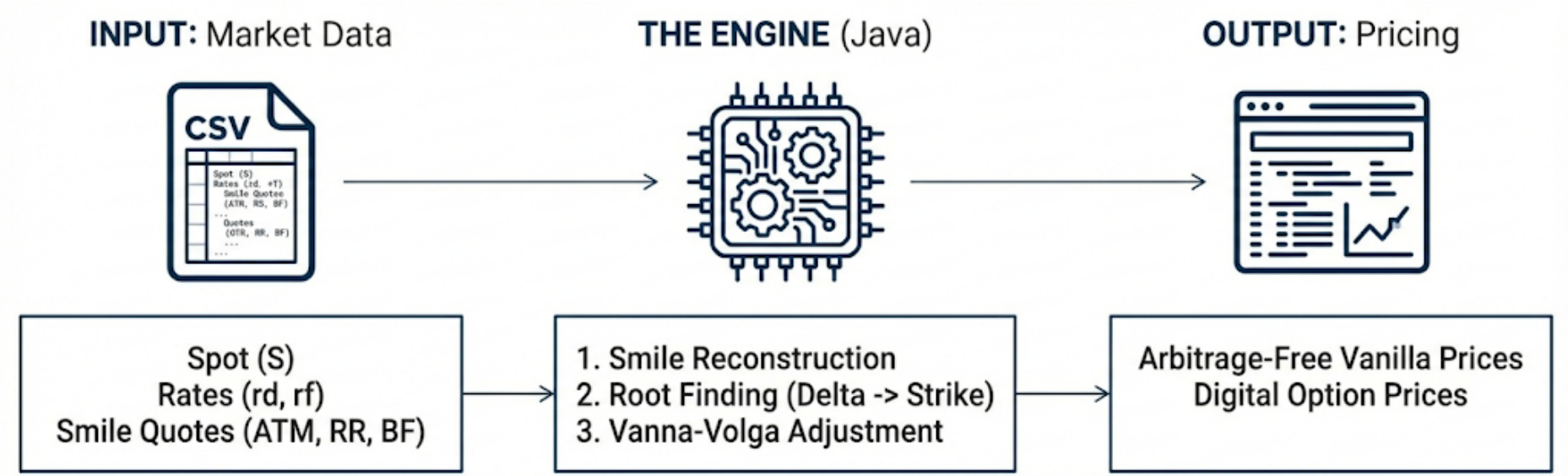
Vanna-Volga method for exotic FX option pricing: A JAVA Pricing Library

