HW 2

Problem 1

Assume in the one-period binomial market of Section 1.1 that both H and T have positive probability of occurring. Show that condition (1.1.2) precludes arbitrage. In other words, show that if $X_0 = 0$ and

$$X_1 = \Delta_0 S_1 + (1+r)(X_0 - \Delta_0 S_0),$$

then we cannot have X_1 strictly positive with positive probability unless X_1 is strictly negative with positive probability as well, and this is the case regardless of the choice of the number Δ_0.

Solution:

Given that $X_0 = 0$, the equation becomes:

$$X_1 = \Delta_0 S_1 - \Delta_0 S_0(1+r) = \Delta_0(S_1 - S_0(1+r)).$$

Suppose $X_1 > 0$, without any probability that $X_1 < 0$.

$$X_1(H) = \Delta_0(S_1(H) - S_0(1+r)) > 0.$$

$$X_1(T) = \Delta_0(S_1(T) - S_0(1+r)) > 0.$$

If $\Delta_0 > 0$, then $S_1(H) > S_0(1+r)$ and $S_1(T) > S_0(1+r)$.

If $\Delta_0 < 0$, then $S_1(H) < S_0(1+r)$ and $S_1(T) < S_0(1+r)$.

By no risk free arbitrage, we have:

$$0 < d < 1 + r < u.$$

i.e.

$$S(T) < (1+r)S(0) < S(H).$$

Both cases lead to a contradiction. Therefore, X_1 cannot be strictly positive with positive probability unless it is also strictly negative with positive probability. Q.E.D.

Problem 2

In the proof of Theorem 1.2.2, show under the induction hypothesis that

$$X_{n+1}(w_1 w_2 \ldots w_n T) = V_{n+1}(w_1 w_2 \ldots w_n T)$$

Solution:

$X_{n+1}(w_1 w_2 \ldots w_n T) = \Delta_n(w_1 w_2 \ldots w_n) S_{n+1}(w_1 w_2 \ldots w_n T) + (1+r)(X_n(w_1 w_2 \ldots$

Substitute $X_n(w_1 w_2 \ldots w_n) = V_n(w_1 w_2 \ldots w_n)$ into the equation, we get:

$X_{n+1}(w_1 w_2 \ldots w_n T) = \Delta_n(w_1 w_2 \ldots w_n) S_{n+1}(w_1 w_2 \ldots w_n T) + (1+r)(V_n(w_1 w_2 \ldots$

Plug in $V_n(w_1 w_2 \ldots w_n) = \frac{1}{1+r}[\tilde{p} V_{n+1}(w_1 w_2 \ldots w_n H) + \tilde{q} V_{n+1}(w_1 w_2 \ldots w_n T)]$, and denoting $V_{n+1}^H := V_{n+1}(w_1 w_2 \ldots w_n H)$, $V_{n+1}^T := V_{n+1}(w_1 w_2 \ldots w_n T)$, samely for S and X, $\Delta_n := \Delta_n(w_1 w_2 \ldots w_n)$, we get:

$$X_{n+1}^T = \Delta_n S_{n+1}^T + (1+r)(\frac{1}{1+r}[\tilde{p} V_{n+1}^H + \tilde{q} V_{n+1}^T] - \Delta_n S_n)$$

$$X_{n+1}^T = \Delta_n S_{n+1}^T + \tilde{p} V_{n+1}^H + \tilde{q} V_{n+1}^T - (1+r)\Delta_n S_n$$

$$X_{n+1}^T = \tilde{p} V_{n+1}^H + \tilde{q} V_{n+1}^T + \Delta_n(S_{n+1}^T - (1+r)S_n)$$

Take the defined values in::

$$X_{n+1}^T = \frac{1+r-d}{u-d} V_{n+1}^H + \frac{u-1-r}{u-d} V_{n+1}^T + \frac{V_{n+1}^H - V_{n+1}^T}{(u-d)S_n}(d-1-r)S_n$$

Which simplifies to:

$$X_{n+1}^T = V_{n+1}^T$$

i.e. $X_{n+1}(w_1 w_2 \ldots w_n T) = V_{n+1}(w_1 w_2 \ldots w_n T)$ Q.E.D.

Problem 3 (Asian Option)

Consider the three-period model of Example 1.2.1, with $S_0 = 4$, $u = 2$, $d = \frac{1}{2}$, and take the interest rate $r = \frac{1}{4}$, so that $\tilde{p} = \tilde{q} = \frac{1}{2}$. For $n = 0, 1, 2, 3$, define $Y_n = \sum_{k=0}^{n} S_k$ to be the sum of the stock prices between times zero and n. Consider an Asian call option that expires at time three and has a strike $K = 4$ (i.e., whose payoff at time three is $(\frac{1}{4}Y_3 - 4)^+$). This is like a European call, except the payoff of the option is based on the average stock price rather than the final stock price. Let $v_n(s, y)$ denote the price of this option at time n if $S_n = s$ and $Y_n = y$. In particular, $v_3(s, y) = (\frac{1}{4}y - 4)^+$.

