Package: femtograd (via r-universe)

May 14, 2026

Title Automatic Differentiation for Statistical Computing

Version 0.3.1

Maintainer Alexander Towell <lex@metafunctor.com>

Description Provides automatic differentiation via reverse-mode AD (backpropagation) for first-order gradients and forward-over-reverse AD for Hessian computation. Includes log-likelihood functions for exponential family distributions (normal, exponential, Poisson, binomial, gamma, beta, negative binomial, Weibull, Pareto), MLE optimization via fit() with base R generics (coef, vcov, confint, logLik, AIC/BIC), hypothesis testing (likelihood ratio, Wald), profile likelihood, bootstrap inference, and model diagnostics. Designed for pedagogy and modern statistics rather than large-scale ML.

License GPL (>= 3)

URL https://queelius.github.io/femtograd/, https://github.com/queelius/femtograd, https://queelius.r-universe.dev/femtograd

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

X-schema.org-keywords automatic-differentiation, backpropagation, maximum-likelihood, hessian, statistics, computational-graph, inference, pedagogy

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports R6

Suggests testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://queelius.r-universe.dev

Date/Publication 2026-04-14 06:34:18 UTC

RemoteUrl https://github.com/queelius/femtograd

RemoteRef HEAD

RemoteSha 50193c0c1dcbd1ffc0d1c268500616d6a5a3f2bc

Contents

-.value . . . 4
*.value . . . 5
/.value . . . 5
^.value . . . 6
+.value . . . 6
abs.value . . . 7
anova.femtofit . . . 7
backward . . . 8
backward.default . . . 8
backward.value . . . 9
bfgs . . . 9
bootstrap . . . 11
bootstrap_fit . . . 11
bounded . . . 12
check_convergence . . . 13
check_hessian . . . 14
compare . . . 16
confint.bootstrap_result . . . 17
confint_mle . . . 18
confint_profile . . . 18
cos.value . . . 19
diagnostics . . . 20
digamma.value . . . 21
distributions . . . 21
div_safe . . . 21
dof . . . 22
dual . . . 22
dual_num . . . 24
exp.value . . . 24
exp_safe . . . 25
femtofit . . . 25
find_mle . . . 27
fisher_information . . . 28
fisher_scoring . . . 29
fit . . . 29
fitting . . . 31
get_data . . . 31
grad . . . 33
grad.default . . . 33
grad.value . . . 34

gradient . . . 34
gradient_ascent . . . 35
gradient_descent . . . 36
hessian . . . 37
hypothesis_tests . . . 38
inv_bounded . . . 38
inv_positive . . . 39
inv_probability . . . 39
inverse_transforms . . . 39
is_dual . . . 40
is_significant_at . . . 40
is_value . . . 41
lbfgs . . . 41
lgamma.value . . . 42
line_search . . . 43
log.value . . . 44
log_safe . . . 44
log_sigmoid . . . 45
log1p.value . . . 45
log1p_safe . . . 46
logit . . . 46
loglik_bernoulli . . . 47
loglik_beta . . . 47
loglik_binomial . . . 48
loglik_exponential . . . 48
loglik_gamma . . . 49
loglik_logistic . . . 50
loglik_negbinom . . . 50
loglik_normal . . . 51
loglik_pareto . . . 51
loglik_poisson . . . 52
loglik_weibull . . . 53
logsumexp . . . 54
lower_bounded . . . 55
lrt . . . 56
mean.value . . . 57
newton_raphson . . . 58
observed_info . . . 59
optimization . . . 60
plot.profile_likelihood . . . 60
positive . . . 60
predict.femtofit . . . 61
primal . . . 62
print.anova.femtofit . . . 63
print.bootstrap_result . . . 63
print.hessian_check . . . 64
print.likelihood_ratio_test . . . 64
print.model_comparison . . . 65

print.profile_likelihood . . . 65
print.value . . . 66
print.wald_test . . . 66
probability . . . 67
profile_likelihood . . . 68
profile_loglik . . . 68
pval . . . 69
relu . . . 70
se . . . 70
se_reliable . . . 71
sigmoid . . . 72
sigmoid_stable . . . 72
sin.value . . . 73
softmax . . . 73
softplus . . . 74
sqrt.value . . . 74
stability . . . 74
std_errors . . . 75
sum.dual . . . 75
sum.value . . . 76
summary.bootstrap_result . . . 76
tangent . . . 77
tanh.value . . . 77
test_stat . . . 78
transforms . . . 78
trigamma.value . . . 79
upper_bounded . . . 79
val . . . 80
value . . . 81
vcov_matrix . . . 82
wald_test . . . 82
zero_grad . . . 84

Index **86**

`-.value` *Subtraction for value objects*

Description

Element-wise subtraction or unary negation. Supports broadcasting when one operand is a 1x1 scalar matrix.

Usage

```
## S3 method for class 'value'
x - y = NULL
```

Arguments

x	A value object or numeric (minuend)
y	A value object or numeric (subtrahend), or NULL for unary negation

Value

A new value object representing x - y or -x

---

*.value *Multiplication for value objects*

Description

Element-wise multiplication of two value objects or a value and numeric. Supports broadcasting when one operand is a 1x1 scalar matrix.

Usage

```
## S3 method for class 'value'
e1 * e2
```

Arguments

e1	A value object or numeric
e2	A value object or numeric

Value

A new value object representing the element-wise product

/.value *Division for value objects*

Description

Element-wise division. Supports broadcasting when one operand is a 1x1 scalar matrix.

Usage

```
## S3 method for class 'value'
x / y
```

Arguments

x	A value object or numeric (numerator)
y	A value object or numeric (denominator)

Value

A new value object representing x / y

`^.value` *Power operation for value objects.*

Description

Element-wise power operation. Supports broadcasting when one operand is a 1x1 scalar matrix.

Usage

```
## S3 method for class 'value'
b ^ e
```

Arguments

`b`	A value object or numeric (base)
`e`	A value object or numeric (exponent)

Value

A new value object representing the element-wise power b^e

`+.value` *Addition for value objects*

Description

Element-wise addition of two value objects or a value and numeric. Both operands are converted to matrices internally. Supports broadcasting when one operand is a 1x1 scalar matrix.

Usage

```
## S3 method for class 'value'
e1 + e2
```

Arguments

`e1`	A value object or numeric
`e2`	A value object or numeric

Value

A new value object representing the element-wise sum

`abs.value` *Absolute value for value objects*

Description

Element-wise absolute value.

Usage

```
## S3 method for class 'value'
abs(x)
```

Arguments

`x`	A value object

Value

A new value object representing abs(x)

`anova.femtofit` *Analysis of variance for femtofit models*

Description

Performs likelihood ratio tests comparing nested models. When given a single model, reports the log-likelihood and degrees of freedom. When given multiple models, performs sequential likelihood ratio tests.

Usage

```
## S3 method for class 'femtofit'
anova(object, ...)
```

Arguments

`object`	A `femtofit` object.
`...`	Additional `femtofit` objects to compare (must be nested).

Details

Models should be provided in order from simplest (fewest parameters) to most complex. The likelihood ratio test compares each model to the previous one.

Value

An `anova.femtofit` object containing the comparison table.

See Also

compare for AIC-based comparison, lrt for explicit likelihood ratio tests

Examples

```
## Not run:
# Compare nested models
m1 <- fit(function(mu) loglik_normal(mu, 1, x), c(mu = 0))
m2 <- fit(function(mu, sigma) loglik_normal(mu, sigma, x), c(mu = 0, sigma = 1))

anova(m1, m2)  # LRT comparing m1 vs m2

## End(Not run)
```

`backward` *Generic function for the Backward pass for automatic differentiation (finds the gradient of every sub-node in the computational graph with respect to* `e`*). In other words, it is responsible for computing the gradient with respect to* `e`*.*

Description

This generic function should be implemented for specific classes. We provide an implementation for `value` objects.

Usage

```
backward(e, ...)
```

Arguments

`e`	An object for which the backward pass should be performed
`...`	additional arguments to pass

`backward.default` *Default implementation does not propagate gradients. For instance, if we have a constant, then the partial of the constant is not meaningful.*

Description

Default implementation does not propagate gradients. For instance, if we have a constant, then the partial of the constant is not meaningful.

Usage

```
## Default S3 method:
backward(e, ...)
```

Arguments

e	An object for which the backward pass should be performed
...	pass additional arguments

---

backward.value *Backward pass for value objects*

Description

Performs the backward pass (gradient computation) for a value object in the computational graph with automatic differentiation. Initializes the gradient of the output node to a matrix of ones (same shape as data).

Usage

```
## S3 method for class 'value'
backward(e, ...)
```

Arguments

e	A value object for which the backward pass should be performed
...	pass additional arguments

---

bfgs *BFGS quasi-Newton optimizer*

Description

Finds the optimum using the BFGS algorithm, which approximates the inverse Hessian using gradient information. More efficient than Newton-Raphson when Hessian computation is expensive.

Usage

```
bfgs(
  objective_fn,
  params,
  max_iter = 1000,
  tol = 1e-06,
  maximize = FALSE,
  line_search_fn = NULL,
  verbose = 0
)
```

Arguments

`objective_fn`	Function taking list of value parameters, returns scalar
`params`	List of value objects (initial parameter values)
`max_iter`	Maximum iterations, default 1000
`tol`	Convergence tolerance on gradient norm, default 1e-6
`maximize`	If TRUE, maximize; if FALSE (default), minimize
`line_search_fn`	Line search function (default: backtracking)
`verbose`	Print progress every N iterations (0 for silent)

Details

BFGS maintains an approximation B to the inverse Hessian, updated as: B_k+1 = (I - ρsy') B_k (I - ρys') + ρss' where s = x_k+1 - x_k, y = g_k+1 - g_k, ρ = 1/(y'*s)

The search direction is d = -B*g, followed by line search.

