AADC Tracer for Model Validation

Case study: QuantLib's down-and-out barrier call

MatLogica

October 2026

Why validators use the Tracer

Model validation has to establish what pricing code actually does. Today that means reading source by hand across dozens of files, with no guarantee the reading matches what runs in production. The AADC Tracer records the calculation as it runs and turns it into evidence a validator can check line by line.

Validation task	Reading the source	With the Tracer
Establish the formula implemented	Inferred from code and documentation	Read from the operations that ran, each tied to file and line
List inputs and hard-coded constants	Manual trace across files	Enumerated automatically, with values and source lines
Verify model dependencies	Taken from the model inventory and documentation	Confirmed from what the calculation actually used
Find market-dependent branches	Search for conditionals; easy to miss inside utilities	Every decision listed, with the inputs that switch it
Test scenarios and compute Greeks	Reprice and bump each scenario	Replay with exact AAD Greeks
See where the work concentrates	Separate profiling	Operation counts per function and line

This case study shows what the Tracer produces when applied to QuantLib 1.41's barrier option, with one production engine and four benchmark engines. From one recording per engine, the Tracer:

- gave the Merton–Reiner–Rubinstein formula, the Douglas finite-difference scheme and three binomial trees as the code computed them, each equation linked to the QuantLib source line that ran it;
- listed every input, including a day count that comes from the market objects, not the trade;
- listed every market-dependent decision in each engine, down to the individual lines inside the error function behind Φ;
- supported replay tests, sensitivity ladders, stress tests and barrier parity, with Greeks by AAD, each checked against fresh QuantLib pricing.

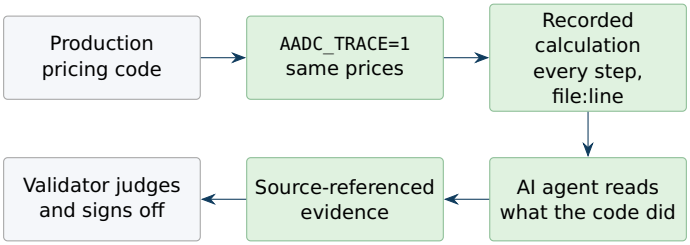
The example is illustrative: it shows the evidence a validation team gets from the Tracer, not a validation outcome for the model.

At a glance

- **One switch, no code change:** `AADC_TRACE=1` on the production pricing program; prices and speed unchanged with it off.
- **Five engines recorded:** analytic, finite differences and three binomial trees, one recording each.
- **The production price in full:** 473 recorded operations, 261 leaves and 30 market-dependent decision sites.
- **Source-linked:** every equation and recorded function linked to its line in the public QuantLib 1.41 source.
- **Exact and fast:** replay agrees with fresh QuantLib pricing to $1.44 \times 10^{-4}\%$ or better at the eleven test points, with AAD Greeks from one reverse pass.
- **At scale:** a QuantLib swaption price gives 74,428 recorded steps, 140 hard-coded numbers and 85 decision branches from a 17 ms recording.

How to read this paper. Items on a green are Tracer output: recording statistics, the functions recorded, decision sites and source references. Function names marked ↗ link to the line in the public QuantLib 1.41 source on GitHub (commit `367ce80ae4f2`). The recordings were made on the AADC tracing build of QuantLib 1.41 (AADC 2.22.2), so a linked upstream line may differ slightly from the line recorded.

The Tracer in brief



One switch. Setting `AADC_TRACE=1` on a normal pricing program returns the same prices plus a complete record of the calculation. With the switch off, the program and its speed are unchanged.

What the record contains.

- Every arithmetic step, comparison and input, each attributed to its source file, line and function.
- The full dependency of each number on market quotes and constants, and the inputs it does not depend on.
- Every hard-coded number that reaches the price (tolerances, cut-offs, day counts, solver brackets), with its value and the line that set it.
- Every market-dependent decision, with the inputs at which it switches.
- Operation counts per function and per source line, including loop trip counts.

An agent that works from evidence. Asked to read source code, a language model infers what the code probably does and can be confidently wrong. Given the record, it reads what the code did. Claims are verifiable (each cites a file and line), results are reproducible (recordings are kept), and the validators make the decisions.

