QUANTIL

Technical Validation Note

Quantil RCAT v4.0 (English edition)

Technical Validation Note v1.2 for tool version 4.0, engine 1.0
Tool version 4.0, engine 1.0
2026-10-03

Quantil is a trading name of Bouman Management B.V.
Bouman Management B.V. | Barendrecht, the Netherlands | Chamber of Commerce (KvK) 98301764
Support is provided through the sales platform. All rights reserved.

Contents

1. Scope and approach
2. Engine description
3. Test program
4. Reference results
5. Sensitivity and limitations
6. Reproducibility instructions for an auditor
7. Conclusion
Appendix A: pseudo-code of the per-iteration loop
Appendix B: known-answer values

1. Scope and approach

This note documents the technical validation of the Quantil RCAT v4.0 Monte Carlo engine (MonteCarloEngine, engine 1.0), the random number generator (modQuantilRandom) and the exact Beta-PERT machinery (modQuantilBetaExact) that together produce the simulated P-levels used in the tool's weighted contingency recommendation.

The approach taken is an independent re-implementation. The complete engine was re-implemented in Python (engine.py), following the VBA source line by line: the same MRG32k3a recurrences, the same seeding rule, the same Box-Muller normal generator, the same exact regularized incomplete beta function and its inverse, the same Cholesky-based correlation handling, and the same percentile and histogram definitions.

The guiding principle of this replica is that it does not merely approximate the VBA engine, it mirrors it exactly, so that wherever both implementations consume the same random sequence, their results must agree to the cent. This is a stronger test than a statistical tolerance check: it verifies that the arithmetic itself, not just its distributional properties, is reproduced correctly.

Eighteen tests were carried out against this replica, covering the random number generator in isolation, the normal and Beta distribution machinery, single-risk and portfolio-level sampling, the effect of the correlation block, run-to-run reproducibility and seed sensitivity, and the percentile and histogram definitions used to write results to the workbook. All 18 tests passed; results are given in chapter 3 and the full reference case figures in chapter 4.

2. Engine description

2.1 Random numbers: MRG32k3a

The engine draws uniform (0,1) random numbers from MRG32k3a, a combined multiple recursive generator (L'Ecuyer, 1999, Operations Research 47(1)), with period approximately 2^191. The generator maintains two sets of three state variables and combines them as follows, using the constants:

Constant	Value
m1	4294967087
m2	4294944443
a12	1403580
a13n	810728
a21	527612
a23n	1370589

Component 1 recurses as p1 = (a12 x s11 − a13n x s10) mod m1; component 2 recurses as p2 = (a21 x s22 − a23n x s20) mod m2. The two components are combined as (p1 − p2 + m1) x NORM when p1 <= p2, and (p1 − p2) x NORM otherwise, where NORM = 1/(m1+1), giving a uniform value strictly between 0 and 1. All arithmetic is exact in double precision because every product stays below 2^53.

Seeding: all six state variables are set to the same seed value (fixed to a whole number). A seed of 0 (or, in the VBA, any value at or below 0) triggers a clock-based seed, which is not reproducible and is therefore never used for a validation run. The workbook default seed is 42 (CONFIG, cfgSeed).

Known answer: with seed 12345, the first draw from the generator is 0.1270111220466 (see Appendix B). The VBA module carries this as a built-in self-test (QRndSelfTest) that runs before every

Monte Carlo simulation; if it fails, the run is aborted with a message asking the user to re-import modQuantilRandom.

2.2 Normal generation: Box-Muller

Standard normal variates are generated by the Box-Muller transform from two uniforms u1 and u2: r = sqrt(-2 ln u1), giving z1 = r cos(2 pi u2) and z2 = r sin(2 pi u2). Both normals are used: z1 is returned immediately and z2 is cached and returned on the next call, so no draw from the underlying generator is wasted. The cache is reset at the start of each Monte Carlo run (QRndResetSpare).