1. Develop an algorithm for computing v_n recursively. In particular, write a formula for v_n in terms of v_{n+1}.
2. Apply the algorithm developed in (i) to compute $v_0(4, 4)$, the price of the Asian option at time zero.
3. Provide a formula for $\delta_n(s, y)$, the number of shares of stock that should be held by the replicating portfolio at time n if $S_n = s$ and $Y_n = y$.

Solution:

1. Develop an algorithm for computing v_n recursively. In particular, write a formula for v_n in terms of v_{n+1}.

By risk neutral pricing, we have:

$$v_n(s,y) = \frac{1}{1+r}[\tilde{p}v_{n+1}(su, y+su) + \tilde{q}v_{n+1}(sd, y+sd)]$$

Taking the given values, we have:

$$v_n(s,y) = \frac{2}{5}[v_{n+1}(2s, y+2s) + v_{n+1}(\frac{s}{2}, y+\frac{s}{2})]$$

2. Apply the algorithm developed in (i) to compute $v_0(4,4)$, the price of the Asian option at time zero.

$$v_0(4,4) = \frac{2}{5}[v_1(8,12) + v_1(2,6)]$$

$$v_0(4,4) = (\frac{2}{5})^2[v_2(16,28) + v_2(4,16) + v_2(4,10) + v_2(1,7)]$$

$$v_0(4,4) = (\frac{2}{5})^3[v_3(32,60) + v_3(8,36) + v_3(8,24) + v_3(2,18) + v_3(8,18) + v_3(2,1$$

$$v_0(4,4) = (\frac{2}{5})^3[11 + 5 + 2 + 1/2 + 1/2 + 0 + 0 + 0] = 152/125 = 1.216$$

3. Provide a formula for $\delta_n(s,y)$, the number of shares of stock that should be held by the replicating portfolio at time n if $S_n = s$ and $Y_n = y$.

$$X_{n+1} = \delta_n S_{n+1} + (1+r)(X_n - \delta_n S_n)$$

We have a system of equations:

$$X_{n+1}(H) = \delta_n su + (1+r)(X_n - \delta_n s) = v_{n+1}(su, y+su)$$

$$X_{n+1}(T) = \delta_n sd + (1+r)(X_n - \delta_n s) = v_{n+1}(sd, y+sd)$$

Subtract the two equations, we get:

$$\delta_n s(u-d) = v_{n+1}(su, y+su) - v_{n+1}(sd, y+sd)$$

Solve for δ_n, we get:

$$\delta_n = \frac{v_{n+1}(su, y+su) - v_{n+1}(sd, y+sd)}{s(u-d)}$$

Problem 4 (Stochastic volatility, random interest rate)

Consider a binomial pricing model, but at each time $n > 1$, the "up factor" $u_n(w_1w_2\dots w_n)$, the "down factor" $d_n(w_1w_2\dots w_n)$, and the interest rate $r_n(w_1w_2\dots w_n)$ are allowed to depend on n and on the first n coin tosses $w_1w_2\dots w_n$. The initial up factor u_0, the initial down factor d_0, and the initial interest rate r_0 are not random. More specifically, the stock price at time one is given by

$$S_1(w_1) = \begin{cases} u_0S_0 & \text{if } w_1 = H \\ d_0S_0 & \text{if } w_1 = T \end{cases}$$

and, for $n > 1$, the stock price at time $n+1$ is given by

$$S_{n+1}(w_1w_2\dots w_nw_{n+1}) = \begin{cases} u_n(w_1w_2\dots w_n)S_n(w_1w_2\dots w_n) & \text{if } w_{n+1} = H \\ d_n(w_1w_2\dots w_n)S_n(w_1w_2\dots w_n) & \text{if } w_{n+1} = T \end{cases}$$

One dollar invested in or borrowed from the money market at time zero grows to an investment or debt of $1 + r_0$ at time one, and, for $n > 1$, one dollar invested in or borrowed from the money market at time n grows to an investment or debt of $1 + r_n(w_1w_2\dots w_n)$ at time $n+1$. We assume that for each n and for all $w_1w_2\dots w_n$, the no arbitrage condition

$$0 < d_n(w_1w_2\dots w_n) < 1 + r_n(w_1w_2\dots w_n) < u_n(w_1w_2\dots w_n)$$

holds. We also assume that $0 < d_0 < 1 + r_0 < u_0$.