At scale. One QuantLib swaption price produced 74,428 recorded steps across about 40 source files, 140 hard-coded numbers reaching the price, three day-count conventions in one valuation and 85 market-dependent branches, all from a single 17 ms recording.

The case study

The example model is QuantLib 1.41's `BarrierOption`, priced as a single-asset down-and-out call under Black–Scholes–Merton. The closed-form analytic engine is in production; four numerical engines serve as benchmarks. One recording per engine gives the statistics below.

Engine	Role	Recorded operations	Leaves	Decision sites
Analytic (Merton–Reiner–Rubinstein)	Production	473	261	30
Finite differences (Douglas)	Benchmark	479,544	127,935	71
Binomial CRR, 200 steps	Benchmark	441,136	145,688	48
Binomial Jarrow–Rudd, 200 steps	Benchmark	476,681	130,241	42
Binomial Leisen–Reimer, 200 steps	Benchmark	435,918	211,867	44

Test configuration. $S_0 = 100$, $K = 100$, $B = 80$, $r = 4\%$, $q = 2\%$, $\sigma = 25\%$, $T = 1$ year (evaluation date 24 Sep 2026, Actual/365 Fixed), continuous monitoring, no rebate. Analytic NPV = 10.277694; AAD delta 0.607119, vega 30.0775, rho 45.5119.

Scope. Down-and-out only, continuous monitoring, flat rates, dividends and volatility. Other barrier types, discrete monitoring and term structures run different code paths and need their own recordings.

1. The model as the code computed it

The pricing formulas below were read off the recorded calculation, not reconstructed from documentation. Under each equation are the functions the Tracer recorded for it, linked to the source; the tables give the operations each function contributed.

Analytic engine (Merton–Reiner–Rubinstein)

Drift parameter μ and λ:

$$\mu = \frac{r-q}{\sigma^2} - \frac{1}{2}, \qquad \lambda = 1 + \mu \tag{1}$$

§ mu()↗ § muSigma()↗

Total standard deviation over $[0, T]$:

$$\hat{\sigma} = \sigma\sqrt{T} \tag{2}$$

§ stdDeviation()↗

Building blocks $A(\phi)$ and $C(\phi, \eta)$:

$$A(\phi) = \phi\, S_0 e^{-qT}\, \Phi(\phi\, x_1) - \phi\, K e^{-rT}\, \Phi(\phi\, x_1 - \phi\, \hat{\sigma}) \tag{3}$$

$$C(\phi, \eta) = \phi\, S_0 e^{-qT} \left(\frac{B}{S_0}\right)^{2\lambda} \Phi(\eta\, y_1) - \phi\, K e^{-rT} \left(\frac{B}{S_0}\right)^{2\lambda - 2} \Phi(\eta\, y_1 - \eta\, \hat{\sigma}) \tag{4}$$

$$x_1 = \frac{\ln(S_0/K)}{\hat{\sigma}} + \lambda\hat{\sigma}, \qquad y_1 = \frac{\ln\!\left(B^2/(S_0 K)\right)}{\hat{\sigma}} + \lambda\hat{\sigma} \tag{5}$$

§ A()↗ § C()↗

Down-and-out call for $B < K$, zero if $S_0 \le B$ at inception:

$$V_{DO} = A(1) - C(1, 1) \tag{6}$$

§ calculate()↗ § triggered()↗

Equation	Description	Recorded function	Operations
(1)	drift μ, λ	mu()↗ muSigma()↗	12, 4
(2)	$\hat{\sigma} = \sigma\sqrt{T}$	stdDeviation()↗	6
(3)	building block $A(\phi)$	A()↗	13
(4)	building block $C(\phi, \eta)$	C()↗	23
(6)	$V_{DO} = A - C$	calculate()↗	6
—	rebate term (zero here)	F()↗	1
—	Φ via the error function	operator()↗ in errorfunction.cpp	135
—	inception check $S_0 \le B$	triggered()↗	1

Finite-difference engine (Douglas scheme)

The engine solves the BSM PDE on a 1D mesh with an absorbing boundary at B:

$$\frac{\partial V}{\partial t} + \tfrac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r-q) S \frac{\partial V}{\partial S} - rV = 0, \qquad V(B, t) = 0 \tag{7}$$