2.3 Normal CDF: Abramowitz and Stegun 26.2.17

Where a standard normal cumulative probability is needed (to convert a correlated normal variate into a correlated uniform), the engine uses the polynomial approximation from Abramowitz and Stegun, formula 26.2.17, which has a maximum absolute error below 7.5e-8 and requires no external function calls, so it behaves identically on Windows and Mac.

2.4 Beta-PERT parametrization

Each risk's three-point estimate (minimum, most likely, maximum) is converted to a standard Beta-PERT distribution with shape parameters a = 1 + lambda (most likely − min)/(max − min) and b = 1 + lambda (max: most likely)/(max − min), where lambda is read from CONFIG (cfgLambda, default 4). This parametrization gives a distribution mean of (min + 4 x most likely + max)/6, the standard PERT expected value, and reduces to a = b when the estimate is symmetric, with no fallback to a plain point estimate.

2.5 Regularized incomplete beta and its inverse

The Beta cumulative distribution function I_x(a,b) is evaluated by a continued-fraction expansion (the Numerical Recipes approach), using a Lanczos approximation for the log-gamma function needed to normalize the Beta function. Both are implemented in pure VBA with no calls to Excel's WorksheetFunction, so results are identical on Windows and Mac and do not depend on Excel's own (slower, bisection-based) Beta_Dist.

The inverse of I_x(a,b), needed to convert a uniform draw into a sampled severity, is found by Newton iteration started from the distribution mean, with a bisection safeguard: whenever a Newton step would leave the current bracket [lo, hi], the step is replaced by a bisection step instead. The iteration stops when the residual is below a tolerance of 1e-12 or the bracket has collapsed.

2.6 Correlation: Cholesky factor of the 12 x 12 block

Correlation between risks is modeled for the first block of register rows R-001 to R-012, using the upper triangle of the matrix entered on 04_CORRELATION (mirrored to the lower triangle). If any off-diagonal coefficient exceeds 0.001 in magnitude, the engine builds the Cholesky factor L of this 12 x 12 matrix and stops with a clear message if the matrix is not positive definite, asking the user to reduce the strongest coefficients (typically below 0.7).

For each iteration, one standard normal is drawn per correlated risk, combined through L to give correlated normals, and each is passed through the normal CDF (section 2.3) to give a correlated uniform u for that risk. A single correlated uniform then drives both the occurrence and the severity of the risk: the risk fires when u is below its probability p, and, when it fires, its severity quantile is u/p, passed through the Beta inverse (section 2.5). Risks outside the 12 x 12 block, and any run without correlation, draw an independent uniform from the generator instead.

2.7 Percentile definition

Percentiles are read from the sorted array of simulated totals at index ceiling(q x N): for example, with N = 10,000 iterations, P80 is the 8,000th sorted result. This index-based definition is monotonic by

construction and matches exactly between the VBA and the Python replica because both use the same ceiling rule rather than an interpolated definition.

2.8 Sample standard deviation

The standard deviation reported alongside the percentiles is the sample standard deviation, using an (N-1) denominator, computed from the same sorted (or unsorted, the order does not affect the sum) array of simulated totals.

2.9 Histogram

The simulated totals are also summarized into a 20-bin histogram, spanning the minimum to the maximum simulated result, with each bin recording its bounds, midpoint, frequency, cumulative frequency and cumulative percentage; this is written to 06_MONTE_CARLO (rows 100-119) and drawn as the histogram and S-curve chart.

3. Test program

The 18 tests below were run against the Python replica (engine.py) Test 1 is also built into the workbook as a self-test that runs before every simulation, and the reference run of test 16 was repeated with the VBA engine in Microsoft Excel itself (section 4.3). Values and criteria are quoted exactly as recorded in the validation run (validation_results.json).

