Package ‘hypothesize’

December 11, 2025

Title A Consistent API for Hypothesis Testing

Version 0.10.0

Description Provides a consistent API for hypothesis testing built on principles from 'Structure and Interpretation of Computer Programs': data abstraction, closure (combining tests yields tests), and higher-order functions (transforming tests). Implements z-tests, Wald tests, likelihood ratio tests, Fisher's method for combining p-values, and multiple testing corrections. Designed for use by other packages that want to wrap their hypothesis tests in a consistent interface.

Encoding UTF-8

Maintainer Alexander Towell <lex@metafunctor.com>

License MIT + file LICENSE

ByteCompile true

Imports stats

URL https://github.com/queelius/hypothesize,
https://queelius.github.io/hypothesize/

BugReports https://github.com/queelius/hypothesize/issues

Suggests testthat (>= 3.0.0), rmarkdown, knitr

VignetteBuilder knitr

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

Author Alexander Towell [aut, cre] (ORCID:
<https://orcid.org/0000-0001-6443-9897>)

Repository CRAN

Date/Publication 2025-12-11 13:20:02 UTC

Contents

`adjust_pval` *Adjust P-Value for Multiple Testing*

Description

Applies a multiple testing correction to a hypothesis test or vector of tests, returning adjusted test object(s).

Usage

```
adjust_pval(x, method = "bonferroni", n = NULL)
```

Arguments

`x`	A `hypothesis_test` object, or a list of such objects.
`method`	Character. Adjustment method (see Details). Default is `"bonferroni"`.
`n`	Integer. Total number of tests in the family. If `x` is a list, defaults to `length(x)`. For a single test, this must be specified.

Details

When performing multiple hypothesis tests, the probability of at least one false positive (Type I error) increases. Multiple testing corrections adjust p-values to control error rates across the family of tests.

This function demonstrates the **higher-order function** pattern: it takes a hypothesis test as input and returns a transformed hypothesis test as output. The adjusted test retains all original properties but with a corrected p-value.

Value

For a single test: a `hypothesis_test` object of subclass `adjusted_test` with the adjusted p-value. For a list of tests: a list of adjusted test objects.

The returned object contains:

stat Original test statistic (unchanged)

p.value Adjusted p-value

dof Original degrees of freedom (unchanged)

adjustment_method The method used

original_pval The unadjusted p-value

n_tests Number of tests in the family

Available Methods

The `method` parameter accepts any method supported by `stats::p.adjust()`:

`"bonferroni"` Multiplies p-values by n. Controls family-wise error rate (FWER). Conservative.

`"holm"` Step-down Bonferroni. Controls FWER. Less conservative than Bonferroni while maintaining strong control.

`"BH"` **or** `"fdr"` Benjamini-Hochberg procedure. Controls false discovery rate (FDR). More powerful for large-scale testing.

`"hochberg"` Step-up procedure. Controls FWER under independence.

`"hommel"` More powerful than Hochberg but computationally intensive.

`"BY"` Benjamini-Yekutieli. Controls FDR under arbitrary dependence.

`"none"` No adjustment (identity transformation).

Higher-Order Function Pattern

This function exemplifies transforming hypothesis tests:

```
adjust_pval : hypothesis_test -> hypothesis_test
```

The output can be used with all standard generics (`pval()`, `test_stat()`, `is_significant_at()`, etc.) and can be further composed.

See Also

`stats::p.adjust()` for the underlying adjustment, `fisher_combine()` for combining (not adjusting) p-values

Examples

```
# Single test adjustment (must specify n)
w <- wald_test(estimate = 2.0, se = 0.8)
pval(w)  # Original p-value

w_adj <- adjust_pval(w, method = "bonferroni", n = 10)
pval(w_adj)  # Adjusted (multiplied by 10, capped at 1)
w_adj$original_pval  # Can still access original

# Adjusting multiple tests at once
tests <- list(
  wald_test(estimate = 2.5, se = 0.8),
  wald_test(estimate = 1.2, se = 0.5),
  wald_test(estimate = 0.8, se = 0.9)
)

# BH (FDR) correction - n is inferred from list length
adjusted <- adjust_pval(tests, method = "BH")
sapply(adjusted, pval)  # Adjusted p-values

# Compare methods
sapply(tests, pval)  # Original
sapply(adjust_pval(tests, method = "bonferroni"), pval)  # Conservative
sapply(adjust_pval(tests, method = "BH"), pval)  # Less conservative
```

---

confint.hypothesis_test

Confidence Interval from Hypothesis Test (Duality)

Description

Extracts a confidence interval from a hypothesis test object, exploiting the fundamental duality between hypothesis tests and confidence intervals.

