Package 'feasiblesize'

November 23, 2026

Type Package

Title Feasible Sample Sizes from Incomplete Design Information

Version 0.1.0

Description Identifies feasible sample sizes under specified false negative and false positive requirements when planning information leaves several compatible distributions. Computes inner and outer set bounds, evaluates normal and paired binary designs, and compares prespecified rules with a finite decision class. For multivariate binary distributions with fixed proper subset margins, rational arithmetic and Sturm sequences give exact feasibility calculations over the full compatibility interval. Implements the framework of Nakamura (2026) <doi:10.3390/stats9050105>.

License BSD_3_clause + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports gmp, stats, graphics, utils

Suggests testthat (>= 3.0.0), knitr, rmarkdown, irr, jsonlite

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 8.0.0

NeedsCompilation no

Author Satoshi Nakamura [aut, cre, cph]

Maintainer Satoshi Nakamura <s-nakamura@med.mie-u.ac.jp>

Repository CRAN

Date/Publication 2026-11-23 14:20:02 UTC

Contents

feasiblesize-package 2
binary_cells 3
binary_compatibility 3

binary_mass 4
classify_rule_family 4
compatible_distribution 5
coordinate_feasibility 6
directional_evidence 7
exact_rational 7
feasible_sizes 8
normal_risks 9
normal_size_bounds 10
operating_polynomial 11
paired_binary_risks 12
paired_discordance_bounds 13
plot.fs_size_bounds 13
polynomial_evaluate 14
polynomial_roots 15
proper_subset_margins 15
rational_polynomial 16
risk_constraint 17
size_bounds 17

Index **19**

`feasiblesize-package` *Feasible Sample Sizes from Incomplete Design Information*

Description

Calculates feasible sets for prespecified rules under false negative and false positive requirements. Compatible states yield inner and outer set bounds. Exact binary calculations use rational arithmetic over the complete affine compatibility interval. Normal and conditional paired binary operating probabilities use numerical distribution functions.

Author(s)

Maintainer: Satoshi Nakamura <s-nakamura@med.mie-u.ac.jp> [copyright holder]

Authors:

- Satoshi Nakamura <s-nakamura@med.mie-u.ac.jp> [copyright holder]

References

Nakamura, S. (2026). Identifying Feasible Sample Sizes from Incomplete Design Information. Stats, 9(5), 105. doi:10.3390/stats9050105.

See Also

`size_bounds()`, `normal_size_bounds()`, `coordinate_feasibility()`, `classify_rule_family()`.

binary_cells *Binary Cells in Lexicographic Order*

Description

Binary Cells in Lexicographic Order

Usage

```
binary_cells(dimension)
```

Arguments

`dimension`	Number of binary variables, an integer from 2 to 20.

Value

Integer matrix, one cell per row; the last coordinate varies fastest.

Examples

```
binary_cells(2)
```

binary_compatibility *Exact Compatibility Interval for Fixed Proper Subset Margins*

Description

Exact Compatibility Interval for Fixed Proper Subset Margins

Usage

```
binary_compatibility(baseline)
```

Arguments

`baseline`	A compatible baseline distribution from `binary_mass()`.

Details

All proper subset margins must be treated as fixed planning inputs. Every compatible mass is `p0(y) + lambda * product(2*y[j]-1)`. The supplied baseline must already be compatible; this function does not solve for a baseline from arbitrary margins. It does not replace sampling uncertainty in estimated margins with an exact population guarantee.

Value

An `fs_compatibility` object with exact `lower`, `upper`, `eta_baseline`, `eta_lower`, `eta_upper`, endpoint zero cells, and the baseline distribution.

Examples

```
binary_compatibility(binary_mass(rep("1/8", 8)))
```

`binary_mass` *An Exact Multivariate Binary Distribution*

Description

An Exact Multivariate Binary Distribution

Usage

```
binary_mass(probabilities, dimension = log2(length(probabilities)))
```

Arguments

`probabilities`	Complete cell probabilities in the order of `binary_cells()`, as exact inputs. May be named by binary cell labels.
`dimension`	Number of binary coordinates; inferred from the vector length if omitted.

Value

An `fs_binary_mass` object with cells and exact probabilities.