Value

A list containing:

`params`	List of value objects at optimum
`value`	Objective function value at optimum
`gradient`	Gradient at optimum
`inv_hessian`	Approximate inverse Hessian
`iterations`	Number of iterations performed
`converged`	TRUE if gradient norm < tol

Examples

```
## Not run:
# Minimize Rosenbrock function
rosenbrock <- function(p) {
  x <- p[[1]]; y <- p[[2]]
  (1 - x)^2 + 100 * (y - x^2)^2
}
result <- bfgs(rosenbrock, list(val(-1), val(-1)))
sapply(result$params, get_data)  # Should be close to c(1, 1)

## End(Not run)
```

`bootstrap` *Bootstrap Inference*

Description

Functions for bootstrap-based inference on fitted models. Bootstrap methods provide non-parametric standard errors and confidence intervals without relying on asymptotic normality.

`bootstrap_fit` *Bootstrap standard errors and confidence intervals*

Description

Performs bootstrap resampling to estimate the sampling distribution of parameter estimates.

Usage

```
bootstrap_fit(loglik_maker, data, params, n_boot = 500, progress = TRUE, ...)
```

Arguments

`loglik_maker`	A function that takes data and returns a log-likelihood function suitable for `fit()`. See examples.
`data`	The data (vector, matrix, or data frame).
`params`	Initial parameter values for `fit()`.
`n_boot`	Number of bootstrap replications (default 500).
`progress`	Logical; show progress messages (default TRUE).
`...`	Additional arguments passed to `fit()`.

Details

The `loglik_maker` function should accept data and return a log-likelihood function that can be passed to `fit()`. This design allows bootstrap to work with any model specification.

Value

A `bootstrap_result` object containing:

estimates Matrix of bootstrap estimates (n_boot x n_params)

original Original parameter estimates

se Bootstrap standard errors

bias Estimated bias

Examples

```
## Not run:
set.seed(42)
x <- rnorm(50, mean = 5, sd = 2)

# Define a function that creates the loglik function from data
make_loglik <- function(data) {
  function(mu, log_sigma) {
    loglik_normal(mu, exp(log_sigma), data)
  }
}

# Bootstrap
boot <- bootstrap_fit(
  make_loglik,
  data = x,
  params = c(mu = 0, log_sigma = 0),
  n_boot = 200
)

# Results
boot$se          # Bootstrap SEs
confint(boot)    # Bootstrap CIs

# Compare with Wald SEs
result <- fit(make_loglik(x), params = c(mu = 0, log_sigma = 0))
se(result)       # Wald SEs

## End(Not run)
```

---

bounded *Transform to bounded interval*

Description

Maps unconstrained values to a bounded interval (lower, upper). Use this for parameters with both lower and upper bounds.

Usage

```
bounded(x, lower, upper)
```

Arguments

`x`	Unconstrained value (scalar, vector, or value object)
`lower`	Lower bound of the interval
`upper`	Upper bound of the interval

Details

The transformation is:

```
bounded(x, a, b) = a + (b - a) * sigmoid(x)
```

This maps (-∞, ∞) to (a, b) smoothly.

Value

Value in (lower, upper)

Examples

```
## Not run:
# Parameter in (0, 10)
raw <- val(0)
param <- bounded(raw, 0, 10)  # sigmoid(0) * 10 = 5
get_data(param)

# Correlation parameter in (-1, 1)
raw_rho <- val(1)
rho <- bounded(raw_rho, -1, 1)

## End(Not run)
```

---

check_convergence *Check convergence diagnostics*

Description

Examines convergence properties of a fitted model.

Usage

```
check_convergence(object, tol = 1e-04)
```

Arguments

`object`	A `femtofit` object.
`tol`	Tolerance for gradient norm (default 1e-4).

Details

At a maximum, the gradient should be (near) zero. Large gradient norms may indicate premature termination or numerical issues.

Value

A `convergence_check` object containing:

converged From the optimizer

gradient_norm Norm of gradient at solution

gradient_ok Is gradient norm below tolerance?

iterations Number of iterations used

loglik Final log-likelihood value

warnings Character vector of any issues detected

Examples

```
## Not run:
set.seed(42)
x <- rnorm(100, mean = 5, sd = 2)

result <- fit(
  function(mu, log_sigma) loglik_normal(mu, exp(log_sigma), x),
  params = c(mu = 0, log_sigma = 0)
)

check <- check_convergence(result)
check

## End(Not run)
```

`check_hessian` *Check Hessian properties*

Description

Examines the Hessian matrix for numerical issues that may indicate problems with the optimization or model specification.

Usage

```
check_hessian(object)
```

Arguments

`object`	A `femtofit` object.

Details

For a proper maximum of the log-likelihood, the Hessian should be negative definite (equivalently, -H should be positive definite).

Common issues:

- **Not negative definite**: May indicate a saddle point rather than maximum
- **High condition number**: Indicates near-singularity, poorly identified parameters
- **Singular matrix**: Parameters are not identifiable from the data

Value

A `hessian_check` object containing:

is_negative_definite Logical: is -H positive definite? (required for MLE)

eigenvalues Eigenvalues of -H (observed information)

condition_number Ratio of largest to smallest eigenvalue

rank Numerical rank of the matrix

is_singular Logical: is the matrix numerically singular?

warnings Character vector of any issues detected

Examples

```
## Not run:
set.seed(42)
x <- rnorm(100, mean = 5, sd = 2)

result <- fit(
  function(mu, log_sigma) loglik_normal(mu, exp(log_sigma), x),
  params = c(mu = 0, log_sigma = 0)
)

check <- check_hessian(result)
check

# Access specific properties
check$is_negative_definite
check$condition_number

## End(Not run)
```

`compare` *Compare multiple fitted models*

Description

Creates a comparison table of AIC, BIC, and other statistics for multiple fitted models. Useful for model selection.

Usage

```
compare(..., names = NULL)
```

Arguments

`...`	One or more `femtofit` objects to compare.
`names`	Optional character vector of model names. If not provided, names are extracted from the call or generated automatically.

Details

Models are ranked by AIC. Delta AIC is calculated relative to the best model (lowest AIC). AIC weights represent the relative likelihood that each model is the best, assuming one of the models is true.

Value

A data.frame with columns for each model's statistics, including log-likelihood, AIC, BIC, delta AIC, and AIC weights.

See Also

`anova.femtofit` for likelihood ratio tests

Examples

```
## Not run:
# Fit competing models
m1 <- fit(function(mu) loglik_normal(mu, 1, x), c(mu = 0))
m2 <- fit(function(mu, sigma) loglik_normal(mu, sigma, x), c(mu = 0, sigma = 1))

# Compare models
compare(m1, m2, names = c("Fixed sigma", "Free sigma"))

## End(Not run)
```

confint.bootstrap_result *Confidence intervals from bootstrap*

Description

Computes confidence intervals using bootstrap methods.

Usage

```
## S3 method for class 'bootstrap_result'
confint(
  object,
  parm = NULL,
  level = 0.95,
  type = c("percentile", "basic", "normal"),
  ...
)
```

Arguments

`object`	A `bootstrap_result` object.
`parm`	Parameter names or indices (default: all).
`level`	Confidence level (default 0.95).
`type`	Type of interval: ”percentile” (default), ”basic”, or ”normal”.
`...`	Additional arguments (ignored).

Details

Available methods:

percentile Uses quantiles of bootstrap distribution directly

basic Uses 2*theta_hat - quantiles (pivot method)

normal Uses bootstrap SE with normal quantiles

Value

Matrix of confidence intervals.

confint_mle *Compute confidence intervals from MLE results*

Description

Compute confidence intervals from MLE results

Usage

```
confint_mle(mle_result, level = 0.95, param_names = NULL)
```

Arguments

mle_result	Result from find_mle
level	Confidence level, default 0.95
param_names	Optional names for parameters

Value

A matrix with columns: estimate, se, lower, upper

confint_profile *Profile confidence intervals*

Description

Computes confidence intervals based on the profile likelihood. These are more accurate than Wald intervals when the likelihood is non-quadratic.

Usage

```
confint_profile(object, parm = NULL, level = 0.95, ...)
```

Arguments

object	A femtofit object.
parm	Parameter name(s) or indices. If NULL, computes for all.
level	Confidence level (default 0.95).
...	Additional arguments passed to profile_loglik.

Details

The profile confidence interval at level 1- is:

$$CI = \{\theta_i : 2(\ell(\hat{\theta}) - pl(\theta_i)) \leq \chi^2_{1,\alpha}\}$$

This is the set of parameter values that would not be rejected by a likelihood ratio test at level .

Value

A matrix with columns for lower and upper bounds.

Examples

```
## Not run:
set.seed(42)
x <- rnorm(100, mean = 5, sd = 2)

result <- fit(
  function(mu, log_sigma) loglik_normal(mu, exp(log_sigma), x),
  params = c(mu = 0, log_sigma = 0)
)

# Profile-based CIs (more accurate for non-quadratic likelihoods)
confint_profile(result)

# Compare with Wald CIs
confint(result)

## End(Not run)
```

---

cos.value *Cosine function for value objects*

Description

Element-wise cosine.

Usage

```
## S3 method for class 'value'
cos(x)
```

Arguments

x	A value object

Value

A new value object representing cos(x)

`diagnostics` *Model Diagnostics*

Description

Functions for checking model fit quality, convergence, and potential numerical issues.

Runs all diagnostic checks on a fitted model.

Usage

```
diagnostics(object, ...)

## S3 method for class 'femtofit'
diagnostics(object, ...)
```

Arguments

`object`	A `femtofit` object.
`...`	Additional arguments passed to individual check functions.

Value

A `model_diagnostics` object containing results from all checks.

Methods (by class)

- `diagnostics(femtofit)`: Diagnostics for femtofit objects

Examples

```
## Not run:
set.seed(42)
x <- rnorm(100, mean = 5, sd = 2)

result <- fit(
  function(mu, log_sigma) loglik_normal(mu, exp(log_sigma), x),
  params = c(mu = 0, log_sigma = 0)
)

diagnostics(result)

## End(Not run)
```

digamma.value *Digamma (psi) function for value objects*

Description

Element-wise digamma: d/dx log(gamma(x)).

