L04_BS-PDE

February 12, 2026

0.1 European Vanilla Options: Basic Properties

Before deriving the Black–Scholes PDE and computing Greeks, it is instructive to analyze the **static properties** of the Black–Scholes pricing formulas. These properties provide intuition, enable fast approximations, and form the basis for many practical trading heuristics.

0.1.1 Recall: Black–Scholes Pricing Formulas

To emphasize the distinction between the **contract specifications** (K, T) and the **model parameter** (σ), we denote the pricing function as $C(K, T; \sigma)$.

The Call Option

$$C_{\text{BS}}(K, T; S_0, \sigma) = S_0 N(d_1) - K e^{-rT} N(d_2),$$

The Put Option

$$P_{\text{BS}}(K, T; S_0, \sigma) = K e^{-rT} N(-d_2) - S_0 N(-d_1),$$

with d_1 and d_2 given by:

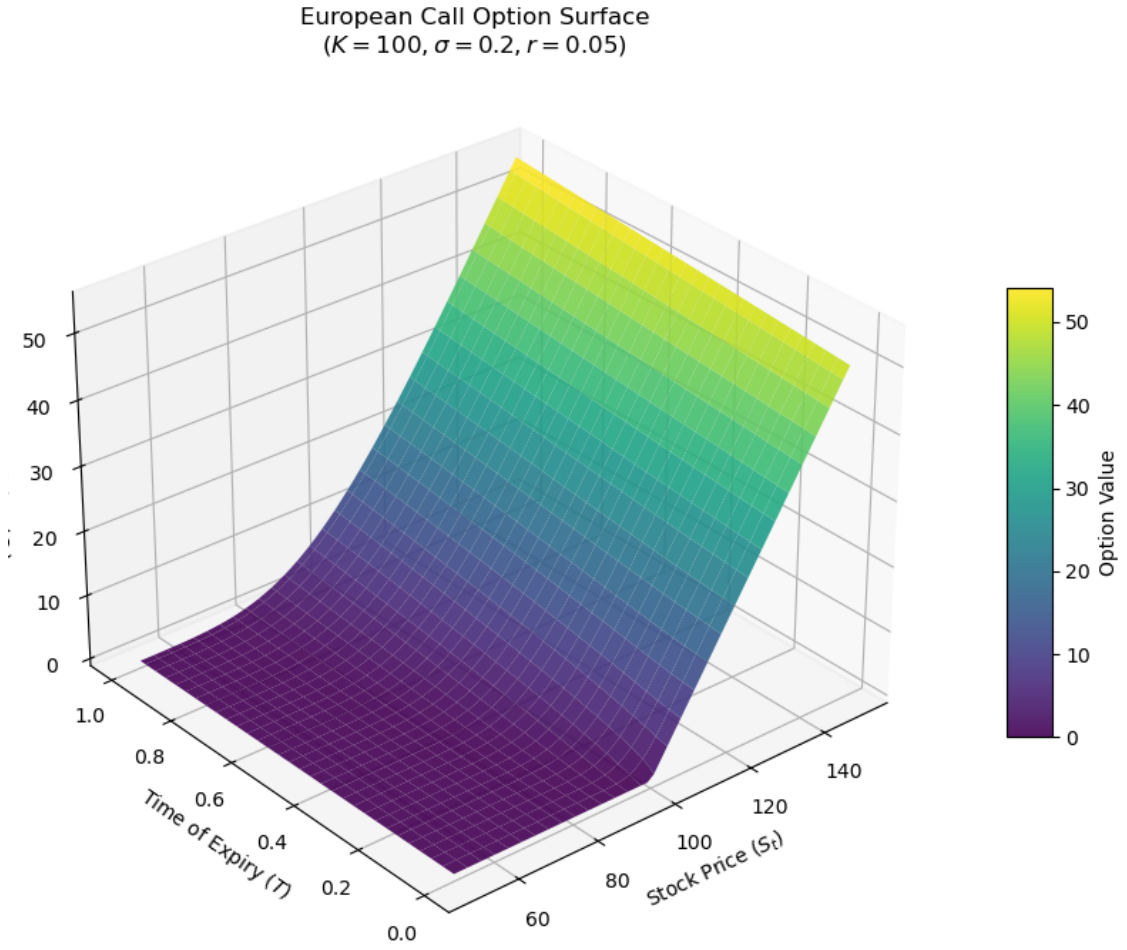
$$d_1 = \frac{\ln(S_0/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \quad \text{and} \quad d_2 = d_1 - \sigma\sqrt{T}$$

Functional Dependencies: * **Variables:** K (Strike) and T (Maturity) define the specific contract in the chain. * **Parameter:** σ (Volatility) is the unobservable model parameter we must calibrate. * **State:** S_0 (Spot) and r (Rate) are observed market data points, treated as constants for the moment of pricing.

[12]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# --- 1. Vectorized Pricing Function ---
def bs_call_price(S, T, K, r, sigma):
    """
    Vectorized Black-Scholes Call Price.
    Inputs S and T can be scalars or numpy arrays/grids.
    """
    # Safety: Ensure T is not zero for division, but keep track of indices
    # We replace 0 with a tiny epsilon to calculate d1/d2 without warning
    safe_T = np.maximum(T, 1e-12)
```

```
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * safe_T) / (sigma * np.
 ↪sqrt(safe_T))
    d2 = d1 - sigma * np.sqrt(safe_T)

    price = S * norm.cdf(d1) - K * np.exp(-r * safe_T) * norm.cdf(d2)

    # Boundary Condition: If T is effectively 0, price is intrinsic value␣
 ↪max(S-K, 0)
    intrinsic_value = np.maximum(S - K, 0.0)
    return np.where(T <= 1e-12, intrinsic_value, price)

# --- 2. Parameters & Grid Generation ---
K = 100
r = 0.05
sigma = 0.2
T_max = 1.0

# Generate Grid
# We include T=0 to explicitly show the "Hockey Stick" payoff
S_vals = np.linspace(50, 150, 60)
T_vals = np.linspace(0.0, T_max, 60)

