{n} k^2\right) \\ &\sim N\left(\frac{1}{n}\left(r - \frac{1}{2}\sigma^2\right)\Delta t\frac{(n+1)}{2}, \frac{1}{n^2}\sigma^2\Delta t\frac{n(n+1)(2n+1)}{6}\right)\end{aligned}$$

But $\Delta t \equiv \frac{T}{n}$, so

$$\begin{aligned}&\frac{1}{n}\sum_{k=1}^{n} kX_k \sim N\left(\left(r - \frac{1}{2}\sigma^2\right)\frac{(n+1)}{2n}T, \frac{\sigma^2(n+1)(2n+1)}{6n^2}T\right) \\ \implies &\frac{1}{n}\sum_{k=1}^{n} kX_k = \left(r - \frac{1}{2}\sigma^2\right)\frac{(n+1)}{2n}T + \sqrt{\frac{(n+1)(2n+1)}{6n^2}}\sigma\sqrt{T}Z \quad \text{for } Z \sim N(0,1)\end{aligned}$$

Therefore

$$\begin{aligned}\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}} &= S_0 \exp\left[\frac{(n+1)}{2}\sum_{k=1}^{n} X_k\right] \\ &= S_0 \exp\left[\left(r-\frac{1}{2}\sigma^2\right)\frac{(n+1)}{2}T + \sqrt{\frac{(n+1)(2n+1)}{6n^2}}\sigma\sqrt{T}Z\right]\end{aligned}$$

We recognize that our difficult problem of pricing a geometric Asian option has been reduced to the the easy problem of pricing a European call option written on an asset with lognormal dynamics

$$\hat{S}_T = S_0 e^{\left(r-\frac{1}{2}\sigma^2\right)\frac{(n+1)}{2n}T + \sqrt{\frac{(n+1)(2n+1)}{6n^2}}\sigma\sqrt{T}Z}$$

for Z a standard normal random variable. Thankfully we know how to price European contingent claims on the process $\hat{S}$ with the above dynamics. Simplifying the notation we let

$$\begin{aligned}\hat{\sigma}^2 &= \frac{\sigma^2(n+1)(2n+1)}{6n^2} \\ \hat{\mu} &= \left(r-\frac{1}{2}\sigma^2\right)\frac{(n+1)}{2n} + \frac{1}{2}\hat{\sigma}^2\end{aligned}$$

Thus,

$$\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}} = \hat{S}_T = S_0 e^{\left(\hat{\mu}-\frac{1}{2}\hat{\sigma}^2\right)T + \hat{\sigma}\sqrt{T}Z}$$

This is precisely the form desired for applying the risk neutral pricing formula to determine the Black-Scholes price of a European contingent claim. Hence,

$$\begin{aligned}C_0^{GA} &= \mathbb{E}_{\mathbb{Q}}\left[e^{-rT}\left(\left[\prod_{i=1}^{n} S_{t_i}\right]^{\frac{1}{n}} - K\right)^+\right] \\ &= \mathbb{E}_{\mathbb{Q}}\left[e^{-rT}\left(\hat{S}_T - K\right)^+\right] \\ &= \mathbb{E}_{\mathbb{Q}}\left[e^{-rT}\hat{S}_T \mathbb{1}_{\hat{S}_T>K}\right] - \mathbb{E}\left[e^{-rT}K\mathbb{1}_{\hat{S}_T>K}\right]\end{aligned}$$

For the binary option we recall that

$$\begin{aligned}\mathbb{E}\left[e^{-rT}K\mathbb{1}_{\hat{S}_T>K}\right] &= e^{-rT}K\mathbb{Q}\left(\hat{S}_T > K\right) \\ &= e^{-rT}K\mathbb{Q}\left(S_0 e^{\left(\hat{\mu}-\frac{1}{2}\hat{\sigma}^2\right)T+\hat{\sigma}\sqrt{T}Z} > K\right) \\ &= e^{-rT}K\mathbb{Q}\left(Z < \frac{\log\frac{S_0}{K} + \left(\hat{\mu}-\frac{1}{2}\hat{\sigma}^2\right)T}{\hat{\sigma}\sqrt{T}}\right) \\ &= e^{-rT}K\Phi[\hat{d}_2] \quad \text{for } \hat{d}_2 = \frac{\log\frac{S_0}{K} + \left(\hat{\mu}-\frac{1}{2}\hat{\sigma}^2\right)T}{\hat{\sigma}\sqrt{T}}\end{aligned}$$

and for the asset-or-nothing option we recall that

$$\begin{aligned}\mathbb{E}_{\mathbb{Q}}\left[e^{-rT}\hat{S}_T\mathbb{1}_{\hat{S}_T>K}\right] &= e^{-rT}\mathbb{E}_{\mathbb{Q}}\left[S_0e^{\left(\hat{\mu}-\frac{1}{2}\hat{\sigma}^2\right)T+\hat{\sigma}\sqrt{T}Z}\mathbb{1}_{\hat{S}_T>K}\right] \\ &= e^{-rT}S_0e^{\left(\hat{\mu}-\frac{1}{2}\hat{\sigma}^2\right)T}\mathbb{E}_{\mathbb{Q}}\left[e^{\hat{\sigma}\sqrt{T}Z}\mathbb{1}_{\hat{S}_T<K}\right]\end{aligned}$$

Manipulating the domain of the indicator function we get

$$\begin{aligned}\hat{S}_T > K \iff S_0e^{\left(\hat{\mu}-\frac{1}{2}\hat{\sigma}^2\right)T+\hat{\sigma}\sqrt{T}Z} &> K \\ \implies Z &> -\hat{d}_2\end{aligned}$$

Thus

$$\begin{aligned}\mathbb{E}_{\mathbb{Q}}\left[e^{-rT}\hat{S}_T\mathbb{1}_{\hat{S}_T>K}\right] &= e^{-rT}S_0e^{\left(\hat{\mu}-\frac{1}{2}\hat{\sigma}^2\right)T}\int_{-\hat{d}_2}^{\infty}\frac{1}{\sqrt{2\pi}}e^{\hat{\sigma}\sqrt{T}z}e^{-\frac{1}{2}z^2}\,dz \\ &= e^{-rT}S_0e^{\left(\hat{\mu}-\frac{1}{2}\hat{\sigma}^2\right)T}\int_{-\hat{d}_2}^{\infty}\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}\left(z-\hat{\sigma}\sqrt{T}\right)^2+\frac{1}{2}\hat{\sigma}^2T}\,dz \\ &= e^{-rT}S_0e^{\hat{\mu}T}\int_{-\hat{d}_2}^{\infty}\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}\left(z-\hat{\sigma}\sqrt{T}\right)^2}\,dz\end{aligned}$$

Performing the substitution

$$\begin{aligned}u = z-\hat{\sigma}\sqrt{T} \implies du &= dz \\ \therefore u\left(-\hat{d}_2\right) &= \frac{\log\frac{K}{S_0}-\left(\hat{\mu}-\frac{1}{2}\hat{\sigma}^2\right)T}{\hat{\sigma}\sqrt{T}}-\hat{\sigma}\sqrt{T} \\ &= \frac{\log\frac{K}{S_0}-\left(\hat{\mu}+\frac{1}{2}\hat{\sigma}^2\right)T}{\hat{\sigma}\sqrt{T}} \\ &= -\frac{\log\frac{S_0}{K}+\left(\hat{\mu}+\frac{1}{2}\hat{\sigma}^2\right)T}{\hat{\sigma}\sqrt{T}} \\ &= -\hat{d}_1\end{aligned}$$

Thus

$$\begin{aligned}\mathbb{E}_{\mathbb{Q}}\left[e^{-rT}\hat{S}_T\mathbb{1}_{\hat{S}_T>K}\right] &= e^{-rT}S_0e^{\hat{\mu}T}\int_{-\hat{d}_1}^{\infty}\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}u^2}\,dz \\ &= e^{-rT}S_0e^{\hat{\mu}T}\left(1-\Phi[-\hat{d}_1]\right) \\ &= e^{-rT}S_0e^{\hat{\mu}T}\Phi[\hat{d}_1]\end{aligned}$$

Finally, putting it all together we have

$$\begin{aligned}C_0^{GA} &= \mathbb{E}_{\mathbb{Q}}\left[e^{-rT}\hat{S}_T\mathbb{1}_{\hat{S}_T>K}\right]-\mathbb{E}\left[e^{-rT}K\mathbb{1}_{\hat{S}_T>K}\right] \\ &= e^{-rT}S_0e^{\hat{\mu}T}\Phi[\hat{d}_1]-e^{-rT}K\Phi[\hat{d}_2] \\ &= e^{-rT}\left(S_0e^{\hat{\mu}T}\Phi[\hat{d}_1]-K\Phi[\hat{d}_2]\right)\end{aligned}$$

as desired.

3 **Question 3:** Arithmetic Asian Option

We wish to price an arithmetic Asian call option written on an underlying asset such that $S_0 = 500, K = 500, r = 0.0175, \sigma = 0.25$, expiry in 1 year, and $n = 52$ monitoring dates. We note that no analytic solution exists and so we must implement some numerical methods to determine its value. We first simulate the price process of the underlying asset. Figure 4 shows us a sample of 20 paths (of 10^5 total simulated paths) generated from a lognormal price process

$$dS_t = rS_t\,dt + \sigma S_t\,dB_t$$

with 52 discretized time steps, in addition to the means and the 95% upper and lower confidence intervals at each monitoring point.

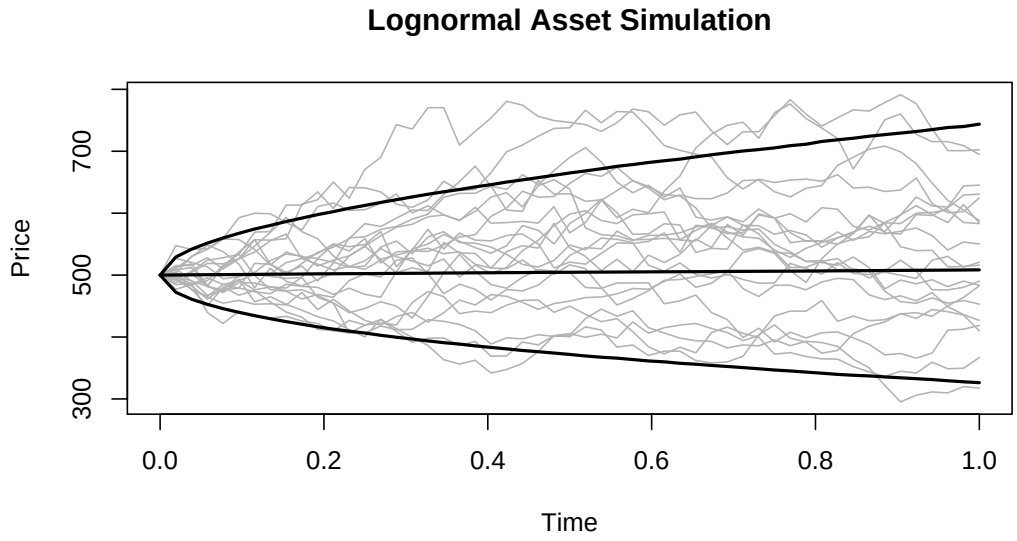

Figure 4: A sample of 20 from a total of 10^5 total lognormal price processes generated for our underlying asset, with the mean and 95% two-tailed confidence interval at each monitoring point plotted in black.

We now consider the price of a geometric Asian call option written on the above asset, whose price has a known analytic form. Table 2 shows the results of the simulation of N random variates for a crude estimator and $N/2$ variates for an antithetic estimator of the price option. Figures 5, 6, and 7 visualizes this information, showing the convergence towards the analytic price for increasing N. We note that the antithetic estimator is more efficiency than the crude estimator for all N.

Table 2: Geometric Asian call option value written on the underlying asset described above.