Usage

```
## S3 method for class 'value'
digamma(x)
```

Arguments

x	A value object

Value

A new value object representing digamma(x)

distributions *Log-likelihood functions for exponential family distributions*

Description

These functions compute log-likelihoods that can be differentiated using femtograd's automatic differentiation. Each function returns a value object suitable for gradient-based optimization.

div_safe *Safe division (handles division by zero)*

Description

Computes x / y with protection against division by zero.

Usage

```
div_safe(x, y, eps = .Machine$double.eps)
```

Arguments

x	Numerator (value object or numeric)
y	Denominator (value object or numeric)
eps	Minimum absolute value for denominator (default: .Machine$double.eps)

Value

x / sign(y) * max(abs(y), eps)

`dof` *Extract degrees of freedom from hypothesis test*

Description

Extract degrees of freedom from hypothesis test

Usage

```
dof(x, ...)

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

Arguments

`x`	A hypothesis test object
`...`	Additional arguments (ignored)

Value

Degrees of freedom

Methods (by class)

- `dof(hypothesis_test)`: Degrees of freedom for hypothesis tests

`dual` *dual R6 class for forward-mode automatic differentiation*

Description

dual R6 class for forward-mode automatic differentiation

dual R6 class for forward-mode automatic differentiation

Details

Represents a dual number (primal, tangent) for computing directional derivatives via forward-mode AD. When combined with reverse-mode (value), enables Hessian computation via forward-over-reverse.

Forward-mode AD propagates derivatives alongside values during the forward pass. For a function f(x), setting tangent(x) = 1 gives f'(x) in the tangent of the output.

For Hessian computation (forward-over-reverse):

- primal is a `value` object (for reverse-mode gradient)
- tangent is also a `value` object (tracks d/dx of the gradient)
- After backward() on primal, tangent of grad gives Hessian entries

Public fields

`primal` The function value (can be numeric or value object)

`tangent` The directional derivative (can be numeric or value object)

Methods

Public methods:

- `dual$new()`
- `dual$clone()`

Method `new()`: Create a new dual number

Usage:

```
dual$new(primal, tangent = 0)
```

Arguments:

`primal` The primal value (numeric or value object)

`tangent` The tangent (derivative direction), default 0

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
dual$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

dual_num *Create a dual number*

Description

Create a dual number

Usage

```
dual_num(primal, tangent = 0)
```

Arguments

primal	The primal value
tangent	The tangent (directional derivative), default 0

Value

A dual object

exp.value *Exponential function for value objects*

Description

Element-wise exponential.

Usage

```
## S3 method for class 'value'
exp(x)
```

Arguments

x	A value object

Value

A new value object representing exp(x)

exp_safe — *Stable exp function (with overflow protection)*

Description

Computes exp(x) with optional clamping to prevent Inf.

Usage

```
exp_safe(x, max_val = 709)
```

Arguments

x	A value object or numeric
max_val	Maximum value for x to prevent overflow (default: 709)

Details

In double precision, exp(710) overflows to Inf. This function clamps the input to prevent overflow while maintaining correct gradients.

Value

exp(min(x, max_val))

femtofit — *Constructor for femtofit objects*

Description

Creates a fitted model object from MLE results. This is primarily an internal constructor; users typically obtain femtofit objects from the `fit()` function.

Usage

```
femtofit(
  coefficients,
  vcov,
  loglik,
  hessian = NULL,
  gradient = NULL,
  converged = TRUE,
  iterations = NA_integer_,
  nobs = NULL,
  call = NULL,
  predict_fn = NULL
```

```
)

## S3 method for class 'femtofit'
coef(object, ...)

## S3 method for class 'femtofit'
vcov(object, ...)

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

## S3 method for class 'femtofit'
logLik(object, ...)

## S3 method for class 'femtofit'
nobs(object, ...)

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

## S3 method for class 'femtofit'
summary(object, ...)

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

Arguments

`coefficients`	Named numeric vector of parameter estimates.
`vcov`	Variance-covariance matrix (named).
`loglik`	Log-likelihood value at the MLE.
`hessian`	Hessian matrix at the MLE.
`gradient`	Gradient vector at the MLE (should be near zero).
`converged`	Logical indicating convergence.
`iterations`	Number of iterations performed.
`nobs`	Number of observations (optional).
`call`	The original function call (optional).
`predict_fn`	Optional prediction function (optional).
`object`	A `femtofit` object.
`...`	Additional arguments (ignored).
`parm`	Parameter names or indices (default: all parameters).
`level`	Confidence level (default: 0.95).
`x`	A `femtofit` object.

Value

A `femtofit` object.

Methods (by generic)

- `coef(femtofit)`: Extract coefficient estimates
- `vcov(femtofit)`: Extract variance-covariance matrix
- `confint(femtofit)`: Compute confidence intervals
- `logLik(femtofit)`: Extract log-likelihood (enables AIC/BIC)
- `nobs(femtofit)`: Number of observations
- `print(femtofit)`: Print fitted model
- `summary(femtofit)`: Summary of fitted model

Functions

- `print(summary.femtofit)`: Print summary

`find_mle` *Find MLE with standard errors*

Description

Convenience function that finds the MLE and computes standard errors from the observed Fisher information.

Usage

```
find_mle(loglik_fn, params, method = "newton", ...)
```

Arguments

`loglik_fn`	Log-likelihood function taking list of value parameters
`params`	List of value objects (initial parameter values)
`method`	Optimization method: ”newton” or ”gradient”
`...`	Additional arguments passed to optimizer

Value

A list containing:

`estimate`	MLE as numeric vector
`se`	Standard errors
`vcov`	Variance-covariance matrix
`loglik`	Log-likelihood at MLE
`hessian`	Hessian at MLE
`converged`	Convergence indicator

Examples

```
## Not run:
x <- rnorm(100, mean = 5, sd = 2)
loglik <- function(p) loglik_normal(p[[1]], p[[2]], x)
result <- find_mle(loglik, list(val(0), val(1)))
result$estimate  # MLEs
result$se        # Standard errors

## End(Not run)
```

---

fisher_information *Compute observed Fisher information matrix*

Description

For a log-likelihood function, the observed Fisher information is the negative Hessian evaluated at the MLE. This is used for computing standard errors of MLEs.

Usage

```
fisher_information(loglik_fn, params)
```

Arguments

`loglik_fn`	Log-likelihood function taking a list of value parameters
`params`	List of value objects at MLE

Details

The observed Fisher information I(^) = -H(^) where H is the Hessian of the log-likelihood. Standard errors are sqrt(diag(I ^{1})).

Value

The observed Fisher information matrix (negative Hessian)

`fisher_scoring` *Fisher scoring optimizer*

Description

Similar to Newton-Raphson but uses expected Fisher information (negative expected Hessian) instead of observed Hessian. More stable for some problems.

Usage

```
fisher_scoring(loglik_fn, params, max_iter = 100, tol = 1e-08, verbose = 0)
```

Arguments

`loglik_fn`	Log-likelihood function
`params`	List of value objects (initial parameter values)
`max_iter`	Maximum iterations, default 100
`tol`	Convergence tolerance, default 1e-8
`verbose`	Print progress every N iterations (0 for silent)

Details

For regular exponential families, Fisher scoring is equivalent to Newton-Raphson since observed = expected information.

Fisher scoring: `theta_{n+1} = theta_n + solve(I(theta_n)) %*% S(theta_n)` where `I = -E(H)` (Fisher information) and S = gradient (score).

This implementation uses the observed Hessian as an approximation to the expected Hessian, making it identical to Newton-Raphson. For a true Fisher scoring implementation, one would need to compute `E(H)` analytically or via Monte Carlo.

Value

Same structure as newton_raphson

`fit` *Fit a model via maximum likelihood*

Description

Finds the maximum likelihood estimates of parameters and returns a fitted model object with standard errors, confidence intervals, and other inference quantities computed automatically via autodiff.

Usage

```
fit(
  loglik,
  params,
  method = c("bfgs", "lbfgs", "newton", "gradient"),
  nobs = NULL,
  predict_fn = NULL,
  ...
)
```

Arguments

`loglik`	A log-likelihood function. Can be specified in two ways: • Named arguments: `function(mu, sigma) loglik_normal(mu, sigma, x)` • Single parameter argument: `function(p) loglik_normal(p$mu, p$sigma, x)` The function should return a scalar value object.
`params`	Named numeric vector of initial parameter values, e.g., `c(mu = 0, sigma = 1)`.
`method`	Optimization method: "bfgs" (default), "lbfgs", "newton", or "gradient".
`nobs`	Optional number of observations used in fitting. Setting this enables `BIC()` (which needs `nobs`) and is reported by `nobs(result)`. If omitted, `BIC()` returns `NA`.
`predict_fn`	Optional prediction function. Should take arguments `(params, newdata)` where `params` is a named list of parameter values and `newdata` is the new data for prediction. If provided, enables use of `predict()` on the fitted model.
`...`	Additional arguments passed to the optimizer.

Value

A `femtofit` object containing:

- Coefficient estimates (accessible via `coef()`)
- Variance-covariance matrix (accessible via `vcov()`)
- Log-likelihood value (accessible via `logLik()`)
- Standard errors, Hessian, convergence info

Examples

```
## Not run:
# Generate data
set.seed(42)
x <- rnorm(100, mean = 5, sd = 2)
```

```
# Fit using named arguments (recommended)
result <- fit(
  function(mu, sigma) loglik_normal(mu, sigma, x),
  params = c(mu = 0, sigma = 1)
)

# Or using single parameter argument
result <- fit(
  function(p) loglik_normal(p$mu, p$sigma, x),
  params = c(mu = 0, sigma = 1)
)

# Standard R generics work
coef(result)      # Parameter estimates
vcov(result)      # Variance-covariance matrix
confint(result)   # 95% confidence intervals
logLik(result)    # Log-likelihood (works with AIC/BIC)
AIC(result)       # Akaike information criterion
summary(result)   # Full summary with p-values

## End(Not run)
```

`fitting` *Statistical model fitting with automatic differentiation*

Description

This module provides the `fit()` function for maximum likelihood estimation and the `femtofit` class for representing fitted models. The interface follows standard R conventions, implementing base R generics like `coef()`, `vcov()`, `confint()`, `logLik()`, etc.