# Create 2D Coordinate Matrices (Meshgrid)
S_grid, T_grid = np.meshgrid(S_vals, T_vals)

# --- 3. Compute Surface (Vectorized) ---
# No loops required - passes entire grids at once
Call_Surface = bs_call_price(S_grid, T_grid, K, r, sigma)

# --- 4. Visualization ---
fig = plt.figure(figsize=(12, 7))
ax = fig.add_subplot(projection='3d')

# Plot Surface with Colormap (viridis)
surf = ax.plot_surface(S_grid, T_grid, Call_Surface, cmap='viridis',
                       edgecolor='none', alpha=0.9)

# Labels and Styling
ax.set_xlabel(r'Stock Price ($S_t$)')
ax.set_ylabel(r'Time of Expiry ($T$)')
ax.set_zlabel(r'Option Price ($C_t$)')
ax.set_title(f'European Call Option Surface\n($K={K}, \sigma={sigma}, r={r}$)')

# Adjust view angle for best perspective
ax.view_init(elev=30, azim=230)
```

```
# Add Colorbar
cbar = fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10)
cbar.set_label('Option Value')

plt.tight_layout()
plt.show()
```



0.1.2 Formulation in Moneyness

A crucial property of the Black-Scholes formula is **Linear Homogeneity** with respect to the asset price S_0 and strike K.

Mathematically, for any constant $\lambda > 0$, we have $C_{\text{BS}}(T, \lambda K; \lambda S_0, \sigma) = \lambda C_{\text{BS}}(T, K; S_0, \sigma)$.

This allows us to reduce the number of variables by normalizing the entire equation by the strike price K. We define **Moneyness** (ξ) as:

$$\xi = \frac{S_0}{K}$$

We can now analyze the **normalized price** $c(\xi)$ (the option price expressed as a percentage of the Strike):

$$c(\xi) := \frac{C_{\text{BS}}(T, K; S_0, \sigma)}{K} = C_{\text{BS}}\left(T, 1; \frac{S_0}{K}, \sigma\right)$$

0.1.3 The Moneyness Formulas

By dividing the standard Black-Scholes equation by K and substituting $S_0 = \xi K$, we obtain:

$$c(\xi) = \xi N(d_1) - e^{-rT} N(d_2)$$

Where the arguments now depend only on moneyness ξ, time to maturity T, and volatility σ:

$$d_1 = \frac{\ln(\xi) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}, \qquad d_2 = d_1 - \sigma\sqrt{T}$$

Computational Advantage: This transformation reduces the dimensionality of the problem. Instead of a 4-dimensional surface (S_0, K, T, σ), we now work with a 3-dimensional surface (ξ, T, σ). This is standard practice when training Machine Learning models for derivatives pricing or building interpolation grids.

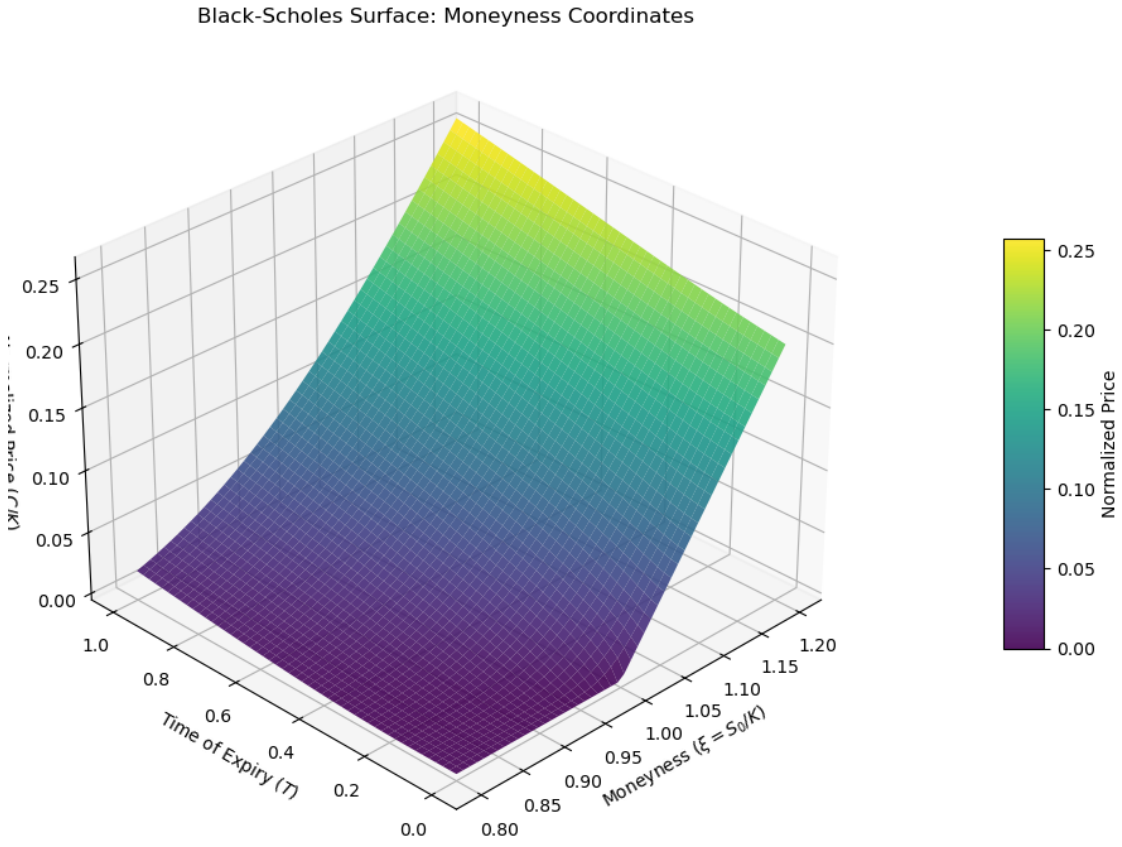
The term **Moneyness** describes the relationship between the current asset price (S) and the strike price (K). It indicates whether an option would result in a positive cash flow if exercised immediately.

Term	Call Option (S vs K)	Put Option (S vs K)	Definition
In the Money (ITM)	$S > K$	$S < K$	The option has **intrinsic value**.
At the Money (ATM)	$S \approx K$	$S \approx K$	The asset price is at the strike.
Out of the Money (OTM)	$S < K$	$S > K$	The option has **zero intrinsic value**.

[13]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Parameters
r = 0.05
sigma = 0.2
T_max = 1.0

# Black-Scholes Normalized Call Price
```

```
# Input: xi = S/K (Moneyness), T = Time to Maturity
# Output: c(xi) = Call Price / K
def call_price_moneyness(xi, T, r=r, sigma=sigma):
    # Handle T=0 edge case to avoid division by zero
    if T <= 1e-8:
        return np.maximum(xi - 1.0, 0.0)

    # Calculate d1 and d2 using xi
    # Note: ln(S/K) becomes ln(xi)
    d1 = (np.log(xi) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    # Normalized price formula: xi * N(d1) - e^{-rT} * N(d2)
    return xi * norm.cdf(d1) - np.exp(-r * T) * norm.cdf(d2)

# Grid Generation
# xi (Moneyness): 0.8 (OTM) to 1.2 (ITM)
xi_vals = np.linspace(0.8, 1.2, 50)
T_vals = np.linspace(1e-4, T_max, 50)

# Create Meshgrid for 3D plotting
Xi_grid, T_grid = np.meshgrid(xi_vals, T_vals)

# Vectorized Calculation
# Compute Z = c(xi) over the grid
Z = np.array([call_price_moneyness(x, t) for x, t in zip(np.ravel(Xi_grid), np.
 ↪ravel(T_grid))])
Z = Z.reshape(Xi_grid.shape)

# Plotting
fig = plt.figure(figsize=(12, 7))
ax = fig.add_subplot(projection='3d')

# Plot the surface
surf = ax.plot_surface(Xi_grid, T_grid, Z, cmap='viridis', edgecolor='none',␣
 ↪alpha=0.9)

# Labels and Styling
ax.set_xlabel(r"Moneyness ($\xi = S_0/K$)")
ax.set_ylabel(r"Time of Expiry ($T$)")
ax.set_zlabel(r"Normalized Price ($C/K$)")
ax.set_title("Black-Scholes Surface: Moneyness Coordinates")

# Adjust view angle for better perspective
ax.view_init(elev=30, azim=225)

# Add colorbar
```

```
plt.colorbar(surf, shrink=0.5, aspect=10, label='Normalized Price', pad=0.1)

plt.tight_layout()
plt.show()
```