Uniform grid in $x = \ln S$ whose lower edge is the barrier:

$$x_i = x_{\min} + ih, \quad h = \frac{x_{\max} - x_{\min}}{N-1}, \quad x_{\min} = \ln B, \quad x_{\max} = \ln S_0 + \sigma\sqrt{T}\,\Phi^{-1}(1-\varepsilon) \tag{8}$$

§ Uniform1dMesher↗ § FdmBlackScholesMesher↗

Drift-diffusion operator, with $D^{(1)}$, $D^{(2)}$ the central-difference stencils:

$$L = \tfrac{1}{2}\sigma^2 D^{(2)} + sD^{(1)} - rI, \qquad s = (r-q) - \tfrac{1}{2}\sigma^2 \tag{9}$$

§ setTime()↗ § axpyb()↗

Douglas time step ($\theta = 0.5$, Crank–Nicolson in one dimension):

$$(I - \theta\Delta t L)\, V^n = V^{n+1} + (1-\theta)\,\Delta t\, L V^{n+1} \tag{10}$$

§ step()↗

Thomas algorithm for the tridiagonal system:

$$\beta_0 = d_0, \quad y_0 = r_0/\beta_0 \tag{11}$$

$$\beta_i = d_i - l_i u_{i-1}/\beta_{i-1}, \quad y_i = (r_i - l_i y_{i-1})/\beta_i \tag{12}$$

$$x_N = y_N, \quad x_i = y_i - u_i x_{i+1}/\beta_i \tag{13}$$

§ solve_splitting()↗ § solveFor()↗

Price extracted by cubic spline:

$$V(x) = a + b(x - x_i) + c(x - x_i)^2 + d(x - x_i)^3 \tag{14}$$

§ value()↗ § derivative()↗

Equation	Description	Recorded function	Operations
(8)	mesh x_i	Uniform1dMesher↗ FdmBlackScholesMesher↗	202, 282
(9)	operator L, drift s	setTime()↗ axpyb()↗	606, 70,700
—	$D^{(1)}$, $D^{(2)}$ stencils	FirstDerivativeOp↗ SecondDerivativeOp↗	994, 794
(10)	Douglas time step	step()↗	1,010
(11)–(13)	Thomas algorithm	solve_splitting()↗ solveFor()↗	150,591, 1,586
—	$y = LV$ product	apply()↗	101,000
(14)	cubic interpolation	value()↗ derivative()↗	21, 14

The default 100×100 grid is visible directly in the counts: `solve_splitting()` records about 15 operations per node per time step.

Binomial engines (CRR, Jarrow–Rudd, Leisen–Reimer)

CRR up/down factors and risk-neutral probability:

$$u = e^{\sigma\sqrt{\Delta t}}, \qquad d = 1/u, \qquad p = \frac{e^{(r-q)\Delta t} - d}{u - d} \tag{15}$$

§ CoxRossRubinstein↗

Node spots, terminal payoff and backward induction:

$$S_{i,j} = S_0 u^{2j-i}, \qquad V_{N,j} = \max(S_{N,j} - K, 0) \tag{16–17}$$

$$V_{i,j} = e^{-r\Delta t}\,[pV_{i+1,j+1} + (1-p)V_{i+1,j}] \tag{18}$$

§ underlying()↗ § operator()↗ § stepback()↗

Barrier knock-out at each step, with proximity threshold:

$$V_{i,j} \leftarrow 0 \ \text{ if } S_{i,j} \le B, \qquad |S_{i,j} - B| < \varepsilon_{\text{rel}} B, \quad \varepsilon_{\text{rel}} = 42\,\epsilon_{\text{mach}} \approx 10^{-14} \tag{19–20}$$

§ checkBarrier() ↗ § adjustBarrier() ↗ § close_enough() ↗

Jarrow–Rudd and Leisen–Reimer calibrations:

$$u_{JR},\, d_{JR} = e^{(r-q-\frac{1}{2}\sigma^2)\Delta t \pm \sigma\sqrt{\Delta t}}, \qquad p_{JR} = \tfrac{1}{2} \tag{21}$$

$$p_{LR} = h(d_2), \qquad p'_{LR} = h(d_1), \qquad u_{LR} = e^{(r-q)\Delta t}\,\frac{p'_{LR}}{p_{LR}} \tag{22}$$