`get_data` *Retrieve the data stored by an object*

Description

Generic function to extract the numeric data from a differentiable object. For value objects, returns the underlying matrix. For plain numeric inputs, converts to matrix representation.

Assignment function to update the data field of a value object without breaking the computational graph reference.

Usage

```
get_data(x, ...)

## S3 method for class 'value'
get_data(x, drop = TRUE, ...)

## Default S3 method:
get_data(x, ...)

get_data(x) <- value

## S3 replacement method for class 'value'
get_data(x) <- value

## Default S3 replacement method:
get_data(x) <- value
```

Arguments

x	A value object
...	additional arguments to pass
drop	If TRUE (default) and result is 1x1, return scalar. Set to FALSE to always return a matrix.
value	The new numeric value to assign (converted to matrix)

Value

The data as scalar (if 1x1 and drop=TRUE) or matrix

The modified value object (invisibly)

Examples

```
x <- val(5)
get_data(x)          # Returns 5 (scalar)
get_data(x, drop = FALSE)  # Returns 1x1 matrix

## Not run:
x <- val(5)
get_data(x) <- 10
get_data(x)  # Returns 10

## End(Not run)
```

`grad` *Gradient of* `x` *with respect to* `e` *in* `backward(e)`*, e.g., dx/de. (applies the chain rule)*

Description

Gradient of `x` with respect to `e` in `backward(e)`, e.g., dx/de. (applies the chain rule)

Usage

```
grad(x, ...)
```

Arguments

`x`	A differential object
`...`	pass additional arguments

Value

The gradient matrix (same dimensions as data)

`grad.default` *Default gradient is zero matrix*

Description

Default gradient is zero matrix

Usage

```
## Default S3 method:
grad(x, ...)
```

Arguments

`x`	A non-value object
`...`	pass additional arguments

Value

Zero matrix of appropriate dimensions

`grad.value` *Gradient of a* `value` *object* `x` *with respect to* `e` *in* `backward(e)`*, e.g., dx/de. (applies the chain rule)*

Description

Gradient of a `value` object `x` with respect to `e` in `backward(e)`, e.g., dx/de. (applies the chain rule)

Usage

```
## S3 method for class 'value'
grad(x, drop = TRUE, ...)
```

Arguments

`x`	A value object
`drop`	If TRUE (default) and result is 1x1, return scalar. Set to FALSE to always return a matrix.
`...`	pass additional arguments

Value

The gradient as scalar (if 1x1 and drop=TRUE) or matrix

`gradient` *Compute gradient as a numeric vector*

Description

Convenience function to compute the gradient of a loss function at the current parameter values.

Usage

```
gradient(loss_fn, params)
```

Arguments

`loss_fn`	A function taking a list of value parameters
`params`	A list of value objects

Value

A numeric vector of gradients (one per parameter)

`gradient_ascent` *Gradient ascent/descent optimizer*

Description

Finds the optimum of a function using gradient-based optimization. Supports gradient clipping and adaptive step sizes.

Usage

```
gradient_ascent(
  objective_fn,
  params,
  lr = 0.01,
  max_iter = 1000,
  tol = 1e-06,
  maximize = TRUE,
  grad_clip = NULL,
  verbose = 0
)
```

Arguments

`objective_fn`	Function taking list of value parameters, returns scalar
`params`	List of value objects (initial parameter values)
`lr`	Learning rate (step size), default 0.01
`max_iter`	Maximum iterations, default 1000
`tol`	Convergence tolerance on gradient norm, default 1e-6
`maximize`	If TRUE (default), maximize; if FALSE, minimize
`grad_clip`	Maximum gradient norm (NULL for no clipping)
`verbose`	Print progress every N iterations (0 for silent)

Value

A list containing:

`params`	List of value objects at optimum
`value`	Objective function value at optimum
`gradient`	Gradient at optimum
`iterations`	Number of iterations performed
`converged`	TRUE if gradient norm $<$ tol

Examples

```
## Not run:
# Find MLE for exponential distribution
x <- rexp(100, rate = 2)
loglik <- function(p) loglik_exponential(p[[1]], x)
result <- gradient_ascent(loglik, list(val(1)))
get_data(result$params[[1]])  # Should be close to 1/mean(x)

## End(Not run)
```

---

gradient_descent *Gradient descent (minimize)*

Description

Convenience wrapper for gradient_ascent with maximize=FALSE.

Usage

```
gradient_descent(
  objective_fn,
  params,
  lr = 0.01,
  max_iter = 1000,
  tol = 1e-06,
  grad_clip = NULL,
  verbose = 0
)
```

Arguments

objective_fn	Function taking list of value parameters, returns scalar
params	List of value objects (initial parameter values)
lr	Learning rate (step size), default 0.01
max_iter	Maximum iterations, default 1000
tol	Convergence tolerance on gradient norm, default 1e-6
grad_clip	Maximum gradient norm (NULL for no clipping)
verbose	Print progress every N iterations (0 for silent)

`hessian` *Compute Hessian matrix via forward-over-reverse automatic differentiation*

Description

Computes the Hessian matrix (matrix of second partial derivatives) of a scalar function with respect to a vector of parameters. Uses forward-mode AD on top of reverse-mode AD for efficient, accurate computation.

Usage

```
hessian(loss_fn, params, value_creator = val)
```

Arguments

`loss_fn`	A function that takes a list of parameters and returns a scalar loss/objective. The function must use femtograd operations.
`params`	A list of value objects (the parameters)
`value_creator`	Function to create value objects (default: val). This allows customization for different value types.

Details

The Hessian is computed using forward-over-reverse mode AD:

1. For each parameter i, create dual numbers where:
   - Primal = value object holding parameter value
   - Tangent = value object (1 for param i, 0 for others)
2. Run the loss function with these dual-value inputs
3. The tangent of the loss is df/d (as a value expression)
4. Call backward() on the tangent loss
5. Each tangent parameter's gradient gives $d^2f/d\ d$

This is the "forward-over-reverse" pattern: forward-mode (dual numbers) computes df/d symbolically, then reverse-mode differentiates that to get $d^2f/d\ d$.

Value

A numeric matrix of dimension (n x n) where n = length(params), containing the Hessian $d^2f/d\ d$

Examples

```
## Not run:
# Hessian of f(x,y) = x^2 + x*y + y^2
loss_fn <- function(p) {
  x <- p[[1]]
  y <- p[[2]]
  x^2 + x*y + y^2
}
params <- list(val(1), val(2))
H <- hessian(loss_fn, params)
# H should be [[2, 1], [1, 2]]

## End(Not run)
```

---

hypothesis_tests *Hypothesis Testing for Fitted Models*

Description

Provides hypothesis testing functions that produce results compatible with the hypothesize package interface. Results have `stat`, `p.value`, and `dof` components accessible via standard methods.

---

inv_bounded *Inverse of bounded transform*

Description

Converts a bounded value back to unconstrained scale.

Usage

```
inv_bounded(x, lower, upper)
```

Arguments

`x`	Value in (lower, upper)
`lower`	Lower bound
`upper`	Upper bound

Value

Unconstrained value

inv_positive *Inverse of positive transform*

Description

Converts a positive value back to unconstrained scale.

Usage

```
inv_positive(x)
```

Arguments

x	Positive value

Value

Unconstrained value (log of x)

inv_probability *Inverse of probability transform*

Description

Converts a probability to unconstrained scale (logit).

Usage

```
inv_probability(x)
```

Arguments

x	Value in (0, 1)

Value

Unconstrained value (logit of x)

inverse_transforms *Inverse transforms for recovering original scale*

Description

These functions convert fitted unconstrained parameters back to their natural scale.

is_dual — *Check if object is a dual number*

Description

Check if object is a dual number

Usage

```
is_dual(x)
```

Arguments

x	Object to check

Value

TRUE if x is a dual object

is_significant_at — *Check if test is significant at given level*

Description

Check if test is significant at given level

Usage

```
is_significant_at(x, alpha = 0.05, ...)

## S3 method for class 'hypothesis_test'
is_significant_at(x, alpha = 0.05, ...)
```

Arguments

x	A hypothesis test object
alpha	Significance level (default 0.05)
...	Additional arguments (ignored)

Value

Logical indicating significance

Methods (by class)

- `is_significant_at(hypothesis_test)`: Significance check for hypothesis tests

is_value *Check if an object is of class value*

Description

Determines if the input object is an instance of the value R6 class.

Usage

```
is_value(x)
```

Arguments

x	The object to be checked

Value

TRUE if the object is of class value, FALSE otherwise

lbfgs *L-BFGS optimizer (limited memory BFGS)*

Description

Memory-efficient variant of BFGS that stores only the last m correction pairs instead of the full inverse Hessian approximation. Suitable for large-scale optimization.

Usage

```
lbfgs(
  objective_fn,
  params,
  m = 10,
  max_iter = 1000,
  tol = 1e-06,
  maximize = FALSE,
  verbose = 0
)
```

Arguments

objective_fn	Function taking list of value parameters, returns scalar
params	List of value objects (initial parameter values)
m	Number of correction pairs to store, default 10
max_iter	Maximum iterations, default 1000

`tol`	Convergence tolerance on gradient norm, default 1e-6
`maximize`	If TRUE, maximize; if FALSE (default), minimize
`verbose`	Print progress every N iterations (0 for silent)

Details

L-BFGS uses the two-loop recursion algorithm to compute the search direction without explicitly forming the inverse Hessian. Memory usage is O(m*n) instead of O(n^2) for full BFGS.