So far, we have treated volatility σ as a known model parameter provided by the user. In reality, volatility is **not directly observable** in the market.

What we *do* observe are the market prices of traded options, which we denote as C_{mkt}.

The Inverse Problem Using our notation where the pricing function is parameterized by σ:

$$C_{\text{BS}}(K, T; S_0, \sigma) = S_0 N(d_1) - K e^{-rT} N(d_2)$$

We seek to invert this relationship.

Definition The **Implied Volatility**, denoted $\sigma_{\text{imp}}(K, T)$, is defined as the unique value of the volatility parameter that, when plugged into the Black-Scholes formula, equates the model price to the observed market price.

Mathematically, it is the solution to the equation:

$$C_{\text{BS}}(K, T; S_0, \sigma_{\text{imp}}) = C_{\text{mkt}}(K, T)$$

Interpretation * **A Quoting Convention:** Traders quote prices in terms of volatility (e.g., "18.5 vols") rather than currency (e.g., "$2.50"). This standardizes the price across different stock levels and time horizons. * **The "Wrong" Number:** It is famously described as *"The wrong number to put in the wrong formula to get the right price"* (Rebonato). It acts as a calibration factor that forces the simplified Black-Scholes model to agree with complex market realities.

0.1.4 The Volatility Surface

If the assumptions of the Black-Scholes model (Geometric Brownian Motion) were perfectly satisfied in the real world, the calibrated σ_{imp} would be **constant** across all strikes K and maturities T.

Empirical Reality: The Smile and Skew When we calibrate σ_{imp} for all liquid options in the market, we do *not* observe a flat constant. Instead, we observe a structure known as the **Volatility Surface** $\Sigma(K, T)$.