No.	Test	Criterion	Value	Pass
1	MRG32k3a known-answer test	first draw with seed 12345 = 0.1270111220466 (L'Ecuyer reference)	0.1270111220466	Pass
2	Uniformity of MRG32k3a output	Kolmogorov-Smirnov p-value > 0.01 on 200,000 draws	D = 0.00138, p = 0.839	Pass
3	Serial correlation of consecutive draws	|lag-1 autocorrelation| < 0.01	0.00111	Pass
4	Normal generator (Box-Muller, both normals used)	mean within 0.01, sd within 0.01 of (0, 1); Shapiro/KS p > 0.01	mean = -0.0014, sd = 1.0014, KS p = 0.656	Pass
5	Normal CDF approximation (A&S 26.2.17)	max absolute error < 1e-7 on [-6, 6]	7.45e-08	Pass
6	Log-gamma (Lanczos)	max absolute error < 1e-9 for x in [1, 60]	7.03e-11	Pass
7	Regularized incomplete beta I_x(a,b)	max absolute error < 1e-12 vs scipy.special.betainc (20 shape pairs, 200 points each)	2.43e-13	Pass
8	Beta inverse (Newton with bisection safeguard)	max absolute error < 1e-9 vs scipy.stats.beta.ppf (20 shape pairs, 400 quantiles each)	2.49e-11	Pass
9	Beta-PERT parametrization	distribution mean equals (Min + 4 ML + Max)/6 exactly	a = 2.3333, b = 3.6667, mean = 6,666,666.67 vs 6,666,666.67	Pass
10	Symmetric three-point estimate	a = b = 3 (no fallback to a point estimate)	a = 3.0, b = 3.0	Pass
11	Single-risk sampling (p = 1, 50,000 iterations)	mean within 0.5% and sd within 2% of the analytic Beta-PERT moments	mean 6,665,271 vs 6,666,667; sd 2,213,343 vs 2,211,083	Pass
12	Portfolio mean equals EMV sum (independent risks, 50,000 iterations)	simulated mean within 1% of the deterministic EMV sum	38,776,617 vs 38,671,000 (+0.27%)	Pass
13	Portfolio standard deviation (independent risks)	simulated sd within 2% of the analytic compound-distribution sd	18,258,303 vs 18,146,508	Pass
14	Effect of Cholesky correlation	mean unchanged within 1%; P90 and sd higher than the independent case	mean 38,737,401 vs 38,776,617; P90 63,886,201 vs 63,476,096; sd 18,468,147 vs 18,258,303	Pass

No.	Test	Criterion	Value	Pass
15	Occurrence correlation of a correlated pair (R-001/R-007, rho = 0.40)	empirical phi coefficient within 0.02 of the bivariate-normal value	0.2374 vs 0.2425	Pass
16	Reproducibility	two runs with seed 42 give identical P80 to the cent	52,963,086.61 / 52,963,086.61	Pass
17	Seed sensitivity	P80 with seed 4242 within 3 standard errors of the mean of seed 42 (order of magnitude check)	53,133,675 vs 52,963,087 (diff 170,588)	Pass
18	Percentile definition and histogram integrity	P80 = sorted result 8,000 of 10,000; percentiles monotonic; histogram frequencies sum to N	P80 index check True, bins sum 10000	Pass

3.1 Random number generator (tests 1 to 3)

These tests establish that the generator itself is correct and behaves as a good source of pseudo-random uniforms before any distribution machinery is built on top of it: the known-answer test (test 1) confirms the exact recurrence and seeding are implemented correctly, a Kolmogorov-Smirnov test on 200,000 draws (test 2) confirms the output is uniformly distributed, and a lag-1 autocorrelation check (test 3) confirms consecutive draws are not detectably correlated, which matters because the engine draws several uniforms per iteration.

3.2 Normal and Beta distribution machinery (tests 4 to 10)

These tests check the building blocks used to turn uniform draws into sampled risk impacts: the Box-Muller normal generator (test 4) and the normal CDF approximation (test 5) that together drive correlated sampling, the log-gamma function (test 6) that underlies the Beta functions, the regularized incomplete beta function and its inverse (tests 7 and 8) against scipy as an independent reference, and the Beta-PERT parametrization itself, both for a general three-point estimate (test 9) and for the symmetric special case (test 10), where the tool must not fall back to a plain point estimate.