Let N be a positive integer. In the model just described:

1. Provide an algorithm for determining the price at time zero for a derivative security that at time N pays off a random amount V_N depending on the result of the first N coin tosses.
2. Provide a formula for the number of shares of stock that should be held at each time n ($0 \le n \le N-1$) by a portfolio that replicates the derivative security V_N.
3. Suppose the initial stock price is $S_0 = 80$, with each head the stock price increases by 10, and with each tail the stock price decreases by 10. In other words, $S_1(H) = 90$, $S_1(T) = 70$, $S_2(HH) = 100$, etc. Assume the interest rate is always zero. Consider a European call with strike price 80, expiring at time five. What is the price of this call at time zero?

Solution:

1. Algorithm for determining the price at time zero:

Model Setup Recap

- At time n, the stock price is $S_n(w_1w_2\dots w_n)$.
- From time n to n+1:
  - If the (n+1)-th toss is heads (H), then $S_{n+1}(w_1\dots w_nH) = u_n(w_1\dots w_n)S_n(w_1\dots w_n)$.

- If the (n+1)-th toss is tails (T), then $S_{n+1}(w_1 \dots w_n T) = d_n(w_1 \dots w_n) S_n(w_1 \dots w_n)$.
- The money-market investment grows from 1 dollar at time n to $1 + r_n(w_1 \dots w_n)$ at time n+1.
- No arbitrage condition: for each n and node $w_1 \dots w_n$, $0 < d_n(w_1 \dots w_n) < 1 + r_n(w_1 \dots w_n) < u_n(w_1 \dots w_n)$.

Risk-Neutral Probability at Each Node

At each node $w_1 \dots w_n$, define the (local) risk-neutral probability $q_n(w_1 \dots w_n) = \frac{(1+r_n(w_1 \dots w_n)) - d_n(w_1 \dots w_n)}{u_n(w_1 \dots w_n) - d_n(w_1 \dots w_n)}$.

Backward-Induction Algorithm

We are given a derivative payoff $V_N(w_1 w_2 \dots w_N)$ at time N, and we want to find the fair price at time 0.

- Initialization (at final time N): $V_N(w_1 \dots w_N)$ is given.
- Backward step (generic node at time n<N): For each history $w_1 w_2 \dots w_n$ up to time n, define $V_n(w_1 \dots w_n) = \frac{1}{1+r_n(w_1 \dots w_n)} [q_n(w_1 \dots w_n) V_{n+1}(w_1 \dots w_n H) + (1 - q_n(w_1 \dots$
.
- Repeat until n=0. Then V_0 is the time-0 fair price.

2. Formula for the Replicating-Portfolio Holdings (Delta)

At time n in state $w_1 \dots w_n$, we replicate the derivative V_{n+1} for the next step by holding $\Delta_n(w_1 \dots w_n)$ shares of stock and an amount $B_n(w_1 \dots w_n)$ in the money market:

$\Delta_n(w_1 \dots w_n) = \frac{V_{n+1}(w_1 \dots w_n H) - V_{n+1}(w_1 \dots w_n T)}{S_n(w_1 \dots w_n)(u_n(w_1 \dots w_n) - d_n(w_1 \dots w_n))}$

Similarly, $B_n(w_1 \dots w_n) = \frac{u_n(\dots) V_{n+1}(\dots, T) - d_n(\dots) V_{n+1}(\dots, H)}{(u_n(\dots) - d_n(\dots))(1 + r_n(\dots))}$.

3. Concrete Example

Given:

- $S_0 = 80$.
- Each heads adds 10 (so S goes up by 10).
- Each tails subtracts 10 (so S goes down by 10).
- The interest rate is always 0.
- A European call with strike K=80, expiring at time N=5.

Model Details:

- Since r=0, the risk-neutral probability at each step is $q = \frac{1}{2}$.
- At each step, a head takes S_n to $S_n + 10$ and a tail takes S_n to $S_n - 10$.

Final Payoffs at n=5:

- The derivative is a call with strike 80: Payoff = $\max\{S_5 - 80, 0\}$
- If X is the number of heads in 5 tosses, $S_5 = 30 + 20X$, for X=0,1,2,3,4,5
- So the payoff is $V_5 = \max\{20X - 50, 0\}$, which equals:
  - 0, if X < 3
  - 10, if X = 3
  - 30, if X = 4
  - 50, if X = 5

Risk-Neutral Pricing:

- Since r=0, the time-0 fair price is simply the expected payoff: $V_0 = (\frac{1}{2})^5[\binom{5}{3} \cdot 10 + \binom{5}{4} \cdot 30 + \binom{5}{5} \cdot 50]$
- Computing: $\binom{5}{3} = 10$, $\binom{5}{4} = 5$, $\binom{5}{5} = 1$
- So $V_0 = \frac{1}{32}[100 + 150 + 50] = \frac{300}{32} = 9.375$

The call price at time zero is $9.375.

