Package: optimflex (via r-universe)

October 29, 2026

Type Package

Title Derivative-Based Optimization with User-Defined Convergence Criteria

Version 0.1.8

Description Provides a derivative-based optimization framework that allows users to combine eight convergence criteria. Unlike standard optimization functions, this package includes a built-in mechanism to verify the positive definiteness of the Hessian matrix at the point of convergence. This additional check helps prevent the solver from falsely identifying non-optimal solutions, such as saddle points, as valid minima.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Imports numDeriv, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/eunscho/optimflex

BugReports https://github.com/eunscho/optimflex/issues

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

Date/Publication 2026-07-01 01:55:21 UTC

RemoteUrl https://github.com/eunscho/optimflex

RemoteRef HEAD

RemoteSha 4e81d4d1710dcfa529c13209e3cd899af205f44b

Contents

bfgs 2
dogleg 4
double_dogleg 5
fast_grad 7
fast_hess 8
fast_jac 8
gauss_newton 9
is_pd_fast 10
l_bfgs_b 11
levenberg_marquardt 12
modified_newton 14
newton_raphson 16

Index **18**

`bfgs` *Broyden-Fletcher-Goldfarb-Shanno (BFGS) Optimization*

Description

Implements the damped BFGS Quasi-Newton algorithm with a Strong Wolfe line search for non-linear optimization, specifically tailored for SEM.

Usage

```
bfgs(
  start,
  objective,
  gradient = NULL,
  hessian = NULL,
  lower = -Inf,
  upper = Inf,
  control = list(),
  ...
)
```

Arguments

`start`	Numeric vector. Starting values for the optimization parameters.
`objective`	Function. The objective function to minimize.
`gradient`	Function (optional). Gradient of the objective function.
`hessian`	Function (optional). Hessian matrix of the objective function.
`lower`	Numeric vector. Lower bounds for box constraints.
`upper`	Numeric vector. Upper bounds for box constraints.

`control` List. Control parameters including convergence flags:

- `use_abs_f`: Logical. Use absolute change in objective for convergence.
- `use_rel_f`: Logical. Use relative change in objective for convergence.
- `use_abs_x`: Logical. Use absolute change in parameters for convergence.
- `use_rel_x`: Logical. Use relative change in parameters for convergence.
- `use_grad`: Logical. Use gradient norm for convergence.
- `use_posdef`: Logical. Verify positive definiteness at convergence.
- `use_pred_f`: Logical. Record predicted objective decrease.
- `use_pred_f_avg`: Logical. Record average predicted decrease.
- `diff_method`: String. Method for numerical differentiation.

`...` Additional arguments passed to objective, gradient, and Hessian functions.

Details

`bfgs` is a Quasi-Newton method that maintains an approximation of the inverse Hessian matrix. It is widely considered the most robust and efficient member of the Broyden family of optimization methods.

BFGS vs. DFP: While both `bfgs` and `dfp` update the inverse Hessian using rank-two formulas, BFGS is generally more tolerant of inaccuracies in the line search. This implementation uses the Sherman-Morrison formula to update the inverse Hessian directly, avoiding the need for matrix inversion at each step.

Strong Wolfe Line Search: To maintain the positive definiteness of the Hessian approximation and ensure global convergence, this algorithm employs a Strong Wolfe line search. This search identifies a step length α that satisfies both sufficient decrease (Armijo condition) and the curvature condition.

Damping for Non-Convexity: In Structural Equation Modeling (SEM), objective functions often exhibit non-convex regions. When `use_damped = TRUE`, Powell's damping strategy is applied to the update vectors to preserve the positive definiteness of the Hessian approximation even when the curvature condition is not naturally met.

Value

A list containing optimization results and iteration metadata.

References

- Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization*. Springer.
- Fletcher, R. (1987). *Practical Methods of Optimization*. Wiley.

Examples

```
# Simple quadratic function optimization
quad <- function(x) (x[1] - 2)^2 + (x[2] + 1)^2
res <- bfgs(start = c(0, 0), objective = quad)
print(res$par)
```

`dogleg` *Dogleg Trust-Region Optimization*

Description

Implements the standard Powell's Dogleg Trust-Region algorithm for non-linear optimization.