Value

A list containing:

`params`	List of value objects at optimum
`value`	Objective function value at optimum
`gradient`	Gradient at optimum
`iterations`	Number of iterations performed
`converged`	TRUE if gradient norm < tol

`lgamma.value` *Log-gamma function for value objects*

Description

Element-wise log(gamma(x)).

Usage

```
## S3 method for class 'value'
lgamma(x)
```

Arguments

`x`	A value object

Value

A new value object representing lgamma(x)

`line_search` *Backtracking line search (Armijo condition)*

Description

Finds a step size that satisfies the Armijo condition for sufficient decrease.

Usage

```
line_search(
  objective_fn,
  param_values,
  direction,
  grad,
  maximize = FALSE,
  alpha = 1,
  c1 = 1e-04,
  rho = 0.5,
  max_iter = 20
)
```

Arguments

`objective_fn`	Function to minimize/maximize
`param_values`	Current parameter values (numeric vector)
`direction`	Search direction (numeric vector)
`grad`	Current gradient (numeric vector)
`maximize`	If TRUE, maximize; if FALSE, minimize
`alpha`	Initial step size, default 1
`c1`	Armijo parameter (sufficient decrease), default 1e-4
`rho`	Backtracking factor, default 0.5
`max_iter`	Maximum backtracking iterations, default 20

Details

Armijo condition (for minimization): f(x + *d) <= f(x) + c1* *g'*d where d is the search direction and g is the gradient.

Value

Step size satisfying Armijo condition

log.value *Natural logarithm for value objects*

Description

Element-wise natural logarithm.

Usage

```
## S3 method for class 'value'
log(x, ...)
```

Arguments

x	A value object
...	Additional arguments (ignored).

Value

A new value object representing log(x)

log_safe *Safe logarithm (handles zeros)*

Description

Computes log(x) with protection against log(0) = -Inf. Optionally clamps input to a minimum value.

Usage

```
log_safe(x, eps = .Machine$double.eps)
```

Arguments

x	A value object or numeric
eps	Minimum value to clamp x to (default: .Machine$double.eps)

Details

This is useful when computing log-likelihoods where probabilities might numerically become zero. Where `x > eps`, the gradient is the true `1/x`. Where the clamp triggered (`x <= eps`), the gradient is zeroed out rather than returning the explosive `1/eps` value — propagating a huge gradient from a region where the primal is known to be wrong would actively mislead the optimizer.

Value

log(max(x, eps))

`log_sigmoid` *Log-sigmoid (numerically stable)*

Description

Computes log(sigmoid(x)) = -log(1 + exp(-x)) stably.

Usage

```
log_sigmoid(x)
```

Arguments

`x`	A value object or numeric

Details

Direct computation of log(sigmoid(x)) fails for large negative x (sigmoid underflows to 0). This uses:

- For x >= 0: -log(1 + exp(-x)) = -softplus(-x)
- For x < 0: x - log(1 + exp(x)) = x - softplus(x)

Value

log(sigmoid(x))

`log1p.value` *Log(1+x) for value objects*

Description

Element-wise log(1+x), numerically stable for small x.

Usage

```
## S3 method for class 'value'
log1p(x)
```

Arguments

`x`	A value object

Value

A new value object representing log(1+x)

log1p_safe — *Log1p with underflow protection (deprecated alias for log1p)*

Description

This is a thin alias for `log1p()`, kept only for backward compatibility. `log1p()` already has `log1p.value` and `log1p.dual` methods registered by femtograd, so it is numerically stable and autodiff-aware on its own. Use `log1p()` directly.

Usage

```
log1p_safe(x)
```

Arguments

`x`	A value object, dual, or numeric.

Value

`log1p(x)`.

logit — *Logit function for value objects*

Description

Element-wise logit: log(p/(1-p)).

Usage

```
logit(x)
```

Arguments

`x`	A value object representing probabilities

Value

A new value object representing logit(x)

`loglik_bernoulli` *Bernoulli distribution log-likelihood*

Description

Special case of binomial with size=1. `L(p|x) = sum(x*log(p) + (1-x)*log(1-p))`

Usage

```
loglik_bernoulli(p, x)
```

Arguments

`p`	Success probability (value object), must be in (0,1)
`x`	Binary vector (0 or 1)

Value

A value object representing the log-likelihood

`loglik_beta` *Beta distribution log-likelihood*

Description

Computes the log-likelihood for i.i.d. beta observations. `L(a,b|x) = n*(lgamma(a+b) - lgamma(a) - lgamma(b)) + (a-1)*sum(log(x)) + (b-1)*sum(log(1-x))`

Usage

```
loglik_beta(alpha, beta, x)
```

Arguments

`alpha`	Shape parameter (value object), must be positive
`beta`	Shape parameter (value object), must be positive
`x`	Numeric vector of observations in (0,1)

Value

A value object representing the log-likelihood

loglik_binomial *Binomial distribution log-likelihood*

Description

Computes the log-likelihood for binomial observations. `L(p|x,n) = sum(x*log(p) + (n-x)*log(1-p) + log(C(n,x)))`

Usage

```
loglik_binomial(p, x, size)
```

Arguments

`p`	Success probability (value object), must be in (0,1)
`x`	Integer vector of successes
`size`	Integer vector of trial counts (or single value if constant)

Value

A value object representing the log-likelihood

Examples

```
## Not run:
x <- rbinom(50, size = 10, prob = 0.3)
p <- val(0.5)
ll <- loglik_binomial(p, x, size = 10)
backward(ll)
# MLE is sum(x) / sum(size)

## End(Not run)
```

loglik_exponential *Exponential distribution log-likelihood*

Description

Computes the log-likelihood for i.i.d. exponential observations. L(|x) = n*log( )* - Σx

Usage

```
loglik_exponential(rate, x)
```

Arguments

`rate`	Rate parameter (value object), must be positive
`x`	Numeric vector of observations (must be non-negative)

Value

A value object representing the log-likelihood

Examples

```
## Not run:
x <- rexp(50, rate = 2)
rate <- val(1)
ll <- loglik_exponential(rate, x)
backward(ll)
grad(rate)  # should be n/rate - sum(x)

## End(Not run)
```

`loglik_gamma` *Gamma distribution log-likelihood*

Description

Computes the log-likelihood for i.i.d. gamma observations. L(, |x) = n log( ) - n*log(Γ( )) + (-1)*Σlog(x)* - Σx

Usage

```
loglik_gamma(shape, rate, x)
```

Arguments

`shape`	Shape parameter (value object), must be positive
`rate`	Rate parameter (value object), must be positive
`x`	Numeric vector of observations (must be positive)

Details

The gamma distribution is parameterized with shape alpha and rate beta, where `E(X) = alpha/beta` and `Var(X) = alpha/beta^2`.

Value

A value object representing the log-likelihood

loglik_logistic *Logistic regression log-likelihood (binary)*

Description

Computes the log-likelihood for binary logistic regression. `L(beta|X,y) = sum(y*log(p) + (1-y)*log(1-p))` where `p = sigmoid(X %*% beta)`

Usage

```
loglik_logistic(beta, X, y)
```

Arguments

`beta`	Coefficient vector (list of value objects)
`X`	Design matrix (n x p numeric matrix)
`y`	Binary response vector (0 or 1)

Details

Uses the numerically stable form: log(p) = -log(1 + exp(-)) and log(1-p) = -log(1 + exp( )) where = X

Value

A value object representing the log-likelihood

loglik_negbinom *Negative binomial log-likelihood*

Description

Computes the log-likelihood for negative binomial (failures until r successes). `L(r,p|x) = sum(lchoose(x+r-1, x) + r*log(p) + x*log(1-p))`

Usage

```
loglik_negbinom(r, p, x)
```

Arguments

`r`	Number of successes parameter (value object or fixed positive)
`p`	Success probability (value object), must be in (0,1)
`x`	Integer vector of failure counts

Value

A value object representing the log-likelihood

loglik_normal *Normal (Gaussian) log-likelihood*

Description

Computes the log-likelihood for i.i.d. normal observations. L(, ²|x) = -n/2 log(2) - n/2 log(²) - Σ(x -)²/(2 ²)

Usage

```
loglik_normal(mu, sigma, x)
```

Arguments

mu	Mean parameter (value object)
sigma	Standard deviation parameter (value object), must be positive
x	Numeric vector of observations

Value

A value object representing the log-likelihood

Examples

```
## Not run:
x <- rnorm(100, mean = 5, sd = 2)
mu <- val(0)
sigma <- val(1)
ll <- loglik_normal(mu, sigma, x)
backward(ll)
grad(mu)  # score for mu

## End(Not run)
```

loglik_pareto *Pareto distribution log-likelihood*

Description

Computes the log-likelihood for i.i.d. Pareto observations. L(, x | x) = n*log( ) + n* *log(x) - (+1)*Σlog(x)

Usage

```
loglik_pareto(alpha, x_min, x)
```

Arguments

alpha	Shape parameter (value object), must be positive
x_min	Minimum/scale parameter x (fixed positive number). All observations must be >= x_min.
x	Numeric vector of observations (must be >= x_min)

Details

The Pareto distribution is used to model heavy-tailed phenomena like income distributions, city sizes, etc. Here x_min is typically known (e.g., min(x)) and alpha is estimated.

Value

A value object representing the log-likelihood

Examples

```
## Not run:
# Generate Pareto data
alpha_true <- 2
x_min <- 1
u <- runif(100)
x <- x_min * (1 - u)^(-1/alpha_true)

# Fit (alpha only, x_min = min(x) is fixed)
result <- fit(
  function(log_alpha) {
    alpha <- exp(log_alpha)
    loglik_pareto(alpha, x_min = min(x), x)
  },
  params = c(log_alpha = 0)
)

## End(Not run)
```

---

loglik_poisson *Poisson distribution log-likelihood*

Description

Computes the log-likelihood for i.i.d. Poisson observations. L(|x) = Σx *log( )* - *n* - Σlog(x !)