3.3 Sampling and portfolio behavior (tests 11 to 15)

These tests move from individual distributions to the simulation as a whole: a single risk sampled 50,000 times should reproduce its analytic Beta-PERT mean and standard deviation (test 11); a portfolio of independent risks should reproduce the deterministic EMV sum as its mean (test 12) and a value close to the analytic compound-distribution standard deviation (test 13); adding the Cholesky correlation block should leave the mean essentially unchanged while raising the P90 and standard deviation, as expected for positively correlated risk (test 14); and the occurrence correlation actually produced for a specific correlated pair should match the value implied by the bivariate normal model within a small tolerance (test 15).

3.4 Reproducibility and output integrity (tests 16 to 18)

The final group checks the properties an auditor relies on directly: that two runs with the same seed give identical results to the cent (test 16); that a different seed changes the result only by an amount consistent with Monte Carlo sampling error, as an order-of-magnitude check rather than a tight tolerance (test 17); and that the percentile index rule and histogram bins are implemented and populated correctly (test 18).

4. Reference results

The workbook ships with the following reference results already written into 06_MONTE_CARLO and drawn on its charts, so that a user opening the full edition for the first time sees a completed, internally consistent run rather than a blank sheet.

4.1 Full edition (seed 42, 10,000 iterations, 30 risks, lambda 4, 12 x 12 Cholesky)

Statistic	Value (EUR)
Deterministic EMV	38,671,000.00
P10	17,609,631.37
P50	35,662,231.27
P80 (recommended budgeting level)	52,963,086.61
P85	57,262,028.66
P90	63,513,883.21
P95	74,192,980.12
Mean	38,594,764.12
Standard deviation	18,444,369.07
Coefficient of variation	0.48
Minimum simulated result	-200,710.70
Maximum simulated result	116,504,671.81

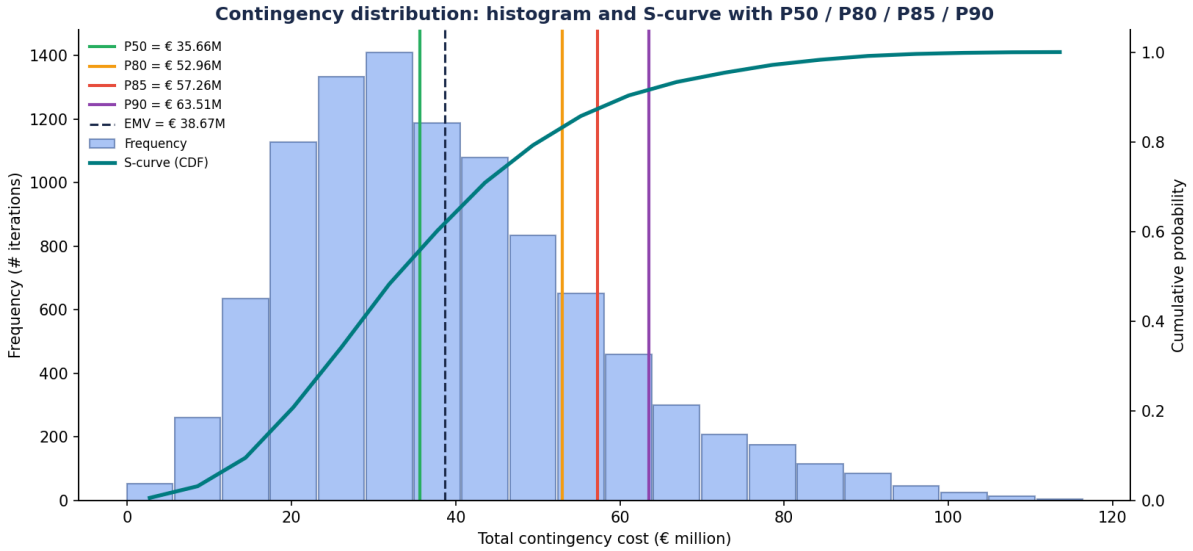


Figure 4.1: histogram and S-curve for the full-edition reference run (06_MONTE_CARLO).