Usage

```
dogleg(
  start,
  objective,
  gradient = NULL,
  hessian = NULL,
  lower = -Inf,
  upper = Inf,
  control = list(),
  ...
)
```

Arguments

`start`	Numeric vector. Starting values for the optimization parameters.
`objective`	Function. The objective function to minimize.
`gradient`	Function (optional). Gradient of the objective function.
`hessian`	Function (optional). Hessian matrix of the objective function.
`lower`	Numeric vector. Lower bounds for box constraints.
`upper`	Numeric vector. Upper bounds for box constraints.
`control`	List. Control parameters including convergence flags: • `use_abs_f`: Logical. Use absolute change in objective for convergence. • `use_rel_f`: Logical. Use relative change in objective for convergence. • `use_abs_x`: Logical. Use absolute change in parameters for convergence. • `use_rel_x`: Logical. Use relative change in parameters for convergence. • `use_grad`: Logical. Use gradient norm for convergence. • `use_posdef`: Logical. Verify positive definiteness at convergence. • `use_pred_f`: Logical. Record predicted objective decrease. • `use_pred_f_avg`: Logical. Record average predicted decrease.
`...`	Additional arguments passed to objective, gradient, and Hessian functions.

Details

This function implements the classic Dogleg method within a Trust-Region framework, based on the strategy proposed by Powell (1970).

Trust-Region vs. Line Search: Trust-Region methods define a neighborhood around the current point (the trust region with radius Δ) where a local quadratic model is assumed to be reliable. Unlike Line Search methods that first determine a search direction and then find an appropriate step length, this approach constrains the step size first and then finds the optimal update within that boundary.

Powell's Dogleg Trajectory: The "Dogleg" trajectory is a piecewise linear path connecting:

1. The current point.
2. The **Cauchy Point** (p_C): The minimizer of the quadratic model along the steepest descent direction.
3. The **Newton Point** (p_N): The unconstrained minimizer of the quadratic model ($B^{-1}g$).

The algorithm selects a step along this path such that it minimizes the quadratic model while remaining within the radius Δ.

Relationship to Double Dogleg: While the `double_dogleg` algorithm (Dennis and Mei, 1979) introduces a bias point to follow the Newton direction more closely, this standard Dogleg follows the original two-segment trajectory.

Value

A list containing optimization results and iteration metadata.

References

- Powell, M. J. D. (1970). A Hybrid Method for Nonlinear Equations. *Numerical Methods for Nonlinear Algebraic Equations*.
- Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization*. Springer.

Examples

```
# Simple quadratic function optimization
quad <- function(x) (x[1] - 2)^2 + (x[2] + 1)^2
res <- dogleg(start = c(0, 0), objective = quad)
print(res$par)
```

`double_dogleg` *Double Dogleg Trust-Region Optimization*

Description

Implements the Double Dogleg Trust-Region algorithm for non-linear optimization.

Usage

```
double_dogleg(
  start,
  objective,
  gradient = NULL,
  hessian = NULL,
  lower = -Inf,
  upper = Inf,
  control = list(),
  ...
)
```

Arguments

`start`	Numeric vector. Starting values for the optimization parameters.
`objective`	Function. The objective function to minimize.
`gradient`	Function (optional). Gradient of the objective function.
`hessian`	Function (optional). Hessian matrix of the objective function.
`lower`	Numeric vector. Lower bounds for box constraints.
`upper`	Numeric vector. Upper bounds for box constraints.
`control`	List. Control parameters including convergence flags starting with 'use_'. • `use_abs_f`: Logical. Use absolute change in objective for convergence. • `use_rel_f`: Logical. Use relative change in objective for convergence. • `use_abs_x`: Logical. Use absolute change in parameters for convergence. • `use_rel_x`: Logical. Use relative change in parameters for convergence. • `use_grad`: Logical. Use gradient norm for convergence. • `use_posdef`: Logical. Verify positive definiteness at convergence. • `use_pred_f`: Logical. Record predicted objective decrease. • `use_pred_f_avg`: Logical. Record average predicted decrease.
`...`	Additional arguments passed to objective, gradient, and Hessian functions.

Details

This function implements the Double Dogleg method within a Trust-Region framework, primarily based on the work of Dennis and Mei (1979).

Trust-Region vs. Line Search: While Line Search methods (like BFGS) first determine a search direction and then find an appropriate step length, Trust-Region methods define a neighborhood around the current point (the trust region with radius Δ) where a local quadratic model is assumed to be reliable. The algorithm then finds a step that minimizes this model within the radius. This approach is generally more robust, especially when the Hessian is not positive definite.

Powell's Dogleg vs. Double Dogleg: Powell's original Dogleg method (1970) constructs a trajectory consisting of two line segments: one from the current point to the Cauchy point, and another from the Cauchy point to the Newton point. The "Double Dogleg" modification by Dennis and Mei (1979) introduces an intermediate "bias" point (p_W) along the Newton direction.