Usage

```
## S3 method for class 'hypothesis_test'
confint(object, parm = NULL, level = 0.95, ...)

## S3 method for class 'wald_test'
confint(object, parm = NULL, level = 0.95, ...)

## S3 method for class 'z_test'
confint(object, parm = NULL, level = 0.95, ...)
```

Arguments

`object`	A `hypothesis_test` object.
`parm`	Ignored (for compatibility with generic).
`level`	Numeric. Confidence level (default 0.95).
`...`	Additional arguments (ignored).

Details

Hypothesis tests and confidence intervals are two views of the same underlying inference. For a test of $H_0 : \theta = \theta_0$ at level α, the $(1 - \alpha)$ confidence interval contains exactly those values of θ_0 that would **not** be rejected.

This duality means:

- A 95% CI contains all values where the two-sided test has p > 0.05
- The CI boundary is where p = 0.05 exactly
- Inverting a test "inverts" it into a confidence set

Value

A named numeric vector with elements `lower` and `upper`.

Available Methods

Confidence intervals are currently implemented for:

- `wald_test`: Uses $\hat{\theta} \pm z_{\alpha/2} \cdot SE$
- `z_test`: Uses $\bar{x} \pm z_{\alpha/2} \cdot \sigma/\sqrt{n}$

Tests without stored estimates (like `lrt` or `fisher_combined_test`) cannot produce confidence intervals directly.

Examples

```
# Wald test stores estimate and SE, so CI is available
w <- wald_test(estimate = 2.5, se = 0.8)
confint(w)              # 95% CI
confint(w, level = 0.99) # 99% CI

# The duality: 2.5 is in the CI, and testing H0: theta = 2.5
# would give p = 1 (not rejected)
wald_test(estimate = 2.5, se = 0.8, null_value = 2.5)

# z-test also supports confint
z <- z_test(rnorm(50, mean = 10, sd = 2), mu0 = 9, sigma = 2)
confint(z)
```

dof — *Generic method for extracting the degrees of freedom from a hypothesis test*

Description

Generic method for extracting the degrees of freedom from a hypothesis test

Usage

```
dof(x, ...)
```

Arguments

`x`	a hypothesis test object
`...`	additional arguments to pass into the method

Value

degrees of freedom

dof.hypothesis_test — *Degrees of freedom method for hypothesis tests*

Description

Degrees of freedom method for hypothesis tests

Usage

```
## S3 method for class 'hypothesis_test'
dof(x, ...)
```

Arguments

`x`	a hypothesis test
`...`	additional arguments

Value

degrees of freedom

`fisher_combine` *Combine Independent P-Values (Fisher's Method)*

Description

Combines p-values from independent hypothesis tests into a single omnibus test using Fisher's method.

Usage

```
fisher_combine(...)
```

Arguments

`...`	`hypothesis_test` objects to combine, or numeric p-values. All tests must be independent.

Details

Fisher's method is a meta-analytic technique for combining evidence from multiple independent tests of the same hypothesis (or related hypotheses). It demonstrates a key principle: **combining hypothesis tests yields a hypothesis test** (the closure property).

Given k independent p-values $p_1, \ldots, p_k$, Fisher's statistic is:

$$X^2 = -2 \sum_{i=1}^{k} \log(p_i)$$

Under the global null hypothesis (all individual nulls are true), this follows a chi-squared distribution with $2k$ degrees of freedom.

Value

A `hypothesis_test` object of subclass `fisher_combined_test` containing:

stat Fisher's chi-squared statistic $-2 \sum \log(p_i)$

p.value P-value from χ^2_{2k} distribution

dof Degrees of freedom ($2k$)

n_tests Number of tests combined

component_pvals Vector of the individual p-values

Why It Works

If p_i is a valid p-value under H_0, then $p_i \sim U(0, 1)$. Therefore $-2 \log(p_i) \sim \chi^2_2$. The sum of independent chi-squared random variables is also chi-squared with summed degrees of freedom, giving $X^2 \sim \chi^2_{2k}$.

Interpretation

A significant combined p-value indicates that **at least one** of the individual null hypotheses is likely false, but does not identify which one(s). Fisher's method is sensitive to any deviation from the global null, making it powerful when effects exist but liberal when assumptions are violated.

Closure Property (SICP Principle)

This function exemplifies the closure property from SICP: the operation of combining hypothesis tests produces another hypothesis test. The result can be further combined, adjusted, or analyzed using the same generic methods (`pval()`, `test_stat()`, `is_significant_at()`, etc.).