N Sim.	Est.	CI High	Mean	CI Low	CI Width	Var.	Geom. Price	Time	Eff.
1e+02	Crude	44.0150	34.5274	25.0397	18.9753	2343.1900	29.6485	0.0030	0.00042
1e+03	Crude	31.1410	28.4957	25.8503	5.2907	1821.5800	29.6485	0.0293	0.00054
1e+04	Crude	30.4829	29.5768	28.6707	1.8122	2137.2000	29.6485	0.2643	0.00046
1e+05	Crude	29.8871	29.5979	29.3086	0.5785	2178.0300	29.6485	2.6812	0.00045
1e+06	Crude	29.6936	29.6023	29.5111	0.1825	2168.0200	29.6485	26.4982	0.00046
1e+07	Crude	29.6784	29.6495	29.6206	0.0578	2170.5400	29.6485	278.1340	0.00046

N Sim.	Est.	CI High	Mean	CI Low	CI Width	Var.	Geom. Price	Time	Eff.
1e+08	Crude	29.6648	29.6556	29.6465	0.0183	2170.7400	29.6485	2681.8100	0.00046
1e+02	Anti.	36.1101	29.0803	22.0505	14.0596	643.1970	29.6485	0.0019	0.00155
1e+03	Anti.	32.6480	30.4409	28.2338	4.4142	634.0000	29.6485	0.0196	0.00157
1e+04	Anti.	30.2943	29.6012	28.9081	1.3862	625.2890	29.6485	0.1864	0.00159
1e+05	Anti.	29.9877	29.7646	29.5416	0.4461	647.7580	29.6485	1.7436	0.00154
1e+06	Anti.	29.7691	29.6986	29.6281	0.1410	646.3250	29.6485	17.8223	0.00154
1e+07	Anti.	29.6918	29.6695	29.6472	0.0446	645.9680	29.6485	180.3340	0.00154
1e+08	Anti.	29.6613	29.6543	29.6472	0.0141	645.4960	29.6485	1764.4300	0.00154

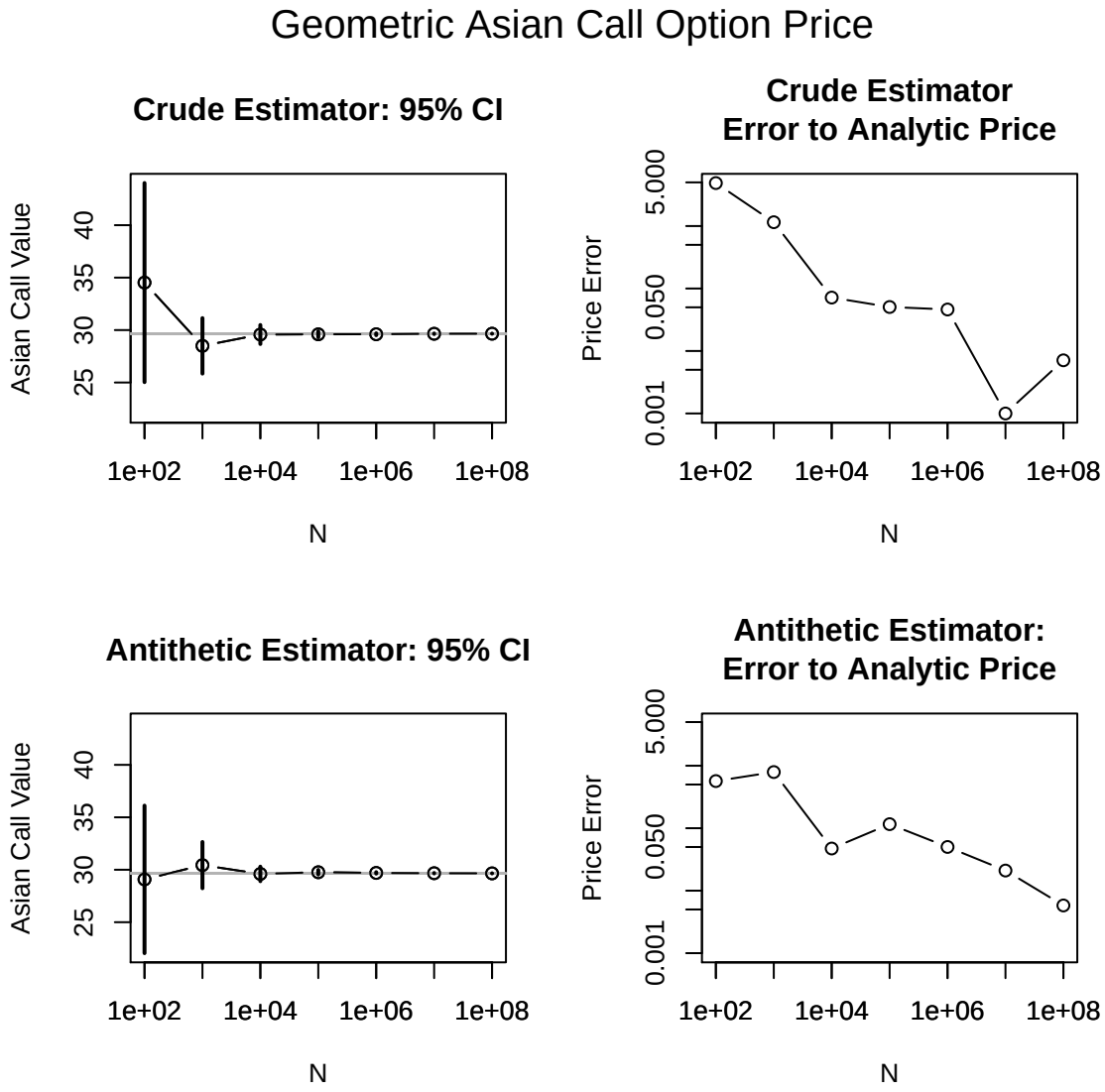

Figure 5: (Top) Crude simulation of a geometric Asian call option written on the above asset. (Bottom) Antithetic simulation of the same geometric Asian call. Note the convergence towards the true price (gray line) with increasing N.

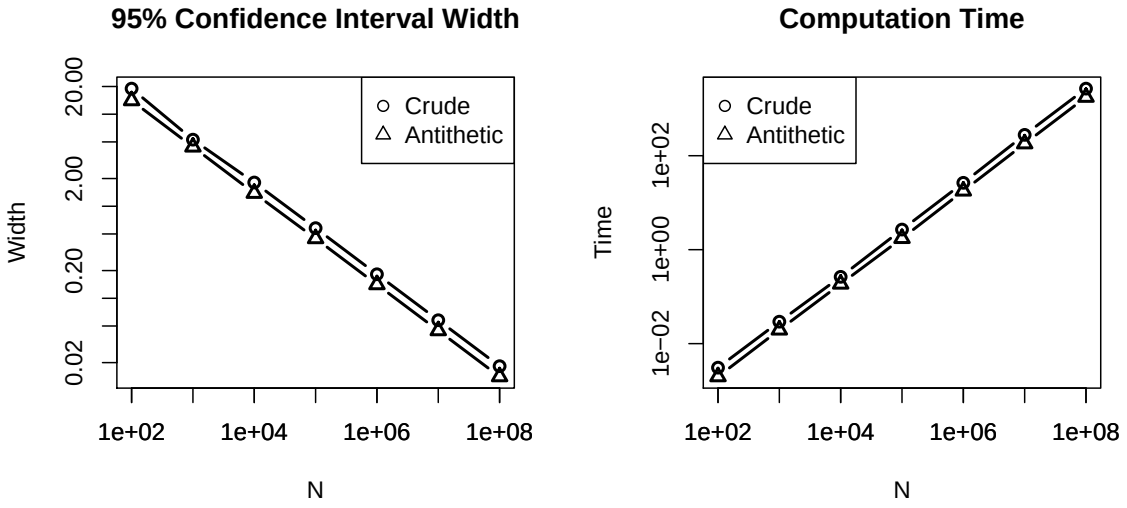

Figure 6: Geometric Asian call option: (Left) Confidence intervals tighten for both crude and antithetic estimators at approximately the same (linear) rate. The antithetic estimator has a persistently narrower interval. (Right) Computation time increases linearly as a function of N for both crude and antithetic estimators.

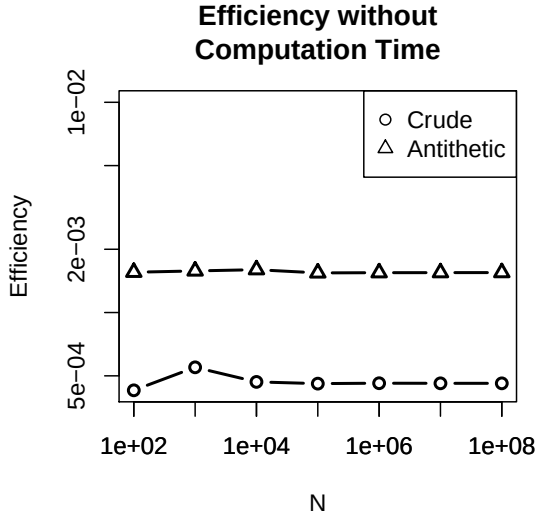

Figure 7: Geometric Asian call option: The antithetic estimator is persistently more efficient than the crude estimator for all N.

We now consider the same simulation for the price of an arithmetic Asian call option, which has no known analytic solution. In addition to the N crude and $N/2$ antithetic random variates, we also introduce a control variate estimator for the arithmetic Asian option using the geometric Asian option price, whose expected value has an analytic solution. That is, we use the control variate

$$X_{Control} = X - c\,(Y - \mathbb{E}\,[Y])$$

such that X is our arithmetic Asian call price, Y the geometric Asian call price, $\mathbb{E}[Y]$ the analytic solution for the geometric Asian call, and the optimal c, denoted c^*, is found by

$$c^* = -\frac{\text{Cov}[X, Y]}{\text{Var}[Y]}$$

For the control variate estimator we first pilot $m = 0.1 \cdot N$ simulation to estimate the optimal c^*. Using this c^* we then run the remaining $M = (N - m)$ simulations to estimate the arithmetic Asian call price itself. Table 3 shows the results of the simulation of N crude variates, $N/2$ antithetic variates, and $0.9 \cdot N$ control variates. Figures 8, 9, and 10 visualizes this information, showing the convergence towards the analytic price for increasing N. We note that while the antithetic estimator is more efficient than the crude estimator, the control variate estimator is profoundly more efficient than the other two.

Table 3: Arithmetic Asian call option value written on the underlying asset described above.

N Sim.	Est.	CI High	Mean	CI Low	CI Width	Var.	Geom. Price	Time	Eff.
1e+02	Crude	36.8093	28.0952	19.3811	17.4282	1976.67000	29.6485	0.0030	0.00050
1e+03	Crude	33.4532	30.3005	27.1478	6.3054	2587.41000	29.6485	0.0351	0.00038
1e+04	Crude	31.4971	30.5624	29.6277	1.8694	2274.15000	29.6485	0.2654	0.00043
1e+05	Crude	31.6320	31.3298	31.0276	0.6044	2377.54000	29.6485	2.7089	0.00042
1e+06	Crude	31.1667	31.0715	30.9763	0.1904	2360.10000	29.6485	26.5557	0.00042
1e+07	Crude	31.1435	31.1134	31.0832	0.0603	2362.48000	29.6485	271.6680	0.00042
1e+08	Crude	31.0947	31.0852	31.0757	0.0190	2360.32000	29.6485	2687.3900	0.00042
1e+02	Anti.	44.9713	37.3294	29.6875	15.2838	760.08000	29.6485	0.0019	0.00131
1e+03	Anti.	33.4747	31.1951	28.9155	4.5592	676.36700	29.6485	0.0183	0.00147
1e+04	Anti.	32.3841	31.6500	30.9158	1.4683	701.53200	29.6485	0.1816	0.00142
1e+05	Anti.	31.3778	31.1450	30.9122	0.4656	705.46100	29.6485	1.7370	0.00141
1e+06	Anti.	31.1151	31.0420	30.9689	0.1462	696.20700	29.6485	17.9376	0.00143
1e+07	Anti.	31.1096	31.0865	31.0634	0.0462	696.61500	29.6485	182.1540	0.00143
1e+08	Anti.	31.0951	31.0878	31.0805	0.0146	697.22400	29.6485	1769.9300	0.00143
1e+02	Control	31.6252	31.2186	30.8120	0.8132	3.87403	29.6485	0.0029	0.25812
1e+03	Control	31.2731	31.1505	31.0278	0.2453	3.52340	29.6485	0.0311	0.28381
1e+04	Control	31.1027	31.0693	31.0359	0.0668	2.61229	29.6485	0.2934	0.38280
1e+05	Control	31.0894	31.0788	31.0681	0.0213	2.65035	29.6485	3.0404	0.37730
1e+06	Control	31.0817	31.0783	31.0749	0.0068	2.64561	29.6485	29.7082	0.37798
1e+07	Control	31.0789	31.0779	31.0768	0.0021	2.61434	29.6485	306.4690	0.38250
1e+08	Control	31.0787	31.0784	31.0780	0.0007	2.61654	29.6485	3016.3400	0.38218