Problem 5

Consider a binomial model with $d = \frac{1}{u}$. Suppose we are concerned with the fluctuations of a call option price due to the underlying stock price and scale factor u . We are not satisfied with the linear approximation and would like to do better. Therefore, we consider the quadratic approximation of the form:

$$V(S, u) = a + b \cdot S + c \cdot u + d \cdot S^2 + e \cdot u^2 + f \cdot S \cdot u$$

1. Extend the analysis from the linear approximation to the quadratic approximation by incorporating second-order derivatives.
2. Calculate the second-order derivatives of the call option and plug into the quadratic approximation (For complicated ones, you can use software).
3. Assume $S_0 = 4$, $u = 2$, $K = 5$, $r = \frac{1}{4}$. Consider $\tilde{S}_0 = 5$ and $\tilde{u} = 2.5$, compare the linear approximation, quadratic approximation with the true option price, and comment.

Solution:

1. Extend the analysis from the linear approximation to the quadratic approximation by incorporating second-order derivatives.

A recap of the linear approximation:

$$V(S_0 + \Delta S,\, u_0 + \Delta u) \approx V(S_0, u_0) + \frac{\partial V}{\partial S}(S_0, u_0)\,\Delta S + \frac{\partial V}{\partial u}(S_0, u_0)\,\Delta u.$$

The quadratic approximation, use the 2nd order Taylor expansion:

$$V(S_0+\Delta S,\ u_0+\Delta u)\ \approx\ V(S_0,u_0)\ +\ \underbrace{\frac{\partial V}{\partial S}(S_0,u_0)\,\Delta S\ +\ \frac{\partial V}{\partial u}(S_0,u_0)\,\Delta u}\,\text{line}$$

$$+\ \underbrace{\frac{1}{2}\,\frac{\partial^2 V}{\partial S^2}(S_0,u_0)\,(\Delta S)^2\ +\ \frac{1}{2}\,\frac{\partial^2 V}{\partial u^2}(S_0,u_0)\,(\Delta u)^2\ +}$$

In Greeks, we have:

$$V(S_0+\Delta S,\ u_0+\Delta u)\approx V(S_0,u_0)+\Delta_S\,\Delta S+\Delta_u\,\Delta u+\tfrac{1}{2}\,\Gamma_S\,(\Delta S)^2+\tfrac{1}{2}\,\Gamma_u\,(\Delta u$$

2. Calculate the second-order derivatives of the call option and plug into the quadratic approximation (For complicated ones, you can use software).

The pricing formula is, with $\tilde{p}=\frac{u+ur-1}{u^2-1}$:

$$V_0=\frac{\tilde{p}}{1+r}(uS_0-K)$$

i.e.

$$V_0=\frac{1}{1+r}\,\frac{(u(1+r)-1)(uS_0-K)}{u^2-1}$$

In [2]:
```python
import sympy

# Define the symbols
S, u, r, K = sympy.symbols('S u r K', real=True)
# We assume 1 + r > 0, u^2 != 1, etc., but do not force positivity in sym

# Define the call-price function:
# V(u,S) = (1/(1+r)) * [ (u(1+r) - 1)*(u*S - K ) / (u^2 - 1 ) ]
V = ( (u*(1+r) - 1)*(u*S - K) ) / (u**2 - 1) / (1+r)

# First-order partial derivatives
dV_dS = V.diff(S)
dV_du = V.diff(u)

# Second-order partial derivatives
d2V_dS2   = dV_dS.diff(S)
d2V_du2   = dV_du.diff(u)
d2V_dSdu  = dV_dS.diff(u)   # or equivalently dV_du.diff(S)

# Print them in a (somewhat) simplified, symbolic form
print("\nV(u,S) =")
sympy.pprint(sympy.simplify(V))

print("\n∂V/∂S =")
sympy.pprint(sympy.simplify(dV_dS))

print("\n∂V/∂u =")
sympy.pprint(sympy.simplify(dV_du))
```

```
print("\n∂²V/∂S² =")
sympy.pprint(sympy.simplify(d2V_dS2))

print("\n∂²V/∂u² =")
sympy.pprint(sympy.simplify(d2V_du2))

print("\n∂²V/∂S∂u =")
sympy.pprint(sympy.simplify(d2V_dSdu))
```

```
V(u,S) =
-(K - S⋅u)⋅(u⋅(r + 1) - 1)
───────────────────────────
          ⎛ 2    ⎞
 (r + 1)⋅⎝u  - 1⎠

∂V/∂S =
u⋅(u⋅(r + 1) - 1)
─────────────────
        ⎛ 2    ⎞
(r + 1)⋅⎝u  - 1⎠

∂V/∂u =
  ⎛ 2    ⎞
↪
S⋅⎝u  - 1⎠⋅(u⋅(r + 1) - 1) + 2⋅u⋅(K - S⋅u)⋅(u⋅(r + 1) - 1) + (-K + S⋅u)⋅(r
+ 1 ↪
──────────────────────────────────────────────────────────────────────────
──── ↪
                                                                  2
↪
                                                          ⎛ 2    ⎞
↪
                                                  (r + 1)⋅⎝u  - 1⎠
↪

↪   ⎛ 2    ⎞
↪ )⋅⎝u  - 1⎠
↪ ──────────
↪
↪
↪

∂²V/∂S² =
0

∂²V/∂u² =
  ⎛                  2
↪
  ⎜          ⎛ 2    ⎞        2
↪
2⋅⎝S⋅(r + 1)⋅⎝u  - 1⎠  - 4⋅u ⋅(K - S⋅u)⋅(u⋅(r + 1) - 1) + 2⋅u⋅(K - S⋅u)⋅(r
+ 1 ↪
──────────────────────────────────────────────────────────────────────────
──── ↪
                                                                                   3
↪
                                                                              ⎛ 2    ⎞
↪
                                                                      (r + 1)⋅⎝u  - 1⎠
↪

↪                                                       ⎞
↪   ⎛ 2    ⎞                  ⎛ 2    ⎞                  ⎟
↪ )⋅⎝u  - 1⎠ + (K - 3⋅S⋅u)⋅⎝u  - 1⎠⋅(u⋅(r + 1) - 1)⎠
↪ ──────────────────────────────────────────────────
↪
↪
↪
```

```
∂²V/∂S∂u =
                 2
      -2⋅r⋅u + u  - 2⋅u + 1
─────────────────────────────────
   4         2         4       2
r⋅u  - 2⋅r⋅u  + r + u  - 2⋅u  + 1
```