Examples

```
binary_mass(rep("1/4", 4))
```

`classify_rule_family` *Classify a Finite Family of Rules over the Full State Interval*

Description

Classify a Finite Family of Rules over the Full State Interval

Usage

```
classify_rule_family(compatibility, n, planned_rule, rule_family)
```

Arguments

`compatibility`	Binary compatibility interval.
`n`	Candidate sample sizes.
`planned_rule`	Name of the prespecified rule.
`rule_family`	Named list of rules. Each rule is a named list indexed by character sample sizes, each containing its polynomial risk constraints.

Details

T1: the planned rule works everywhere. T2: one common alternative works everywhere. T3: some rule works at each state, but no one rule works everywhere. T4: a rule works at some states only. T5: no rule works anywhere. Common roots from all rules are used to preserve the quantifier order. The planned rule outer bound differs from the decision class outer bound.

Value

A list with four nested sets, five cases T1 through T5, prespecified rule `inner` and `outer` bounds, and candidate specific evidence.

Examples

```
cpt <- binary_compatibility(binary_mass(rep("1/4", 4)))
ok <- list(risk_constraint("fp", "false_positive", 0L, "1/2"))
classify_rule_family(cpt, 1L, "planned", list(planned = list("1" = ok)))
```

`compatible_distribution` *Evaluate a Compatible Binary Distribution*

Description

Evaluate a Compatible Binary Distribution

Usage

```
compatible_distribution(compatibility, coordinate)
```

Arguments

`compatibility`	An object from `binary_compatibility()`.
`coordinate`	One exact lambda coordinate in the closed interval.

Value

An exact binary mass.

Examples

```
cpt <- binary_compatibility(binary_mass(rep("1/4", 4)))
compatible_distribution(cpt, "1/8")
```

`coordinate_feasibility` *Exact Feasibility over a Binary Compatibility Interval*

Description

Exact Feasibility over a Binary Compatibility Interval

Usage

```
coordinate_feasibility(compatibility, constraints, refinement_steps = 24L)
```

Arguments

`compatibility`	Object from `binary_compatibility()`.
`constraints`	Nonempty list of uniquely named `risk_constraint()` objects.
`refinement_steps`	Positive number of exact root bisections.

Details

Certifies the entire closed interval, including isolated feasible points, using exact rational arithmetic and Sturm root isolation. The classification is `"feasible_everywhere"`, `"unidentified"`, or `"infeasible_everywhere"`. A finite collection of constraint polynomials must correctly represent the supplied statistical problem; their statistical derivation is the caller's responsibility.

Value

An `fs_coordinate_result` with classification, all boundary points, signs, rational witnesses between boundaries, and feasible components. Components contain one based boundary indices `lower` and `upper`.

Examples

```
cpt <- binary_compatibility(binary_mass(rep("1/4", 4)))
p <- rational_polynomial(c("1/2", 2L))
coordinate_feasibility(cpt, list(
    risk_constraint("fn", "false_negative", 1L-p, "3/4"),
    risk_constraint("fp", "false_positive", p, "3/4")))
```

directional_evidence *Directional Evidence Relative to a Feasible Comparator*

Description

Directional Evidence Relative to a Feasible Comparator

Usage

```
directional_evidence(n, risks, beta = 0.2, gamma = 0.2)
```

Arguments

n	One candidate size.
risks	Risk table as in feasible_sizes().
beta, gamma	Error tolerances.

Details

A failed constraint alone does not justify a directional claim. A smaller feasible comparator is required for "too large", and a larger feasible comparator for "too small", under the same rule and state.

Value

A list with a status and feasible smaller or larger comparators.

Examples

```
directional_evidence(1000, normal_risks(1:1000))
```

exact_rational *Exact Rational Inputs*

Description

Exact Rational Inputs

Usage

```
exact_rational(x)
```

Arguments

x	An integer, rational or decimal character vector, or a bigq rational vector from package 'gmp'. Noninteger double inputs are rejected.

Details

Use character input such as `"0.1"` or `"1/10"` to state an exact probability. Binary floating point inputs such as `0.1` are not silently treated as exact decimal probabilities. Integer doubles up to 2^53 are accepted.

Value

A `bigq` vector of exact rational numbers.