§ JarrowRudd ↗ § LeisenReimer ↗

Equation	Description	Recorded function	Operations
(15)	CRR u, d, p	CoxRossRubinstein ↗	8
(21)	JR u, d, $p = \frac{1}{2}$ (JR recording)	JarrowRudd ↗	2
(22)	LR Peizer–Pratt (LR recording)	LeisenReimer ↗	21
(16)	node spots $S_{i,j}$	underlying() ↗	137,400
(17)	terminal payoff	operator() ↗ in payoffs.cpp	878
(18)	backward induction	stepback() ↗	180,624
(19)	barrier knock-out	checkBarrier() ↗ adjustBarrier() ↗	45,808, 34,278
(20)	proximity threshold	close_enough() ↗	20,399

The three trees share backward induction, knock-out and the proximity test; the recording isolates the only part that differs, the calibration of u, d and p.

2. Every input, found automatically

The Tracer listed every input that reaches the price and how it enters the calculation, so the input classification required by model governance starts from evidence rather than a manual trace.

Input	How it enters (from the recording)	Production source
S_0, spot	User input	Equity feed, to be specified
σ, implied volatility	Single flat value via `BlackConstantVol`	Volatility surface model
r, risk-free rate	Single flat value via `FlatForward`	Yield curve model
q, dividend yield	Single flat value via `FlatForward`	Dividend model
K, B	Trade terms	Contract
Day count	Actual/365 Fixed, from the day counters on the curves and volatility surface	Set by the caller's market objects

The day count is the kind of detail a document review tends to miss: it comes from the market objects, not the trade terms. The recording surfaces it because it feeds every time-to-maturity calculation.

Materiality. With r and q marked as inputs, a 100bp move in each was repriced by replay. Both exceed the 0.10 bid/ask spread of a 100-notional option:

Parameter	NPV at -100bp	NPV at $+100$bp	Total swing	Material
r (base 4%)	-0.450	$+0.461$	0.910	Yes
q (base 2%)	$+0.569$	-0.547	1.116	Yes

This gives a validator a quantified basis for asking the model owner to document the production source and valuation controls for r and q.

3. Every market-dependent decision

The Tracer records every point where a market input decides which code runs, and reports the inputs at which each decision changes side. This is where code review is weakest: many of these decisions sit inside utilities a reviewer would treat as trusted black boxes.

Decision sites per engine. Each site is a source line where a recorded value is turned into a plain number to evaluate a condition. The table gives the total conversions, the distinct sites and the source files where they concentrate.

Engine	Conversions	Sites	Top source files (conversions)
Analytic	129	30	interestrate.cpp↗ (38), termstructure.cpp↗ (20), voltermstructure.cpp↗ (18), simplequote.hpp↗ (16), errorfunction.cpp↗ (16), yieldtermstructure.cpp↗ (6)
FD	17,885	71	triplebandlinearop.cpp↗ (10,100), interestrate.cpp↗ (1,606), cubicinterpolation.hpp↗ (1,564), termstructure.cpp↗ (1,204), comparison.hpp↗ (800), simpsonintegral.hpp↗ (589)
Binomial CRR	104,925	48	discretizedbarrieroption.cpp↗ (78,260), predefined_ops.h (13,564), comparison.hpp↗ (11,084), timegrid.cpp↗ (1,713), lattice.hpp↗ (215), interestrate.cpp↗ (24)

Decisions that switch within the tested ranges. Recording the analytic engine with branch-switch registration and replaying it across every test scenario identified the sites that change side within the tested input ranges:

analyticbarrierengine.cpp:210↗ errorfunction.cpp:207↗ errorfunction.cpp:209↗ errorfunction.cpp:210↗ errorfunction.cpp:221↗ errorfunction.cpp:225↗ errorfunction.cpp:227↗ errorfunction.cpp:234↗ interestrate.cpp:79↗ normaldistribution.cpp:36↗

Seven of the ten are in the error function: they select the approximation region behind Φ, and line 225 sets the sign of the result. Static review by function name does not reveal them; the recording does. With these sites and their switching inputs in hand, the validator knows exactly where a recording applies and where a scenario should be re-recorded.