- **Cauchy Point (p_C):** The minimizer of the quadratic model along the steepest descent direction.
- **Newton Point (p_N):** The minimizer of the quadratic model $(B^{-1}g)$.
- **Double Dogleg Point (p_W):** A point defined as $\gamma \cdot p_N$, where γ is a scaling factor (bias) that ensures the path stays closer to the Newton direction while maintaining monotonic descent in the model.

This modification allows the algorithm to perform more like a Newton method earlier in the optimization process compared to the standard Dogleg.

Value

A list containing optimization results and iteration metadata.

References

- Dennis, J. E., & Mei, H. H. (1979). Two New Unconstrained Optimization Algorithms which use Function and Gradient Values. *Journal of Optimization Theory and Applications*, 28(4), 453-482.
- Powell, M. J. D. (1970). A Hybrid Method for Nonlinear Equations. *Numerical Methods for Nonlinear Algebraic Equations*.
- Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization*. Springer.

Examples

```
# Simple quadratic function optimization
quad <- function(x) (x[1] - 2)^2 + (x[2] + 1)^2
res <- double_dogleg(start = c(0, 0), objective = quad)
print(res$par)
```

`fast_grad` *Fast Numerical Gradient*

Description

Provides a high-speed numerical gradient using forward or central differences.

Usage

```
fast_grad(f, x, diff_method = c("forward", "central"), ...)
```

Arguments

`f`	Function. The objective function.
`x`	Numeric vector. Parameters at which to evaluate the gradient.
`diff_method`	String. Differentiation method: "forward" or "central".
`...`	Additional arguments passed to `f`.

Value

A numeric vector of gradients.

`fast_hess` *Fast Numerical Hessian*

Description

High-speed numerical Hessian calculation using finite differences.

Usage

```
fast_hess(f, x, diff_method = c("forward", "central"), ...)
```

Arguments

`f`	Function. The objective function.
`x`	Numeric vector. Parameters at which to evaluate the Hessian.
`diff_method`	String. Differentiation method: "forward" or "central".
`...`	Additional arguments passed to `f`.

Value

A symmetric Hessian matrix.

`fast_jac` *Fast Numerical Jacobian*

Description

Calculates the Jacobian matrix for a vector-valued function (e.g., residuals).

Usage

```
fast_jac(f_res, x, diff_method = c("forward", "central"), ...)
```

Arguments

`f_res`	Function. A function returning a vector of residuals.
`x`	Numeric vector. Parameters at which to evaluate the Jacobian.
`diff_method`	String. Differentiation method: "forward" or "central".
`...`	Additional arguments passed to `f_res`.

Value

A Jacobian matrix of dimension (m x n).

gauss_newton *Gauss-Newton Optimization*

Description

Implements a full-featured Gauss-Newton algorithm for non-linear optimization, specifically optimized for Structural Equation Modeling (SEM).

Usage

```
gauss_newton(
  start,
  objective,
  residual = NULL,
  gradient = NULL,
  hessian = NULL,
  jac = NULL,
  control = list(),
  ...
)
```

Arguments

start	Numeric vector. Starting values for the optimization parameters.
objective	Function. The objective function to minimize.
residual	Function (optional). Function that returns the residuals vector.
gradient	Function (optional). Gradient of the objective function.
hessian	Function (optional). Hessian matrix of the objective function.
jac	Function (optional). Jacobian matrix of the residuals.
control	List. Control parameters including convergence flags: • use_abs_f: Logical. Use absolute change in objective for convergence. • use_rel_f: Logical. Use relative change in objective for convergence. • use_abs_x: Logical. Use absolute change in parameters for convergence. • use_rel_x: Logical. Use relative change in parameters for convergence. • use_grad: Logical. Use gradient norm for convergence. • use_posdef: Logical. Verify positive definiteness at convergence. • use_pred_f: Logical. Record predicted objective decrease. • use_pred_f_avg: Logical. Record average predicted decrease. • diff_method: String. Method for numerical differentiation.
...	Additional arguments passed to objective, gradient, and Hessian functions.

Details

gauss_newton is a specialized optimization algorithm for least-squares and Maximum Likelihood problems where the objective function can be expressed as a sum of squared residuals.

Scaling and SEM Consistency: To ensure consistent simulation results and standard error (SE) calculations, this implementation adjusts the Gradient $(2J^T r)$ and the Approximate Hessian $(2J^T J)$ to match the scale of the Maximum Likelihood (ML) fitting function F_{ML}. This alignment is critical when calculating asymptotic covariance matrices using the formula $\frac{2}{n}H^{-1}$.