Emanuele Saccoliti

11/02/2026

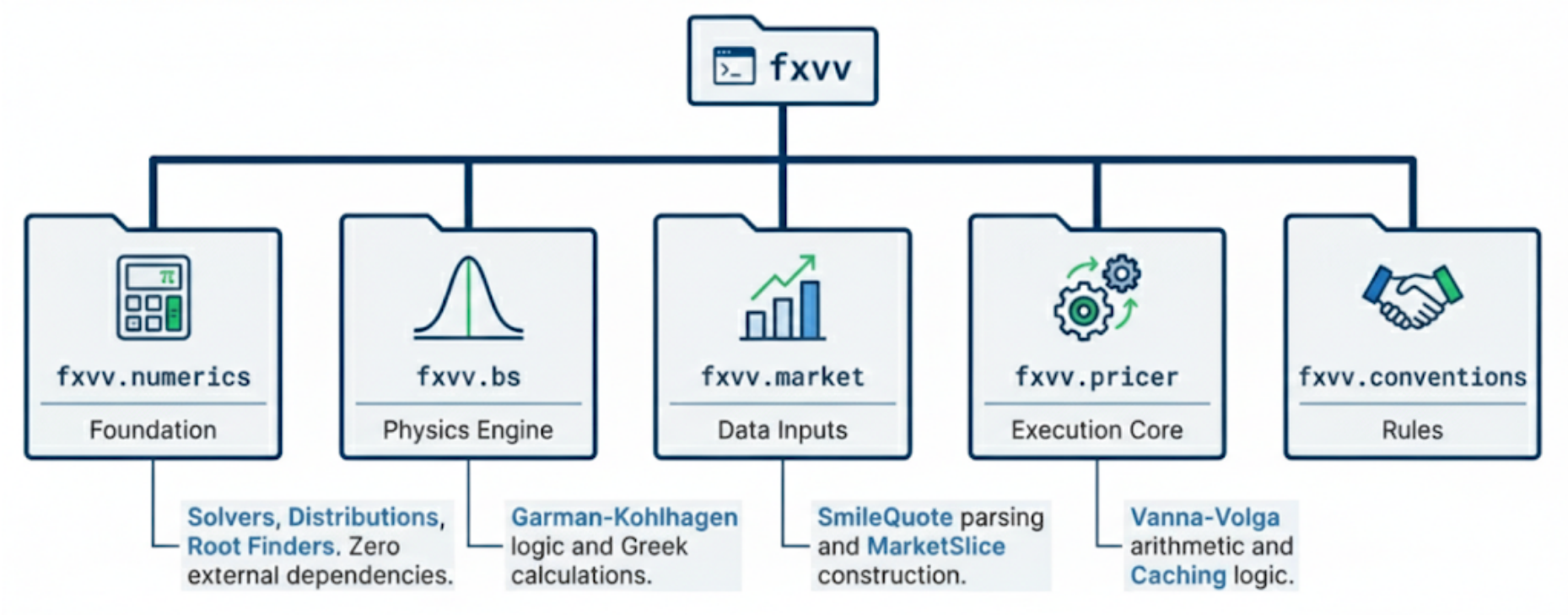
Pipeline

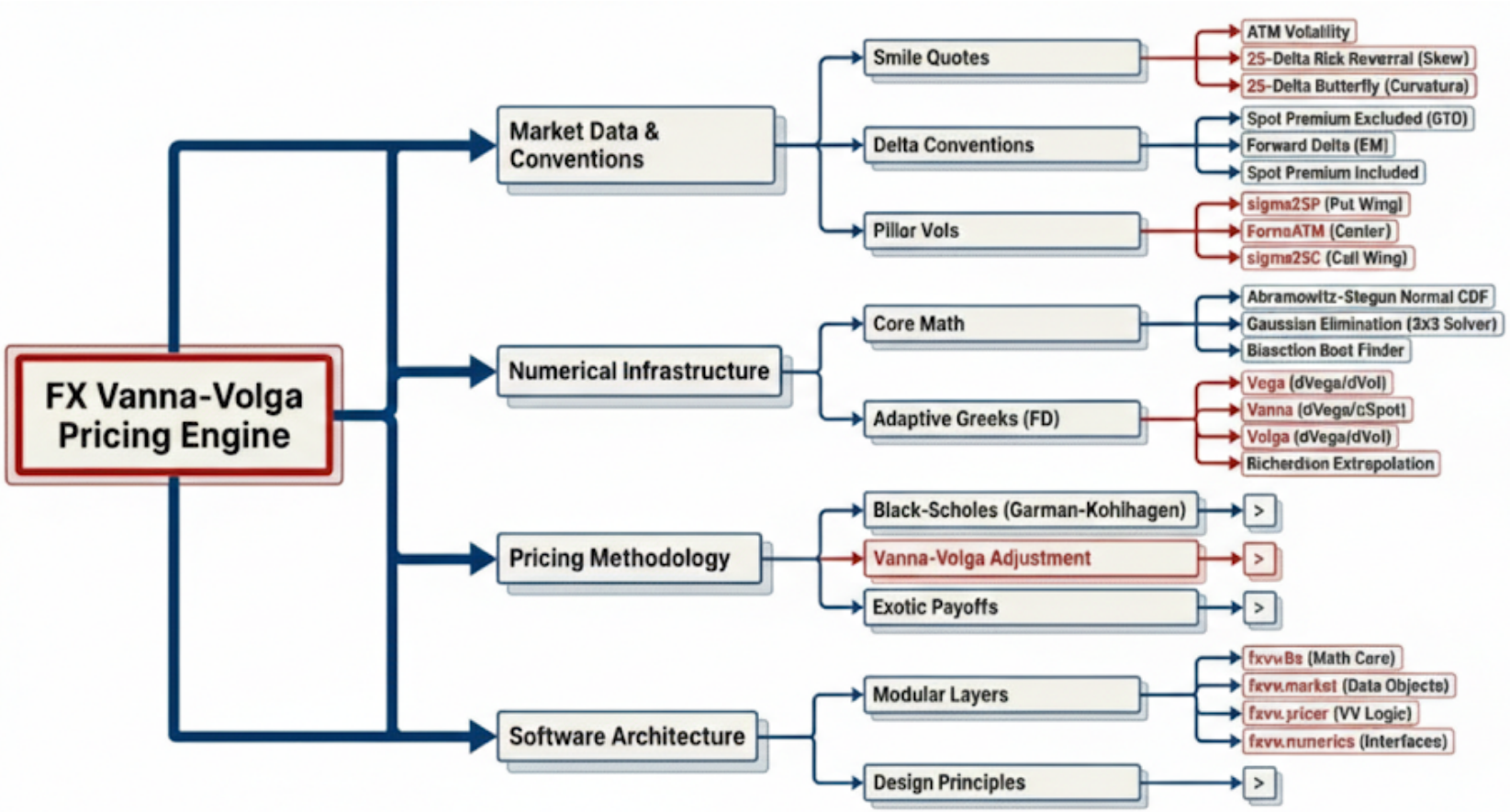

INPUT: Market Data
THE ENGINE (Java)
OUTPUT: Pricing
CSV
Spot (S)
Rates (rd, rf)
Smile Quotes (ATM, RR, BF)
1. Smile Reconstruction
2. Root Finding (Delta -> Strike)
3. Vanna-Volga Adjustment
Arbitrage-Free Vanilla Prices
Digital Option Prices