Usage

```
loglik_poisson(lambda, x)
```

Arguments

`lambda`	Rate parameter (value object), must be positive
`x`	Integer vector of observations (counts)

Details

The term Σlog(x !) is constant w.r.t. and is included for completeness.

Value

A value object representing the log-likelihood

Examples

```
## Not run:
x <- rpois(100, lambda = 3)
lambda <- val(1)
ll <- loglik_poisson(lambda, x)
backward(ll)
# MLE is mean(x)

## End(Not run)
```

`loglik_weibull` *Weibull distribution log-likelihood*

Description

Computes the log-likelihood for i.i.d. Weibull observations. L(k, | x) = n*log(k)* - *n*k*log( ) + (k-1)*Σlog(x) - Σ(x /)^k

Usage

```
loglik_weibull(shape, scale, x)
```

Arguments

`shape`	Shape parameter k (value object), must be positive
`scale`	Scale parameter (value object), must be positive
`x`	Numeric vector of observations (must be positive)

Details

The Weibull distribution is commonly used in survival analysis and reliability engineering. It generalizes the exponential distribution (k=1 gives exponential with rate 1/).

Value

A value object representing the log-likelihood

Examples

```
## Not run:
x <- rweibull(100, shape = 2, scale = 3)

# Using log-parameterization for positivity
result <- fit(
  function(log_shape, log_scale) {
    shape <- exp(log_shape)
    scale <- exp(log_scale)
    loglik_weibull(shape, scale, x)
  },
  params = c(log_shape = 0, log_scale = 0)
)

## End(Not run)
```

---

logsumexp *Log-Sum-Exp (numerically stable)*

Description

Computes log(sum(exp(x))) in a numerically stable way by factoring out the maximum value: log(sum(exp(x))) = m + log(sum(exp(x - m))) where m = max(x).

Usage

```
logsumexp(..., na.rm = FALSE)
```

Arguments

`...`	value objects and/or numeric vectors to include in the sum
`na.rm`	Logical, whether to remove NA values

Details

The naive computation of log(sum(exp(x))) overflows when x contains large values (exp(710) = Inf in double precision) and underflows when x contains very negative values. This implementation is stable for any finite input.

Value

A value object representing log(sum(exp(x)))

Examples

```
## Not run:
x <- val(c(1000, 1001, 1002))  # Would overflow with naive exp()
logsumexp(x)  # Returns ~1002.41 correctly

## End(Not run)
```

`lower_bounded` *Transform to lower-bounded interval*

Description

Maps unconstrained values to (lower, ∞). Equivalent to lower + positive(x).

Usage

```
lower_bounded(x, lower)
```

Arguments

`x`	Unconstrained value
`lower`	Lower bound

Details

The transformation is: lower_bounded(x, a) = a + exp(x)

Value

Value > lower

Examples

```
## Not run:
# Parameter > 2
raw <- val(0)
param <- lower_bounded(raw, 2)  # 2 + exp(0) = 3

## End(Not run)
```

`lrt` *Likelihood Ratio Test*

Description

Computes the likelihood ratio test (LRT) for comparing nested models. Works with femtofit objects or raw log-likelihood values.

Usage

```
lrt(null_model, alt_model, df = NULL)
```

Arguments

`null_model`	Either a `femtofit` object (simpler model) or a numeric log-likelihood value.
`alt_model`	Either a `femtofit` object (more complex model) or a numeric log-likelihood value.
`df`	Degrees of freedom. If NULL and both arguments are femtofit objects, computed as the difference in number of parameters.

Details

The likelihood ratio test statistic is:

$$\Lambda = -2(\ell_0 - \ell_1)$$

where ℓ_0 is the log-likelihood under the null (simpler) model and ℓ_1 is the log-likelihood under the alternative (complex) model.

Under the null hypothesis and regularity conditions, Λ asymptotically follows a chi-squared distribution with degrees of freedom equal to the difference in number of free parameters.

Value

A `likelihood_ratio_test` object (also inherits from `hypothesis_test`) containing:

stat The LRT statistic: -2 * (loglik_null - loglik_alt)

p.value P-value from chi-squared distribution

dof Degrees of freedom

null_loglik Log-likelihood of null model

alt_loglik Log-likelihood of alternative model

Examples

```
## Not run:
# Generate data
set.seed(42)
x <- rnorm(100, mean = 5, sd = 2)

# Fit full model (estimate both mu and sigma)
full <- fit(
  function(mu, log_sigma) {
    sigma <- exp(log_sigma)
    loglik_normal(mu, sigma, x)
  },
  params = c(mu = 0, log_sigma = 0)
)

# Fit null model (mu fixed at 0)
null <- fit(
  function(log_sigma) {
    sigma <- exp(log_sigma)
    loglik_normal(0, sigma, x)  # mu = 0
  },
  params = c(log_sigma = 0)
)

# Likelihood ratio test
test <- lrt(null, full)
test
pval(test)
is_significant_at(test, 0.05)

# Can also use raw log-likelihoods
lrt(null_loglik = -150, alt_loglik = -140, df = 1)

## End(Not run)
```

---

mean.value *Mean for value objects*

Description

Computes the arithmetic mean of all elements across value objects.

Usage

```
## S3 method for class 'value'
mean(x, ...)
```

Arguments

x	A value object or list of value objects
...	Additional arguments (for compatibility with base::mean)

Value

A new value object (1x1 matrix) representing the mean

---

newton_raphson *Newton-Raphson optimizer*

Description

Finds the optimum using Newton-Raphson method with exact Hessian. Uses second-order information for faster convergence near optimum.

Usage

```
newton_raphson(
  objective_fn,
  params,
  max_iter = 100,
  tol = 1e-08,
  maximize = TRUE,
  step_scale = 1,
  verbose = 0
)
```

Arguments

objective_fn	Function taking list of value parameters, returns scalar
params	List of value objects (initial parameter values)
max_iter	Maximum iterations, default 100
tol	Convergence tolerance on step size, default 1e-8
maximize	If TRUE (default), maximize; if FALSE, minimize
step_scale	Scale factor for Newton step (< 1 for damping), default 1
verbose	Print progress every N iterations (0 for silent)

Details

Newton-Raphson update: _n+1 = _n - H ¹ g For maximization, uses: _n+1 = _n - H ¹ g (H is negative definite) For minimization, uses: _n+1 = _n - H ¹ g (H is positive definite)

The Hessian is computed via forward-over-reverse AD at each iteration. This is exact but can be slow for many parameters.

Value

A list containing:

`params`	List of value objects at optimum
`value`	Objective function value at optimum
`gradient`	Gradient at optimum
`hessian`	Hessian at optimum
`iterations`	Number of iterations performed
`converged`	TRUE if step size < tol

Examples

```
## Not run:
# Find MLE for normal distribution
x <- rnorm(100, mean = 5, sd = 2)
loglik <- function(p) loglik_normal(p[[1]], p[[2]], x)
result <- newton_raphson(loglik, list(val(0), val(1)))
sapply(result$params, get_data)  # Should be close to c(mean(x), sd(x))

## End(Not run)
```

`observed_info` *Observed Fisher information matrix*

Description

Returns the observed Fisher information matrix, which is the negative Hessian of the log-likelihood at the MLE.

Usage

```
observed_info(object, ...)

## S3 method for class 'femtofit'
observed_info(object, ...)
```

Arguments

`object`	A fitted model object.
`...`	Additional arguments.

Value

The observed information matrix.

Methods (by class)

- `observed_info(femtofit)`: Observed information for femtofit

optimization *Optimization routines for maximum likelihood estimation*

Description

These functions find the optimum (maximum or minimum) of differentiable objective functions using femtograd's automatic differentiation.

plot.profile_likelihood *Plot profile likelihood*

Description

Plot profile likelihood

Usage

```
## S3 method for class 'profile_likelihood'
plot(x, level = 0.95, ...)
```

Arguments

x	A profile_likelihood object.
level	Confidence level for adding threshold line (default 0.95).
...	Additional arguments passed to plot.

positive *Transform to positive values*

Description

Maps unconstrained values to positive values via exp(x). Use this for parameters like standard deviations, rates, or variances.

Usage

```
positive(x)
```

Arguments

x	Unconstrained value (scalar, vector, or value object)

Details

The transformation is: positive(x) = exp(x)

For optimization, work with the unconstrained parameter and transform:

```
result <- fit(
  function(mu, log_sigma) {
    sigma <- positive(log_sigma)  # exp(log_sigma)
    loglik_normal(mu, sigma, data)
  },
  params = c(mu = 0, log_sigma = 0)  # log(1) = 0
)
# To recover sigma: exp(coef(result)["log_sigma"])
```

Value

Positive value (always > 0)

Examples

```
## Not run:
# Parameter that must be positive
log_sigma <- val(-1)
sigma <- positive(log_sigma)  # exp(-1)  0.368
get_data(sigma)

# Works in optimization
fit(
  function(mu, log_sigma) loglik_normal(mu, positive(log_sigma), x),
  params = c(mu = 0, log_sigma = 0)
)

## End(Not run)
```

---

predict.femtofit *Predictions from a fitted model*

Description

Generate predictions from a `femtofit` object using the prediction function supplied at fit time.

Usage

```
## S3 method for class 'femtofit'
predict(object, newdata, ...)
```

Arguments

`object`	A `femtofit` object.
`newdata`	New data for prediction. The format depends on the prediction function provided to `fit()`.
`...`	Additional arguments passed to the prediction function.

Details

For `predict()` to work, a `predict_fn` must have been provided when calling `fit()`. The prediction function should have signature `function(params, newdata, ...)` where `params` is a named list of parameter values.

Value

Predictions, as returned by the prediction function.