Comparison with Newton-Raphson: Unlike newton_raphson or modified_newton, which require the full second-order Hessian, Gauss-Newton approximates the Hessian using the Jacobian of the residuals. This is computationally more efficient and provides a naturally positive-semidefinite approximation, though a ridge adjustment is still provided for numerical stability.

Ridge Adjustment Strategy: The function includes a "Ridge Rescue" mechanism. If the approximate Hessian is singular or poorly conditioned for Cholesky decomposition, it iteratively adds a diagonal ridge (τI) until numerical stability is achieved.

Value

A list containing optimization results and iteration metadata.

References

- Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization*. Springer.
- Bollen, K. A. (1989). *Structural Equations with Latent Variables*. Wiley.

Examples

```
# Simple quadratic function optimization
quad <- function(x) (x[1] - 2)^2 + (x[2] + 1)^2
res <- gauss_newton(start = c(0, 0), objective = quad)
print(res$par)
```

---

is_pd_fast *Fast Positive Definiteness Check*

Description

Efficiently checks if a matrix is positive definite using Cholesky decomposition.

Usage

```
is_pd_fast(M)
```

Arguments

M	Matrix. The matrix to verify.

Value

Logical. TRUE if positive definite, FALSE otherwise.

`l_bfgs_b` *Limited-memory BFGS with Box Constraints (L-BFGS-B)*

Description

Performs bound-constrained minimization using the L-BFGS-B algorithm. This implementation handles box constraints via Generalized Cauchy Point (GCP) estimation and subspace minimization, featuring a limited-memory (two-loop recursion) inverse Hessian approximation.

Usage

```
l_bfgs_b(
  start,
  objective,
  gradient = NULL,
  hessian = NULL,
  lower = -Inf,
  upper = Inf,
  control = list(),
  ...
)
```

Arguments

`start`	Numeric vector. Initial values for the parameters.
`objective`	Function. The scalar objective function to be minimized.
`gradient`	Function (optional). Returns the gradient vector. If `NULL`, numerical derivatives are used.
`hessian`	Function (optional). Returns the Hessian matrix. Used for final positive definiteness verification if `use_posdef = TRUE`.
`lower, upper`	Numeric vectors. Lower and upper bounds for the parameters. Can be scalars if all parameters share the same bounds.
`control`	A list of control parameters:

- `use_abs_f`: Logical. Criterion: $|f_{new} - f_{old}| <$ `tol_abs_f`.
- `use_rel_f`: Logical. Criterion: $|(f_{new} - f_{old})/f_{old}| <$ `tol_rel_f`.
- `use_abs_x`: Logical. Criterion: $\max|x_{new} - x_{old}| <$ `tol_abs_x`.
- `use_rel_x`: Logical. Criterion: $\max|(x_{new} - x_{old})/x_{old}| <$ `tol_rel_x`.
- `use_grad`: Logical. Criterion: $\|g\|_\infty <$ `tol_grad`.
- `use_posdef`: Logical. Criterion: Positive definiteness of the Hessian.
- `use_pred_f`: Logical. Criterion: predicted decrease $<$ `tol_pred_f`.

- `use_pred_f_avg`: Logical. Criterion: predicted decrease per parameter $<$ `tol_pred_f_avg`.
- `max_iter`: Maximum number of iterations (default: 10000).
- `m`: Number of L-BFGS memory updates (default: 5).
- `tol_abs_f`, `tol_rel_f`: Tolerances for function value change.
- `tol_abs_x`, `tol_rel_x`: Tolerances for parameter change.
- `tol_grad`: Tolerance for the projected gradient (default: 1e-4).
- `tol_pred_f`, `tol_pred_f_avg`: Tolerances for the predicted decrease.

... Additional arguments passed to objective, gradient, and Hessian functions.

Value

A list containing optimization results and metadata.

Comparison with Existing Functions

This function adds three features for rigorous convergence control. First, it applies an **AND rule**: all selected convergence criteria must be satisfied simultaneously. Second, users can choose among **eight distinct criteria** (e.g., changes in f, x, gradient, or predicted decrease) instead of relying on fixed defaults. Third, it provides an **optional verification** using the Hessian computed from derivatives (analytically when provided, or via numerical differentiation). Checking the positive definiteness of this Hessian at the final solution reduces the risk of declaring convergence at non-minimizing stationary points, such as saddle points.

References

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5), 1190-1208.

Morales, J. L., & Nocedal, J. (2011). L-BFGS-B: Remark on algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software*, 38(1), 1-4.

Examples

```
# Simple quadratic function optimization
quad <- function(x) (x[1] - 2)^2 + (x[2] + 1)^2
res <- l_bfgs_b(start = c(0, 0), objective = quad)
print(res$par)
```

---

levenberg_marquardt *Levenberg-Marquardt Optimization*

Description

Implements a full-featured Levenberg-Marquardt algorithm for non-linear optimization, specifically optimized for Structural Equation Modeling (SEM).