Key Features

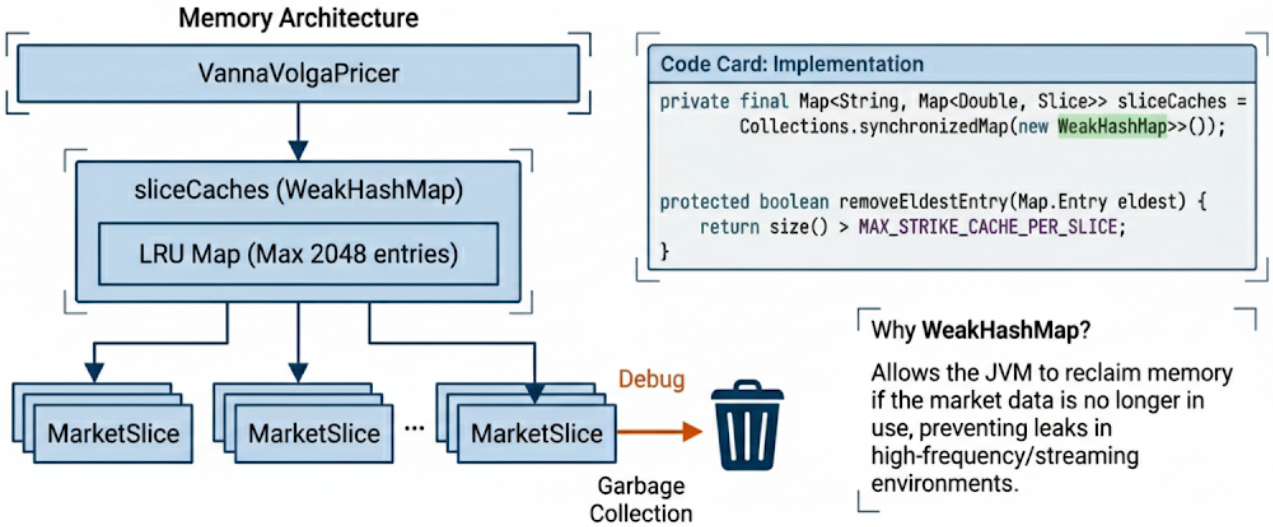
- **Advanced Numerical Greeks**: The system computes second-order sensitivities Vanna and Volga using adaptive finite differences and Richardson extrapolation to ensure numerical stability even near market bound.
- **Performance Optimization (Caching)**: To avoid redundant calculations, the pricer caches Vanna-Volga weights and the pillar 3x3 Greek matrix per market slice, significantly speeding up the pricing of multiple strikes.
- **Configurable FX Conventions**: The engine is built to handle different market conventions, specifically allowing the user to switch between Spot Premium Excluded and Forward Premium Excluded delta logic.
- **Modular Architecture**: The library uses a decoupled design where pricing methodology, numerical infrastructure, and market conventions are independent modules that can be replaced or extended.
- **Self-Contained Frameworks**: The library is written from scratch with zero external dependencies, implementing built-in mathematical utilities and numerical approximations (e.g. Abramowitz-Stegun for the Normal CDF)

Architecture




FX Vanna-Volga Pricing Engine
Market Data & Conventions
Smile Quotes
ATM Volatility
25-Delta Risk Reversal (Skew)
25-Delta Butterfly (Curvatura)
Delta Conventions
Spot Premium Excluded (GTO)
Forward Delta (EM)
Spot Premium Included
Pillar Vols
sigma2SP (Put Wing)
FormaATM (Center)
sigma2SC (Call Wing)
Numerical Infrastructure
Core Math
Abramowitz-Stegun Normal CDF
Gaussian Elimination (3x3 Solver)
Bisection Root Finder
Adaptive Greeks (FD)
Vega (dVega/dVol)
Vanna (dVega/dSpot)
Volga (dVega/dVol)
Richardson Extrapolation
Pricing Methodology
Black-Scholes (Garman-Kohlhagen)
Vanna-Volga Adjustment
Exotic Payoffs
Software Architecture
Modular Layers
fxvv.bs (Math Core)
fxvv.market (Data Objects)
fxvv.pricer (VV Logic)
fxvv.numerics (Interfaces)
Design Principles

Caching



- Each *MarketSlice* is associated with a cache of precomputed results, avoiding repeated recalculation of Greeks and linear systems.
- **Per-Slice Caching**: It uses a WeakHashMap to store a SliceCache for each MarketSlice. This allows the engine to compute the pillar 3x3 Greek matrix once per market state and reuse it for all subsequent strikes.
- **Strike-Level Caching**: Within each slice, it employs a LinkedHashMap configured as an LRU (Least Recently Used) cache to store weights by strike, preventing redundant linear system solves during risk runs or surface generations.

Handling Market Complexity

- Rather than hardcoding logic for specific currency pairs, the architecture uses the Strategy Pattern via the *DeltaConvention enum*.
- This allows the same *MarketSliceBuilder* and *GKBlackScholes* instance to handle diverse market practices (e.g. Spot Premium Excluded or Forward Delta) by simply passing a different parameter at runtime.

Code Card: **GKBlackScholes.deltaInstance**

```
switch (conv) {
  case SPOT_PREM_EXCLUDED:
    return isCall ? DFf * N.cdf(d1) : -DFf * N.cdf(-d1); // Example logic for

  case FWD_PREM_EXCLUDED:
    return isCall ? N.cdf(d1) : -N.cdf(-d1); // Example logic for put

  case SPOT_PREM_INCLUDED:
    double premiumAdjustedScale = DFd * (K / S);
    return ...; // Implementation for premium included
}
```

A '25 Delta' is not a universal constant. It depends on the market protocol.

1. **Spot Premium Excluded**
   Standard G10 currency convention.
2. **Fwd Premium Excluded**
   Common in Emerging Markets.
3. **Spot Premium Included**
   Mathematically complex, non-monotonic.