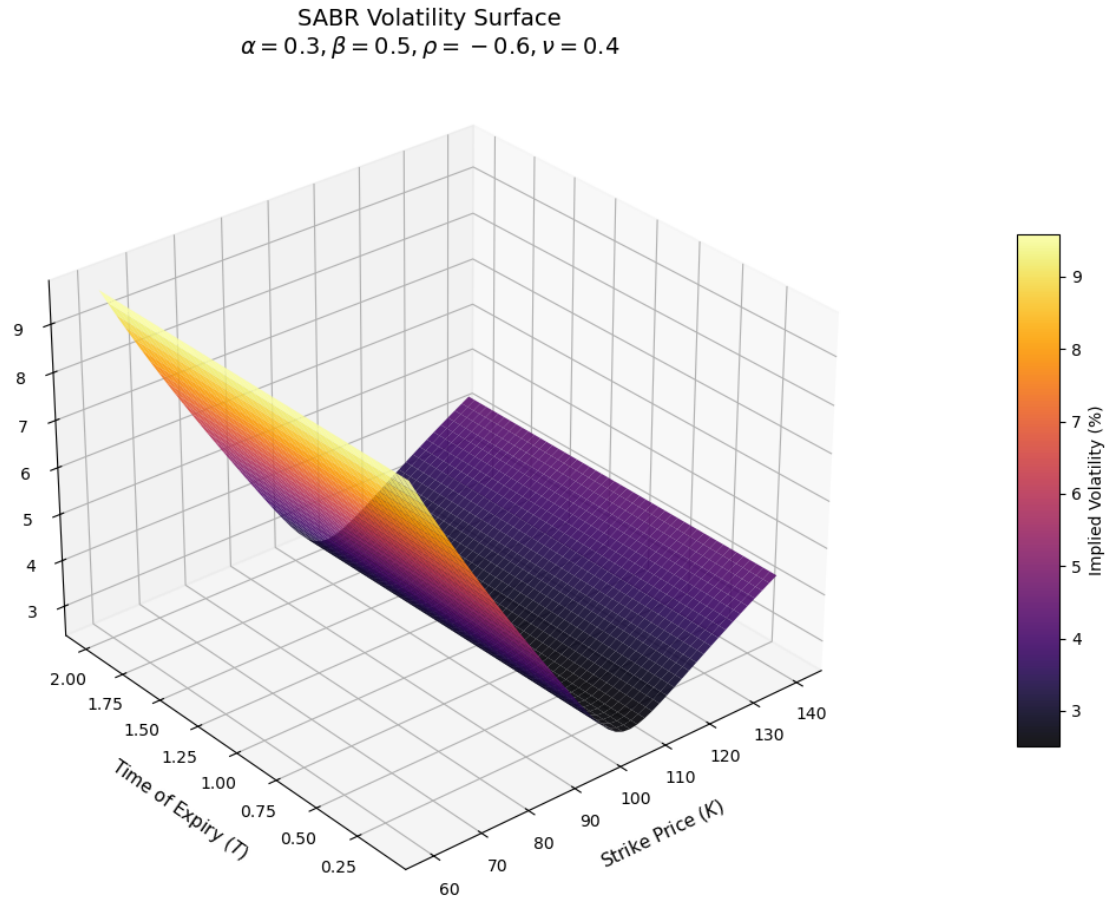
The surface exhibits two main stylized facts:

1. **The Smile/Skew (Dependence on K):** For a fixed maturity T, implied volatility varies with the strike.
   - **Equity Skew:** Typically, σ_{imp} decreases as Strike increases (OTM Puts trade at higher vols than OTM Calls due to crash fear).
   - **FX Smile:** Typically U-shaped (Both OTM Puts and Calls trade at premiums due to fat-tailed return distributions).
2. **The Term Structure (Dependence on T):** For a fixed strike K, implied volatility varies with time to maturity. Short-term options often react more violently to news (earnings, macro events) than long-term options.

[18]:
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

def sabr_vol(K, T, F, alpha, beta, rho, nu):
    """
    Hagan et al. (2002) SABR Implied Volatility Approximation.
    Vectorized for K and T.
    """
    # Avoid division by zero for ATM where K=F
    # We add a tiny epsilon to K where it equals F
    K = np.where(np.isclose(K, F), F * (1 + 1e-7), K)

    # 1. Variables X and Z
    # FK_beta = (F * K)**((1 - beta) / 2)
    log_FK = np.log(F / K)
    FK_beta = (F * K)**((1 - beta) / 2)

    z = (nu / alpha) * FK_beta * log_FK
```

```
    # x(z) function
    # Numerator inside the log: sqrt(1 - 2*rho*z + z^2) + z - rho
    # Denominator inside the log: 1 - rho
    x_z = np.log((np.sqrt(1 - 2*rho*z + z**2) + z - rho) / (1 - rho))

    # 2. The Expansion Terms
    # A = alpha / (FK_beta * (1 + ...))
    term1 = (1 - beta)**2 / 24 * (log_FK**2)
    term2 = (1 - beta)**4 / 1920 * (log_FK**4)
    denominator_A = FK_beta * (1 + term1 + term2)
    A = alpha / denominator_A

    # B = z / x(z)
    B = z / x_z

    # C = Expansion factor (1 + [...] * T)
    bracket1 = ((1 - beta)**2 / 24) * (alpha**2 / (F * K)**(1 - beta))
    bracket2 = (rho * beta * nu * alpha) / (4 * (F * K)**((1 - beta) / 2))
    bracket3 = (2 - 3 * rho**2) / 24 * nu**2
    C = 1 + (bracket1 + bracket2 + bracket3) * T

    return A * B * C

# --- Parameters ---
F0 = 100.0       # Forward Price
alpha = 0.3      # ATM Volatility Level (roughly)
beta = 0.5       # Elasticity (0.5 = CIR/Square Root process, 1 = Lognormal)
rho = -0.6       # Correlation (Controls SKEW). Negative = Downward slope
nu = 0.4         # Vol of Vol (Controls SMILE/Convexity)

# --- Grid Generation ---
strikes = np.linspace(60, 140, 50)  # Strike range
maturities = np.linspace(0.1, 2.0, 50) # Time range

K_grid, T_grid = np.meshgrid(strikes, maturities)

# --- Calculate Surface ---
# We assume Forward price F is constant (approx) or F = S*exp(rT).
# For simplicity here, F=F0.
Z_sabr = sabr_vol(K_grid, T_grid, F0, alpha, beta, rho, nu)

# --- Visualization ---
fig = plt.figure(figsize=(14, 8))
ax = fig.add_subplot(projection='3d')

surf = ax.plot_surface(K_grid, T_grid, Z_sabr * 100,
                       cmap=cm.inferno,
```

```
                        edgecolor='none',
                        alpha=0.9)

# Styling
ax.set_xlabel('Strike Price ($K$)', fontsize=11, labelpad=10)
ax.set_ylabel('Time of Expiry ($T$)', fontsize=11, labelpad=10)
ax.set_zlabel('Implied Volatility (%)', fontsize=11, labelpad=10)
ax.set_title(f'SABR Volatility Surface\n'
             f'$\\alpha={alpha}, \\beta={beta}, \\rho={rho}, \\nu={nu}$',␣
 ↪fontsize=14)