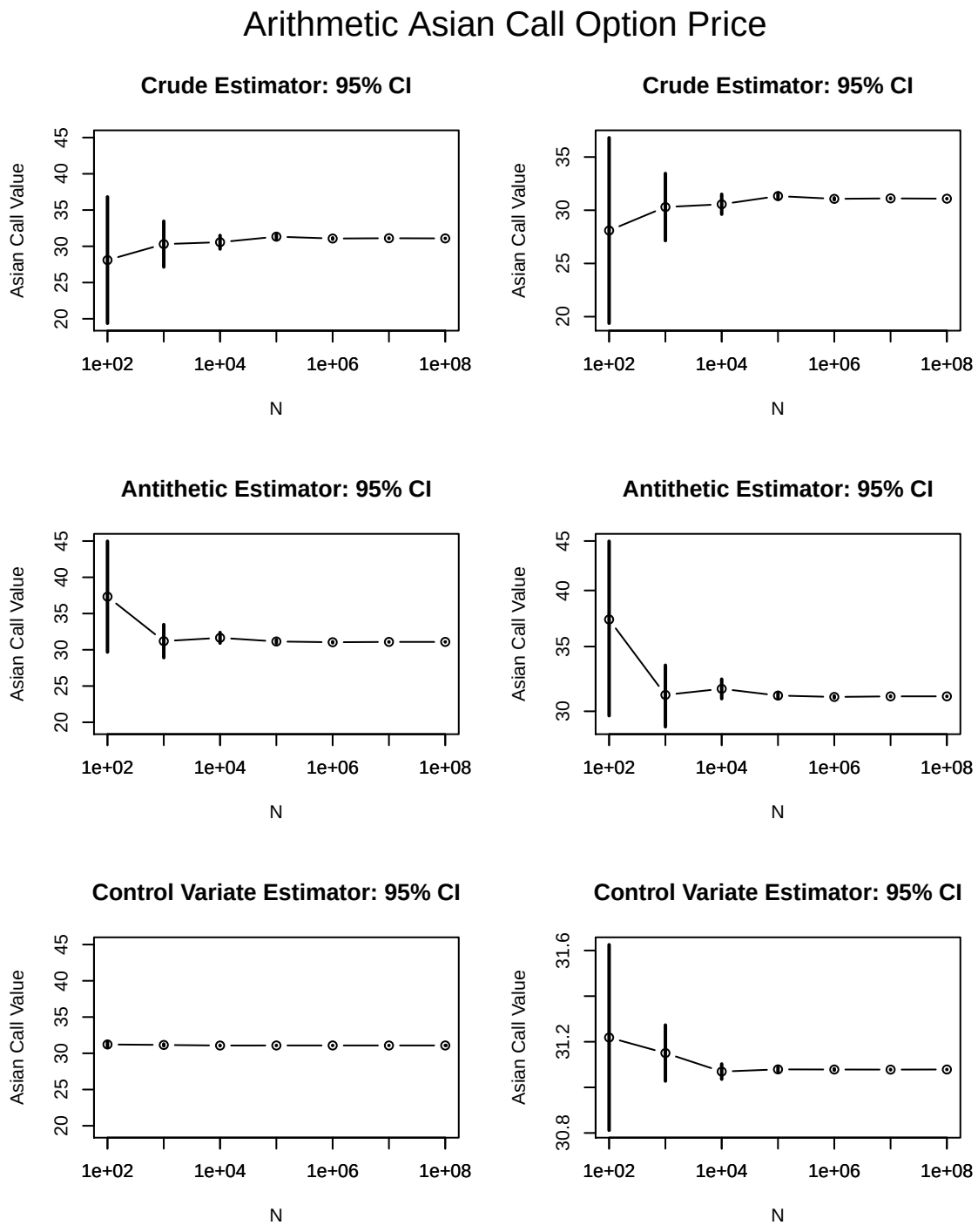

Figure 8: Arithmetic Asian call option: (Top) Crude simulation of a arithmetic Asian call option written on the above asset. (Middle) Antithetic simulation of the same arithmetic Asian call. (Bottom) Control variate simulation of the arithmetic Asian call.

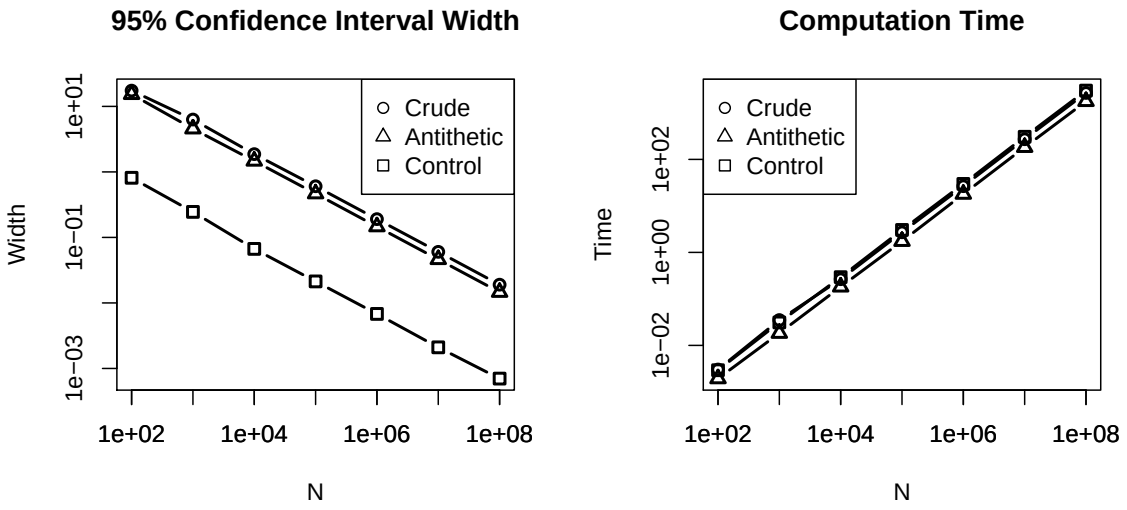

Figure 9: Arithmetic Asian call option: (Left) Confidence interval widths tighten linearly with increasing N. Note the dramatic reduction in width using the control variate estimator. (Right) Computation time increases linearly as a function of N for all estimators.

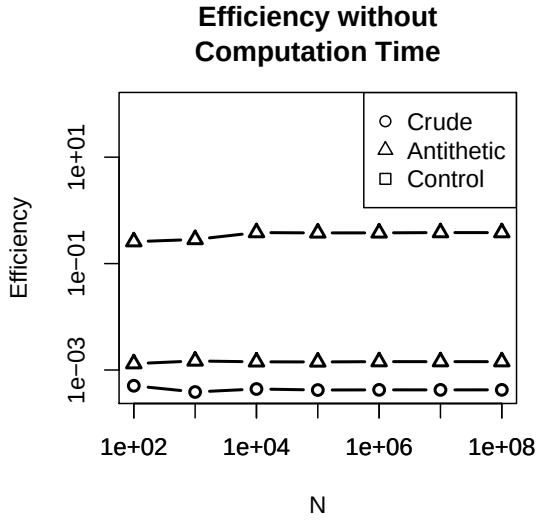

Figure 10: Arithmetic Asian call option: Although the antithetic variate is persistently more efficient than the crude estimator, we see a dramatic increase in efficiency using the control variate estimator.

4 **Question 4:** Heston Stochastic Volatility Model

We wish to simulate an asset using the Heston stochastic volatility model

$$dS_t = rS_t\,dt + \sqrt{v_t}S_t\left[\rho\,dB_t^{(1)} + \sqrt{1-\rho^2}\,dB_t^{(2)}\right]$$
$$dv_t = \kappa(\theta - v_t)\,dt + \sigma\sqrt{v_t}\,dB_t^{(1)}$$

We generate paths from the Heston model such that $S_0 = 100, v_0 = 0.1, T = 0.5$, and $n = 125$ discretization points using an Euler-discretization. Figure 11 shows us a sample of 20 paths (from 10^4 total simulated paths) generated from the discretization of the Heston model. Using the same volatility process and identical Brownian increments we see the sample price processes for different ρ, in addition the sample means and the 95% two-tailed confidence intervals at each discretized point.

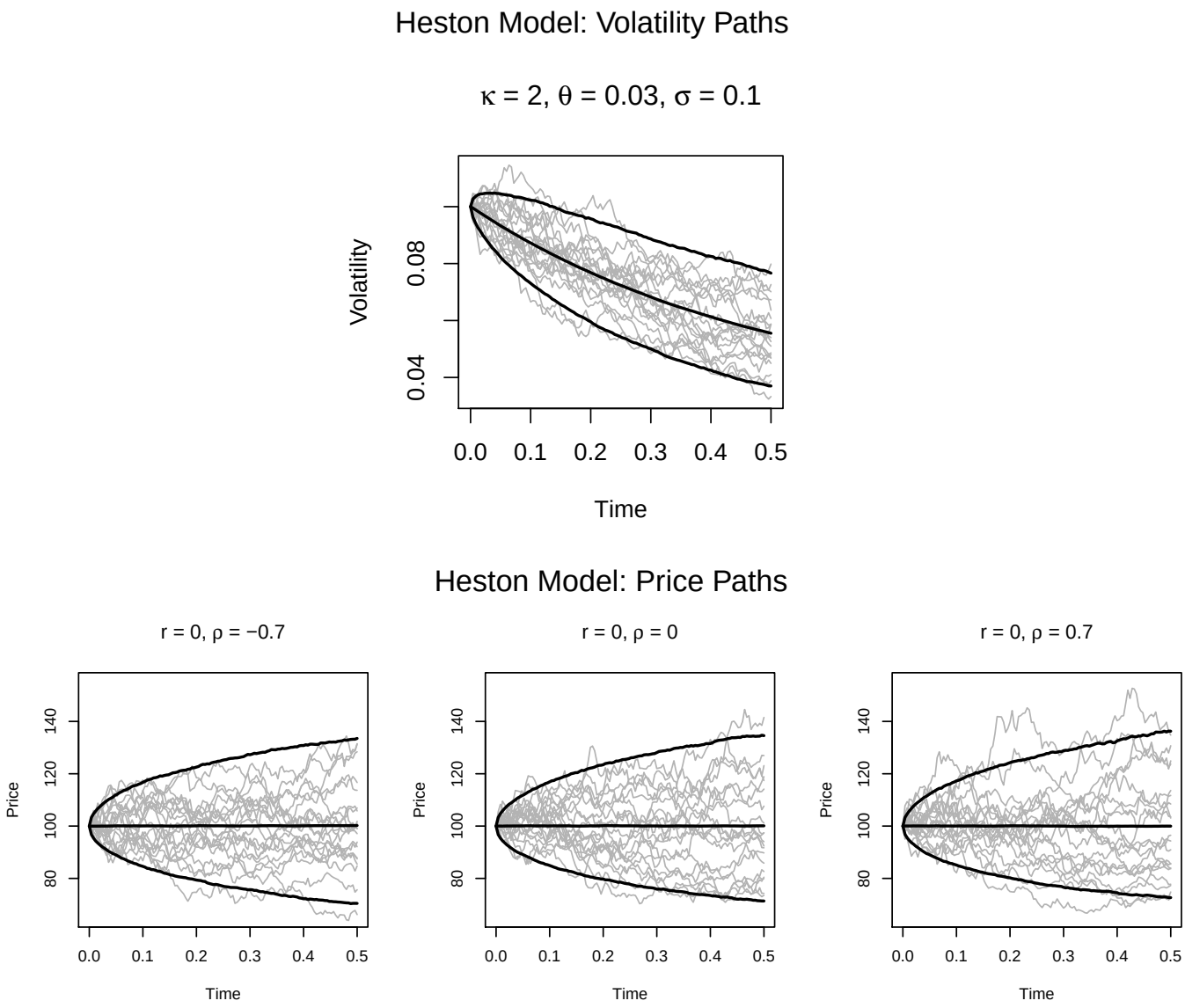

Figure 11: (Top) A sample of 20 from 10^4 total volatility processes. (Bottom) A sample of 20 from a total of 10^4 total price processes for different values of ρ. Means and 95% two-tailed confidence intervals are also plotted (black) at each discretized point.