Advanced Numerical Greeks

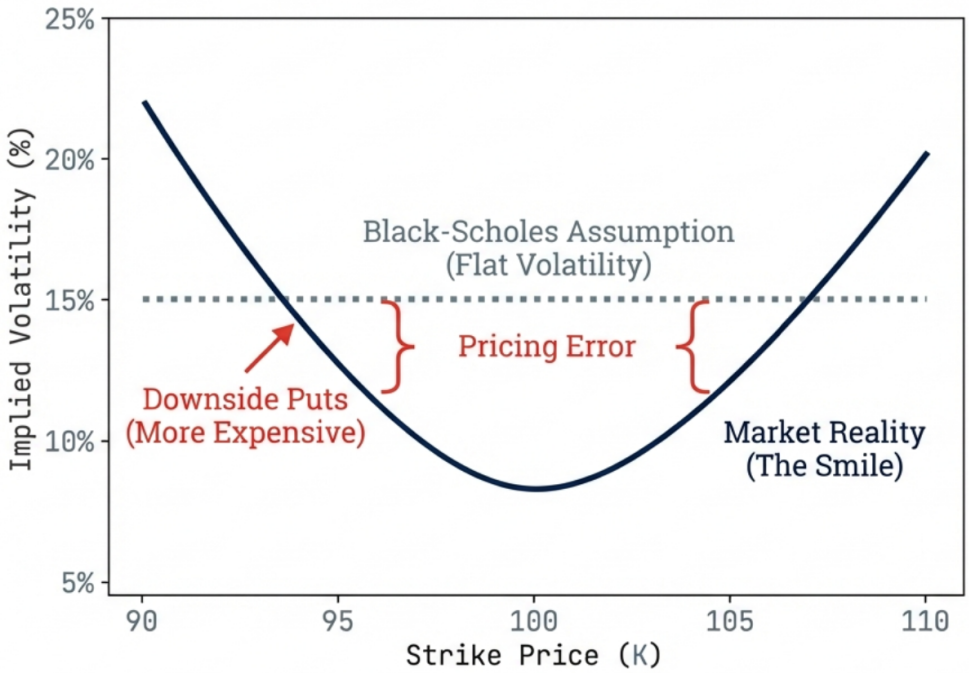
The engine computes sensitivities via Richardson extrapolation and refinement loops. Instead of using a single fixed increment, the library follows this process:

- **Adaptive Refinement**: The library starts with an initial step size (h_0) and progressively halve it.
- **Stability Checking**: At each step, the library compares the new result with the previous one using the isStable method, which checks if the difference falls within defined absolute and relative tolerances.
- **Richardson extrapolation:** combines multiple step sizes to obtain higher-order accurate derivative estimates.

To prevent errors in extreme market conditions, the library implements several safeguards:

- **Boundary Handling**: It includes logic to detect if a variable is too close to its lower bound. If a symmetric central difference would go below the allowed minimum (e.g., negative volatility), the engine automatically switches to one-sided (forward) differences to maintain validity.
- **Safety Constraints**: The system enforces strict minimums for volatility and spot prices to avoid division by zero errors or NaNs during the differentiation process.

Black-Scholes Isn't Enough



THE ASSUMPTION

Black-Scholes assumes volatility is constant across all strikes.

THE REALITY

Out-of-the-money options trade at different volatilities due to:

- Crash fears (skew)
- Tail risks (curvature)

THE CONSEQUENCE

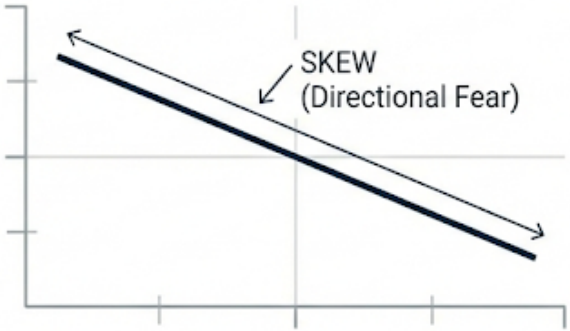
Using flat volatility drastically misprices exotic derivatives, such as digital and barrier options.

Decoding FX Market

ATM (At-The-Money)

The baseline level of volatility.
The anchor of the curve.

Risk Reversal (25 Delta RR)

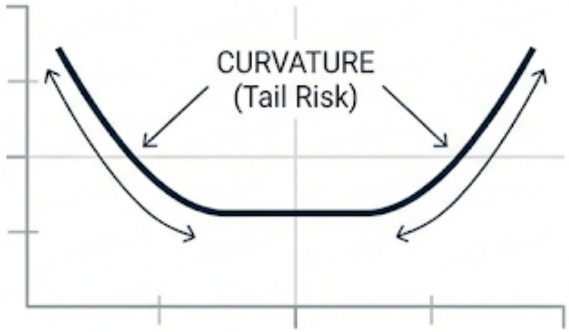


Measures SKEW (Directional Fear).

Formula: Vol(25C) − Vol(25P)

Negative RR implies the market fears the downside (Puts are expensive).

Butterfly (25 Delta BF)



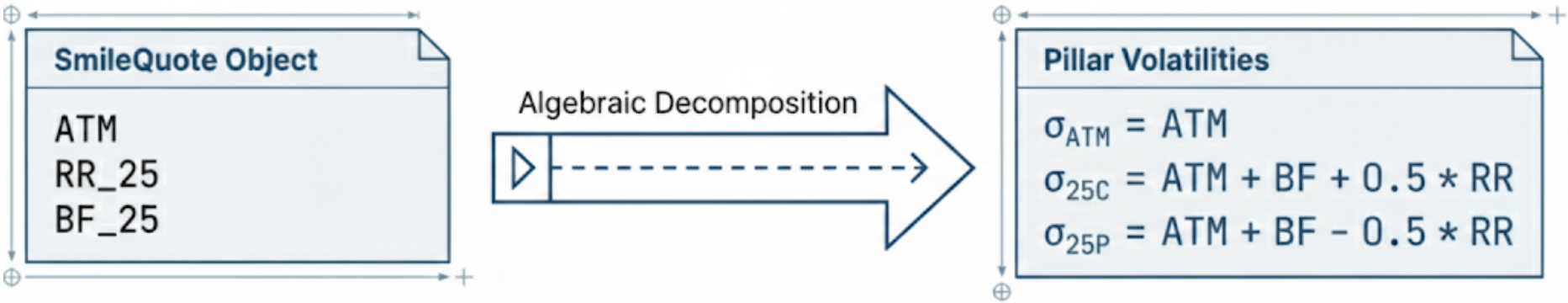
Measures CURVATURE (Tail Risk).