# Adjust view to see the Skew clearly
ax.view_init(elev=30, azim=230)

# Colorbar
cbar = fig.colorbar(surf, ax=ax, shrink=0.6, aspect=12, pad=0.1)
cbar.set_label('Implied Volatility (%)')

plt.tight_layout()
plt.show()
```



0.1.5 Black–Scholes Analytics and Integral Representations

Throughout this section we assume **zero interest rates**, $r = 0$.

Notation - S_0: Current Spot price. - F: **Forward price** at maturity T. - General case: $F = S_0 e^{rT}$. - In this section ($r = 0$): $F = S_0$. - K: Strike price. - σ: Annualized volatility parameter. - T: Time to maturity. - $\sigma_{\text{tot}} := \sigma\sqrt{T}$: **Total volatility** (standard deviation over the life of the option). - $N(\cdot)$ and $n(\cdot)$: Standard normal CDF and PDF, where:

$$n(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

0.1.6 The Pricing Formulas

By expressing the Black-Scholes equation using the Forward price F, we decouple the interest rate (drift) from the volatility (diffusion).

$$\begin{aligned} \text{Call}_{BS}(K, F) &= FN(d_1) - KN(d_2) \\ \text{Put}_{BS}(K, F) &= KN(-d_2) - FN(-d_1) \end{aligned}$$

where the standardized coordinates d_1 and d_2 are defined using **log-moneyness** and **total volatility**:

$$\begin{aligned} d_1 &= \frac{\ln(F/K) + \frac{1}{2}\sigma_{\text{tot}}^2}{\sigma_{\text{tot}}} \\ d_2 &= d_1 - \sigma_{\text{tot}} \end{aligned}$$

0.1.7 The Fundamental Identity (Symmetry)

A crucial algebraic relationship between the standard normal density evaluated at d_1 and d_2 is:

$$F\, n(d_1) = K\, n(d_2)$$

Proof: Recall that $d_1 = d_2 + \sigma_{\text{tot}}$. Expanding the ratio of the densities:

$$\frac{n(d_1)}{n(d_2)} = \frac{e^{-d_1^2/2}}{e^{-d_2^2/2}} = \exp\left(-\frac{1}{2}(d_1^2 - d_2^2)\right)$$

Using the difference of squares $d_1^2 - d_2^2 = (d_1 - d_2)(d_1 + d_2)$:

$$\begin{aligned} d_1 - d_2 &= \sigma_{\text{tot}} \\ d_1 + d_2 &= \frac{2\ln(F/K)}{\sigma_{\text{tot}}} \end{aligned}$$

Multiplying these gives $d_1^2 - d_2^2 = 2\ln(F/K)$. Substituting back into the exponent:

$$\frac{n(d_1)}{n(d_2)} = \exp\left(-\ln\left(\frac{F}{K}\right)\right) = \frac{K}{F} \quad \implies \quad Fn(d_1) = Kn(d_2).$$

This identity acts as a "cancellation trick" that simplifies derivatives significantly. It is the reason why the Gamma (Γ) and Vega ($\mathcal{V}$) formulas are compact: - **Vega Derivation:** When differentiating the price $FN(d_1) - KN(d_2)$, the terms involving $\frac{\partial d}{\partial \sigma}$ cancel out perfectly because of this identity, leaving only the $N(d_1)$ term. - **Asymptotic Analysis:** It allows us to switch between measures (asset numeraire vs. cash numeraire) effortlessly when estimating tail probabilities.

0.1.8 Put-Call Parity

Before moving to integral representations, we verify that our Black-Scholes formulas satisfy the fundamental **Put-Call Parity** relationship.

The Relationship (for $r = 0$)

$$\text{Call}_{BS}(K, F) - \text{Put}_{BS}(K, F) = F - K$$

Computational Note: > In a trading engine, never call the costly `norm.cdf` function four times to price a Call and a Put. > 1. Calculate the Call: $C = FN(d_1) - KN(d_2)$. > 2. Calculate the Put via Parity: $P = C - (F - K)$. > This reduces computational cost by ~50%.

Proof: Put-Call Parity We substitute the pricing formulas into the left-hand side:

$$\begin{aligned} C - P &= [FN(d_1) - KN(d_2)] - [KN(-d_2) - FN(-d_1)] \\ &= F[N(d_1) + N(-d_1)] - K[N(d_2) + N(-d_2)] \end{aligned}$$

Using the symmetry property of the Normal CDF, $N(x) + N(-x) = 1$ (the total probability mass is 1), we obtain:

$$\begin{aligned} C - P &= F(1) - K(1) \\ &= F - K \end{aligned}$$

0.1.9 Alternative Integral Representation via Mills' Ratio

We define the **Mills' ratio** $R(x)$ as the ratio of the survival function (or CDF of $-x$) to the density:

$$R(x) := \frac{N(-x)}{n(x)} = \frac{1 - N(x)}{n(x)}.$$

Using the integral identity for the normal cumulative distribution:

$$N(-x) = n(x) \int_0^\infty e^{-xt} e^{-t^2/2}\, dt,$$

we can substitute this into the pricing equations to obtain the **deterministic integral representations**:

Put Option Integral:

$$\text{Put}_{BS}(K, F) = Kn(d_2) \int_0^\infty e^{-d_2 t} \left(1 - e^{-\sigma_{\text{tot}} t}\right) e^{-t^2/2}\, dt$$

Call Option Integral:

$$\text{Call}_{BS}(K, F) = Fn(d_1) \int_0^\infty e^{d_1 t} \left(1 - e^{-\sigma_{\text{tot}} t}\right) e^{-t^2/2} \, dt$$

0.1.10 Compact Representation

We can further compact these formulas by introducing the auxiliary function $\phi(x, y)$:

$$\phi(x, y) := n(x) \left[R(x) - R(x + y)\right].$$

This allows us to express the option prices purely in terms of ϕ, the standardized arguments, and total volatility:

$$\begin{aligned} \text{Put}_{BS}(K, F) &= K\phi(d_2, \sigma_{\text{tot}}) \\ \text{Call}_{BS}(K, F) &= F\phi(-d_1, \sigma_{\text{tot}}) \end{aligned}$$

Applications: This representation is particularly useful for computational finance because it facilitates: 1. **Asymptotic Analysis:** Studying behavior as $T \to 0$ or $K \to \infty$. 2. **Bounds:** Deriving tight upper and lower bounds using inequalities for Mills' ratio. 3. **Approximations:** Creating fast numerical approximation schemes that avoid calling the expensive `CDF` function.

0.2 Some Static Properties of the Black-Scholes Prices

0.2.1 1. Positivity

Since $F > 0$ and the density $n(x) > 0$, the sign of the Call and Put option of the price is determined entirely by the term in the brackets. 1. **Strictly Decreasing:** The Mills' ratio $R(z) = \frac{1-N(z)}{n(z)}$ is a strictly decreasing function for all $z \in \mathbb{R}$.

2. **Ordering:** Since volatility is positive ($\sigma_{\text{tot}} > 0$), we have arguments $x < x + y$.
3. **Conclusion:** By the monotonicity of R:

$$R(x) > R(x + y) \implies \phi(x, y) > 0.$$

Thus, the Black-Scholes formula guarantees positive prices **mathematically**, consistent with the **financial** requirement.

0.2.2 2. Monotonicity in Strikes

Basic no-arbitrage logic dictates the direction of prices relative to the strike:

- **Calls:** The right to buy at a higher price is strictly less valuable.

$$K_1 < K_2 \quad \Rightarrow \quad \text{Call}_{BS}(K_1) \geq \text{Call}_{BS}(K_2)$$

- **Puts:** The right to sell at a higher price is strictly more valuable.

$$K_1 < K_2 \quad \Rightarrow \quad \text{Put}_{BS}(K_1) \leq \text{Put}_{BS}(K_2)$$

0.2.3 3. Monotonicity in Volatility

Differentiating the pricing formula with respect to σ yields the **Vega** (the gradient):

$$\frac{\partial \text{Call}_{BS}}{\partial \sigma} = \mathcal{V} = Fn(d_1)\sqrt{T} > 0.$$

Computational Consequence: Since the derivative is strictly positive, the price is strictly increasing in σ. This implies: 1. **Uniqueness:** For any valid market price, there is exactly one Implied Volatility σ_{imp}. 2. **Convergence:** The function is "well-behaved" (convex in relevant regions), guaranteeing that gradient-based root finders like **Newton-Raphson** will converge rapidly.

0.2.4 4. Tail Bounds via Mills' Ratio

For deep Out-of-the-Money (OTM) options, evaluating the cumulative distribution $N(\cdot)$ is numerically expensive and prone to floating-point underflow.

We can use bounds on Mills' ratio $R(x)$ (such as the **Birnbaum bound**) to approximate prices using elementary functions.

Birnbaum-Type Upper Bound (OTM Call) For large d_1 (deep OTM), the call price is bounded by:

$$\text{Call}_{BS} \lesssim Fn(d_1)\left[\frac{\sqrt{d_1^2+4}-d_1}{2} - \frac{\sqrt{(d_1+\sigma_{\text{tot}})^2+4}-(d_1+\sigma_{\text{tot}})}{2}\right].$$

Applications: * **Tail Risk Control:** Estimating probabilities of extreme events (e.g., VaR). * **Screening:** Quickly rejecting candidate calibrations that are too far off. * **Numerical Stability:** Replacing `CDF` calls with algebraic bounds in asymptotic regimes.

0.2.5 5. ATM Expansion: A Trader's Approximation

A standard "mental math" shortcut is derived by expanding the formula around the At-The-Money (ATM) point where $F = K$.

The Setup At $F = K$, we have $\ln(F/K) = 0$, so:

$$d_1 = \frac{\sigma_{\text{tot}}}{2}, \qquad d_2 = -\frac{\sigma_{\text{tot}}}{2}.$$

Substituting this into the Call formula:

$$\text{Call}_{BS}(F,F) = F\left[N\left(\frac{\sigma_{\text{tot}}}{2}\right) - N\left(-\frac{\sigma_{\text{tot}}}{2}\right)\right] = F\left[2N\left(\frac{\sigma_{\text{tot}}}{2}\right) - 1\right].$$

Taylor Expansion Using the first-order Taylor expansion $2N(x) - 1 \approx \frac{2}{\sqrt{2\pi}}x$ for small x:

$$\text{Call}_{BS}(F,F) \approx F \cdot \frac{2}{\sqrt{2\pi}} \cdot \frac{\sigma_{\text{tot}}}{2} = \frac{F\sigma_{\text{tot}}}{\sqrt{2\pi}}.$$

Numerically, $1/\sqrt{2\pi} \approx 0.3989$, leading to the famous approximation:

$$\text{Call}_{BS}(F, F) \approx 0.4\, F\, \sigma_{\text{tot}}.$$

0.2.6 6. Trader's Rule of Thumb (The Straddle)

An **ATM Straddle** involves buying both a Call and a Put at strike $K = F$. Since $\text{Call}_{ATM} = \text{Put}_{ATM}$ (when $r = 0$), the price of the straddle is simply $2 \times \text{Call}_{ATM}$.

$$\text{Straddle}_{ATM} \approx 2 \times 0.4\, F\, \sigma_{\text{tot}} = 0.8\, F\, \sigma\sqrt{T}.$$

Usage: * **Sanity Check:** If a trader sees a 1-year ATM straddle trading at \$16 on a \$100 stock, the implied volatility is approximately $16/(0.8 \times 100) = 20\%$. * **Daily Volatility:** Since $\sqrt{T_{day}} \approx 1/16$ (assuming 256 trading days), the daily breakeven move is roughly $\frac{0.8}{16}\sigma F = 0.05\sigma F$.