See Also

`adjust_pval()` for multiple testing correction (different goal)

Examples

```
# Scenario: Three independent studies test the same drug effect
# Study 1: p = 0.08 (trend, not significant)
# Study 2: p = 0.12 (not significant)
# Study 3: p = 0.04 (significant at 0.05)

# Combine using raw p-values
combined <- fisher_combine(0.08, 0.12, 0.04)
combined
is_significant_at(combined, 0.05)  # Stronger evidence together

# Or combine hypothesis_test objects directly
t1 <- wald_test(estimate = 1.5, se = 0.9)
t2 <- wald_test(estimate = 0.8, se = 0.5)
t3 <- z_test(rnorm(30, mean = 0.3), mu0 = 0, sigma = 1)

fisher_combine(t1, t2, t3)

# The result is itself a hypothesis_test, so it composes
# (though combining non-independent tests is invalid)
```

`hypothesis_test` *Create a Hypothesis Test Object*

Description

Constructs a hypothesis test object that implements the `hypothesize` API. This is the base constructor used by specific test functions like `lrt()`, `wald_test()`, and `z_test()`.

Usage

```
hypothesis_test(stat, p.value, dof, superclasses = NULL, ...)
```

Arguments

`stat`	Numeric. The test statistic.
`p.value`	Numeric. The p-value (probability of observing a test statistic as extreme as `stat` under the null hypothesis).
`dof`	Numeric. Degrees of freedom. Use `Inf` for tests based on the normal distribution.
`superclasses`	Character vector. Additional S3 classes to prepend, creating a subclass of `hypothesis_test`.
`...`	Additional named arguments stored in the object for introspection (e.g., input data, null hypothesis value).

Details

The `hypothesis_test` object is the fundamental data abstraction in this package. It represents the result of a statistical hypothesis test and provides a consistent interface for extracting results.

This design follows the principle of **data abstraction**: the internal representation (a list) is hidden behind accessor functions (`pval()`, `test_stat()`, `dof()`, `is_significant_at()`).

Value

An S3 object of class `hypothesis_test` (and any `superclasses`), which is a list containing at least `stat`, `p.value`, `dof`, plus any additional arguments passed via `...`.

Extending the Package

To create a new type of hypothesis test:

1. Create a constructor function that computes the test statistic and p-value.
2. Call `hypothesis_test()` with appropriate `superclasses`.
3. The new test automatically inherits all generic methods.

Example:

```
my_test <- function(data, null_value) {
  stat <- compute_statistic(data, null_value)
  p.value <- compute_pvalue(stat)
  hypothesis_test(
    stat = stat, p.value = p.value, dof = length(data) - 1,
    superclasses = "my_test",
    data = data, null_value = null_value
  )
}
```

See Also

`lrt()`, `wald_test()`, `z_test()` for specific test constructors; `pval()`, `test_stat()`, `dof()`, `is_significant_at()` for accessors

Examples

```
# Direct construction (usually use specific constructors instead)
test <- hypothesis_test(stat = 1.96, p.value = 0.05, dof = 1)
test

# Extract components using the API
pval(test)
test_stat(test)
dof(test)
is_significant_at(test, 0.05)

# Create a custom test type
custom <- hypothesis_test(
  stat = 2.5, p.value = 0.01, dof = 10,
  superclasses = "custom_test",
  method = "bootstrap", n_replicates = 1000
)
class(custom)  # c("custom_test", "hypothesis_test")
custom$method  # "bootstrap"
```

---

is_significant_at — *Generic method for checking if a hypothesis test is significant at a given significance level.*

Description

Generic method for checking if a hypothesis test is significant at a given significance level.

Usage

```
is_significant_at(x, alpha, ...)
```

Arguments

x	a hypothesis test object
alpha	significance level
...	additional arguments passed to methods

Value

logical indicating whether the test is significant at the given significance level alpha (e.g., 0.05) or not.

`is_significant_at.hypothesis_test` *Significance test for the hypothesis_test class.*

Description

Significance test for the hypothesis_test class.

Usage

```
## S3 method for class 'hypothesis_test'
is_significant_at(x, alpha, ...)
```

Arguments

`x`	a hypothesis test object
`alpha`	significance level
`...`	additional arguments (ignored)

Value

logical indicating whether the test is significant at the given significance level alpha (e.g., 0.05) or not.

`lrt` *Likelihood Ratio Test*

Description

Computes the likelihood ratio test (LRT) statistic and p-value for comparing nested models.