3. Assume $S_0 = 4$, $u = 2$, $K = 5$, $r = \frac{1}{4}$. Consider $\tilde{S}_0 = 5$ and $\tilde{u} = 2.5$, compare the linear approximation, quadratic approximation with the true option price, and comment.

In [6]:
```python
import sympy

# ----------------------------------------------------------
# Minimal example comparing exact, linear, and quadratic prices
# for the one-period binomial call:
#     V(S,u) = [(u*(1+r)-1)*(u*S - K)] / [(1+r)*(u^2 - 1)]
# around (S0,u0) = (4,2). We'll evaluate at (S_new,u_new) = (5,2.5).
# ----------------------------------------------------------

# 1) Define symbols and the exact pricing formula:
S, u = sympy.symbols('S u', real=True, positive=True)
r_val, K_val = 0.25, 5
V_expr = ((u*(1+r_val) - 1) * (u*S - K_val)) / ((1+r_val)*(u**2 - 1))

# 2) Make a callable function for V(S,u):
V_func = sympy.lambdify((S,u), V_expr, 'math')

# 3) Compute exact price at base (S0,u0) and new (S_new,u_new):
S0, u0 = 4, 2
S_new, u_new = 5, 2.5
V0          = V_func(S0, u0)          # exact at (4,2)
V_new_exact= V_func(S_new, u_new)    # exact at (5,2.5)

# 4) Get partial derivatives symbolically and evaluate at (S0,u0):
dV_dS_expr = V_expr.diff(S)
dV_du_expr = V_expr.diff(u)
d2V_dS2_expr   = dV_dS_expr.diff(S)
d2V_du2_expr   = dV_du_expr.diff(u)
d2V_dSdu_expr  = dV_dS_expr.diff(u)

dV_dS    = dV_dS_expr.subs({S:S0, u:u0})
dV_du    = dV_du_expr.subs({S:S0, u:u0})
d2V_dS2  = d2V_dS2_expr.subs({S:S0, u:u0})
d2V_du2  = d2V_du2_expr.subs({S:S0, u:u0})
d2V_dSdu = d2V_dSdu_expr.subs({S:S0, u:u0})

# 5) Define the linear and quadratic approximations at (S_new,u_new):
dS, dU = (S_new - S0), (u_new - u0)

V_lin_approx = V0 + dV_dS*dS + dV_du*dU
V_quad_approx= (V0
                + dV_dS*dS + dV_du*dU
                + 0.5*d2V_dS2*(dS**2)
                + 0.5*d2V_du2*(dU**2)
                + d2V_dSdu*(dS*dU))

# 6) Print results and errors:
```

```
print("Exact price at (4,2)          = ", float(V0))
print("Exact price at (5,2.5)        = ", float(V_new_exact))

print("\n--- Partial derivatives at (4,2) ---")
print("∂V/∂S   =", float(dV_dS))
print("∂V/∂u   =", float(dV_du))
print("∂²V/∂S² =", float(d2V_dS2))
print("∂²V/∂u² =", float(d2V_du2))
print("∂²V/∂S∂u=", float(d2V_dSdu))

print("\n--- Approximations at (5,2.5) ---")
print("Linear approximation  =", float(V_lin_approx))
print("Quadratic approximation =", float(V_quad_approx))

err_lin  = float(V_lin_approx - V_new_exact)
err_quad = float(V_quad_approx - V_new_exact)
print("\nErrors vs exact:")
print("Linear error    =", round(err_lin, 4))
print("Quadratic error =", round(err_quad, 4))
```

```
Exact price at (4,2)          =  1.2
Exact price at (5,2.5)        =  2.4285714285714284

--- Partial derivatives at (4,2) ---
∂V/∂S   = 0.8
∂V/∂u   = 1.0
∂²V/∂S² = 0.0
∂²V/∂u² = -0.7999999999999998
∂²V/∂S∂u= 0.0

--- Approximations at (5,2.5) ---
Linear approximation  = 2.5
Quadratic approximation = 2.4

Errors vs exact:
Linear error    = 0.0714
Quadratic error = -0.0286
```

