Scientific validation

```
library(testflow)
```

Purpose

Passing tests only proves a package is internally consistent: a formula reproduces its own citation, a search converges to *some* answer, `R CMD check` runs clean. None of that proves the formula is the *right* one, or that the search converges to the *correct* answer. This vignette documents a different, stronger standard applied across `testflow`'s statistical surface: every formula with real risk of being wrong - anything bespoke, iterative, or approximate - was checked against an independent source: a hand re-derivation, a Monte Carlo simulation, or an established, separately maintained R package that implements the same calculation.

That distinction is not academic. Two real defects were found this way, both silent - no error, no warning, a clean `R CMD check`, and numbers plausible enough that nothing about them looked wrong until they were checked against an outside reference. Both are described below, together with the wider set of checks that turned up nothing.

What "validated" means here

For a formula to count as validated in this document, one of the following had to hold:

- **External package agreement.** The result matches an independently maintained R package that implements the same statistic, to numerical precision (typically 1e-6 or better; sometimes exact, e.g. `1e-16` for the Wilson interval against `stats::prop.test()`).
- **Independent re-derivation.** The formula was re-derived from the underlying probability model (e.g. integrating the exponential hazard over uniform accrual by hand) and checked against `stats::integrate()` or a closed-form solution, without re-reading the package's own code.
- **Simulation.** For quantities defined by a data-generating process (e.g. the probability of an event under staggered trial accrual), a large Monte Carlo simulation (millions of draws) confirmed the closed form to within simulation error.

Internal-only checks - "the code reproduces the formula written in its own roxygen block" - do not count on their own, precisely because that is the standard that let both defects below ship undetected.

Two defects found and fixed

Bioequivalence sample size: `iterative_tost` was not exact

`sample_size_bioequivalence(method = "iterative_tost")` searches for the smallest sample size achieving a target two-one-sided-test (TOST) power for average bioequivalence. Until this validation pass, the power calculation used a **normal approximation** (`stats::pnorm()`/`qnorm()`), even though it was documented as exact. The correct calculation uses the *noncentral* t distribution, because the estimated standard error carries its own sampling variability that a normal approximation ignores.

The consequence was concrete: at `GMR = 0.95`, `CV = 15%`, 90% power, the old (normal-approximation) code returned `n = 13`; the correct noncentral-t calculation requires `n = 16` - an under-sized study by about 23%.

```
# Current (fixed) implementation: testflow treats n as one continuous total
# and rounds up to the nearest even number, since a 2x2 crossover has two
# equal-sized sequences
```

```r
x <- sample_size_bioequivalence(
  design = "crossover", gmr = 0.95, cv_within = 0.15,
  alpha = 0.05, power = 0.90, method = "iterative_tost"
)
n_testflow <- 2 * ceiling(x$n / 2)
n_testflow
#> [1] 16

# Cross-check against PowerTOST's Owen's-Q-based exact calculation (the
# calculation used in real regulatory bioequivalence submissions)
PowerTOST::sampleN.TOST(
  alpha = 0.05, targetpower = 0.90, theta0 = 0.95,
  theta1 = 0.80, theta2 = 1.25, CV = 0.15, design = "2x2", print = FALSE
)$`Sample size`
#> [1] 16
```

The two match exactly. See `?sample_size_bioequivalence` for the full noncentral-t formula.

Cohen's kappa: the hypothesis test used the wrong standard error

`test_agreement()` reports Cohen's kappa with a confidence interval and a test of $H_0 : \kappa = 0$. Two *different* standard errors are needed for these two purposes, because the sampling variance of $\hat{\kappa}$ depends on κ itself:

- the CI needs the SE *at* $\hat{\kappa}$ (Fleiss, Cohen & Everitt, 1969);
- the null test needs the SE *at* $\kappa = 0$ (Fleiss, 1981) - a different, generally larger, quantity whenever agreement is materially above chance, which is the typical case.