Usage

```
lrt(null_loglik, alt_loglik, dof)
```

Arguments

`null_loglik`	Numeric. The maximized log-likelihood under the null (simpler) model.
`alt_loglik`	Numeric. The maximized log-likelihood under the alternative (more complex) model.
`dof`	Positive integer. Degrees of freedom, typically the difference in the number of free parameters between models.

Details

The likelihood ratio test is a fundamental method for comparing nested statistical models. Given a null model M_0 (simpler, fewer parameters) nested within an alternative model M_1 (more complex), the LRT tests whether the additional complexity of M_1 is justified by the data.

The test statistic is:

$$\Lambda = -2\left(\ell_0 - \ell_1\right) = -2\log\frac{L_0}{L_1}$$

where ℓ_0 and ℓ_1 are the maximized log-likelihoods under the null and alternative models, respectively.

Under H_0 and regularity conditions, Λ is asymptotically chi-squared distributed with degrees of freedom equal to the difference in the number of free parameters between models.

Value

A `hypothesis_test` object of subclass `likelihood_ratio_test` containing:

stat The LRT statistic $\Lambda = -2(\ell_0 - \ell_1)$

p.value P-value from chi-squared distribution with `dof` degrees of freedom

dof The degrees of freedom

null_loglik The input null model log-likelihood

alt_loglik The input alternative model log-likelihood

Assumptions

1. The null model must be nested within the alternative model (i.e., obtainable by constraining parameters of the alternative).
2. Both likelihoods must be computed from the same dataset.
3. Standard regularity conditions for asymptotic chi-squared distribution must hold (true parameter not on boundary, etc.).

Relationship to Other Tests

The LRT is one of the "holy trinity" of likelihood-based tests, alongside the Wald test (`wald_test()`) and the score (Lagrange multiplier) test. All three are asymptotically equivalent under H_0, but the LRT is often preferred because it is invariant to reparameterization.

See Also

`wald_test()` for testing individual parameters

Examples

```
# Comparing nested regression models
# Null model: y ~ x1 (log-likelihood = -150)
# Alt model:  y ~ x1 + x2 + x3 (log-likelihood = -140)
# Difference: 3 additional parameters

test <- lrt(null_loglik = -150, alt_loglik = -140, dof = 3)
test

# Is the more complex model significantly better?
is_significant_at(test, 0.05)

# Extract the test statistic (should be 20)
test_stat(test)

# Access stored inputs for inspection
test$null_loglik
test$alt_loglik
```

---

print.hypothesis_test *Print method for hypothesis tests*

Description

Print method for hypothesis tests

Usage

```
## S3 method for class 'hypothesis_test'
print(x, ...)
```

Arguments

x	a hypothesis test
...	additional arguments

Value

a string representation of the hypothesis test

pval — *Generic method for extracting the p-value from a hypothesis test*

Description

Generic method for extracting the p-value from a hypothesis test

Usage

```
pval(x, ...)
```

Arguments

`x`	a hypothesis test object
`...`	additional arguments to pass into the method

Value

p-value

pval.hypothesis_test — *p-value method for hypothesis tests*

Description

p-value method for hypothesis tests

Usage

```
## S3 method for class 'hypothesis_test'
pval(x, ...)
```

Arguments

`x`	a hypothesis test
`...`	additional arguments

Value

p-value

test_stat — *Generic method for extracting the test statistic from a hypothesis test*

Description

Generic method for extracting the test statistic from a hypothesis test

Usage

```
test_stat(x, ...)
```

Arguments

x	a hypothesis test object
...	additional arguments to pass into the method

Value

test statistic

test_stat.hypothesis_test — *Test statistic method for hypothesis tests*

Description

Test statistic method for hypothesis tests

Usage

```
## S3 method for class 'hypothesis_test'
test_stat(x, ...)
```

Arguments

x	a hypothesis test
...	additional arguments

Value

test statistic

`wald_test` *Wald Test*

Description

Computes the Wald test statistic and p-value for testing whether a parameter equals a hypothesized value.

Usage

```
wald_test(estimate, se, null_value = 0)
```

Arguments

`estimate`	Numeric. The estimated parameter value $\hat{\theta}$.
`se`	Numeric. The standard error of the estimate, $SE(\hat{\theta})$.
`null_value`	Numeric. The hypothesized value θ_0 under the null hypothesis. Default is 0.

Details

The Wald test is a fundamental tool in statistical inference, used to test the null hypothesis $H_0 : \theta = \theta_0$ against the alternative $H_1 : \theta \neq \theta_0$.