4.1 European Call Value – Crude Estimator

With our discretized price process in place we wish to find the value of a European call option with strike price $K = S_0 = 100$. We first use a crude estimator to determine its value. Table 4 shows the results of the simulation of N random variates for a crude estimator. We visualize this information below in Figure 12 when comparing the results to that of the conditional estimator.

Table 4: Crude estimator of the value of a European call option written on an underlying asset with dynamics described by the Heston stochastic volatility model above.

N Sim.	rho	CI High	Mean	CI Low	CI Width	Var.	Time	Eff.
1e+02	-0.7000	10.13080	7.49356	4.85634	5.27446	181.0430	0.012430	0.005523550

N Sim.	rho	CI High	Mean	CI Low	CI Width	Var.	Time	Eff.
1e+03	-0.7000	8.22815	7.51869	6.80924	1.41891	131.0200	0.130569	0.007632423
1e+04	-0.7000	7.88599	7.65112	7.41625	0.46974	143.6010	1.232050	0.006963740
1e+05	-0.7000	7.74065	7.66625	7.59186	0.14879	144.0800	12.266100	0.006940589
1e+06	-0.7000	7.66651	7.64302	7.61953	0.04698	143.6300	127.201000	0.006962334
1e+02	0.0000	9.18576	7.10696	5.02816	4.15760	112.4890	0.014976	0.008889758
1e+03	0.0000	7.81848	7.06557	6.31265	1.50583	147.5640	0.128000	0.006776721
1e+04	0.0000	8.07385	7.82368	7.57350	0.50035	162.9210	1.193250	0.006137944
1e+05	0.0000	7.63553	7.55789	7.48026	0.15527	156.8940	12.101600	0.006373730
1e+06	0.0000	7.69234	7.66756	7.64278	0.04956	159.8500	121.366000	0.006255865
1e+02	0.7000	10.90440	8.09769	5.29099	5.61341	205.0590	0.017504	0.004876645
1e+03	0.7000	8.03211	7.26507	6.49803	1.53408	153.1520	0.124928	0.006529461
1e+04	0.7000	7.98748	7.72654	7.46560	0.52188	177.2470	1.232510	0.005641844
1e+05	0.7000	7.79207	7.70968	7.62729	0.16478	176.7110	12.300800	0.005658957
1e+06	0.7000	7.71334	7.68732	7.66130	0.05204	176.2120	122.705000	0.005674982

4.2 European Call Value – Conditional Estimator

Similar to the crude estimator, we now wish to find the price of a European call option with strike price $K = S_0 = 100$ on the above asset using a conditional Monte-Carlo estimator. To implement a conditional Monte-Carlo estimator we use the following result

$$\mathbb{E}_{\mathbb{Q}}\left[e^{-rT}\left(S_T - K\right)^+ \,\middle|\, \left\{B_t^{(1)} : 0 \le t \le T\right\}\right] = S_0 \xi \Phi(\tilde{d}_1) - K e^{-rT}\Phi(\tilde{d}_2)$$

where

$$\begin{aligned}
\xi &= \exp\left(-\frac{\rho^2}{2}\int_0^T v_t\,dt + \rho\int_0^T \sqrt{v_t}\,dB_t^{(1)}\right)\\
\tilde{d}_1 &= \frac{\log\frac{S_0\xi}{K} + \left(r + \frac{1}{2}\tilde{\sigma}^2\left(1-\rho^2\right)\right)}{\tilde{\sigma}\sqrt{T(1-\rho^2)}}\\
\tilde{d}_2 &= \tilde{d}_1 - \tilde{\sigma}\sqrt{T(1-\rho^2)}\\
\tilde{\sigma}^2 &= \frac{1}{T}\int_0^T v_t\,dt
\end{aligned}$$

With this estimator we wish to find the value of a European call option written on the asset described by the same Heston process. Table 5 shows the results of the simulation of N random variates for a the conditional Monte-Carlo estimator.

Table 5: Conditional Monte-Carlo estimator of the value of a European call option written on the underlying asset with dynamics described by the Heston stochastic volatility model above.

N Sim.	rho	CI High	Mean	CI Low	CI Width	Var.	Time	Eff.
1e+02	-0.7000	10.15730	8.50169	6.84609	3.31121	71.350800	0.006799	0.01401526
1e+03	-0.7000	7.98546	7.52603	7.06660	0.91886	54.944500	0.073334	0.01820018
1e+04	-0.7000	7.85080	7.70412	7.55743	0.29337	56.009400	0.675183	0.01785415
1e+05	-0.7000	7.67245	7.62698	7.58151	0.09094	53.819200	6.642650	0.01858073
1e+06	-0.7000	7.65977	7.64537	7.63097	0.02880	53.997800	68.223800	0.01851927
1e+02	0.0000	7.80971	7.71987	7.63003	0.17968	0.210107	0.007744	4.75947969
1e+03	0.0000	7.67563	7.64909	7.62255	0.05308	0.183370	0.069584	5.45345476
1e+04	0.0000	7.67593	7.66754	7.65915	0.01678	0.183235	0.680360	5.45747264
1e+05	0.0000	7.67184	7.66917	7.66649	0.00535	0.186365	6.803000	5.36581440

N Sim.	rho	CI High	Mean	CI Low	CI Width	Var.	Time	Eff.
1e+06	0.0000	7.66877	7.66792	7.66707	0.00170	0.186354	63.393300	5.36613113
1e+02	0.7000	7.91821	6.45114	4.98408	2.93413	56.025600	0.010768	0.01784898
1e+03	0.7000	8.05375	7.53357	7.01339	1.04036	70.436500	0.069008	0.01419718
1e+04	0.7000	7.87585	7.70359	7.53133	0.34452	77.241800	0.662128	0.01294636
1e+05	0.7000	7.73581	7.68204	7.62826	0.10755	75.278300	6.682030	0.01328404
1e+06	0.7000	7.71023	7.69321	7.67620	0.03403	75.373300	66.398200	0.01326730

4.3 Crude and Conditional Estimator Comparison

We now take a moment to visualize the data in Tables 4 and 5 in order to compare the performance of both European call estimators in the Heston model. Figures 12, 13 and 14 show us the convergence for the value of the European call option with increased N over the three values for ρ. Of note is the superiority of the conditional Monte-Carlo estimator made apparent in Figures 13 and 14: The conditional estimator is consistently more efficient than its crude sibling. Furthermore, the conditional estimator requires less computational effort than its crude counterpart since, while the crude estimator requires the simulation of the price process (thereby requiring the volatility process), the conditional estimator only requires the simulation of the volatility process.

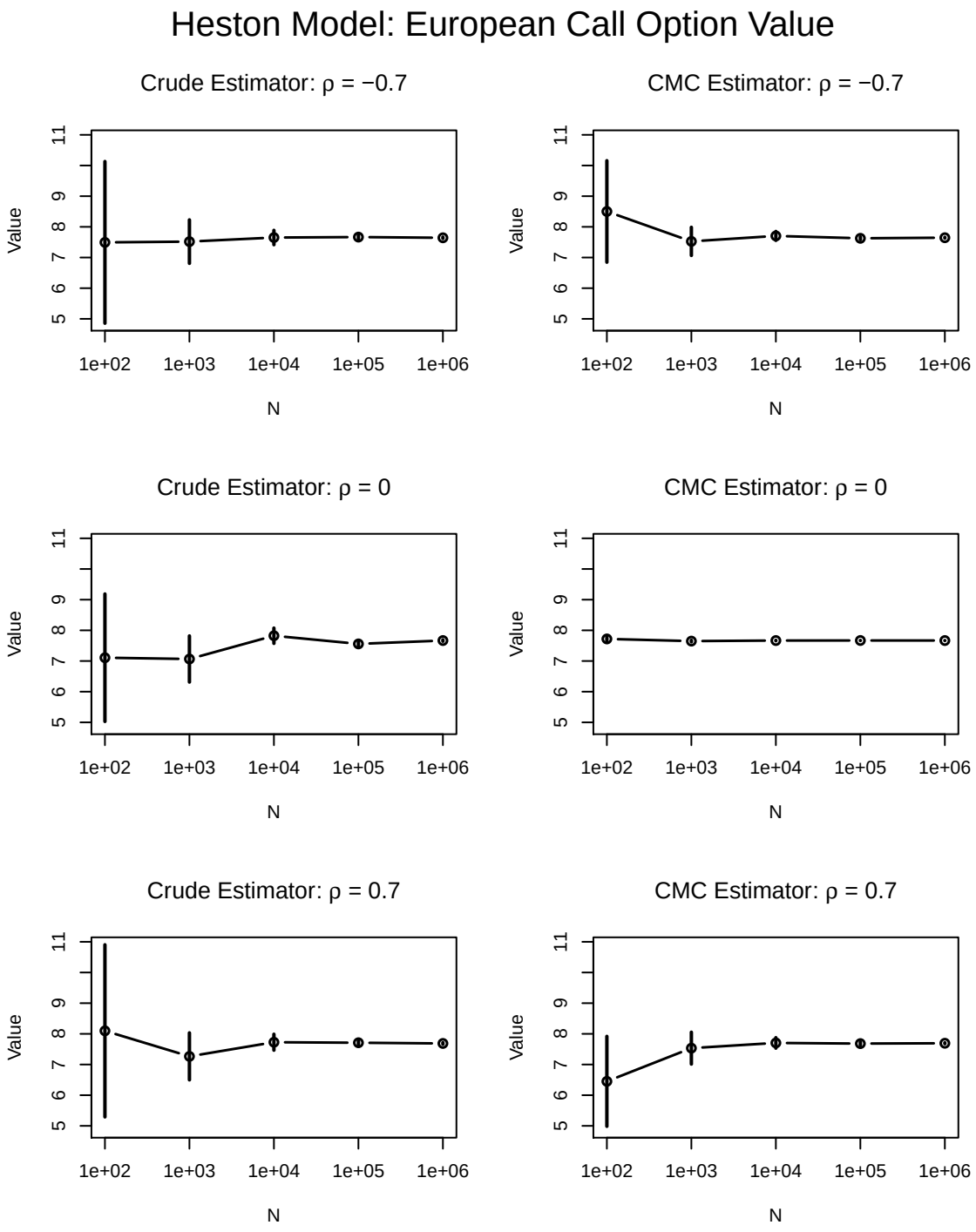

Figure 12: (Left) Crude and (Right) conditional Monte-Carlo simulations for the value of a European call option written on an underlying asset following the Heston stochastic volatility model.