Examples

```
## Not run:
# Linear regression example
x <- 1:10
y <- 2 + 3*x + rnorm(10, sd = 0.5)

# Fit with prediction function
result <- fit(
  function(a, b, log_sigma) {
    sigma <- exp(log_sigma)
    predicted <- a + b * x
    loglik_normal(predicted, sigma, y)
  },
  params = c(a = 0, b = 1, log_sigma = 0),
  predict_fn = function(params, newdata) {
    params$a + params$b * newdata
  }
)

# Predict for new x values
predict(result, newdata = c(11, 12, 13))

## End(Not run)
```

`primal` *Extract primal from dual or return value unchanged*

Description

Extract primal from dual or return value unchanged

Usage

```
primal(x)
```

Arguments

`x`	A dual or non-dual object

Value

The primal value

`print.anova.femtofit` *Print anova table for femtofit*

Description

Print anova table for femtofit

Usage

```
## S3 method for class 'anova.femtofit'
print(x, digits = 4, ...)
```

Arguments

`x`	An `anova.femtofit` object from `anova.femtofit()`.
`digits`	Number of significant digits for printing.
`...`	Additional arguments (ignored).

`print.bootstrap_result` *Print method for bootstrap results*

Description

Print method for bootstrap results

Usage

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

Arguments

`x`	A `bootstrap_result` object.
`...`	Additional arguments (ignored).

`print.hessian_check` *Print methods for diagnostic objects*

Description

Print methods for diagnostic objects

Usage

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

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

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

Arguments

`x`	A diagnostic object.
`...`	Additional arguments (ignored).

`print.likelihood_ratio_test`
Print method for likelihood ratio test

Description

Print method for likelihood ratio test

Usage

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

Arguments

`x`	A likelihood_ratio_test object
`...`	Additional arguments (ignored)

`print.model_comparison` *Print model comparison*

Description

Print model comparison

Usage

```
## S3 method for class 'model_comparison'
print(x, digits = 2, ...)
```

Arguments

`x`	A `model_comparison` object from `compare()`.
`digits`	Number of digits for rounding.
`...`	Additional arguments (ignored).

`print.profile_likelihood` *Print method for profile likelihood*

Description

Print method for profile likelihood

Usage

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

Arguments

`x`	A `profile_likelihood` object.
`...`	Additional arguments (ignored).

print.value *Print value object and its computational graph*

Description

Print value object and its computational graph

Usage

```
## S3 method for class 'value'
print(x, depth = Inf, indent = "  ", ...)
```

Arguments

x	A value object
depth	Integer indicates depth (dfs) to recurse down the graph
indent	A character string specifying the indentation for each level in the computational graph (default: " ")
...	Additional arguments (ignored).

print.wald_test *Print method for Wald test*

Description

Print method for Wald test

Usage

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

Arguments

x	A wald_test object
...	Additional arguments (ignored)

`probability` *Transform to probability values*

Description

Maps unconstrained values to (0, 1) via the sigmoid function. Use this for probability parameters.

Usage

```
probability(x)
```

Arguments

`x`	Unconstrained value (scalar, vector, or value object)

Details

The transformation is: probability(x) = 1 / (1 + exp(-x)) = sigmoid(x)

For optimization:

```
result <- fit(
  function(logit_p) {
    p <- probability(logit_p)
    loglik_bernoulli(p, data)
  },
  params = c(logit_p = 0)  # sigmoid(0) = 0.5
)
# To recover p: sigmoid(coef(result)["logit_p"])
```

Value

Value in (0, 1)

Examples

```
## Not run:
# Parameter that must be in (0, 1)
logit_p <- val(2)
p <- probability(logit_p)  # sigmoid(2)  0.88
get_data(p)

## End(Not run)
```

profile_likelihood *Profile Likelihood*

Description

Functions for computing profile likelihoods and profile-based confidence intervals. Profile likelihood provides more accurate inference than Wald-based methods when the likelihood is non-quadratic.

profile_loglik *Compute profile likelihood for a parameter*

Description

Computes the profile log-likelihood for a single parameter by maximizing over all other parameters at each fixed value.

Usage

```
profile_loglik(object, parm, values = NULL, n_points = 20, range_mult = 3, ...)
```

Arguments

`object`	A `femtofit` object from `fit()`.
`parm`	The parameter name or index to profile.
`values`	Optional vector of values at which to evaluate the profile. If NULL, a grid is computed based on the MLE and standard error.
`n_points`	Number of points in the grid (default 20). Ignored if `values` is provided.
`range_mult`	Multiplier for the range around MLE (default 3). The grid spans MLE ± range_mult * SE.
`...`	Additional arguments passed to the optimizer.

Details

The profile log-likelihood for parameter is defined as:

$$pl(\theta_i) = \max_{\theta_{-i}} \ell(\theta_i, \theta_{-i})$$

where denotes all parameters except . For each grid value of , this re-optimizes over the remaining parameters, warm-started from the MLE.

If the `femtofit` object was produced by an older version of `fit()` that did not stash the wrapped objective, a quadratic approximation based on the stored Hessian is returned instead and `is_quadratic_approx = TRUE` is set on the result.

Value

A `profile_likelihood` object containing:

parameter Name of the profiled parameter

values Grid of parameter values

profile_loglik Profile log-likelihood at each value

mle MLE value of the parameter

max_loglik Maximum log-likelihood

Examples

```
## Not run:
set.seed(42)
x <- rnorm(100, mean = 5, sd = 2)

result <- fit(
  function(mu, log_sigma) loglik_normal(mu, exp(log_sigma), x),
  params = c(mu = 0, log_sigma = 0)
)

# Profile likelihood for mu
prof <- profile_loglik(result, "mu")
plot(prof)

## End(Not run)
```

---

pval *Extract p-value from hypothesis test*

Description

Extract p-value from hypothesis test

Usage

```
pval(x, ...)

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

Arguments

`x`	A hypothesis test object
`...`	Additional arguments (ignored)

Value

The p-value

Methods (by class)

- `pval(hypothesis_test)`: p-value for hypothesis tests

`relu` *ReLU activation function for value objects*

Description

Element-wise ReLU: max(0, x)

Usage

```
relu(x)
```

Arguments

`x`	A value object

Value

A new value object representing max(0, x)

`se` *Standard errors from a fitted model*

Description

Extracts standard errors of coefficient estimates.

Usage

```
se(object, ...)

## S3 method for class 'femtofit'
se(object, ...)
```

Arguments

`object`	A `femtofit` object.
`...`	Additional arguments (ignored).

Value

Named numeric vector of standard errors.

Methods (by class)

- `se(femtofit)`: Standard errors for femtofit

`se_reliable` *Check if standard errors are reliable*

Description

A quick check of whether standard errors from the variance-covariance matrix are likely to be reliable.

Usage

```
se_reliable(object)
```

Arguments

`object`	A `femtofit` object.

Details

Standard errors may be unreliable if:

- The Hessian is not negative definite
- The model did not converge
- Any SE is NA or negative

Value

Logical indicating whether SEs are likely reliable.

`sigmoid` *Sigmoid activation function for value objects*

Description

Element-wise sigmoid: 1/(1+exp(-x))

Usage

```
sigmoid(x)
```

Arguments

`x`	A value object

Value

A new value object representing sigmoid(x)

`sigmoid_stable` *Stable sigmoid function*

Description

Computes sigmoid(x) = 1/(1+exp(-x)) with overflow protection.

Usage

```
sigmoid_stable(x)
```

Arguments

`x`	A value object or numeric

Details

For large positive x, exp(-x) underflows to 0, giving sigmoid = 1 (correct). For large negative x, we use sigmoid(x) = exp(x)/(1+exp(x)) to avoid overflow.

Value

Sigmoid values in (0, 1)

`sin.value` *Sine function for value objects*

Description

Element-wise sine.

Usage

```
## S3 method for class 'value'
sin(x)
```

Arguments

`x`	A value object

Value

A new value object representing sin(x)

`softmax` *Softmax function (numerically stable)*

Description

Computes softmax(x)_i = exp(x_i) / sum(exp(x)) in a numerically stable way.

Usage

```
softmax(x)
```

Arguments

`x`	A value object or numeric vector

Details

Uses the identity softmax(x) = softmax(x - max(x)) to prevent overflow.

Value

A value object (or numeric) with softmax probabilities

softplus *Softplus function for value objects*

Description

Element-wise softplus: log(1 + exp(x)).

Usage

```
softplus(x)
```

Arguments

x	A value object

Value

A new value object representing softplus(x)

sqrt.value *Square root for value objects*

Description

Element-wise square root.

Usage

```
## S3 method for class 'value'
sqrt(x)
```

Arguments

x	A value object

Value

A new value object representing sqrt(x)

stability *Numerical stability utilities for automatic differentiation*

Description

These functions provide numerically stable implementations of common operations that can overflow or underflow with naive implementations.

`std_errors` *Compute standard errors from Hessian*

Description

Extracts standard errors from the Hessian of a log-likelihood. `SE(theta_i) = sqrt(diag(solve(I(theta))))` where `I = -H`.

Usage

```
std_errors(hess, is_loglik = TRUE)
```

Arguments

`hess`	The Hessian matrix (from `hessian()`)
`is_loglik`	If TRUE, treats hess as Hessian of log-likelihood and uses -H ^{1}. If FALSE, uses H ^{1} directly.

Value

A numeric vector of standard errors

`sum.dual` *Sum for dual numbers*

Description

Sum for dual numbers

Usage

```
## S3 method for class 'dual'
sum(..., na.rm = FALSE)
```

Arguments

`...`	Dual numbers or numeric values to sum.
`na.rm`	Logical; should missing values be removed?

sum.value *Summation for value objects*

Description

Sums all elements of value objects, returning a 1x1 value.

Usage

```
## S3 method for class 'value'
sum(..., na.rm = FALSE)
```

Arguments

...	value objects and/or numeric values to sum
na.rm	Logical, whether to remove NA values

Value

A new value object (1x1 matrix) representing the sum

summary.bootstrap_result *Summary for bootstrap results*

Description

Summary for bootstrap results

Usage

```
## S3 method for class 'bootstrap_result'
summary(object, level = 0.95, ...)
```

Arguments

object	A bootstrap_result object.
level	Confidence level (default 0.95).
...	Additional arguments (ignored).

`tangent` *Extract tangent from dual or return 0*

Description

Extract tangent from dual or return 0

Usage

```
tangent(x)
```

Arguments

`x`	A dual or non-dual object

Value

The tangent value

`tanh.value` *Hyperbolic tangent for value objects*

Description

Element-wise tanh.

Usage

```
## S3 method for class 'value'
tanh(x)
```

Arguments

`x`	A value object

Value

A new value object representing tanh(x)

test_stat *Extract test statistic from hypothesis test*

Description

Extract test statistic from hypothesis test

Usage

```
test_stat(x, ...)