Examples

```
exact_rational(c("0.1", "1/3", "1e-3"))
```

`feasible_sizes` *Feasible Sizes under Two Error Requirements*

Description

Feasible Sizes under Two Error Requirements

Usage

```
feasible_sizes(risks, beta = 0.2, gamma = 0.2)
```

Arguments

`risks`	Data frame with columns `n`, `false_negative`, and `false_positive`, containing one row per candidate.
`beta`	Maximum probability of missing a meaningful effect.
`gamma`	Maximum probability of a positive decision for a negligible effect.

Details

Both inequalities are inclusive. The calculation is conditional on the supplied risks; it does not validate their statistical derivation.

Value

Sorted integer vector of feasible candidate sizes.

Examples

```
feasible_sizes(normal_risks(1:100))
```

normal_risks *Operating Probabilities for a Normal Mean*

Description

Operating Probabilities for a Normal Mean

Usage

```
normal_risks(
  n,
  d_negligible = 0.1,
  d_meaningful = 0.5,
  sigma = 1,
  alpha = 0.05,
  rule = c("point", "minimum"),
  calibration_sigma = sigma
)
```

Arguments

n	Distinct positive candidate sizes.
d_negligible	Upper boundary of negligible effects.
d_meaningful	Lower boundary of meaningful effects; must exceed d_negligible.
sigma	Positive standard deviation of one observation.
alpha	One sided nominal significance level.
rule	"point" for a zero null boundary or "minimum" for the negligible effect boundary.
calibration_sigma	Positive, fixed standard deviation used to set the critical value. Defaults to sigma for a fully specified known variance model. Hold this fixed when evaluating one rule over uncertain sigma.

Details

Assumes independent normal observations. Rejects when the sample mean exceeds null_boundary + qnorm(1-alpha)*calibration_sigma/sqrt(n). Risks are evaluated at the declared effect boundaries. Floating point normal probabilities are used, rather than exact arithmetic.

Value

A data frame of candidate sizes, both risks, and power.

Examples

```
feasible_sizes(normal_risks(1:1000))
feasible_sizes(normal_risks(1:1000, rule = "minimum"))
```

`normal_size_bounds` *Normal Bounds over a Continuous Standard Deviation Interval*

Description

Normal Bounds over a Continuous Standard Deviation Interval

Usage

```
normal_size_bounds(
  n,
  sigma_range,
  d_negligible = 0.1,
  d_meaningful = 0.5,
  alpha = 0.05,
  rule = c("point", "minimum"),
  beta = 0.2,
  gamma = 0.2,
  calibration_sigma = max(sigma_range)
)
```

Arguments

`n`	Distinct positive candidate sizes.
`sigma_range`	Two ordered positive standard deviation endpoints.
`d_negligible`	Upper boundary of negligible effects.
`d_meaningful`	Lower boundary of meaningful effects; must exceed `d_negligible`.
`alpha`	One sided nominal significance level.
`rule`	`"point"` for a zero null boundary or `"minimum"` for the negligible effect boundary.
`beta`, `gamma`	Error tolerances, each strictly between zero and one half.
`calibration_sigma`	Positive, fixed standard deviation used to set the critical value. Defaults to `sigma` for a fully specified known variance model. Hold this fixed when evaluating one rule over uncertain `sigma`.

Details

Uses one common critical value calibrated with `calibration_sigma`, which defaults to the upper endpoint. When both error tolerances are below one half, a feasible critical mean must lie between the effect boundaries. Both risks then increase with sigma, so endpoint calculations give the exact inner and outer sets for this normal model. If sigma were known separately in each state and the critical value changed with it, that would be a different family of rules.

Value

An `fs_size_bounds` object over the complete closed `sigma_range`.

Examples

```
b <- normal_size_bounds(1:800, sigma_range = c(0.22, 0.30),
                        d_negligible = 0.02, d_meaningful = 0.05,
                        rule = "minimum")
min(b$inner)
```

---

operating_polynomial *Exact Operating Probability Polynomial for a Fixed Rule*

Description

Exact Operating Probability Polynomial for a Fixed Rule

Usage

```
operating_polynomial(compatibility, n, rule, max_sequences = 1000000L)
```

Arguments

`compatibility`	Binary compatibility interval.
`n`	A positive integer sample size.
`rule`	Function taking an integer matrix with n rows and returning an exact conditional positive decision probability in `[0,1]`.
`max_sequences`	Maximum ordered samples to enumerate; positive integer.