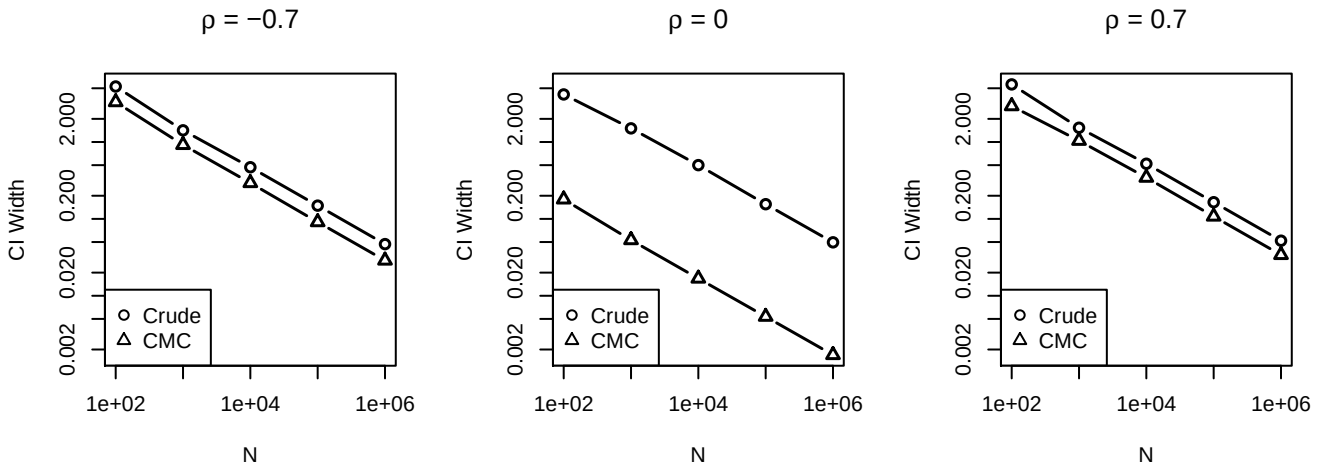

Figure 13: Comparison of the 95% confidence intervals generated by the crude and conditional Monte-Carlo estimators given various values for ρ. Note the persistent reduction in interval width by using the conditional estimator, with a profound reduction when $\rho = 0$.

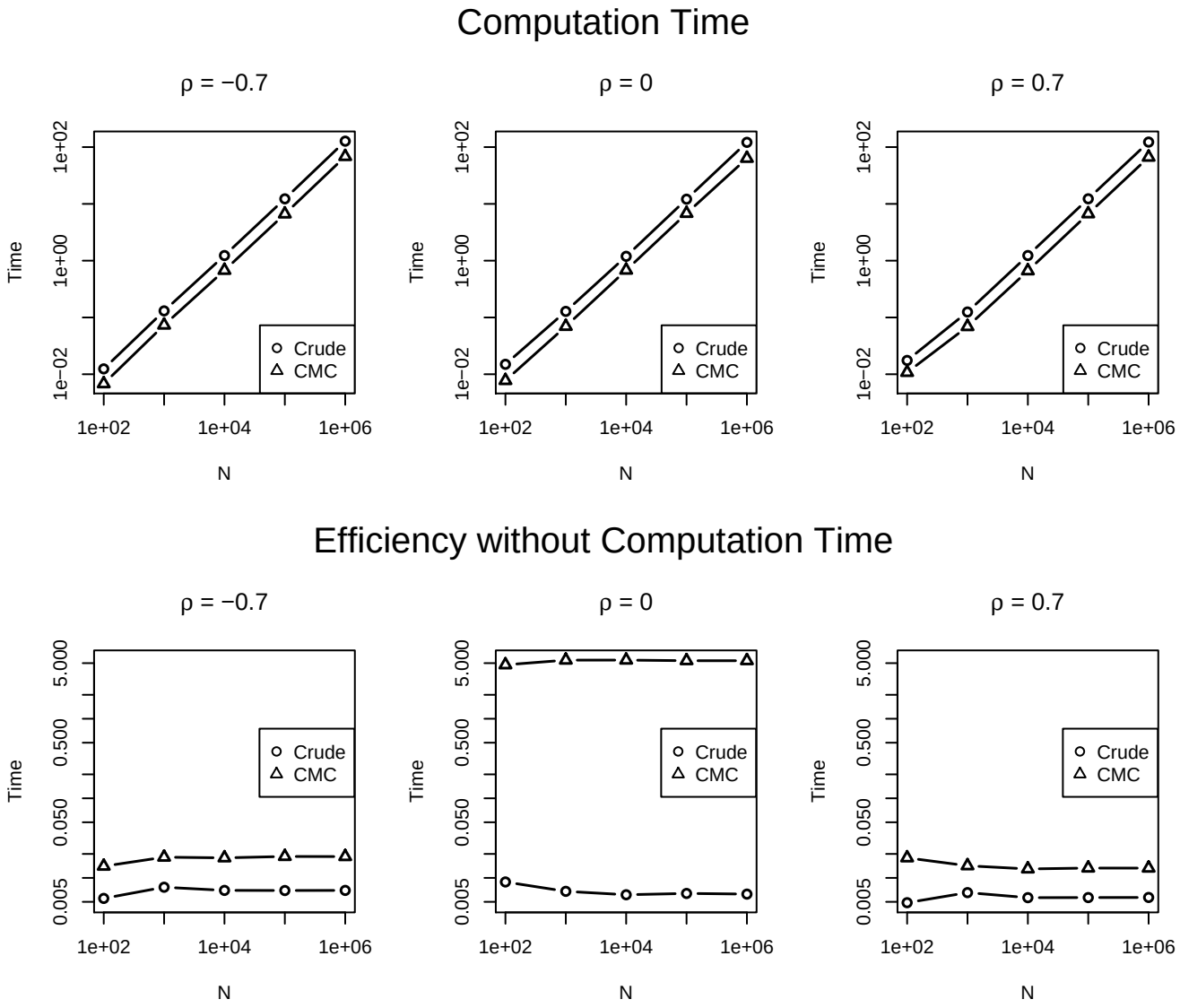

Figure 14: Comparison of the (Top) computation time and (Bottom) estimator efficiency for the crude and conditional Monte-Carlo estimators. Of interest is the increased efficiency of the conditional Monte-Carlo estimator which is particularly apparent when $\rho = 0$.

Appendix A Code

A.1 main_q1.cpp

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <time.h>
#include "rng.h"
#include "MathHelp.h"
#include "Asset.h"
#include "VanillaOption.h"
#include "AriAsianOption.h"
#include "GeoAsianOption.h"

using namespace std;

vector<double> CI(vector<double> v, double Z_a_2) {
   vector<double> conf_int(3);
   int n = v.size();
   double avg = MathHelp::mean(v);
   double stddev = MathHelp::sd(v);

   conf_int.at(0) = avg - Z_a_2 * stddev / sqrt(n);
   conf_int.at(1) = avg;
   conf_int.at(2) = avg + Z_a_2 * stddev / sqrt(n);

   return conf_int;
}
vector<double> crudeSim(VanillaOption VO, double T, int N) {
   double K = VO.getStrike();
   Asset A = VO.getAsset();
   double r = A.getR();
   vector<double> h(N); // payoff vector
   double z; // random normal variate

   for (int i = 0; i < N; i++) {
      z = rng::rnorm();
      h.at(i) = exp(-r * T) * fmax(K - A.generatePath(T, z), 0);
   }
   return h;
}
double antitheticNormal(double z, double mu) {
   return 2 * mu - z;
}
vector<double> antitheticNormal(vector<double> z, double mu) {
   vector<double> z_AT(z.size());
   for (int i = 0; i < z.size(); i++) {
      z_AT.at(i) = antitheticNormal(z.at(i), mu);
   }
   return z_AT;
}
vector<double> antitheticSim(VanillaOption VO, double T, int N) {
   N /= 2;
   double K = VO.getStrike();
   Asset A = VO.getAsset();
   double r = A.getR();
   vector<double> h(N); // payoff vector
   double z1, z2, S1, S2, h1, h2; // normal variates, asset prices, payoffs

   for (int i = 0; i < N; i++) {
      z1 = rng::rnorm();
      z2 = antitheticNormal(z1, 0); // antithetic variate to z1
      S1 = A.generatePath(T, z1);
      S2 = A.generatePath(T, z2);
      h1 = fmax(K - S1, 0);
      h2 = fmax(K - S2, 0);
      h.at(i) = 0.5 * (h1 + h2);
```

```
    }
    return h;
}
void writeMatrix(vector<vector<double> > data, string filepath, string header) {
    ofstream outfile;
    outfile.precision(6);

    outfile.open(filepath);
    outfile << header << endl;
    for (int i = 0; i < data.size(); i++) {
        for (int j = 0; j < data.at(0).size(); j++) {
            outfile << data.at(i).at(j) << ",";
        }
        outfile << endl;
    }
    outfile.close();
}
int main() {
    srand(402);
    double r = 0.0175;
    double sigma = 0.4;
    double S0 = 100;
    double K = 40;
    double t = 0;
    double T = 1;
    char type = 'p';
    int log10Nmin = 2;
    int log10Nmax = 8;
    double Z_a_2 = 1.96; // two tailed Z value for 1 - alpha confidence level
    // 95% = 1.96, 99% = 2.575, 99.5% = 2.81, 99.9% = 3.29, 99.99% = 3.89

    Asset A(S0, r, sigma);
    VanillaOption VO(A, K, t, T, type);

    vector<int> N;
    for (int i = log10Nmin; i <= log10Nmax; i++) {
        N.push_back( pow(10, i) );
    }

    vector<vector<double> > crude_CI( N.size() );
    vector<vector<double> > anti_CI( N.size() );
    clock_t t1, t2; float diff;
    vector<double> h_crude, h_anti;

    cout << "crude\t\tantithetic" << endl;
    for (int i = 0; i < N.size(); i++) {
        cout << N.at(i) << "\t\t";

        t1 = clock();
        h_crude = crudeSim(VO, T, N.at(i));
        t2 = clock();
        diff = ((float)t2 - (float)t1) / CLOCKS_PER_SEC;
        cout << printf("%.5f", diff) << "\t\t";
        crude_CI.at(i) = CI(h_crude, Z_a_2);
        crude_CI.at(i).push_back( pow(MathHelp::sd(h_crude), 2) );
        crude_CI.at(i).push_back( N.at(i) );
        crude_CI.at(i).push_back( diff );

        t1 = clock();
        h_anti = antitheticSim(VO, T, N.at(i));
        t2 = clock();
        diff = ((float)t2 - (float)t1) / CLOCKS_PER_SEC;
        cout << printf("%.5f", diff) << endl;
        anti_CI.at(i) = CI(h_anti, Z_a_2);
        anti_CI.at(i).push_back( pow(MathHelp::sd(h_anti), 2) );
        anti_CI.at(i).push_back( N.at(i) );
        anti_CI.at(i).push_back( diff );
    }

    string header = "CI_lo,mean,CI_hi,var,N,time,";
    string filepath_crude = "../../data/q1_crude_est.csv";
    string filepath_anti = "../../data/q1_anti_est.csv";
```