We observe that the linear approximation overestimates the option price, while the quadratic approximation underestimates the option price. Overally, the quadratic approximation is more accurate by the much smaller absolute value of error.

Problem 6

Download the S&P 500 data (use close prices) online (e.g., from Nasdaq) between 2020.1.1 and 2024.12.31 (excluding all weekends and holidays). Denote the stock prices in order as S_i and daily log return as $R_i = \log(S_{i+1}) - \log(S_i)$ (assuming nothing happens during weekends and holidays). Denote the total number of daily return as n.

1. Visualize the trajectory of log prices.

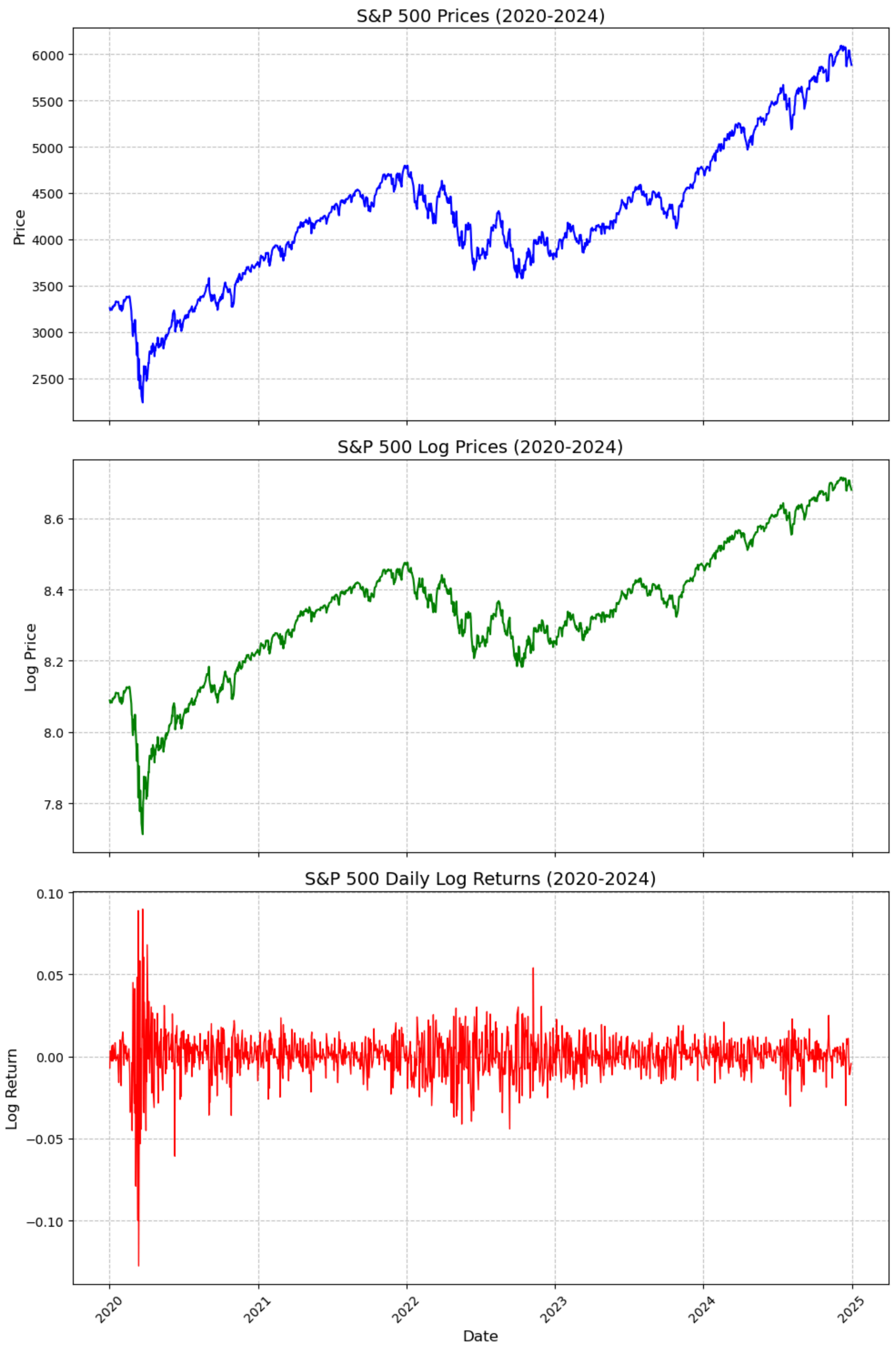
2. Estimate the mean and variance of R_i as $\hat{\mu}$ and $\hat{\sigma}^2$. Assuming the estimated variance $\hat{\sigma}^2$ is the true variance σ^2, calculate the accuracy of the mean estimator