0.3 Calibration to Vanilla Options and The Volatility Surface

In a functioning market, we do not observe just one option price. We observe a **chain** of option prices across different strikes (K) and maturities (T).

0.3.1 The Calibration Problem

Calibration in the context of Black-Scholes is the process of solving the **inverse problem** for every quoted option in the market.

Given a set of market prices $C_{\text{mkt}}(K_i, T_j)$, we seek a matrix of volatilities $\Sigma_{i,j}$ such that:

$$C_{\text{BS}}(K_i, T_j, \Sigma_{i,j}) = C_{\text{mkt}}(K_i, T_j)$$

0.3.2 The Implied Volatility Smile

If the Black-Scholes assumptions (Geometric Brownian Motion) were perfectly true, the calibrated volatility $\Sigma_{i,j}$ would be a flat constant for all K and T.

Empirical Reality: When we calibrate to real market data, we observe a distinct "Smile" or "Skew":

1. **The Smile/Skew:** For a fixed maturity T, implied volatility varies with strike K.
   - *Equity Markets:* Typically downward sloping (Skew). OTM Puts have higher implied volatility than OTM Calls (Crash protection is expensive).
   - *FX Markets:* Typically U-shaped (Smile). Extreme moves in either direction are more likely than the normal distribution predicts (Fat tails).
2. **Term Structure:** Implied volatility varies with time to maturity T. Short-term expirations often exhibit sharper smiles than long-term expirations due to the Mean Reversion of volatility.

0.3.3 Computational Algorithm: Point-wise Calibration

To construct this surface computationally, we apply the Newton-Raphson solver to each data point in the grid independently.

Input: * Matrix of Strikes: $\mathbf{K}$ * Matrix of Maturities: $\mathbf{T}$ * Matrix of Market Prices: $\mathbf{C}_{\text{mkt}}$

Algorithm: Iterate over every option (i, j) in the chain: 1. **Check Arbitrage:** Verify $C_{i,j} > \max(F - K_i, 0)$. If this fails, the price is invalid (arbitrage exists). 2. **Root Finding:** Solve $C_{\text{BS}}(\sigma) - C_{i,j} = 0$ using Newton-Raphson. 3. **Store:** Place result σ_{imp} into the surface matrix $\Sigma_{i,j}$.

The Interpolation Challenge: Market data is discrete (specific strikes and monthly expirations), but risk management requires a continuous surface. After point-wise calibration, we typically fit a parametric model (like **SABR** or **SVI**) to smooth the dots into a continuous surface $\Sigma(K, T)$.

0.3.4 The Newton-Raphson Method

Since $C_{BS}(\sigma)$ is non-linear, we cannot isolate σ algebraically. We must solve for it numerically. Strictt monotonicity in the σ guarantees that the inversion is well posed; however, the inverse function of the Black-Scholes pricer is not known in closed form.

Numerically, we frame the problem as finding the root of the objective function $f(\sigma)$:

$$f(\sigma) = C_{BS}(\sigma) - C_{\text{mkt}} = 0$$

The **Newton-Raphson method** finds this root by iteratively approximating the function $f(\sigma)$ by its tangent line.

Given an initial guess σ_0, we update our estimate using the ratio of the pricing error to the slope (Vega):

$$\sigma_{n+1} = \sigma_n - \frac{f(\sigma_n)}{f'(\sigma_n)}$$

Substituting the financial terms: 1. **Numerator:** $f(\sigma_n) = \text{ModelPrice}(\sigma_n) - \text{MarketPrice}$. 2. **Denominator:** $f'(\sigma_n) = \frac{\partial C}{\partial \sigma} = \text{Vega}(\sigma_n)$.

The iteration becomes:

$$\sigma_{n+1} = \sigma_n - \frac{C_{BS}(\sigma_n) - C_{\text{mkt}}}{\mathcal{V}(\sigma_n)}$$

0.3.5 Computational Challenges

While Newton-Raphson is quadratically convergent (extremely fast), it has specific pitfalls in option pricing:

1. **The "Small Vega" Problem:** For deep Out-of-the-Money (OTM) or deep In-the-Money (ITM) options, the Vega approaches zero ($\mathcal{V} \to 0$).
   - *Result:* The denominator becomes tiny, causing the update step $\Delta\sigma$ to explode towards infinity.
   - *Fix:* Hybrid algorithms (like Brent's Method) switch to **Bisection** if the Newton step jumps out of bounds.

2. **The Initial Guess:** Bad initial guesses can slow down convergence. A popular heuristic (from the Brenner-Subrahmanyam approximation) for At-The-Money options is:

$$\sigma_{start} \approx \sqrt{\frac{2\pi}{T}} \frac{C_{\text{mkt}}}{F}$$

0.3.6 Python Implementation: Implied Volatility Solver

Below is a robust implementation. Note the `try-except` block to handle cases where Vega is zero (division by zero).

[17]:
```python
import numpy as np
from scipy.stats import norm