Details

Enumeration is exponential. This constructor is intended for small exact examples. For larger designs supply analytically derived risk polynomials to `risk_constraint()` or use a supported model calculation. The rule is fixed across lambda; it must not access the unknown coordinate.

Value

Exact rational polynomial in lambda.

Examples

```
cpt <- binary_compatibility(binary_mass(rep("1/4", 4)))
operating_polynomial(cpt, 1L, function(s) as.integer(prod(2*s[1,]-1) == 1))
```

paired_binary_risks *Operating Probabilities for Independent Paired Binary Observations*

Description

Operating Probabilities for Independent Paired Binary Observations

Usage

```
paired_binary_risks(
  n,
  discordance,
  d_negligible = 0.01,
  d_meaningful = 0.05,
  alpha = 0.05,
  rule = c("point", "minimum"),
  calibration_discordance = NULL
)
```

Arguments

n	Distinct positive candidate sizes.
discordance	Probability that the paired outcomes disagree, in (0,1].
d_negligible, d_meaningful	Boundaries for the difference P(1,0)-P(0,1).
alpha	One sided nominal level.
rule	"point" for the zero boundary conditional binomial rule or "minimum" for a rule calibrated at a supplied known discordance.
calibration_discordance	Discordance used to calibrate the minimum effect rule. Must be supplied explicitly for that rule. Hold it fixed across states if the actual discordance is uncertain.

Details

Conditions on the discordant count M, which is Binomial(n,r). Given M, positive discordances are Binomial(M,(r+d)/(2*r)). Uses the strict critical count qbinom(1-alpha,M,p0)+1. Summing the exact binomial model probabilities over M removes Monte Carlo error, but the numerical probabilities use floating point arithmetic. Summation roundoff within 1e-12 of [0,1] is clipped to that interval; larger violations are errors. The zero boundary critical counts do not depend on r. A minimum effect rule calibrated separately with each unknown state is not a common implementable rule. This function therefore requires its calibration discordance to be explicit.

Value

Data frame with both error probabilities and power.

Examples

```
paired_binary_risks(c(300, 500, 1000), 0.11275)
paired_binary_risks(c(500, 1000), 0.11275, rule = "minimum",
                    calibration_discordance = 0.11275)
```

`paired_discordance_bounds` *Discordance Bounds from Paired Binary Margins*

Description

Discordance Bounds from Paired Binary Margins

Usage

```
paired_discordance_bounds(p_first, p_second, minimum_difference = 0)
```

Arguments

`p_first`, `p_second`	Marginal success probabilities.
`minimum_difference`	Optional nonnegative boundary that the discordance must permit, for example the meaningful difference.

Value

Named vector with the sharp lower and upper discordance bounds, restricted by `minimum_difference`. Errors if the restricted set is empty.

Examples

```
paired_discordance_bounds(0.4, 0.35, 0.05)
```

`plot.fs_size_bounds` *Plot Inner and Outer Sample Size Bounds*

Description

Plot Inner and Outer Sample Size Bounds

Usage

```
## S3 method for class 'fs_size_bounds'
plot(x, y = NULL, ...)
```

Arguments

x	An object returned by size_bounds() or normal_size_bounds().
y	Unused.
...	Additional graphical arguments passed to graphics::plot().

Value

x, invisibly. Produces a plot.

---

polynomial_evaluate *Evaluate an Exact Polynomial*

Description

Evaluate an Exact Polynomial

Usage

```
polynomial_evaluate(polynomial, x)
```

Arguments

polynomial	An object from rational_polynomial().
x	One exact rational coordinate.

Value

One exact bigq rational.

Examples

```
polynomial_evaluate(rational_polynomial(c("1/2", 1L)), "1/4")
```

polynomial_roots *Isolate Real Roots Using Exact Sturm Sequences*

Description

Isolate Real Roots Using Exact Sturm Sequences

Usage

```
polynomial_roots(polynomial, lower, upper, refinement_steps = 24L)
```

Arguments

`polynomial`	An exact rational polynomial.
`lower`, `upper`	Exact endpoints of a closed interval.
`refinement_steps`	Positive integer number of bisections once an interval contains exactly one root. This controls the displayed isolating width, not the validity of the root count.

Details

Repeated roots are counted once. Identically zero polynomials are rejected. No finite grid or floating point root finder is used.