by $\sqrt{\mathbb{E}[(\mu - \frac{1}{n}\sum_{i=1}^{n} R_i)^2]} = \frac{\sigma}{\sqrt{n}}$. Assume return data is i.i.d., provide a 95% confidence interval of $\hat{\mu}$.

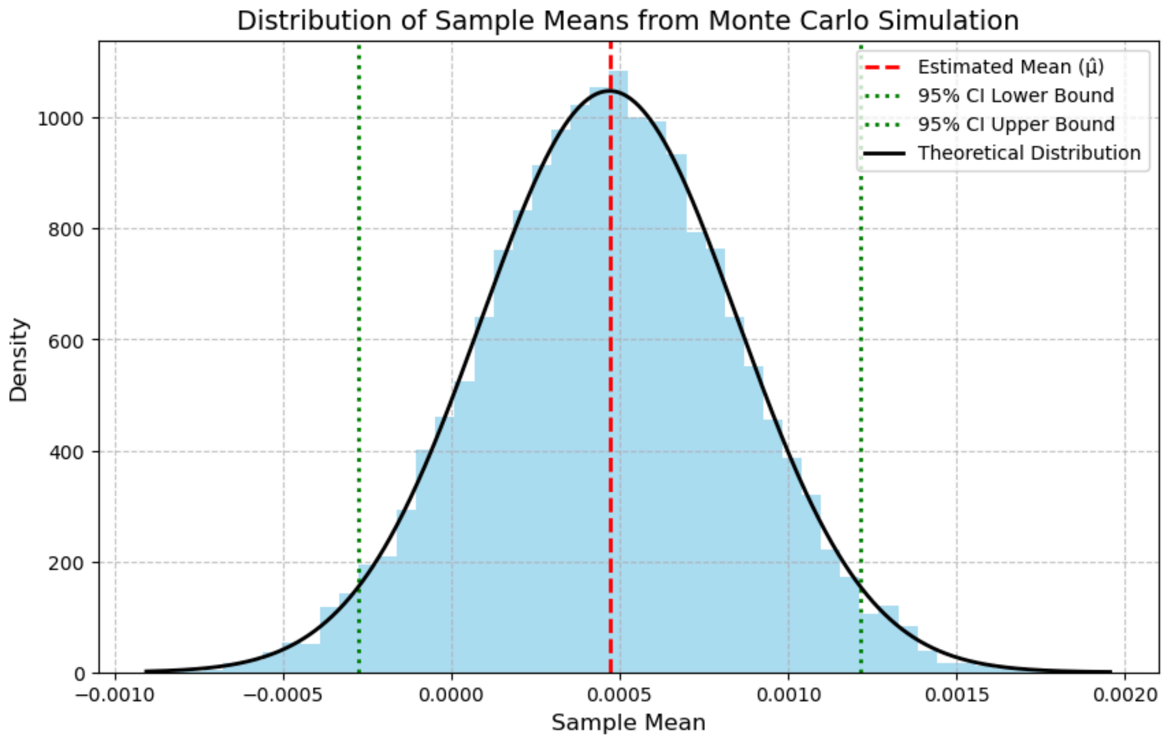
3. Assume the log return follows the normal distribution with the estimated mean and variance that $R_i \sim \mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$. Numerically verify the accuracy of the mean estimator above by Monte Carlo simulation. To do this, generate M (e.g., 10000) sequences with n i.i.d normal random variables $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$. For each sequence, estimate its mean. Then calculate the standard deviation of the mean estimator and compare it with the formula above. Convince yourself that the mean estimation of log return for S&P 500 is inaccurate.

In [29]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Read the data
dataframe = pd.read_csv('data/SP500_HistoricalData.csv', parse_dates=['Da
S = pd.to_numeric(dataframe['Close/Last'])
R = np.log(S.shift(-1) / S)

# 1. Visualize the trajectory of log prices.
# Create a figure with 3 subplots
fig, axs = plt.subplots(3, 1, figsize=(10, 15), sharex=True)

# Plot 1: S&P 500 Prices
axs[0].plot(S.index, S, color='blue', linewidth=1.5)
axs[0].set_title('S&P 500 Prices (2020-2024)', fontsize=14)
axs[0].set_ylabel('Price', fontsize=12)
axs[0].grid(True, linestyle='--', alpha=0.7)