Formula: Average of wings − ATM

High BF implies "fat tails" in the distribution.

- FX Traders don't quote the full surface. They only quote few liquid instruments.
- We must reverse-engineer the full smile from this sparse data.

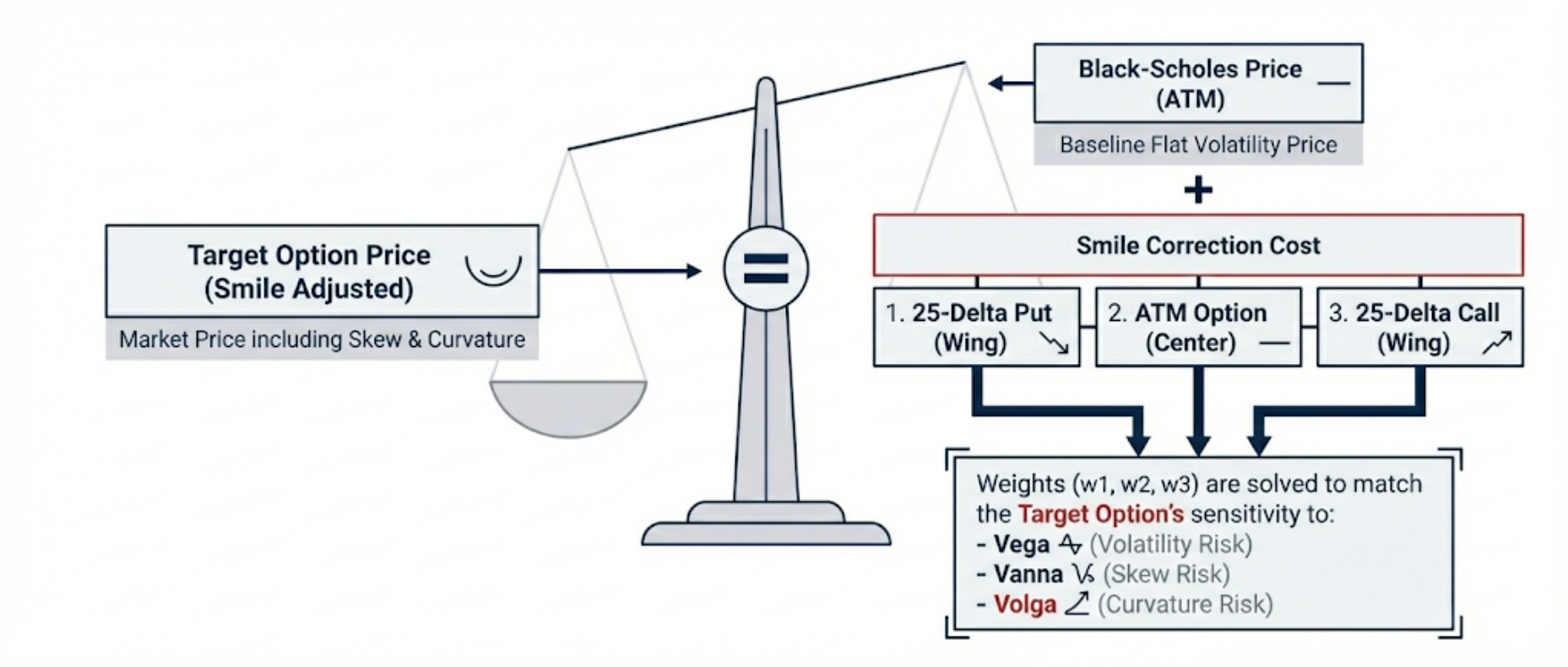
Parsing the Smile: From Quotes to Pillars



Code Card: Pillar Calculations

```
public double sigma25C() {
   return sigmaATM + bf25 + 0.5 * rr25;
}
public double sigma25P() {
   return sigmaATM + bf25 - 0.5 * rr25;
}
```

The Solution: Vanna-Volga Replication



The central idea of Vanna-Volga method is to **replicate the smile cost** (namely the difference between the price computed with/without including the smile effect) of a target option, using a portfolio of these three liquid instruments that carry mainly Volga and Vanna risks.

The replication consists of finding the weights (w_i) to assign to these three options such that the synthetic portfolio has the same Greeks (Vega, Vanna and Volga) of the target option.

These Greeks can be replicated by the weighted sum of three instruments:

$$\boxed{X_i = w_{ATM}\, ATM_i + w_{RR}\, RR_i + w_{BF}\, BF_i} \qquad i = \text{Vega, Vanna, Volga} \qquad (1)$$

where the weights are obtained by solving the linear system:

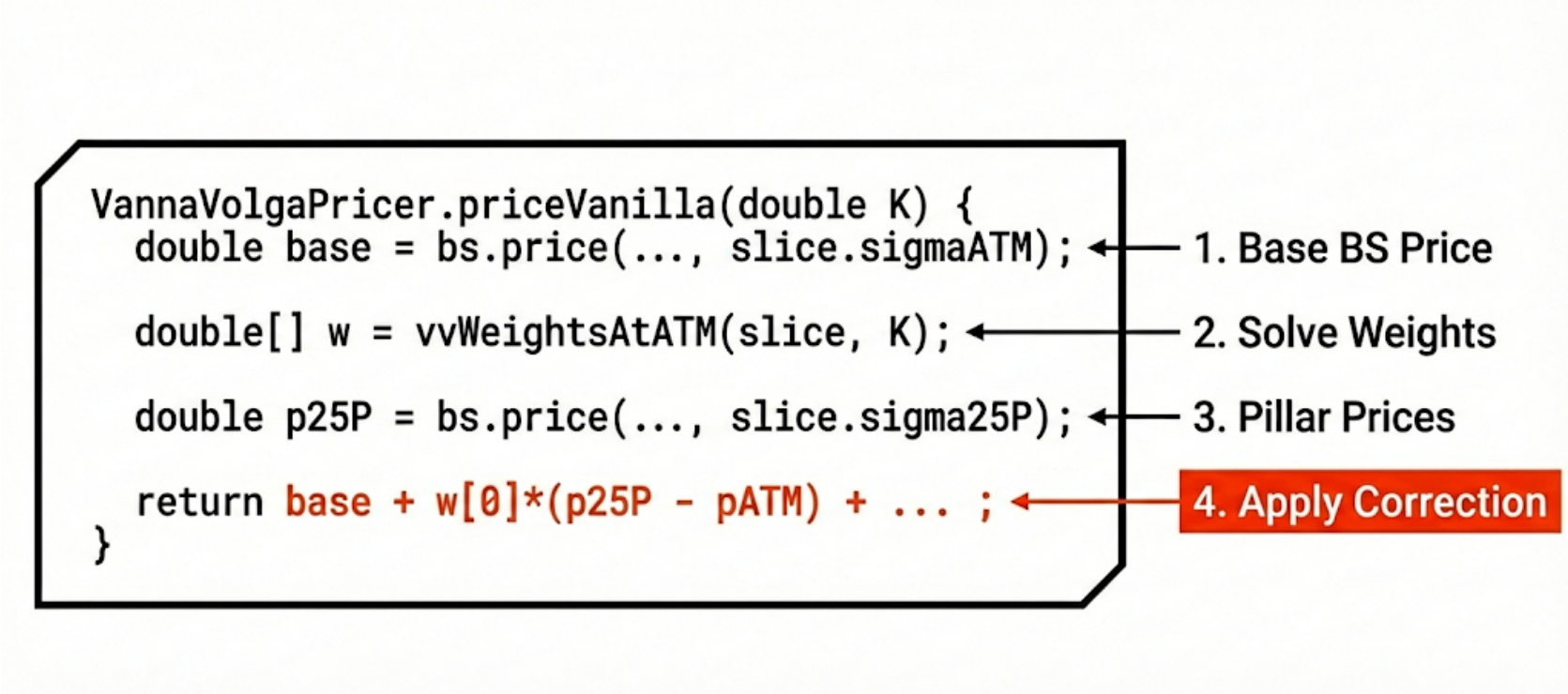
$$\vec{x} = A\vec{w}$$

with

$$A = \begin{pmatrix} ATM_{\text{vega}} & RR_{\text{vega}} & BF_{\text{vega}} \\ ATM_{\text{vanna}} & RR_{\text{vanna}} & BF_{\text{vanna}} \\ ATM_{\text{volga}} & RR_{\text{volga}} & BF_{\text{volga}} \end{pmatrix}, \qquad \vec{w} = \begin{pmatrix} w_{ATM} \\ w_{RR} \\ w_{BF} \end{pmatrix}, \qquad \vec{x} = \begin{pmatrix} X_{\text{vega}} \\ X_{\text{vanna}} \\ X_{\text{volga}} \end{pmatrix}.$$

Once the replication weights have been determined, the Vanna–Volga price is computed as:

1. **Black-Scholes price**: Calculated using only the ATM volatility.
2. **Smile correction**: An additional adjustment is added to account for the extra cost (or saving) arising from the fact that the three market pillars are priced differently from Black-Scholes.



How the smile correction is computed?

For each pillar i (25Δ Put, ATM, 25Δ Call), we compute the **smile cost** as the difference between the Black-Scholes price of the target option at strike K computed with the pillar volatility σ_i and the price computed using the ATM volatility σ_{ATM}:

$$\Delta P_i = C^{BS}(K, \sigma_i) - C^{BS}(K, \sigma_{ATM})$$

Specifically we have

- 25Δ Put: $C^{BS}(K, \sigma_{25P}) - C^{BS}(K, \sigma_{ATM})$
- ATM: $C^{BS}(K, \sigma_{ATM}) - C^{BS}(K, \sigma_{ATM}) = 0$
- 25Δ Call: $C^{BS}(K, \sigma_{25C}) - C^{BS}(K, \sigma_{ATM})$

These quantities measure the ***extra price*** charged by the market due to the volatility smile.