Value

Ordered list of roots, each represented by exact rational `lower` and `upper`. Equal bounds denote an exact rational root. Unequal bounds are an open isolating interval containing exactly one distinct root.

Examples

```
polynomial_roots(rational_polynomial(c(-2L, 0L, 1L)), 0L, 2L)
```

proper_subset_margins *Proper Subset Margins of an Exact Binary Mass*

Description

Proper Subset Margins of an Exact Binary Mass

Usage

```
proper_subset_margins(mass)
```

Arguments

mass	An object from binary_mass().

Value

A list of exact marginal tables for every proper coordinate subset, including the empty subset. Coordinates in names use R's one based indexing.

Examples

```
proper_subset_margins(binary_mass(rep("1/4", 4)))
```

---

rational_polynomial *Rational Polynomials in Ascending Coefficient Order*

Description

Rational Polynomials in Ascending Coefficient Order

Usage

```
rational_polynomial(coefficients = 0L)
```

Arguments

coefficients	Exact inputs, in ascending powers of the variable.

Details

Integer and character constants can be used directly in polynomial arithmetic. Wrap a bigq constant in rational_polynomial() before mixed arithmetic to avoid conflicting S3 arithmetic methods from package 'gmp'.

Value

An fs_polynomial object. Arithmetic +, -, *, and nonnegative integer powers preserve exact rational coefficients.

Examples

```
x <- rational_polynomial(c(0L, 1L))
p <- (x - "1/3")^2
polynomial_evaluate(p, "1/3")
```

risk_constraint *A Polynomial Error Constraint*

Description

A Polynomial Error Constraint

Usage

```
risk_constraint(name, kind, polynomial, upper_bound)
```

Arguments

name	Unique, nonempty name.
kind	One of "false_negative", "false_positive", or "other".
polynomial	Exact risk polynomial in lambda.
upper_bound	Exact tolerance in [0,1].

Details

Risk polynomials are verified to stay in [0,1] over the complete interval by coordinate_feasibility(). Generic "other" constraints omit this range requirement.

Value

An fs_constraint object representing polynomial <= upper_bound.

Examples

```
risk_constraint("fp", "false_positive", rational_polynomial(c("1/2", 1L)), "1/2")
```

size_bounds *Inner and Outer Bounds over a Finite State Set*

Description

Inner and Outer Bounds over a Finite State Set

Usage

```
size_bounds(risks, beta = 0.2, gamma = 0.2)
```

Arguments

risks	Data frame with n, state, false_negative, and false_positive. Every state must supply every candidate exactly once.
beta, gamma	Error tolerances as in feasible_sizes().

Details

The inner bound is the intersection and the outer bound the union of the state specific feasible sets. A finite sample of nuisance values does not certify bounds over a continuous identified set. Use `coordinate_feasibility()` for the supported exact polynomial calculation.

Value

An object of class `fs_size_bounds` with `inner`, `outer`, `unidentified`, `infeasible`, `by_state`, and the complete risk table.

Examples

```
r <- rbind(transform(normal_risks(1:100), state = "a"),
           transform(normal_risks(1:100, sigma = 1.2), state = "b"))
size_bounds(r)
```

Index

binary_cells, 3
binary_cells(), *4*
binary_compatibility, 3
binary_compatibility(), *5, 6*
binary_mass, 4
binary_mass(), *3*, *16*

classify_rule_family, 4
classify_rule_family(), *2*
compatible_distribution, 5
coordinate_feasibility, 6
coordinate_feasibility(), *2*, *17, 18*

directional_evidence, 7

exact_rational, 7

feasible_sizes, 8
feasible_sizes(), *7*, *17*
feasiblesize *(*feasiblesize-package*)*, 2
feasiblesize-package, 2

graphics::plot(), *14*

normal_risks, 9
normal_size_bounds, 10
normal_size_bounds(), *2*, *14*

operating_polynomial, 11

paired_binary_risks, 12
paired_discordance_bounds, 13
plot.fs_size_bounds, 13
polynomial_evaluate, 14
polynomial_roots, 15
proper_subset_margins, 15

rational_polynomial, 16
rational_polynomial(), *14*
risk_constraint, 17
risk_constraint(), *6*, *11*

size_bounds, 17
size_bounds(), *2*, *14*