# Plot 2: Log Prices
log_prices = np.log(S)
axs[1].plot(S.index, log_prices, color='green', linewidth=1.5)
axs[1].set_title('S&P 500 Log Prices (2020-2024)', fontsize=14)
axs[1].set_ylabel('Log Price', fontsize=12)
axs[1].grid(True, linestyle='--', alpha=0.7)

# Plot 3: Log Returns
axs[2].plot(R.index, R, color='red', linewidth=1)
axs[2].set_title('S&P 500 Daily Log Returns (2020-2024)', fontsize=14)
axs[2].set_xlabel('Date', fontsize=12)
axs[2].set_ylabel('Log Return', fontsize=12)
axs[2].grid(True, linestyle='--', alpha=0.7)

plt.xticks(rotation=45)
plt.tight_layout()
plt.margins(x=0.01)
plt.show()

# 2. Estimate the mean and variance of $R_i$ as $\hat{\mu}$ and $\hat{\si
mu_hat = R.mean()
var_hat = R.var()

# Calculate the accuracy of the mean estimator
n = len(R)
accuracy = np.sqrt(var_hat / n)
```

```
# Calculate the 95% confidence interval
import scipy.stats as stats
confidence_level = 0.95
z_score = stats.norm.ppf((1 + confidence_level) / 2)  # z-score for 95%
margin_of_error = z_score * accuracy
ci_lower = mu_hat - margin_of_error
ci_upper = mu_hat + margin_of_error

print(f"Estimated mean (μ̂): {mu_hat:.6f}")
print(f"Estimated variance (σ̂²): {var_hat:.6f}")
print(f"Standard error of the mean: {accuracy:.6f}")
print(f"95% Confidence Interval: [{ci_lower:.6f}, {ci_upper:.6f}]")

# 3. Assume the log return follows the normal distribution with the estim

# Monte Carlo simulation to verify the accuracy of the mean estimator
np.random.seed(42)  # For reproducibility
M = 10000  # Number of simulations
sample_means = np.zeros(M)
for i in range(M):
    simulated_returns = np.random.normal(mu_hat, np.sqrt(var_hat), n)
    sample_means[i] = np.mean(simulated_returns)

empirical_std = np.std(sample_means)

print(f"\nMonte Carlo Verification:")
print(f"Number of simulations (M): {M}")
print(f"Theoretical standard error: {accuracy:.6f}")
print(f"Empirical standard error from simulation: {empirical_std:.6f}")
print(f"Relative difference: {100 * abs(empirical_std - accuracy) / accur

# Visualize the distribution of sample means
plt.figure(figsize=(10, 6))
plt.hist(sample_means, bins=50, alpha=0.7, color='skyblue', density=True)
plt.axvline(mu_hat, color='red', linestyle='--', linewidth=2, label='Esti
plt.axvline(ci_lower, color='green', linestyle=':', linewidth=2, label='9
plt.axvline(ci_upper, color='green', linestyle=':', linewidth=2, label='9

# Add a normal curve with the theoretical mean and standard error
x = np.linspace(min(sample_means), max(sample_means), 1000)
plt.plot(x, stats.norm.pdf(x, mu_hat, accuracy), 'k-', linewidth=2, label
plt.title('Distribution of Sample Means from Monte Carlo Simulation', fon
plt.xlabel('Sample Mean', fontsize=12)
plt.ylabel('Density', fontsize=12)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()

# Assess the accuracy of the mean estimator
print("\nAssessment of Mean Estimator Accuracy:")
print(f"Width of 95% CI: {2 * margin_of_error:.6f}")
print(f"Relative width compared to mean: {100 * 2 * margin_of_error / abs

if 100 * 2 * margin_of_error / abs(mu_hat) > 50:
    print("The mean estimation is inaccurate as the confidence interval i
    print("This suggests high uncertainty in the estimation of the true m
```


S&P 500 Prices (2020-2024)
Price
6000
5500
5000
4500
4000
3500
3000
2500
S&P 500 Log Prices (2020-2024)
Log Price
8.6
8.4
8.2
8.0
7.8
S&P 500 Daily Log Returns (2020-2024)
Log Return
0.10
0.05
0.00
−0.05
−0.10
2020
2021
2022
2023
2024
2025
Date

```
Estimated mean (μ̂): 0.000470
Estimated variance (σ̂²): 0.000182
Standard error of the mean: 0.000381
95% Confidence Interval: [-0.000276, 0.001216]

Monte Carlo Verification:
Number of simulations (M): 10000
Theoretical standard error: 0.000381
Empirical standard error from simulation: 0.000378
Relative difference: 0.7403%
```



```
Assessment of Mean Estimator Accuracy:
Width of 95% CI: 0.001492
Relative width compared to mean: 317.45%
The mean estimation is inaccurate as the confidence interval is very wide
relative to the mean value.
This suggests high uncertainty in the estimation of the true mean of log r
eturns.
```