What a source review would likely miss

Three details from this example that sit outside the files a reviewer normally reads, each surfaced by the recording:

- **A day count that comes from the market objects, not the trade.** Time to maturity follows the day counters on the curves and volatility surface the caller passes in, Actual/365 Fixed in this example. Set the curves and the volatility surface to Actual/360 and the price moves from 10.2777 to 10.3393. The recording names the convention each term actually used.
- **A sign decision inside Φ.** errorfunction.cpp:225↗ chooses the sign of erf in the region $0.84375 \le |x| < 1.25$; every Φ in the price is evaluated through this function.
- **A switch to an asymptotic expansion.** normaldistribution.cpp:36↗ replaces the error-function result with an asymptotic series once the cumulative normal falls to 10^{-8} or below, which deep out-of-the-money terms can reach.

4. Fast, exact testing

Once recorded, the analytic engine was repriced by replay with exact AAD Greeks from a single reverse pass, and each replayed price was checked against an independent fresh QuantLib evaluation.

Replay against fresh pricing. At eleven points across S_0, r, q and σ, replay and fresh pricing agree to $1.44 \times 10^{-4}\%$ at most:

S_0	r	q	σ	Replay	Fresh	Rel. difference
100	0.04	0.02	0.25	10.277694	10.277694	0
85	0.04	0.02	0.25	2.339095	2.339095	1.53e-06%
120	0.04	0.02	0.25	24.623229	24.623229	2.42e-07%
100	0.02	0.02	0.25	9.389708	9.389708	0
100	0.06	0.02	0.25	11.209629	11.209629	0
100	0.04	0.00	0.25	11.436079	11.436079	4.66e-14%
100	0.04	0.04	0.25	9.203779	9.203779	0
100	0.04	0.02	0.15	6.817526	6.817536	1.44e-04%
100	0.04	0.02	0.40	13.713691	13.713691	2.02e-08%
90	0.04	0.02	0.25	4.761567	4.761567	2.51e-09%
110	0.04	0.02	0.25	16.927702	16.927702	1.93e-10%

Further tests.

Test	What was checked	Result
Spot ladder, S_0 82 to 160	Replay vs fresh; price $\to 0$ at barrier; smooth delta	Agree to 1.2e-03%; delta 0.466 to 0.962
Rate ladder, r -2% to 10%	Replay vs fresh; positive rho	Agree to 1.7e-12%; rho $\geq$ 38.62
Volatility ladder, σ 5% to 60%	Price vs σ	Price rises from 3.06 to 16.10
Barrier parity	Down-in + down-out = European	Holds to 3.33e-14%
Cross-engine benchmark	Each benchmark engine within 0.5% of analytic	FD 0.0053%, CRR 0.0646%, JR 0.0487%; LR 0.508% (Finding 3)
Convergence	FD and trees approach analytic as resolution grows	FD 0.0013% and CRR 0.0646% at 200

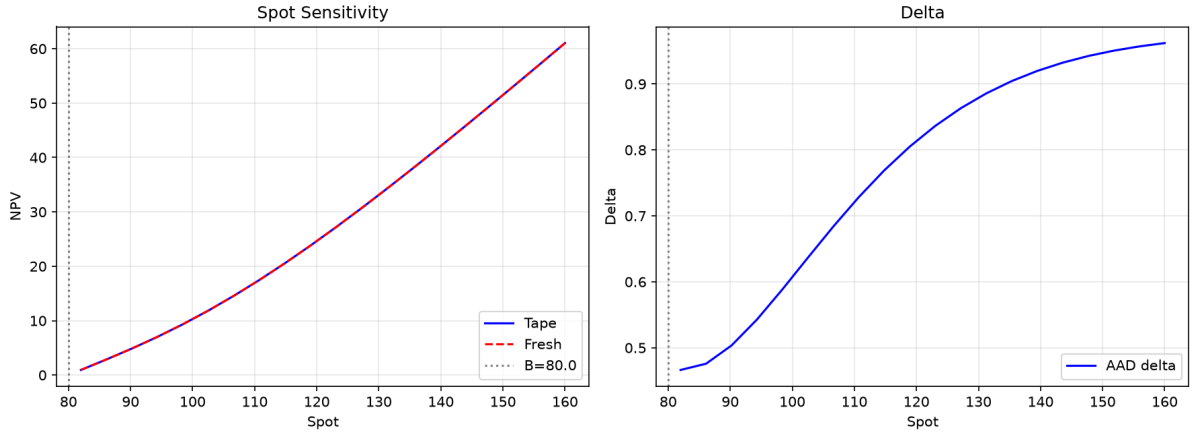


Figure 1: Spot ladder: price by replay and fresh pricing (left), AAD delta (right).