The code computed the CI's SE correctly, then reused it as the test denominator instead of the null SE. This inflated the z-statistic and made p-values anti-conservative (too small - a false sense of significance), though the point estimate and CI themselves were unaffected.

```r
set.seed(2)
n <- 200
cats <- c("A", "B", "C")
rater1 <- sample(cats, n, replace = TRUE, prob = c(0.5, 0.3, 0.2))
rater2 <- ifelse(runif(n) < 0.75, rater1, sample(cats, n, replace = TRUE))
dat <- data.frame(r1 = factor(rater1, levels = cats), r2 = factor(rater2, levels = cats))

x <- test_agreement(dat, rater1 = r1, rater2 = r2)
x$primary_test[, c("statistic", "p.value")]
#> # A tibble: 1 x 2
#>   statistic  p.value
#>       <dbl>    <dbl>
#> 1      14.4 9.86e-47

irr::kappa2(dat[, c("r1", "r2")])[c("statistic", "p.value")]
#> $statistic
#> [1] 14.35536
#>
#> $p.value
#> [1] 0
```

The two now match exactly. Before the fix, `testflow` reported z = 17.2 here instead of the correct 14.4.

Everything else checked, with nothing found

The rest of the package's statistical surface was checked the same way. No further defects were found; results are summarized rather than reproduced live here, since exhaustively re-running every cross-check would require installing a dozen reference packages just to build this vignette.

Area	Checked against	Result
One-proportion precision (Wald/Wilson/exact), minimum-n search	`stats::prop.test()`, `stats::binom.test()`, exhaustive brute-force scan	Exact match; brute-force testing caught and fixed a separate non-monotonicity bug in the search algorithm itself (see `NEWS.md`)
Survival: uniform-accrual event probability	`stats::integrate()`, Monte Carlo simulation (2M draws)	Matches to 1e-14 / within simulation error
Survival: Schoenfeld and Freedman event formulas	Hand-derivation from the cited references	Exact match
Precision planning: risk-difference (unequal allocation), log-odds-ratio	Independent variance-formula derivation	Exact match
Continuous/binary sample size (superiority, NI, equivalence)	`TrialSize` (same methodology), `pwr` (different, exact-t methodology; small expected divergence confirmed)	Exact match vs. `TrialSize`; expected small gap vs. `pwr`
Linear regression (F-test, R2, adjusted R2), logistic regression (LR-test, McFadden R2), VIF	`stats::lm/glm`, `lmtest::lrtest()`, `pscl::pR2()`	Exact match
Symmetry test (Cabilio-Masaro)	`lawstat::symmetry.test(option = "CM")`	Exact match
Factorial ANOVA (Type I/II/III), eta squared	`car::Anova()`, `effectsize::eta_squared()`	Exact match
Repeated-measures ANOVA, Mauchly's W, generalized eta squared	`rstatix::anova_test()`	Exact match
Friedman test effect size, Cochran's Q	Hand-derivation, `DescTools::CochranQTest()`	Exact match
Diagnostic accuracy (sensitivity, specificity, PPV, NPV, LR+, LR-)	`epiR::epi.tests()`	Exact match
ROC / AUC (Hanley-McNeil), Youden's J	Independent re-derivation, `pROC`	Exact match
ICC(1,1), ICC(2,1), ICC(3,1), including the Satterthwaite-approximated CI	`psych::ICC()`	Matches to 5+ significant figures
Cohen's d (independent, one-sample, paired), Cramer's V, rank-biserial correlation	`effectsize`	Exact match
Kruskal-Wallis epsilon squared	`rstatix::kruskal_effsize()`	Exact match
Cox regression (LR-test, concordance)	`survival::coxph()` internals	Exact match
Correlation, categorical association, proportion tests	Thin `stats::` wrappers	Delegate correctly by construction

What this does and does not guarantee

This validation targeted correctness of the implemented formulas against their stated methodology - not every possible methodological choice. A formula can be correctly implemented and still not be the specific convention you expect (for example, `testflow`'s Cramer's V is the raw, non-bias-corrected statistic, and its rank-biserial correlation uses the opposite sign convention to some other packages) - these are documented choices, not defects, and worth checking against your own convention if it matters for your use case.

It also does not certify code paths with no numerically bespoke logic to get wrong in the first place (formatting, plotting, printing) - those carry essentially no statistical risk and were not part of this exercise.

If you find a discrepancy against another tool, please open an issue with a reproducible example: `https://github.com/ielbadisy/testflow/issues`. The two defects above were both silent for some time before being checked this way, and the most reliable way to keep it from happening again is the same one that found them - compare against an independent source, not just the package's own tests.