```
   writeMatrix(crude_CI, filepath_crude, header);
   writeMatrix(anti_CI, filepath_anti, header);
}
```

A.2 main_q3.cpp

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <time.h>
#include "rng.h"
#include "MathHelp.h"
#include "Asset.h"
#include "AriAsianOption.h"
#include "GeoAsianOption.h"

using namespace std;

vector<double> CI(vector<double> v, double Z_a_2) {
   vector<double> conf_int(3);
   int n = v.size();
   double avg = MathHelp::mean(v);
   double stddev = MathHelp::sd(v);

   conf_int.at(0) = avg - Z_a_2 * stddev / sqrt(n);
   conf_int.at(1) = avg;
   conf_int.at(2) = avg + Z_a_2 * stddev / sqrt(n);

   return conf_int;
}
vector<double> crudeSim(AsianOption* AO, int N) {
   Asset A = AO->getAsset();
   int n = AO->getNMonitoringPoints();
   double delta_t = AO->getTEnd() - AO->getTStart();

   vector<double> h(N), z(n);

   for (int i = 0; i < N; i++) {
      z = rng::rnorm(n);
      AO->setPath( A.generatePath(delta_t, n, z) );
      h.at(i) = AO->payoff();
   }
   return h;
}
double antitheticNormal(double z, double mu) {
   return 2 * mu - z;
}
vector<double> antitheticNormal(vector<double> z, double mu) {
   vector<double> z_AT(z.size());
   for (int i = 0; i < z.size(); i++) {
      z_AT.at(i) = antitheticNormal(z.at(i), mu);
   }
   return z_AT;
}
vector<double> antitheticSim(AsianOption* AO, int N) {
   N /= 2;
   Asset A = AO->getAsset();
   int n = AO->getNMonitoringPoints();
   double delta_t = AO->getTEnd() - AO->getTStart();

   vector<double> h(N), z1(n), z2(n);
   double h1, h2;

   for (int i = 0; i < N; i++) {
      z1 = rng::rnorm(n);
      z2 = antitheticNormal(z1, 0);
```

```
      AO->setPath( A.generatePath(delta_t, n, z1) );
      h1 = AO->payoff();
      AO->setPath( A.generatePath(delta_t, n, z2) );
      h2 = AO->payoff();
      h.at(i) = 0.5 * (h1 + h2);
   }
   return h;
}
double controlC(std::vector<double> x, std::vector<double> y) {
   int m = x.size(); // we should check if x, y dimensions agree
   double theta_hat_x = MathHelp::mean(x);
   double theta_hat_y = MathHelp::mean(y);
   double s_hat_y = MathHelp::sd(y);

   double z = 0;
   for (int i = 0; i < m; i++) {
      z += x.at(i) * y.at(i);
   }
   return (z - m * theta_hat_x * theta_hat_y) / ((m - 1) * pow(s_hat_y, 2));
}
vector<double> controlSim(AriAsianOption AAO, GeoAsianOption GAO, int N, int m) {
   int M = N - m;
   Asset A = AAO.getAsset(); // check if assets are the same
   int n = AAO.getNMonitoringPoints();
   double delta_t = AAO.getTEnd() - AAO.getTStart();

   vector<double> z(n), S(n + 1), p1(m), p2(m), h(M);
   for (int i = 0; i < m; i++) { // M = nb of pilot runs to compute c
      z = rng::rnorm(n);
      S = A.generatePath(delta_t, n, z);
      AAO.setPath(S);
      GAO.setPath(S);
      p1.at(i) = AAO.payoff();
      p2.at(i) = GAO.payoff();
   }
   double c = controlC(p1, p2);

   for (int i = 0; i < M; i++) {
      z = rng::rnorm(n);
      S = A.generatePath(delta_t, n, z);
      AAO.setPath(S);
      GAO.setPath(S);
      h.at(i) = AAO.payoff() - c * ( GAO.payoff() - GAO.price() );
   }
   return h;
}
void writeMatrix(vector<vector<double> > data, string filepath, string header) {
   ofstream outfile;
   outfile.precision(6);

   outfile.open(filepath);
   outfile << header << endl;
   for (int i = 0; i < data.size(); i++) {
      for (int j = 0; j < data.at(0).size(); j++) {
         outfile << data.at(i).at(j) << ",";
      }
      outfile << endl;
   }
   outfile.close();
}

int main() {
   srand(402);
   double r = 0.0175;
   double sigma = 0.25;
   double S0 = 500;
   double K = 500;
   double t = 0;
   double T = 1;
   int n = 52;
   char type = 'c';
   int log10Nmin = 2;
```

```
int log10Nmax = 8;
double Z_a_2 = 1.96; // two tailed Z value for 1 - alpha confidence level
// 95% = 1.96, 99% = 2.575, 99.5% = 2.81, 99.9% = 3.29, 99.99% = 3.89

Asset A(S0, r, sigma);
AriAsianOption AAO(A, K, t, T, n , type);
GeoAsianOption GAO(A, K, t, T, n, type);
double geo_price = GAO.price();

vector<int> N, m;
for (int i = log10Nmin; i <= log10Nmax; i++) {
   N.push_back( pow(10, i) );
   m.push_back( pow(10, i - 1) );
}

vector<vector<double> > crude_CI( N.size() );
vector<vector<double> > anti_CI( N.size() );
vector<vector<double> > cont_CI( N.size() );
vector<vector<double> > geo_crude_CI( N.size() );
vector<vector<double> > geo_anti_CI( N.size() );

clock_t t1, t2; float diff;
vector<double> h_crude, h_anti, h_cont, geo_h_crude, geo_h_anti;
cout << "crude\tgeo_crude\tanti\tgeo_anti\tcontrol" << endl;
for (int i = 0; i < N.size(); i++) {
   t1 = clock();
   h_crude = crudeSim(&AAO, N.at(i));
   t2 = clock();
   diff = ((float)t2 - (float)t1) / CLOCKS_PER_SEC;
   cout << printf("%.5f", diff) << "\t";
   crude_CI.at(i) = CI(h_crude, Z_a_2);
   crude_CI.at(i).push_back( pow(MathHelp::sd(h_crude), 2) );
   crude_CI.at(i).push_back( geo_price );
   crude_CI.at(i).push_back( N.at(i) );
   crude_CI.at(i).push_back( diff );

   t1 = clock();
   geo_h_crude = crudeSim(&GAO, N.at(i));
   t2 = clock();
   diff = ((float)t2 - (float)t1) / CLOCKS_PER_SEC;
   cout << printf("%.5f", diff) << "\t";
   geo_crude_CI.at(i) = CI(geo_h_crude, Z_a_2);
   geo_crude_CI.at(i).push_back( pow(MathHelp::sd(geo_h_crude), 2) );
   geo_crude_CI.at(i).push_back( geo_price );
   geo_crude_CI.at(i).push_back( N.at(i) );
   geo_crude_CI.at(i).push_back( diff );

   t1 = clock();
   h_anti = antitheticSim(&AAO, N.at(i));
   t2 = clock();
   diff = ((float)t2 - (float)t1) / CLOCKS_PER_SEC;
   cout << printf("%.5f", diff) << "\t";
   anti_CI.at(i) = CI(h_anti, Z_a_2);
   anti_CI.at(i).push_back( pow(MathHelp::sd(h_anti), 2) );
   anti_CI.at(i).push_back( geo_price );
   anti_CI.at(i).push_back( N.at(i) );
   anti_CI.at(i).push_back( diff );

   t1 = clock();
   geo_h_anti = antitheticSim(&GAO, N.at(i));
   t2 = clock();
   diff = ((float)t2 - (float)t1) / CLOCKS_PER_SEC;
   cout << printf("%.5f", diff) << "\t";
   geo_anti_CI.at(i) = CI(geo_h_anti, Z_a_2);
   geo_anti_CI.at(i).push_back( pow(MathHelp::sd(geo_h_anti), 2) );
   geo_anti_CI.at(i).push_back( geo_price );
   geo_anti_CI.at(i).push_back( N.at(i) );
   geo_anti_CI.at(i).push_back( diff );

   t1 = clock();
   h_cont = controlSim(AAO, GAO, N.at(i), m.at(i));
   t2 = clock();
```

```
      diff = ((float)t2 - (float)t1) / CLOCKS_PER_SEC;
      cout << printf("%.5f", diff) << endl;
      cont_CI.at(i) = CI(h_cont, Z_a_2);
      cont_CI.at(i).push_back( pow(MathHelp::sd(h_cont), 2) );
      cont_CI.at(i).push_back( geo_price );
      cont_CI.at(i).push_back( N.at(i) );
      cont_CI.at(i).push_back( diff );
   }

   string header = "CI_lo,mean,CI_hi,var,geo_price,N,time,";
   string filepath_crude = "../../data/q3_crude_est.csv";
   string filepath_anti = "../../data/q3_anti_est.csv";
   string filepath_cont = "../../data/q3_cont_est.csv";
   string filepath_geo_crude = "../../data/q3_geo_crude_est.csv";
   string filepath_geo_anti = "../../data/q3_geo_anti_est.csv";

   writeMatrix(crude_CI, filepath_crude, header);
   writeMatrix(anti_CI, filepath_anti, header);
   writeMatrix(cont_CI, filepath_cont, header);
   writeMatrix(geo_crude_CI, filepath_geo_crude, header);
   writeMatrix(geo_anti_CI, filepath_geo_anti, header);
}
```

A.3 main_q4a.cpp

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <time.h>
#include <string.h>
#include "rng.h"
#include "MathHelp.h"
#include "HestonAsset.h"
#include "VanillaOption.h"

using namespace std;

vector<double> CI(vector<double> v, double Z_a_2) {
   vector<double> conf_int(3);
   int n = v.size();
   double avg = MathHelp::mean(v);
   double stddev = MathHelp::sd(v);

   conf_int.at(0) = avg - Z_a_2 * stddev / sqrt(n);
   conf_int.at(1) = avg;
   conf_int.at(2) = avg + Z_a_2 * stddev / sqrt(n);

   return conf_int;
}
void writeMatrix(vector<vector<double> > data, string filepath, string header) {
   ofstream outfile;
   outfile.precision(6);

   outfile.open(filepath);
   outfile << header << endl;
   for (int i = 0; i < data.size(); i++) {
      for (int j = 0; j < data.at(0).size(); j++) {
         outfile << data.at(i).at(j) << ",";
      }
      outfile << endl;
   }
   outfile.close();
}
vector<double> crudeSim(HestonAsset HA, double K, double T, int n, int N) {
   double rho = HA.getRho();
   double S0 = HA.getSpot();
   double r = HA.getR();
```

```
   double S;
   vector<double> z1(n), z2(n), crude(N);

   for (int i = 0; i < N; i++) {
      z1 = rng::rnorm(n);
      z2 = rng::rnorm(n);
      S = HA.generatePricePath(T, n, z1, z2).at(n); // terminal value
      crude.at(i) = exp(-r * T) * fmax(S - K, 0);
   }
   return crude;
}
int main() {
   srand(402);
   double Z_a_2 = 1.96; // two tailed Z value for 1 - alpha confidence level
   // 95% = 1.96, 99% = 2.575, 99.5% = 2.81, 99.9% = 3.29, 99.99% = 3.89
   int log10Nmin = 2;
   int log10Nmax = 6;

   vector<int> N (log10Nmax - log10Nmin + 1); // vector of N to compute
   for (int i = 0; i < N.size(); i++) {
      N.at(i) = pow(10, log10Nmin + i);
   }

   // some parameters for the heston model
   double S0 = 100;
   double K = 100;
   char type = 'c';
   double v0 = 0.1;
   double t = 0;
   double T = 0.5;
   int n = 125;

   double r = 0;
   double rho_v[] = {-0.7, 0, 0.7};
   string rho_s[] = {"-7","0","7"};
   double kappa = 2;
   double theta = 0.03;
   double sigma = 0.1;

   for (int k = 0; k < sizeof(rho_v)/sizeof(rho_v[0]); k++) {
      double rho = rho_v[k];
      string filepath = "../../data/q4_heston_crude_est";
      filepath = filepath + "_" + rho_s[k] + ".csv";
      cout << filepath << endl;