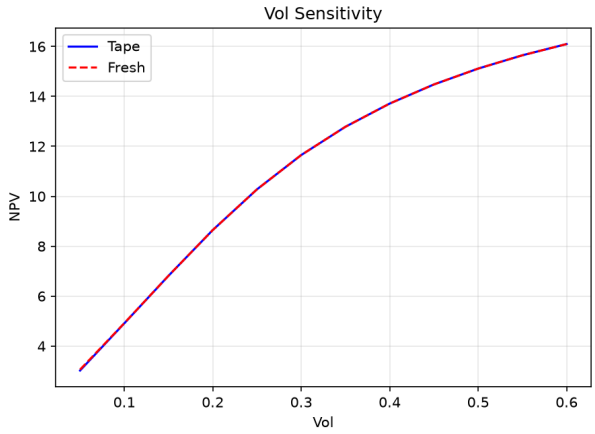


Figure 2: Volatility ladder: price by replay and fresh pricing.

Replay makes this coverage cheap. The analytic recording has 473 operations and replay runs at millions of evaluations per second, so ladders and scenario sets can be as fine as the reviewer wants.

5. Where the computation concentrates

The recording counts operations per function and per source line, so the reviewer can see where each engine spends its work.

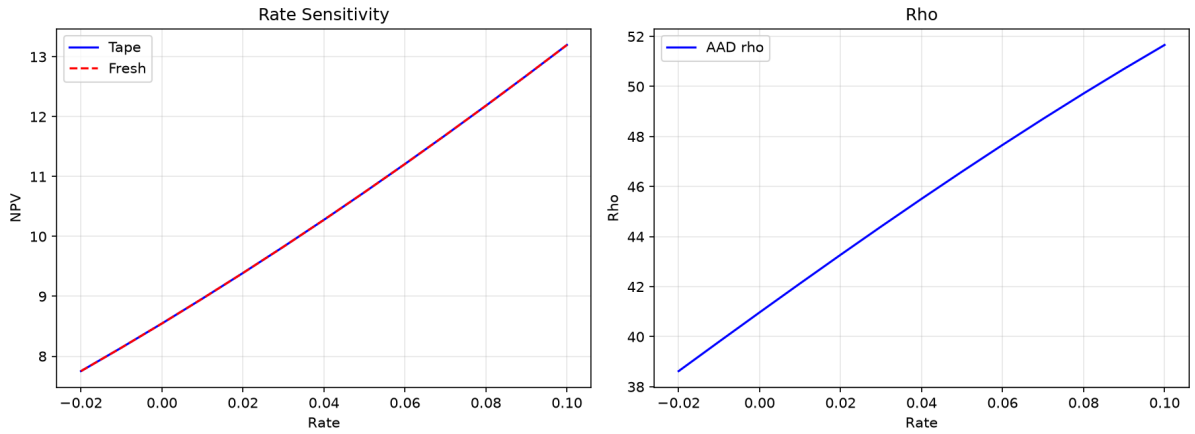


Figure 3: Rate ladder: price by replay and fresh pricing (left), AAD rho (right).

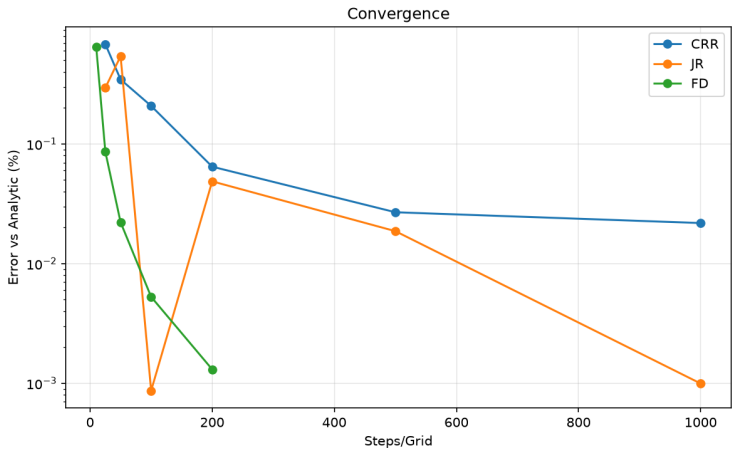


Figure 4: Convergence of the finite-difference and binomial engines to the analytic price.