The test is based on the asymptotic normality of maximum likelihood estimators. Under regularity conditions, if $\hat{\theta}$ is the MLE with standard error $SE(\hat{\theta})$, then:

$$z = \frac{\hat{\theta} - \theta_0}{SE(\hat{\theta})} \sim N(0, 1)$$

The Wald statistic is typically reported as $W = z^2$, which follows a chi-squared distribution with 1 degree of freedom under H_0. This formulation generalizes naturally to multivariate parameters.

The p-value is computed as $P(\chi^2_1 \geq W)$, giving a two-sided test. The z-score is stored in the returned object for reference.

Value

A `hypothesis_test` object of subclass `wald_test` containing:

stat The Wald statistic $W = z^2$

p.value Two-sided p-value from chi-squared(1) distribution

dof Degrees of freedom (always 1 for univariate Wald test)

z The z-score $(\hat{\theta} - \theta_0)/SE$

estimate The input estimate

se The input standard error

null_value The input null hypothesis value

Relationship to Other Tests

The Wald test is one of the "holy trinity" of likelihood-based tests, alongside the likelihood ratio test (`lrt()`) and the score test. For large samples, all three are asymptotically equivalent, but they can differ substantially in finite samples.

See Also

`lrt()` for likelihood ratio tests, `z_test()` for testing means

Examples

```
# Test whether a regression coefficient differs from zero
# Suppose we estimated beta = 2.5 with SE = 0.8
w <- wald_test(estimate = 2.5, se = 0.8, null_value = 0)
w

# Extract components
test_stat(w)       # Wald statistic (chi-squared)
w$z                # z-score
pval(w)            # p-value
is_significant_at(w, 0.05)

# Test against a non-zero null
# H0: theta = 2 vs H1: theta != 2
wald_test(estimate = 2.5, se = 0.8, null_value = 2)
```

---

z_test *One-Sample Z-Test*

Description

Tests whether a population mean equals a hypothesized value when the population standard deviation is known.

Usage

```
z_test(x, mu0 = 0, sigma, alternative = c("two.sided", "less", "greater"))
```

Arguments

`x`	Numeric vector. The sample data.
`mu0`	Numeric. The hypothesized population mean under H_0. Default is 0.
`sigma`	Numeric. The known population standard deviation.
`alternative`	Character. Type of alternative hypothesis: `"two.sided"` (default), `"less"`, or `"greater"`.

Details

The z-test is one of the simplest and most fundamental hypothesis tests. It tests $H_0 : \mu = \mu_0$ against various alternatives when the population standard deviation σ is known.

Given a sample $x_1, \ldots, x_n$, the test statistic is:

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

Under H_0, this follows a standard normal distribution. The p-value depends on the alternative hypothesis:

- Two-sided ($H_1 : \mu \neq \mu_0$): $2 \cdot P(Z > |z|)$
- Less ($H_1 : \mu < \mu_0$): $P(Z < z)$
- Greater ($H_1 : \mu > \mu_0$): $P(Z > z)$

Value

A `hypothesis_test` object of subclass `z_test` containing:

stat The z-statistic

p.value The p-value for the specified alternative

dof Degrees of freedom (Inf for normal distribution)

alternative The alternative hypothesis used

null_value The hypothesized mean μ_0

estimate The sample mean $\bar{x}$

sigma The known population standard deviation

n The sample size

When to Use

The z-test requires knowing the population standard deviation, which is rare in practice. When σ is unknown and estimated from data, use a t-test instead. The z-test is primarily pedagogical, illustrating the logic of hypothesis testing in its simplest form.

Relationship to Wald Test

The z-test is a special case of the Wald test (`wald_test()`) where the parameter is a mean and the standard error is $\sigma/\sqrt{n}$. The Wald test generalizes this to any asymptotically normal estimator.

See Also

`wald_test()` for the general case with estimated standard errors

Examples

```
# A light bulb manufacturer claims bulbs last 1000 hours on average.
# We test 50 bulbs and know from historical data that sigma = 100 hours.
lifetimes <- c(980, 1020, 950, 1010, 990, 1005, 970, 1030, 985, 995,
               1000, 1015, 960, 1025, 975, 1008, 992, 1012, 988, 1002,
               978, 1018, 965, 1022, 982, 1005, 995, 1010, 972, 1028,
               990, 1000, 985, 1015, 968, 1020, 980, 1008, 992, 1012,
               975, 1018, 962, 1025, 985, 1002, 988, 1010, 978, 1020)

# Two-sided test: H0: mu = 1000 vs H1: mu != 1000
z_test(lifetimes, mu0 = 1000, sigma = 100)

# One-sided test: are bulbs lasting less than claimed?
z_test(lifetimes, mu0 = 1000, sigma = 100, alternative = "less")
```

Index