Strathmore
UNIVERSITY

Computational Methods Assignment 2: No. 2

Omondi EdJoel, 095892

September 2025

1 Question 2

1.1 Generalized Black–Scholes PDE with Stochastic Volatility (Derivation)

Model. Let the underlying and its (instantaneous) variance evolve as

$$\begin{aligned} dS_t &= \mu\, S_t\, dt + \sigma_t\, S_t\, dZ_t, \qquad v_t := \sigma_t^2, \\ dv_t &= \kappa(\theta - v_t)\, dt + \gamma\sqrt{v_t}\, dW_t, \end{aligned}$$

with correlated Brownian motions Z, W such that

$$\mathrm{Cov}(dZ_t, dW_t) = \rho\, dt, \qquad |\rho| \leq 1.$$

Let $U = U(S, v, t)$ denote the time-t value of a European-style claim maturing at T.

Itô's formula for $U(S, v, t)$. The quadratic (co)variations are

$$d\langle S\rangle_t = \sigma_t^2 S_t^2\, dt = v_t S_t^2\, dt, \qquad d\langle v\rangle_t = \gamma^2 v_t\, dt, \qquad d\langle S, v\rangle_t = \mathrm{Cov}(dS_t, dv_t) = \rho\,\gamma\, v_t S_t\, dt.$$

Applying Itô to $U(S_t, v_t, t)$,

$$\begin{aligned} dU_t &= U_t\, dt + U_S\, dS_t + U_v\, dv_t + \tfrac{1}{2} U_{SS}\, d\langle S\rangle_t + \tfrac{1}{2} U_{vv}\, d\langle v\rangle_t + U_{Sv}\, d\langle S, v\rangle_t \\ &= \Big(U_t + \mu S\, U_S + \kappa(\theta - v)\, U_v + \tfrac{1}{2} v S^2\, U_{SS} + \rho\,\gamma\, v S\, U_{Sv} + \tfrac{1}{2}\gamma^2 v\, U_{vv}\Big) dt \\ &\quad + U_S\, \sigma S\, dZ_t + U_v\, \gamma\sqrt{v}\, dW_t, \end{aligned}$$

where, for brevity, we write $S = S_t$, $v = v_t$, $\sigma = \sigma_t$, and $U_t = \partial U/\partial t$, $U_S = \partial U/\partial S$, etc.

Risk-neutral valuation and volatility risk premium. In incomplete stochastic-volatility markets, the variance factor v carries an (unspanned) risk. Introduce a (possibly affine) market price of volatility risk $\lambda(v)$; a common linear specification is $\lambda(v) = \lambda v$. Under the risk-neutral measure Q:

$$\frac{dS_t}{S_t} = r\, dt + \sigma_t\, dZ_t^Q, \qquad dv_t = \big[\kappa(\theta - v_t) - \lambda(v_t)\big]\, dt + \gamma\sqrt{v_t}\, dW_t^Q,$$

with $\mathrm{Cov}(dZ_t^Q, dW_t^Q) = \rho\, dt$. The Q-generator $\mathcal{L}^Q$ acting on U is then

$$\mathcal{L}^Q U = rS\, U_S + \big[\kappa(\theta - v) - \lambda(v)\big]\, U_v + \tfrac{1}{2} v S^2\, U_{SS} + \rho\,\gamma\, v S\, U_{Sv} + \tfrac{1}{2}\gamma^2 v\, U_{vv}.$$

Pricing PDE (no-arbitrage / Feynman–Kac). Discounted claim values are Q-martingales; equivalently, U solves

$$U_t + \mathcal{L}^Q U - rU = 0.$$

With $\lambda(v) = \lambda v$ this becomes the *generalized Black–Scholes PDE*:

$$\boxed{\frac{1}{2}vS^2\frac{\partial^2 U}{\partial S^2} + \rho\gamma vS\frac{\partial^2 U}{\partial v\,\partial S} + \frac{1}{2}\gamma^2 v\frac{\partial^2 U}{\partial v^2} + rS\frac{\partial U}{\partial S} + \big[\kappa(\theta - v) - \lambda v\big]\frac{\partial U}{\partial v} - rU + \frac{\partial U}{\partial t} = 0.}$$

Terminal/Boundary conditions. For a European payoff $\Phi(S_T)$ one imposes $U(S, v, T) = \Phi(S)$ together with appropriate growth and boundary conditions in (S, v) to ensure uniqueness.

Appendix: Starter Code (Simulation of (S_t, v_t))

Purpose. The below shows Python code to simulate correlated (S_t, v_t) over $[0, T]$ under the physical measure (or Q if you set $\mu = r$ and replace the v-drift with $\kappa(\theta - v) - \lambda v$). This is *not* needed for the PDE derivation, but handy if you want to *illustrate* the model dynamics alongside your write-up.

Listing 1: Stochastic volatility (CIR-type variance) with correlated Brownian motions

```python
import numpy as np

def simulate_sv_paths(
    S0=100.0, v0=0.04, mu=0.05, r=0.05,
    kappa=1.5, theta=0.04, gamma=0.5,
    rho=-0.6, T=1.0, steps=1000, paths=10000, seed=42,
    risk_neutral=False, lamb=0.0
):
    """
    Log-Euler for S (keeps S>0), full truncation Euler for v (keeps v >= 0).
    If risk_neutral=True, use mu=r and v-drift kappa(theta-v)-lambda*v.
    """
    rng = np.random.default_rng(seed)
    dt  = T / steps
    S   = np.full(paths, S0, dtype=float)
    v   = np.full(paths, v0, dtype=float)

    for _ in range(steps):
        Z1 = rng.standard_normal(paths)
        Z2 = rng.standard_normal(paths)
        dW1 = np.sqrt(dt) * Z1
        dW2 = np.sqrt(dt) * (rho * Z1 + np.sqrt(max(1 - rho**2, 0.0)) * Z2)

        vpos = np.maximum(v, 0.0)
        # choose drift for v
        if risk_neutral:
            drift_v = kappa * (theta - vpos) - lamb * vpos
            drift_S = r
        else:
            drift_v = kappa * (theta - vpos)
            drift_S = mu

        # log-Euler for S_t:
        S *= np.exp((drift_S - 0.5 * vpos) * dt + np.sqrt(vpos) * dW1)
```

```
        # full truncation Euler for v_t:
        v += drift_v * dt + gamma * np.sqrt(vpos) * dW2

    return S, np.maximum(v, 0.0)

# Example (test):
if __name__ == "__main__":
    ST, vT = simulate_sv_paths(
        S0=100, v0=0.04, mu=0.05, r=0.05,
        kappa=1.0, theta=0.04, gamma=0.5,
        rho=-0.5, T=1.0, steps=4000, paths=20000,
        risk_neutral=True, lamb=0.0, seed=123
    )
    print("ST mean:", ST.mean(), "  vT mean:", vT.mean())
```