Engine	Recorded operations	Leaves	Sites	Dominant function (operations)
Analytic	473	261	30	operator()↗ in errorfunction.cpp (135)
Finite differences	479,544	127,935	71	solve_splitting()↗ in triplebandlinearop.cpp (150,591)
Binomial CRR	441,136	145,688	48	stepback()↗ in bsmlattice.hpp (180,624)

By operation count the analytic recording is about 1,000 times smaller than either numerical benchmark (479,544 / 473 ≈ 1,014; 441,136 / 473 ≈ 933), and more than a quarter of it is the error function, which is also where most of its market-dependent decisions sit (§3). These are counts of recorded operations, a proxy for computational work rather than a timing profile.

Findings supported by the Tracer evidence

Each finding below rests on evidence the Tracer produced from the recordings, not on a reading of the source:

- **Verification of model dependencies (1).** A model inventory normally lists feeding models from documentation. The recording verifies them from the calculation itself: σ, r and q enter the price only through `BlackConstantVol` and `FlatForward`, each a single flat value, and no other upstream input reaches the price (§2). This confirms the dependencies on the volatility surface, yield curve and dividend models, and that their validation reports are needed.
- **Classification of r and q (2).** With r and q marked as inputs on the recording, a ± 100bp move in each was repriced by replay (§2). The swings of 0.910 and 1.116 against a 0.10 bid/ask establish materiality without rebuilding the pricing setup.
- **Benchmark engines (3, 4).** Each engine was recorded separately (§1). The recordings show the three binomial trees share backward induction, knock-out and the proximity test, and differ

only in the calibration of u, d and p: 8 operations for CRR, 2 for Jarrow–Rudd and 21 for Leisen–Reimer. The differences in accuracy and convergence between the trees are therefore localised to that calibration step. Findings 3 and 4 concern QuantLib's own benchmark trees, not the Tracer: the Tracer detected that one of the library's benchmark engines misses its tolerance, which is the evidence a validation team needs before relying on it.

The findings in this example were:

#	Severity	Finding	Recommendation
1	Material	Validation reports for the feeding yield curve, dividend and volatility surface models not available	Validate the feeding models independently
2	Medium	Production source of r and q not documented; both are material (swing 0.910 and 1.116 vs 0.10 bid/ask, §2)	Confirm the classification; if marked, document valuation controls (IPV, reserve)
3	Low	QuantLib's Leisen–Reimer implementation at 200 steps is 0.508% from analytic, outside the 0.5% benchmark tolerance	Use a higher resolution, or exclude it from barrier benchmarks
4	Low	Jarrow–Rudd convergence not monotonic (error rises 25→50 and 100→200 steps)	Use at least the production resolution

Evidence and regulatory alignment

The Tracer produces the evidence; the validators judge fitness and sign off. Three properties make that evidence usable in a regulated validation:

- **Verifiable.** Each formula, input and decision cites the file and line that produced it, so an independent party can check the claim against the source.
- **Reproducible.** Recordings and re-recordings are retained, and any test can be re-run by replay.
- **Complete within scope.** Inputs, constants and market-dependent decisions are enumerated from the calculation, not sampled. Decisions that cannot be reached within the tested input ranges are reported, not assumed safe.

These map to the expectations of PRA SS1/23 (independent validation; documentation sufficient for a third party), Fed SR 26-2 (conceptual soundness, effective challenge, ongoing monitoring) and the ECB Guide to internal models (independent validation and quantitative testing).

Who it is for and getting started

The Tracer is built for model validation and audit teams, model documentation teams, and quant developers working in C++ or Python. It runs on the production pricing code: switch it on with `AADC_TRACE=1` and the same program returns its prices plus the full record.

To try the Tracer on QuantLib or apply it to your own models, contact MatLogica.