Then, the total smile correction is obtained as the weighted sum of the pillar smile costs:

$$\text{Adjustment} = w_{25P}\left(\phi_{25P} - \phi_{ATM}\right) + w_{25C}\left(\phi_{25C} - \phi_{ATM}\right) \tag{2}$$

where ϕ_i denotes the price of the target option computed using the volatility of the corresponding pillar:

$$\phi_{ATM} = C^{BS}(K, \sigma_{ATM})$$

$$\phi_{25P} = C^{BS}(K, \sigma_{25P})$$

$$\phi_{25C} = C^{BS}(K, \sigma_{25C})$$

Finally, the smile adjustment is added to the Black-Scholes price computed with ATM volatility:

$$\boxed{Price_{VV}(K) = Price_{BS}(K, \sigma_{ATM}) + \text{Adjustment}} \tag{3}$$

From Deltas to Strikes

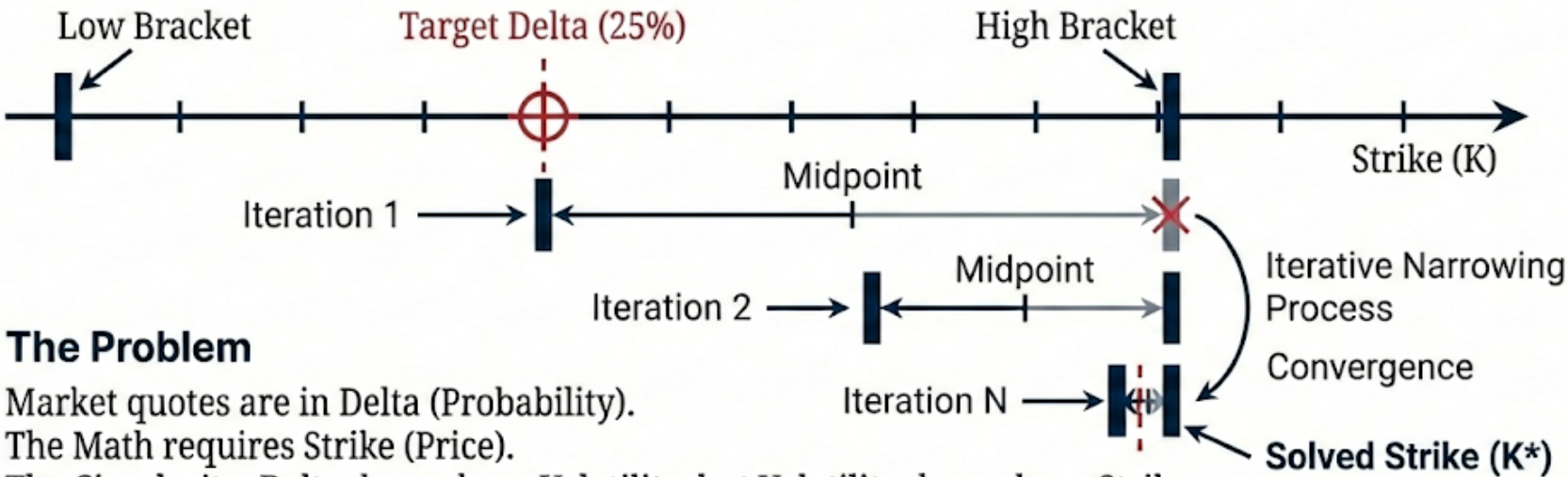
- Market FX options are quoted in terms of delta, while pricing models require the strike as input.
- This creates an inverse problem: the option delta depends on volatility, while the volatility smile itself is parameterized as a function of the strike. As a consequence, strike and volatility must be solved simultaneously.
- To resolve this circular dependency, we can use a bisection algorithm. Starting from an initial low and high strike bracket, the algorithm evaluates the model-implied delta at the midpoint strike and compares it to the market quoted delta. Depending on the sign of the error, the bracket is iteratively updated, progressively narrowing the admissible strike interval.

This monotonic root-finding procedure guarantees convergence to a unique strike K^* such that

$$\Delta_{\text{model}}(K^*) = \Delta_{\text{market}}.$$

The resulting K^* is therefore fully consistent with market quoting conventions.

Solving the Inverse Problem



The Problem

Market quotes are in Delta (Probability).
The Math requires Strike (Price).
The Circularity: Delta depends on Volatility, but Volatility depends on Strike.

The Solution ⟶ A robust Bisection Algorithm that iteratively narrows the strike range until the calculated Delta **matches the market quote.**

From Vanilla Calls to Digital Options

The Call price under the domestic risk–neutral measure is given by

$$C(K) = e^{-r_d T}\, \mathbb{E}^{\mathbb{Q}^d}\left[(S_T - K)^+\right],$$

where the payoff is $\varphi(S_T) = (S_T - K)^+$.

If we take the derivative of the payoff with respect to K

$$\frac{\partial \varphi(S_T)}{\partial K} = \frac{\partial}{\partial K}(S_T - K)^+ = -1_{\{S_T > K\}}.$$

Therefore,

$$\frac{\partial C(K)}{\partial K} = -e^{-r_d T}\, \mathbb{P}^{\mathbb{Q}^d}(S_T > K) \tag{4}$$

Digital Options via Strike Differentiation

A digital call instead pays a fixed amount if the option finishes in-the-money:

$$1_{\{S_T > K\}}$$

Its price is therefore the discounted risk–neutral probability that the option finishes in the money:

$$\text{DigitalCall}(K) = e^{-r_d T}\, \mathbb{P}^{\mathbb{Q}^d}(S_T > K)$$

Remark

A digital option measures the probability of crossing the strike.