4.2 Demo edition (seed 42, 2,000 iterations, 15 risks)

Statistic	Value (EUR)
P10	7,898,204
P50	17,948,914
P80 (recommended budgeting level)	26,487,060
P85	28,444,806
P90	30,682,559
P95	35,018,889
Mean	18,841,698
Standard deviation	9,066,370
Weighted recommendation	25,796,330 (7.6% of base)

These reference values let an auditor open either shipped edition, re-run the simulation with the same seed, and confirm the same figures reappear before examining any project-specific data.

4.3 Confirmation run in Microsoft Excel

After the v4.0 modules were imported into the full edition, the simulation was run with the VBA engine in desktop Microsoft Excel on Windows, with the shipped defaults (seed 42, 10,000 iterations, lambda 4, 12 x 12 Cholesky correlation). The built-in known-answer self-test of the generator passed (otherwise the run would have been aborted), and the workbook reproduced the Python replica as follows.

Statistic	Python replica (EUR)	VBA engine in Excel (EUR)
Iterations / risks included	10,000 / 30	10,000 / 30
Deterministic EMV	38,671,000	38,671,000
P10	17,609,631	17,609,631
P50	35,662,231	35,662,231
P80 (recommended budgeting level)	52,963,087	52,963,087
P85	57,262,029	57,262,029
P90	63,513,883	63,513,883
P95	74,192,980	74,192,980
Mean	38,594,764	38,594,764
Standard deviation	18,444,369	18,444,369
Coefficient of variation	0.48	0.48

Run information recorded by the workbook on 06_MONTE_CARLO: seed 42 (H85), engine "RCAT v4.0 engine 1.0 (MRG32k3a, exact Beta inverse)" (H86), timestamp 2026-10-03 10:27 (H87), correlation mode Cholesky (12 x 12) (H88), run time 0.9 seconds (H89) and lambda 4 (H90). All values agree to the euro at the precision displayed by the workbook. The demo edition was confirmed in the same way on 2026-10-03 at 10:33 with its own capped modules (seed 42, 2,000 iterations, 15 risks, Cholesky 12 x 12, run time 0.1 seconds): P50 17,948,914, P80 26,487,060 and P90 30,682,559, identical to the reference values in section 4.2.

5. Sensitivity and limitations

5.1 Seed sensitivity

Test 17 reran the full-edition reference case with a different seed (4242 in place of 42) and compared the resulting P80 with the seed-42 result, as an order-of-magnitude check on Monte Carlo sampling error rather than a tight reproducibility test (that is test 16, which requires exact agreement for the same seed). The result was P80 = 53,133,675 against 52,963,087 for seed 42, a difference of 170,588, small relative to the width of the distribution (P10 to P90 spans roughly 46 million).

5.2 Monte Carlo standard error of the P80 estimate

The seed-to-seed difference in section 5.1 is consistent, in order of magnitude, with the Monte Carlo standard error expected for a P80 estimate from 10,000 iterations of a distribution with the reference case's standard deviation (18,444,369): percentile estimators of this kind have a standard error on the order of a few hundred thousand euros at this sample size and this percentile. This is quoted as an order-of-magnitude statement only; it is not a formal confidence interval on the P80, and users who need a tighter bound should increase the iteration count (CONFIG allows 1,000 to 100,000) and compare results across seeds directly.

5.3 Limits of the current engine

- Correlation is modeled for a block of 12 risks only (register rows R-001 to R-012); risks beyond this block are sampled independently, regardless of any real-world correlation with other risks.
- The engine has no schedule model: it simulates cost impact only, not the interaction between schedule risk and cost risk that an integrated cost-and-schedule model would capture.
- The what-if section on 10_SCENARIO (base cost change, additional risks, probability uplift, impact escalation, correlation factor change, and a combined total) uses linear approximations rather than re-running the simulation, and should be read as indicative sensitivity, not as an alternative simulated result.

6. Reproducibility instructions for an auditor

An auditor who wishes to confirm a set of results independently should proceed as follows.