      // build heston asset w/ given params
      HestonAsset HA(S0, r, v0, sigma, kappa, theta, rho);
      // start conditional sim
      vector<vector<double> > hest_crude_CI( N.size() ); // confidence intervals over all N
      clock_t t1, t2; float diff;

      for (int i = 0; i < N.size(); i++) {
         t1 = clock(); // time code
         vector<double> crude_price = crudeSim(HA, K, T, n, N.at(i));
         t2 = clock();
         diff = ((float)t2 - (float)t1) / CLOCKS_PER_SEC;
         cout << N.at(i) << "\t\t" << printf("%.5f", diff) << endl;
         hest_crude_CI.at(i) = CI( crude_price, Z_a_2 );
         hest_crude_CI.at(i).push_back( pow(MathHelp::sd(crude_price), 2) );
         hest_crude_CI.at(i).push_back( N.at(i) );
         hest_crude_CI.at(i).push_back( diff );
      }
      cout << "\n";

      string header = "CI_lo,mean,CI_hi,var,N,time,";
      writeMatrix(hest_crude_CI, filepath, header);
   }
}
```

A.4 main_q4b.cpp

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <time.h>
#include "rng.h"
#include "MathHelp.h"
#include "HestonAsset.h"
#include "VanillaOption.h"

using namespace std;

vector<double> CI(vector<double> v, double Z_a_2) {
   vector<double> conf_int(3);
   int n = v.size();
   double avg = MathHelp::mean(v);
   double stddev = MathHelp::sd(v);

   conf_int.at(0) = avg - Z_a_2 * stddev / sqrt(n);
   conf_int.at(1) = avg;
   conf_int.at(2) = avg + Z_a_2 * stddev / sqrt(n);

   return conf_int;
}
void writeMatrix(vector<vector<double> > data, string filepath, string header) {
   ofstream outfile;
   outfile.precision(6);

   outfile.open(filepath);
   outfile << header << endl;
   for (int i = 0; i < data.size(); i++) {
      for (int j = 0; j < data.at(0).size(); j++) {
         outfile << data.at(i).at(j) << ",";
      }
      outfile << endl;
   }
   outfile.close();
}
double conditionalXi(double rho, vector<double> v, double dt, vector<double> z) {
   double xi = 0;
   for (int i = 0; i < v.size(); i++) {
      xi += -0.5 * pow(rho, 2) * v.at(i) * dt + rho * sqrt(v.at(i) * dt) * z.at(i);
   }
   return exp(xi);
}
double conditionalSigma(vector<double> v, double dt) {
   double T = v.size() * dt;
   double sigma = 0;
   for (int i = 0; i < v.size(); i++) {
      sigma += v.at(i) * dt;
   }
   return sqrt(sigma/T);
}
double conditionalD1(double S, double K, double r, double sigma, double rho, double xi, double T) {
   double num = log( S * xi / K ) + (r + 0.5 * pow(sigma, 2) * (1 - pow(rho, 2))) * T;
   double den = sigma * sqrt(T * (1 - pow(rho, 2)));
   return num/den;
}
double conditionalD2(double S, double K, double r, double sigma, double rho, double xi, double T) {
   return conditionalD1(S, K, r, sigma, rho, xi, T) - sigma * sqrt(T * (1 - pow(rho, 2)));
}
vector<double> conditionalMC(HestonAsset HA, double K, double T, int n, int N) {
   double rho = HA.getRho();
   double S0 = HA.getSpot();
   double r = HA.getR();
   vector<double> z1(n), v(n + 1), cmc(N);
   double xi, sigma, d1, d2;

   for (int i = 0; i < N; i++) {
```

```
      z1 = rng::rnorm(n);
      v = HA.generateVolPath(T, n, z1);
      v.pop_back(); // remove terminal element
      xi = conditionalXi(rho, v, T/n, z1);
      sigma = conditionalSigma(v, T/n);
      d1 = conditionalD1(S0, K, r, sigma, rho, xi, T);
      d2 = conditionalD2(S0, K, r, sigma, rho, xi, T);
      cmc.at(i) = S0 * xi * MathHelp::stdNormCdf(d1) -
         K * exp(-r * T) * MathHelp::stdNormCdf(d2);
   }
   return cmc;
}
int main() {
   srand(402);
   double Z_a_2 = 1.96; // two tailed Z value for 1 - alpha confidence level
   // 95% = 1.96, 99% = 2.575, 99.5% = 2.81, 99.9% = 3.29, 99.99% = 3.89
   int log10Nmin = 2;
   int log10Nmax = 6;

   vector<int> N (log10Nmax - log10Nmin + 1); // vector of N to compute
   for (int i = 0; i < N.size(); i++) {
      N.at(i) = pow(10, log10Nmin + i);
   }

   // some parameters for the heston model
   double S0 = 100;
   double K = 100;
   char type = 'c';
   double v0 = 0.1;
   double t = 0;
   double T = 0.5;
   int n = 125;

   double r = 0;
   double rho_v[] = {-0.7, 0, 0.7};
   string rho_s[] = {"-7","0","7"};
   double kappa = 2;
   double theta = 0.03;
   double sigma = 0.1;

   for (int k = 0; k < sizeof(rho_v)/sizeof(rho_v[0]); k++) {
      double rho = rho_v[k];
      string filepath = "../../data/q4_heston_cmc_est";
      filepath = filepath + "_" + rho_s[k] + ".csv";
      cout << filepath << endl;

      // build heston asset w/ given params
      HestonAsset HA(S0, r, v0, sigma, kappa, theta, rho);
      // start conditional sim
      vector<vector<double> > hest_cmc_CI( N.size() ); // confidence intervals over all N
      clock_t t1, t2; float diff;

      for (int i = 0; i < N.size(); i++) {
         t1 = clock(); // time code
         vector<double> cmc_price = conditionalMC(HA, K, T, n, N.at(i));
         t2 = clock();
         diff = ((float)t2 - (float)t1) / CLOCKS_PER_SEC;
         cout << N.at(i) << "\t\t" << printf("%.5f", diff) << endl;
         hest_cmc_CI.at(i) = CI( cmc_price, Z_a_2 );
         hest_cmc_CI.at(i).push_back( pow(MathHelp::sd(cmc_price), 2) );
         hest_cmc_CI.at(i).push_back( N.at(i) );
         hest_cmc_CI.at(i).push_back( diff );
      }
      cout << "\n";

      string header = "CI_lo,mean,CI_hi,var,N,time,";
      writeMatrix(hest_cmc_CI, filepath, header);
   }

}
```

A.5 MathHelp.h

```
/*
   Header file for MathHelp
   Contains some useful functions for our purposes
*/
#ifndef MATHHELP_H
#define MATHHELP_H

#include <vector>

class MathHelp {
   public:
      static double stdNormPdf(double x);
      static double stdNormCdf(double x);
      static double mean(std::vector<double> v);
      static double sd(std::vector<double> v);
};
#endif
```

A.6 MathHelp.cpp

```
#include "MathHelp.h"
#include <cmath>

double MathHelp::stdNormPdf(double x) {
   return 1/sqrt(2 * M_PI) * exp(-1/2.0 * pow(x, 2));
}
double MathHelp::stdNormCdf(double x_in) {
   // we notice that the approximation is much better for positive x
   // use symmetry of the normal CDF to compute the negative x values
   double x = fabs(x_in);

   double b0 = 0.2316419;
   double b1 = 0.31938530;
   double b2 = -0.356563782;
   double b3 = 1.781477937;
   double b4 = -1.821255978;
   double b5 = 1.330274429;
   double t = 1/(1 + b0 * x);

   double y = 1 - stdNormPdf(x) * (b1*t + b2*pow(t,2) + b3*pow(t,3) + b4*pow(t,4) + b5*pow(t,5));

   if (x_in < 0) {
      y = 1 - y;
   }
   return y;
}
double MathHelp::mean(std::vector<double> v) {
   double sum = 0;
   for (int i = 0; i < v.size(); i++) {
      sum += v.at(i);
   }
   return sum / v.size();
}
double MathHelp::sd(std::vector<double> v) {
   double xbar = mean(v);
   double sq_sum = 0;
   for (int i = 0; i < v.size(); i++) {
      sq_sum += pow((v.at(i) - xbar), 2);
   }
   return sqrt( sq_sum / (v.size() - 1) );
}
```

A.7 rng.h

```
/*
   Header file for the random number generator
*/
#ifndef RNG_H
#define RNG_H

#include <vector>

class rng {
   public:
      static double runif();
      static double rnorm();
      static std::vector<double> rnorm(int n);
};
#endif
```

A.8 rng.cpp

```
#include "rng.h"
#include <stdlib.h>   /* srand, rand */
#include <cmath>

// constants for rnorm
const double a0 = 2.50662823884;
const double a1 = -18.61500062529;
const double a2 = 41.39119773534;
const double a3 = -25.44106049637;
const double a[] = {a0,a1,a2,a3};
const double b0 = -8.47351093090;
const double b1 = 23.08336743743;
const double b2 = -21.06224101826;
const double b3 = 3.13082909833;
const double b[] = {b0,b1,b2,b3};
const double c0 = 0.3374754822726147;
const double c1 = 0.9761690190917186;
const double c2 = 0.1607979714918209;
const double c3 = 0.0276438810333863;
const double c4 = 0.0038405729373609;
const double c5 = 0.0003951896511919;
const double c6 = 0.0000321767881768;
const double c7 = 0.0000002888167364;
const double c8 = 0.0000003960315187;
const double c[] = {c0,c1,c2,c3,c4,c5,c6,c7,c8};

double rng::runif() {
   return ((double) rand()) / RAND_MAX;
}

double rng::rnorm() {
   double v = runif();
   double u;
   double z = 0;
   double A = 0;
   double B = 0;

   u = v;
   if (v < 0.5) {
      u = 1 - v;
   }

   if (u <= 0.92 & u >= 0.5) {
      for (int i = 0; i <= 3; i++) {
         A += a[i] * pow(u - 0.5, 2 * i + 1);
         B += b[i] * pow(u - 0.5, 2 * i + 2);
      }
```

```
      z = A / (1 + B);
   }
   else {
      for (int i = 0; i <= 8; i++) {
         z += c[i] * pow(log( -log(1 - u) ), i);
      }
   }

   if (v < 0.5) return -z;
   else return z;
}
std::vector<double> rng::rnorm(int n) {
   std::vector<double> z(n);
   for (int i = 0; i < n; i++) {
      z.at(i) = rnorm();
   }
   return z;
}
```

A.9 Asset.h

```
/*
   Header file for an Asset
   An asset is an object with a spot, interest rate, and (optional) volatility
*/
#ifndef ASSET_H
#define ASSET_H

#include <vector>

class Asset {
   protected:
      double spot; // asset spot value
      double r; // cont. comp. interest rate
      double sigma; // vol (lognormal)

      double lognormalModel(double S, double T, int n, double Z);

   public:
      Asset(); // default constructor
      virtual ~Asset(){};
      Asset(double spot, double r); // constructor w/o vol
      Asset(double spot, double r, double sigma);
      void setSpot(double spot);
      double getSpot();
      void setR(double r);
      double getR();
      void setSigma(double sigma);
      double getSigma();

      double generatePath(double T, double z);
      std::vector<double> generatePath(double T, int n, std::vector<double> z);
};
#endif
```

A.10 Asset.cpp

```
#include "Asset.h"
#include <cmath> // pow

Asset::Asset() {}
Asset::Asset(double spot, double r) {
   this->spot = spot;
   this->r = r;
```

```
}
Asset::Asset(double spot, double r, double sigma) {
   this->spot = spot;
   this->r = r;
   this->sigma = sigma;
}
void Asset::setSpot(double spot) {
   this->spot = spot;
}
double Asset::getSpot() {
   return spot;
}
void Asset::setR(double r) {
   this->r = r;
}
double Asset::getR() {
   return r;
}
void Asset::setSigma(double sigma) {
   this->sigma = sigma;
}
double Asset::getSigma() {
   return sigma;
}
double Asset::generatePath(double T, double z) {
   return lognormalModel(spot, T, 1, z);
}
std::vector<double> Asset::generatePath(double T, int n, std::vector<double> z) {
   // to do, check if size of z matches

   std::vector<double> S(n + 1);
   S.at(0) = spot;

   for (int i = 1; i < S.size(); i++) {
      S.at(i) = lognormalModel(S.at(i - 1), T, n, z.at(i - 1));
   }
   return S;
}
double Asset::lognormalModel(double S, double T, int n, double Z) {
   return S * exp( (r - 0.5 * pow(sigma, 2)) * T/n + sigma * sqrt(T/n) * Z );
}
```

A.11 HestonAsset.h

```
/*
   Header file for an HestonAsset
*/
#ifndef HESTONASSET_H
#define HESTONASSET_H

#include "Asset.h"
#include <vector>

class HestonAsset: public Asset {
   private:
      double vol; // vol
      double sigma; // vol of vol
      double kappa; // reversion rate
      double theta; // long run variance
      double rho;

      double hestonVol(double v, double T, int n, double z);
      double hestonPrice(double S, double v, double T, int n, double z1, double z2);

   public:
      HestonAsset(); // default constructor
      HestonAsset(double spot, double r, double vol, double sigma, double kappa,
         double theta, double rho); // heston
```

```
    void setVol(double vol);
    double getVol();
    void setSigma(double sigma);
    double getSigma();
    void setKappa(double kappa);
    double getKappa();
    void setTheta(double theta);
    double getTheta();
    void setRho(double rho);
    double getRho();

    double generateVolPath(double T, double z);
    std::vector<double> generateVolPath(double T, int n, std::vector<double> z);
    double generatePricePath(double T, double z1, double z2);
    std::vector<double> generatePricePath(double T, int n,
      std::vector<double> z1, std::vector<double> z2);
};
#endif
```