Therefore, from (4) it is straightforward to say:

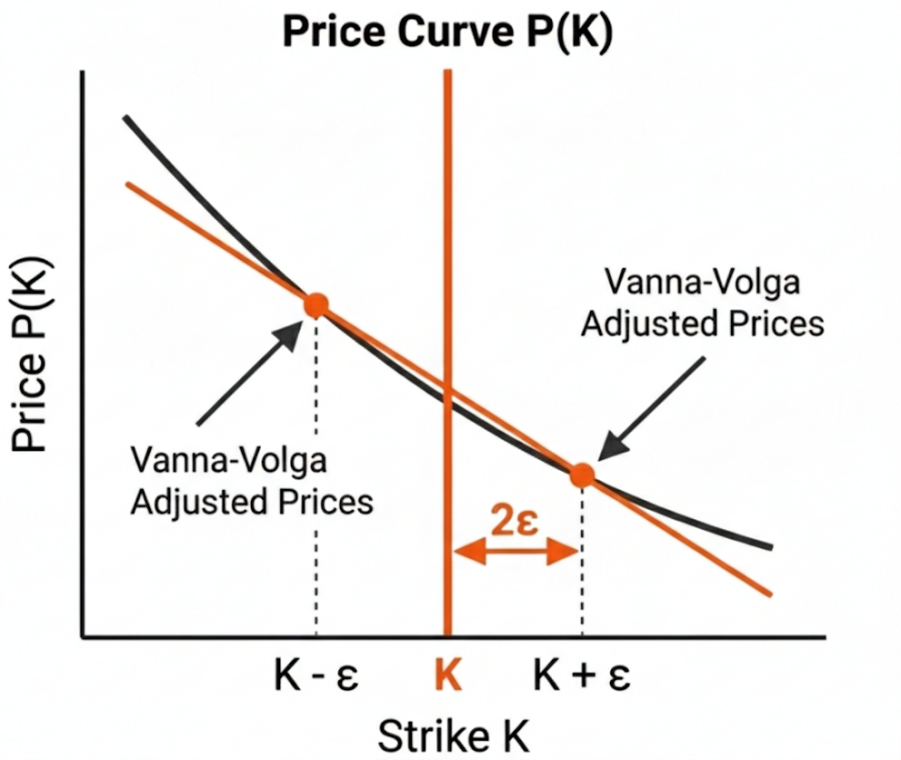
$$\text{DigitalCall}(K) = -\frac{\partial C(K)}{\partial K} \tag{5}$$

Numerical implementation

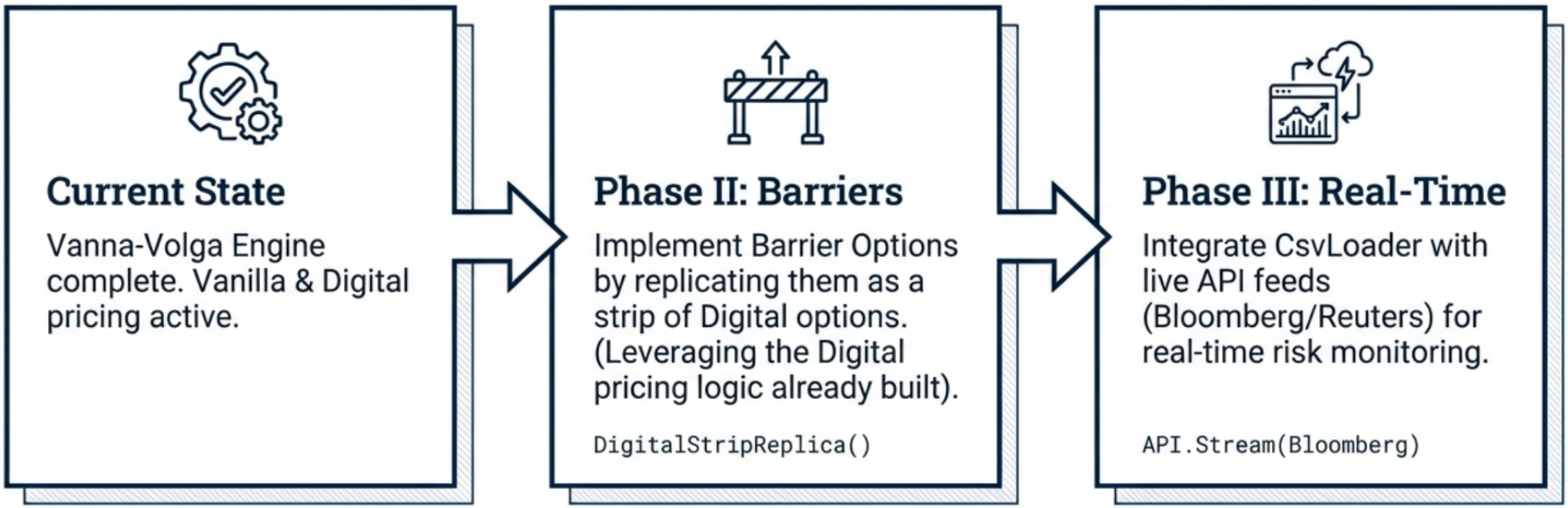
We can approximate (5) using central finite differences.

The idea is to evaluate the Vanna-Volga adjusted Call price at two nearby strikes, $K - \varepsilon$ and $K + \varepsilon$, and estimate the slope of the price curve around K:

$$\boxed{\text{DigitalCall}_{VV}(K) \approx \frac{C_{VV}(K - \varepsilon) - C_{VV}(K + \varepsilon)}{2\varepsilon}} \tag{6}$$



Future Extensions & Scalability



References

- Bossens, F., Rayée, G., Skantzos, N. S., Deelstra, G. (2010): "Vanna-Volga methods applied to FX derivatives: from theory to market practice".
- Castagna, A., Mercurio, F. (2007): The Vanna-Volga method for implied volatilities: tractability and robustness".
- Downey, A. B., Mayfield, C. (2020). Think Java: How to think like a computer scientist. O'Reilly Media.
- Garman, B. M., Kohlhagen, S. W. (1983): "Foreign currency option values".