def implied_volatility(price_mkt, F, K, T, r=0, tol=1e-6, max_iter=100):
    """
    Calculate Implied Volatility using Newton-Raphson.

    Parameters:
    price_mkt : float - Observed market price of the option
    F         : float - Forward Price
    K         : float - Strike Price
    T         : float - Time to maturity
    r         : float - Risk-free rate (default 0)

    Returns:
    sigma_imp : float - Implied Volatility
    """

    # 1. Initial Guess (Brenner-Subrahmanyam approximation)
    # This places us close to the solution for ATM options
    sigma = np.sqrt(2 * np.pi / T) * (price_mkt / F)

    for i in range(max_iter):
        # Calculate d1, d2
        sigma_sqrt_T = sigma * np.sqrt(T)
        d1 = (np.log(F / K) + 0.5 * sigma**2 * T) / sigma_sqrt_T
        d2 = d1 - sigma_sqrt_T

        # Calculate Price and Vega
        # Note: We use r=0 formulas here based on lecture assumptions,
        # but generic implementation would include discount factors.
        price_model = F * norm.cdf(d1) - K * norm.cdf(d2)
        vega = F * norm.pdf(d1) * np.sqrt(T)

        # Calculate Error
        diff = price_mkt - price_model

        # Check Convergence
```

```
        if abs(diff) < tol:
            return sigma

        # Newton-Raphson Step
        # Protection against Zero Vega (Deep OTM/ITM)
        if abs(vega) < 1e-8:
            print("Warning: Vega is zero, Newton method fails.")
            return np.nan

        sigma = sigma + diff / vega

    print("Warning: Max iterations reached without convergence.")
    return sigma

# Example Usage
F = 100
K = 100
T = 1.0
market_price = 7.96  # Assume this is the observed price

iv = implied_volatility(market_price, F, K, T)
print(f"Market Price: {market_price}")
print(f"Implied Vol:  {iv:.4f} (Expected ~20%)")
```

```
Market Price: 7.96
Implied Vol:  0.1999 (Expected ~20%)
```

0.4 Lecture 4: Black-Scholes PDE and the Greeks

0.4.1 Objectives

After this lecture, you should be able to:

1. Derive the Black–Scholes Partial Differential Equation (PDE) using a dynamic replication (delta-hedging) argument.
2. Explain why the drift μ of the stock disappears from the pricing equation and why pricing depends only on r, σ, and the payoff.
3. Formulate the Black–Scholes PDE together with appropriate terminal and boundary conditions for European options.
4. Understand the connection between the PDE approach and the probabilistic (risk-neutral expectation) approach via the Feynman–Kac theorem.
5. Solve the Black–Scholes PDE for European call and put options and recover the closed-form pricing formula.
6. Define and interpret the main Greeks:
   - Delta (Δ),
   - Gamma (Γ),
   - Vega (ν),
   - Theta (Θ),
   - Rho (ρ).

7. Compute Greeks analytically in the Black–Scholes model and explain their economic meaning in terms of hedging and risk management.

1 The Black-Scholes PDE

While the **Risk-Neutral Valuation** formula derived in the previous section gives us the *fair price* of an option to avoid arbitrage, it does not tell a trader how to manage the risk of selling that option.

In this section, we shift perspective from **Probability Theory** (Expectations) to **Financial Engineering** (Replication). We will demonstrate that the option price $V(S,t)$ must satisfy a partial differential equation (PDE) essentially identical to the Heat Equation in physics.

1.1 The Replication Argument

Suppose a financial institution sells a European Call option. They are now exposed to the risk that the stock price S_t will rise. To hedge this risk, they construct a portfolio Π_t consisting of: 1. **Short** one option position $-V(S,t)$. 2. **Long** Δ_t shares of the underlying stock S_t.

The value of this portfolio at time t is:

$$\Pi_t = -V(S_t,t) + \Delta_t S_t$$

We want to choose Δ_t such that this portfolio becomes **risk-free** over a tiny time interval dt.

1.1.1 Step 1: Dynamics of the Portfolio

Assuming the standard geometric Brownian motion for the stock:

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

We apply **Itô's Lemma** to determine the change in the option value dV_t:

$$dV = \left(\frac{\partial V}{\partial t} + \mu S\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\right) dt + \sigma S \frac{\partial V}{\partial S} dW_t$$

Therefore, the change in the portfolio value $d\Pi$ is:

$$d\Pi = -dV + \Delta dS$$

$$d\Pi = -\left[\left(\frac{\partial V}{\partial t} + \mu S\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\right) dt + \sigma S \frac{\partial V}{\partial S} dW_t\right] + \Delta(\mu S dt + \sigma S dW_t)$$

1.1.2 Step 2: Eliminating Risk (Delta Hedging)

We group the terms involving the source of risk, dW_t:

$$\text{Stochastic Term} = \left(\Delta - \frac{\partial V}{\partial S}\right)\sigma S dW_t$$

To make the portfolio risk-free (deterministic), we must eliminate this term. We choose the hedge ratio Δ as:

$$\Delta_t = \frac{\partial V}{\partial S}$$

This is the so-called **Delta** of the option. It represents the sensitivity of the option price to changes in the underlying asset.

1.1.3 Step 3: No-Arbitrage Condition

If the portfolio Π is risk-free, it must earn exactly the risk-free rate r to prevent arbitrage. Any other return would allow a trader to make risk-less profit.

$$d\Pi = r\Pi dt$$

Substituting our expressions back into this equation:

$$\left(-\frac{\partial V}{\partial t} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\right) dt = r\left(-V + S\frac{\partial V}{\partial S}\right) dt$$

Rearranging terms leads to the **Black-Scholes Partial Differential Equation**:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0$$

This equation holds for **any** derivative on S (Call, Put, Binary, etc.). The specific type of option is determined only by the **Boundary Condition** at time T: * **Call:** $V(S,T) = \max(S-K, 0)$ * **Put:** $V(S,T) = \max(K-S, 0)$

1.2 The Feynman-Kac Theorem: Bridging Theory and Computation

We have now derived the option price in two completely different ways: 1. **Martingale Approach (Lecture 3):** $V_0 = e^{-rT}\mathbb{E}^{\mathbb{Q}}[h(S_T)]$ 2. **PDE Approach (Lecture 4):** $\frac{\partial V}{\partial t} + \mathcal{L}V - rV = 0$

The **Feynman-Kac Theorem** proves that these two solutions are mathematically identical. This is crucial for Computational Finance because it gives us a choice of numerical methods:

Problem Characteristics	Preferred Method	Computational Technique
High Dimensions (e.g., Basket Options)	Probabilistic	**Monte Carlo Simulation**
Path Dependence (e.g., Asian Options)	Probabilistic	**Monte Carlo Simulation**
Early Exercise (e.g., American Options)	PDE	**Finite Difference Methods**
Complex Barriers	PDE	**Finite Difference Methods**