A.12 HestonAsset.cpp

```
#include "HestonAsset.h"
#include <cmath> // pow

HestonAsset::HestonAsset() {}
HestonAsset::HestonAsset(double spot, double r, double vol, double sigma, double kappa,
  double theta, double rho):Asset(spot, r) {

  this->vol = vol;
  this->sigma = sigma;
  this->kappa = kappa;
  this->theta = theta;
  this->rho = rho;
}
void HestonAsset::setVol(double vol) {
  this->vol = vol;
}
double HestonAsset::getVol() {
  return vol;
}
void HestonAsset::setSigma(double sigma) {
  this->sigma = sigma;
}
double HestonAsset::getSigma() {
  return sigma;
}
void HestonAsset::setKappa(double kappa) {
  this->kappa = kappa;
}
double HestonAsset::getKappa() {
  return kappa;
}
void HestonAsset::setTheta(double theta) {
  this->theta = theta;
}
double HestonAsset::getTheta() {
  return theta;
}
void HestonAsset::setRho(double rho) {
  this->rho = rho;
}
double HestonAsset::getRho() {
  return rho;
}
double HestonAsset::generateVolPath(double T, double z) {
  return hestonVol(vol, T, 1, z);
}
std::vector<double> HestonAsset::generateVolPath(double T, int n, std::vector<double> z) {
```

```
   // to do, check if sizes match
   std::vector<double> v(n + 1);
   v.at(0) = vol;

   for (int i = 1; i < v.size(); i++) {
      v.at(i) = hestonVol(v.at(i - 1), T, n, z.at(i - 1));
   }
   return v;
}
double HestonAsset::generatePricePath(double T, double z1, double z2) {
   return hestonPrice(spot, vol, T, 1, z1, z2);
}
std::vector<double> HestonAsset::generatePricePath(double T, int n,
   std::vector<double> z1, std::vector<double> z2) { // to do, check if size of z matches
   std::vector<double> S(n + 1), v(n + 1);
   S.at(0) = spot;
   v = generateVolPath(T, n, z1);

   for (int i = 1; i < S.size(); i++) {
      S.at(i) = hestonPrice(S.at(i - 1), v.at(i - 1), T, n, z1.at(i - 1), z2.at(i - 1));
   }
   return S;
}
double HestonAsset::hestonVol(double v, double T, int n, double z) {
   double v_out = v + kappa * (theta - v) * (T/n) + sigma * sqrt(v * (T/n)) * z;
   if (v_out < 0) return 0;
   else return v_out;
}
double HestonAsset::hestonPrice(double S, double v, double T, int n, double z1, double z2) {
   double x = log(S);

   return exp(x + (r - 0.5 * v) * (T/n) + sqrt(v) * (rho * sqrt(T/n) * z1 +
      sqrt(1 - pow(rho, 2)) * sqrt(T/n) * z2));
}
```

A.13 Option.h

```
/*
   Header file for an Option
   An option is an object that contains an asset, a strike, and a time to maturity
   Maybe it makes sense to implement this as an abstract class?
   It doesn't make sense to be able to instantiate a general 'Option'
*/
#ifndef OPTION_H
#define OPTION_H

#include "Asset.h"

class Option {
   protected:
      Asset A; // underlying asset
      double K; // strike price
      double t; // current time
      double T; // expiry time
      char type; // call or put

   public:
      Option(); // default constructor
      Option(Asset A, double K, double t, double T, char type);
      void setAsset(Asset A);
      Asset getAsset();
      void setStrike(double K);
      double getStrike();
      void setTStart(double t);
      double getTStart();
      void setTEnd(double T);
      double getTEnd();
      void setType(char type);
```

```
    char getType();

    virtual double payoff() = 0;
};
#endif
```

A.14 Option.cpp

```
#include "Option.h"

Option::Option() {}
Option::Option(Asset A, double K, double t, double T, char type) {
   this->A = A;
   this->K = K;
   this->t = t;
   this->T = T;
   this->type = type;
}
void Option::setAsset(Asset A) {
   this->A = A;
}
Asset Option::getAsset() {
   return A;
}
void Option::setStrike(double K) {
   this->K = K;
}
double Option::getStrike() {
   return K;
}
void Option::setTStart(double t) {
   this->t = t;
}
double Option::getTStart() {
   return t;
}
void Option::setTEnd(double T) {
   this->T = T;
}
double Option::getTEnd() {
   return T;
}
void Option::setType(char type) {
   this->type = type;
}
char Option::getType() {
   return type;
}
```

A.15 VanillaOption.h

```
/*
   Header file for a VanillaOption
   A vanilla option is a child of an option that contains a specific implementation
   of a payoff function
*/
#ifndef VANILLAOPTION_H
#define VANILLAOPTION_H

#include "Option.h"

class VanillaOption: public Option {
   private:
      double computeD1();
```

```
      double computeD2();

   public:
      VanillaOption(); // default constructor
      VanillaOption(Asset A, double K, double t, double T, char type);

      double payoff();
      double blackScholesPrice();
      double blackScholesDelta();
};
#endif
```

A.16 VanillaOption.cpp

```
#include "VanillaOption.h"
#include <cmath> // fmax
#include <stdlib.h> // exit
#include "MathHelp.h"

VanillaOption::VanillaOption() {}
VanillaOption::VanillaOption(Asset A, double K, double t, double T, char type)
   :Option(A, K, t, T, type){} // call the parent constructor
double VanillaOption::payoff() {
   if (type == 'c') return fmax(A.getSpot() - K, 0);
   else if (type == 'p') return fmax(K - A.getSpot(), 0);
   else exit(EXIT_FAILURE);
}
double VanillaOption::blackScholesPrice() {
   double R = A.getR();
   double S = A.getSpot();
   double d1 = computeD1();
   double d2 = computeD2();

   if (type == 'c') {
      return MathHelp::stdNormCdf(d1) * S - MathHelp::stdNormCdf(d2) * K *
         exp(-R * (T - t));
   } else if (type == 'p') {
      return -MathHelp::stdNormCdf(-d1) * S + MathHelp::stdNormCdf(-d2) * K *
      exp(-R * (T - t));
   } else exit(EXIT_FAILURE);
}
double VanillaOption::blackScholesDelta() {
   double d1 = computeD1();

   if (type == 'c') return MathHelp::stdNormCdf(d1);
   else if (type == 'p') return -MathHelp::stdNormCdf(-d1);
   else exit(EXIT_FAILURE);
}
double VanillaOption::computeD1() {
   double S = A.getSpot();
   double R = A.getR();
   double sigma = A.getSigma();

   return 1/(sigma * sqrt(T - t)) * (log(S/K) + (R + 0.5 * pow(sigma,2)) * (T - t));
}
double VanillaOption::computeD2() {
   double sigma = A.getSigma();
   return computeD1() - sigma * sqrt(T - t);
}
```

A.17 AsianOption.h

```
/*
   Header file for an AsianOption
```

```
*/
#ifndef ASIANOPTION_H
#define ASIANOPTION_H

#include "Option.h"
#include "Asset.h"
#include <vector>

class AsianOption: public Option {
   protected:
      int n;
      std::vector<double> path;

   public:
      AsianOption(); // default constructor
      AsianOption(Asset A, double K, double t, double T, int n, char type);
      double getNMonitoringPoints();
      void setNMonitoringPoints(int n);
      std::vector<double> getPath();
      void setPath(std::vector<double> path);

      virtual double payoff() = 0;
      virtual double price() = 0;
};
#endif
```

A.18 AsianOption.cpp

```
#include "AsianOption.h"

AsianOption::AsianOption() {}
AsianOption::AsianOption(Asset A, double K, double t, double T, int n, char type)
   :Option(A, K, t, T, type) {
   this->n = n;
}
double AsianOption::getNMonitoringPoints() {
   return n;
}
void AsianOption::setNMonitoringPoints(int n) {
   this->n = n;
}
std::vector<double> AsianOption::getPath() {
   return path;
}
void AsianOption::setPath(std::vector<double> path) {
   this->path = path;
}
```

A.19 AriAsianOption.h

```
/*
   Header file for a AriAsianOption
*/
#ifndef ARIASIANOPTION_H
#define ARIASIANOPTION_H

#include "AsianOption.h"
#include <vector>

class AriAsianOption: public AsianOption {
   public:
      AriAsianOption(); // default constructor
      AriAsianOption(Asset A, double K, double t, double T, int n, char type);
```

```
      double payoff();
      double price();
};
#endif
```

A.20 AriAsianOption.cpp

```
#include "AriAsianOption.h"
#include <cmath>

AriAsianOption::AriAsianOption() {}
AriAsianOption::AriAsianOption(Asset A, double K, double t, double T, int n, char type)
   :AsianOption(A, K, t, T, n, type){} // call the parent constructor
double AriAsianOption::payoff() {
   double avg;
   double df = exp(-A.getR() * (T - t));
   double sum = 0;

   for (int i = 1; i < path.size(); i++) {
      sum += path.at(i);
   }
   avg = sum/n;

   if (type == 'c') return df * fmax(avg - K, 0);
   else if (type == 'p') return df * fmax(K - avg, 0);
   else return nan("1");
}
double AriAsianOption::price() {
   return nan("1"); // maybe numeric implementation w/ default control variate algo
}
```

A.21 GeoAsianOption.h

```
/*
   Header file for a GeoAsianOption
*/
#ifndef GEOASIANOPTION_H
#define GEOASIANOPTION_H

#include "AsianOption.h"
#include <vector>

class GeoAsianOption: public AsianOption {
   private:
      double computeSigmaHat();
      double computeMuHat();
      double computeD1();
      double computeD2();

   public:
      GeoAsianOption(); // default constructor
      GeoAsianOption(Asset A, double K, double t, double T, int n, char type);

      double payoff();
      double price();
};
#endif
```

A.22 GeoAsianOption.cpp

```cpp
#include "GeoAsianOption.h"
#include <cmath>
#include "MathHelp.h"

GeoAsianOption::GeoAsianOption() {}
GeoAsianOption::GeoAsianOption(Asset A, double K, double t, double T, int n, char type)
   :AsianOption(A, K, t, T, n, type){} // call the parent constructor
double GeoAsianOption::payoff() {
   double g_avg;
   double df = exp(-A.getR() * (T - t));
   double prod = 1;

   for (int i = 1; i < path.size(); i++) {
      prod *= path.at(i);
   }
   g_avg = pow(prod, 1.0/n);

   if (type == 'c') return df * fmax(g_avg - K, 0);
   else if (type == 'p') return df * fmax(K - g_avg, 0);
   else return nan("1");
}
double GeoAsianOption::price() {
   double S = A.getSpot();
   double r = A.getR();
   double mu_hat = computeMuHat();
   double d1 = computeD1();
   double d2 = computeD2();

   if (type == 'c') {
      return exp(-r * (T - t)) * (S * exp(mu_hat * (T - t)) * MathHelp::stdNormCdf(d1) -
         K * MathHelp::stdNormCdf(d2));
   } else if (type == 'p') {
      return 0;
   }
   else return nan("1");
}
double GeoAsianOption::computeSigmaHat() {
   double sigma = A.getSigma();

   return sqrt( pow(sigma, 2) * (n + 1) * (2 * n + 1) / (6 * pow(n, 2)) );
}
double GeoAsianOption::computeMuHat() {
   double r = A.getR();
   double sigma = A.getSigma();
   double sigma_hat = computeSigmaHat();

   return (r - 0.5 * pow(sigma, 2)) * (n + 1) / (2 * n) + 0.5 * pow(sigma_hat, 2);
}
double GeoAsianOption::computeD1() {
   double S = A.getSpot();
   double r = A.getR();
   double sigma = A.getSigma();

   double sigma_hat = computeSigmaHat();
   double mu_hat = computeMuHat();

   return (log(S / K) + (mu_hat + 0.5 * pow(sigma_hat, 2)) * (T - t)) / (sigma_hat * sqrt(T - t));
}
double GeoAsianOption::computeD2() {
   double sigma = A.getSigma();
   double sigma_hat = computeSigmaHat();

   return computeD1() - sigma_hat * sqrt(T - t);
}
```