1. Record the run information block on 06_MONTE_CARLO before doing anything else: the seed used (H85), the engine label (H86), the run timestamp (H87), the correlation mode (H88) and the run time (H89), together with the lambda used (H90).
2. Record the statistics block (E85:E96): iterations, risks included, deterministic EMV, and P10, P50, P80, P85, P90, P95, mean, standard deviation and coefficient of variation.
3. Confirm CONFIG holds the same seed (cfgSeed), iteration count (cfgIterations) and lambda (cfgLambda) that the run information block reports as having been used.
4. Re-run the simulation (the Run Monte Carlo button on 12_DASHBOARD or 06_MONTE_CARLO, or Ctrl+Shift+M, which both call RunMonteCarlo) with the same seed and iteration count, and confirm the statistics block reproduces the same values to the cent (this is the property verified independently as test 16).
5. To check the calculation itself rather than only its reproducibility, extract the risk register (probability, minimum, most likely, maximum, Include? flag) and the 04_CORRELATION matrix, and run them through the Python replica (engine.py) with the same seed, iteration count and lambda; compare the resulting P-levels and statistics against the workbook's statistics block.
6. Where the comparison is against the shipped reference case rather than a re-extracted register, compare directly against the values in chapter 4 of this note.

Because the engine is seeded and the percentile and standard deviation definitions are documented in chapter 2, this comparison should agree exactly for the full edition and demo edition reference cases, and to the cent for any other register when the same seed, iteration count and lambda are used in both the workbook and the replica.

7. Conclusion

The Quantil RCAT v4.0 Monte Carlo engine has been independently re-implemented in Python and checked against 18 tests spanning the random number generator, the normal and Beta distribution machinery, portfolio-level sampling, the effect of correlation, reproducibility, seed sensitivity and output integrity. All 18 tests passed, and the full-edition and demo-edition reference cases shipped with the tool were confirmed to match the values recorded in this note. On 2026-10-03 the VBA engine itself was run in Microsoft Excel on the full-edition and demo-edition reference cases and reproduced the P-levels of the Python replica to the euro (section 4.3): the workbook engine and the validated replica are arithmetically identical.

The engine's known limitations, a 12-risk correlation block, no schedule model, and linear approximations in the what-if section, are stated in chapter 5 so that users and auditors can weigh the

simulated P-levels appropriately alongside the bottom-up and factor-based methods described in the companion Quality and Benchmark Dossier.

Appendix A: pseudo-code of the per-iteration loop

The following pseudo-code summarizes one iteration of the simulation, as implemented identically in MonteCarloEngine (VBA) and engine.py (Python).

Step	Operation
1	iterTotal <- 0
2	If correlation active: draw one standard normal per correlated risk (1..12); combine through the Cholesky factor L; pass each combined normal through the normal CDF to obtain a correlated uniform u(1..12).
3	For each included risk j in register order:
3a	If risk j is in the correlated block, u <- u(row of j); otherwise draw an independent uniform u.
3b	If u < probability(j): the risk fires.
3c	If it fires and has a valid three-point estimate: severity quantile <- u / probability(j); x <- BetaInverse(quantile, a(j), b(j)); impact <- min(j) + x (max(j) − min(j)); iterTotal <- iterTotal + impact.
3d	If it fires but has no valid range (min = max): iterTotal <- iterTotal + mostLikely(j).
4	Store iterTotal as the result of this iteration.
5	After all iterations: sort the results; read P10..P95 at index ceiling(q x N); compute the mean and the (N-1) sample standard deviation; build the 20-bin histogram; write everything, with the seed, engine label, timestamp and run time, to 06_MONTE_CARLO.

Appendix B: known-answer values

These two values are the fixed reference points used to confirm that a given implementation, VBA or the Python replica, is arithmetically identical to the one this note describes.

Check	Value
MRG32k3a, first draw with seed 12345	0.1270111220466
Beta inverse, I_x^-1(0.3; a = 2.5, b = 3.5)	0.3017723232475

The first is the built-in self-test run by the VBA engine before every simulation (QRndSelfTest); the second is the self-test of the exact Beta inverse (QBetaSelfTest in modQuantilBetaExact, which can be run from the VBA editor), checked in this note against scipy.stats.beta.ppf(0.3, 2.5, 3.5).