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

Arguments

x	A hypothesis test object
...	Additional arguments (ignored)

Value

The test statistic

Methods (by class)

- `test_stat(hypothesis_test)`: Test statistic for hypothesis tests

transforms *Parameter Transformation Helpers*

Description

These functions provide clean, differentiable transformations for constrained parameters. They allow optimization in unconstrained space while ensuring parameters stay in valid ranges.

`trigamma.value` *Trigamma function for value objects*

Description

Element-wise trigamma: second derivative of lgamma.

Usage

```
## S3 method for class 'value'
trigamma(x)
```

Arguments

`x`	A value object

Value

A new value object representing trigamma(x)

`upper_bounded` *Transform to upper-bounded interval*

Description

Maps unconstrained values to (-∞, upper). Equivalent to upper - positive(x).

Usage

```
upper_bounded(x, upper)
```

Arguments

`x`	Unconstrained value
`upper`	Upper bound

Details

The transformation is: upper_bounded(x, b) = b - exp(x)

Value

Value < upper

Examples

```
## Not run:
# Parameter < 10
raw <- val(0)
param <- upper_bounded(raw, 10)  # 10 - exp(0) = 9

## End(Not run)
```

`val` `value` *object constructor*

Description

Creates a new value node in the computational graph with automatic differentiation. Input is converted to matrix representation internally.

Usage

```
val(data, label = "")
```

Arguments

`data`	Numeric value (scalar, vector, or matrix)
`label`	Optional character label for debugging

Details

All values are stored as matrices internally:

- `val(5)` creates a 1x1 matrix
- `val(c(1,2,3))` creates a 3x1 column vector
- `val(matrix(...))` preserves matrix dimensions

Value

A value object

`value` *value R6 class*

Description

value R6 class

value R6 class

Details

Represents a node in the computational graph with automatic differentiation. Each value object holds data as a matrix, gradient as a matrix of the same dimensions, a backward function, previous nodes, and an optional label.

All data in femtograd uses matrix representation:

- Scalars are 1x1 matrices
- Column vectors are n x 1 matrices
- Row vectors are 1 x n matrices
- Matrices are m x n matrices

This ensures consistent behavior for get_data(), grad(), and hessian().

Public fields

`data` Numeric matrix containing the value

`grad` Gradient matrix (same dimensions as data), initially zeros

`backward_fn` A function that performs the backward pass (gradient computation)

`prev` A list of previous nodes in the computational graph

`label` Optional character label for debugging (default: "")

Methods

Public methods:

- `value$new()`
- `value$clone()`

Method `new()`: Initializes a new value object with the given data, list of children, and optional label.

Usage:

```
value$new(data, children = list(), label = "")
```

Arguments:

`data` Numeric value (scalar, vector, or matrix) - will be converted to matrix

`children` List of previous nodes in the computational graph (default: empty list)

`label` Optional character label for debugging

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
value$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`vcov_matrix` *Compute variance-covariance matrix from Hessian*

Description

Compute variance-covariance matrix from Hessian

Usage

```
vcov_matrix(hess, is_loglik = TRUE)
```

Arguments

`hess`	The Hessian matrix
`is_loglik`	If TRUE, returns -H ^{1} (for log-likelihood)

Value

The variance-covariance matrix

`wald_test` *Wald Test for Model Parameters*

Description

Performs Wald tests for individual parameters or linear combinations of parameters in a fitted model.

Usage

```
wald_test(object, ...)

## S3 method for class 'femtofit'
wald_test(object, parm = NULL, null_value = 0, ...)

## Default S3 method:
wald_test(object, se, null_value = 0, ...)
```

Arguments

`object`	A `femtofit` object, or for the generic: the parameter estimate.
`...`	Additional arguments.
`parm`	Parameter name(s) or indices to test. If NULL, tests all parameters.
`null_value`	The null hypothesis value(s). Default is 0.
`se`	Standard error (for default method)

Details

The Wald test statistic for a single parameter is:

$$W = \left(\frac{\hat{\theta} - \theta_0}{SE(\hat{\theta})} \right)^2$$

which follows a chi-squared distribution with 1 degree of freedom under H0.

Value

For single parameter tests, a `wald_test` object (also inherits from `hypothesis_test`) containing:

stat The Wald chi-squared statistic

p.value P-value from chi-squared(1) distribution

dof Degrees of freedom (1 for single parameter)

z The z-score

estimate Parameter estimate

se Standard error

null_value The null hypothesis value

For multiple parameters, returns a list of wald_test objects.

Methods (by class)

- `wald_test(femtofit)`: Wald test for femtofit objects
- `wald_test(default)`: Wald test from raw estimate and SE

Examples

```
## Not run:
set.seed(42)
x <- rnorm(100, mean = 5, sd = 2)

result <- fit(
  function(mu, log_sigma) loglik_normal(mu, exp(log_sigma), x),
  params = c(mu = 0, log_sigma = 0)
)
```

```
# Test if mu = 0
test <- wald_test(result, "mu")
pval(test)

# Test if mu = 5
wald_test(result, "mu", null_value = 5)

# Test all parameters
wald_test(result)

## End(Not run)
```

`zero_grad` *Reset gradients to zero*

Description

Resets the gradient of a value object (and all its ancestors in the computational graph) to zero. This is useful when reusing parameters across multiple optimization steps.

Usage

```
zero_grad(e)

## S3 method for class 'value'
zero_grad(e)

## S3 method for class 'list'
zero_grad(e)

## Default S3 method:
zero_grad(e)
```

Arguments

`e`	A value object or list of value objects

Details

After calling `backward()`, gradients accumulate in the computational graph. Before computing new gradients, you should reset them with `zero_grad()`.

Value

Invisibly returns the input (for chaining)

See Also

backward, grad

Examples

```
x <- val(3)
y <- x^2
backward(y)
grad(x)      # 6
zero_grad(y)
grad(x)      # 0
```

Index

*.value, 5
+.value, 6
-.value, 4
/.value, 5
^.value, 6

abs.value, 7
anova.femtofit, 7, *16*
anova.femtofit(), *63*

backward, 8, *84*
backward.default, 8
backward.value, 9
bfgs, 9
bootstrap, 11
bootstrap_fit, 11
bounded, 12

check_convergence, 13
check_hessian, 14
coef.femtofit *(femtofit)*, 25
compare, *8*, 16
compare(), *65*
confint.bootstrap_result, 17
confint.femtofit *(femtofit)*, 25
confint_mle, 18
confint_profile, 18
cos.value, 19

diagnostics, 20
digamma.value, 21
distributions, 21
div_safe, 21
dof, 22
dual, 22
dual_num, 24

exp.value, 24
exp_safe, 25

femtofit, 25
find_mle, 27
fisher_information, 28
fisher_scoring, 29
fit, 29
fitting, 31

get_data, 31
get_data<- *(get_data)*, 31
grad, 33, *84*
grad.default, 33
grad.value, 34
gradient, 34
gradient_ascent, 35
gradient_descent, 36

hessian, 37
hypothesis_tests, 38

inv_bounded, 38
inv_positive, 39
inv_probability, 39
inverse_transforms, 39
is_dual, 40
is_significant_at, 40
is_value, 41

lbfgs, 41
lgamma.value, 42
line_search, 43
log.value, 44
log1p(), *46*
log1p.value, 45
log1p_safe, 46
log_safe, 44
log_sigmoid, 45
logit, 46
logLik.femtofit *(femtofit)*, 25
loglik_bernoulli, 47
loglik_beta, 47
loglik_binomial, 48

loglik_exponential, 48
loglik_gamma, 49
loglik_logistic, 50
loglik_negbinom, 50
loglik_normal, 51
loglik_pareto, 51
loglik_poisson, 52
loglik_weibull, 53
logsumexp, 54
lower_bounded, 55
lrt, *8*, 56

mean.value, 57

newton_raphson, 58
nobs.femtofit *(femtofit)*, 25

observed_info, 59
optimization, 60

plot.profile_likelihood, 60
positive, 60
predict.femtofit, 61
primal, 62
print.anova.femtofit, 63
print.bootstrap_result, 63
print.convergence_check
(print.hessian_check), 64
print.femtofit *(femtofit)*, 25
print.hessian_check, 64
print.likelihood_ratio_test, 64
print.model_comparison, 65
print.model_diagnostics
(print.hessian_check), 64
print.profile_likelihood, 65
print.summary.femtofit *(femtofit)*, 25
print.value, 66
print.wald_test, 66
probability, 67
profile_likelihood, 68
profile_loglik, 68
pval, 69

relu, 70

se, 70
se_reliable, 71
sigmoid, 72
sigmoid_stable, 72
sin.value, 73
softmax, 73
softplus, 74
sqrt.value, 74
stability, 74
std_errors, 75
sum.dual, 75
sum.value, 76
summary.bootstrap_result, 76
summary.femtofit *(femtofit)*, 25

tangent, 77
tanh.value, 77
test_stat, 78
transforms, 78
trigamma.value, 79

upper_bounded, 79

val, 80
value, 81
vcov.femtofit *(femtofit)*, 25
vcov_matrix, 82

wald_test, 82

zero_grad, 84