Usage

```
levenberg_marquardt(
  start,
  objective,
  residual = NULL,
  gradient = NULL,
  hessian = NULL,
  jac = NULL,
  lower = -Inf,
  upper = Inf,
  control = list(),
  ...
)
```

Arguments

`start`	Numeric vector. Starting values for the optimization parameters.
`objective`	Function. The objective function to minimize.
`residual`	Function (optional). Function that returns the residuals vector.
`gradient`	Function (optional). Gradient of the objective function.
`hessian`	Function (optional). Hessian matrix of the objective function.
`jac`	Function (optional). Jacobian matrix of the residuals.
`lower`	Numeric vector. Lower bounds for box constraints.
`upper`	Numeric vector. Upper bounds for box constraints.
`control`	List. Control parameters including convergence flags: • `use_abs_f`: Logical. Use absolute change in objective for convergence. • `use_rel_f`: Logical. Use relative change in objective for convergence. • `use_abs_x`: Logical. Use absolute change in parameters for convergence. • `use_rel_x`: Logical. Use relative change in parameters for convergence. • `use_grad`: Logical. Use gradient norm for convergence. • `use_posdef`: Logical. Verify positive definiteness at convergence. • `use_pred_f`: Logical. Record predicted objective decrease. • `use_pred_f_avg`: Logical. Record average predicted decrease. • `diff_method`: String. Method for numerical differentiation. • `lambda_init`: Numeric. Initial damping parameter (LM-specific). • `lambda_factor`: Numeric. Multiplier for lambda on failure (LM-specific). • `lambda_max`: Numeric. Upper limit for lambda (LM-specific). • `max_time`: Numeric. Time limit in seconds (LM-specific).
`...`	Additional arguments passed to objective, residual, gradient, hessian, and jac functions.

Details

`levenberg_marquardt` is a specialized optimization algorithm for least-squares and Maximum Likelihood problems where the objective function can be expressed as a sum of squared residuals.

Scaling and SEM Consistency: To ensure consistent simulation results and standard error (SE) calculations, this implementation adjusts the Gradient $(2J^T r)$ and the Approximate Hessian $(2J^T J)$ to match the scale of the Maximum Likelihood (ML) fitting function F_{ML}. The damped Newton step, the gain ratio, and the gradient-based convergence test all operate on this same scale so that the algorithm minimizes the actual objective rather than a residual surrogate.

Comparison with Gauss-Newton: Unlike `gauss_newton`, which uses a backtracking line search (Armijo), Levenberg-Marquardt controls step size via a damping parameter λ that interpolates between Gauss-Newton and gradient descent. The gain ratio (actual vs. predicted decrease) governs λ adjustment following Nielsen's (1999) update rule.

Value

A list containing optimization results and iteration metadata.

References

- Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, **2**(2), 164–168. doi:10.1090/qam/10666
- Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, **11**(2), 431–441. doi:10.1137/0111030
- Nielsen, H. B. (1999). *Damping parameter in Marquardt's method* (Technical Report IMM-REP-1999-05). Technical University of Denmark.
- Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization* (2nd ed.). Springer. doi:10.1007/9780387400655

Examples

```
# Simple quadratic function optimization
quad <- function(x) (x[1] - 2)^2 + (x[2] + 1)^2
quad_res <- function(x) c(x[1] - 2, x[2] + 1)
res <- levenberg_marquardt(start = c(0, 0), objective = quad, residual = quad_res)
print(res$par)
```

`modified_newton` *Modified Newton-Raphson Optimization*

Description

Implements an optimized Newton-Raphson algorithm for non-linear optimization featuring dynamic ridge adjustment and backtracking line search.

Usage

```
modified_newton(
  start,
  objective,
  gradient = NULL,
  hessian = NULL,
  control = list(),
  ...
)
```

Arguments

start	Numeric vector. Starting values for the optimization parameters.
objective	Function. The objective function to minimize.
gradient	Function (optional). Gradient of the objective function.
hessian	Function (optional). Hessian matrix of the objective function.
control	List. Control parameters including convergence flags: • use_abs_f: Logical. Use absolute change in objective for convergence. • use_rel_f: Logical. Use relative change in objective for convergence. • use_abs_x: Logical. Use absolute change in parameters for convergence. • use_rel_x: Logical. Use relative change in parameters for convergence. • use_grad: Logical. Use gradient norm for convergence. • use_posdef: Logical. Verify positive definiteness at convergence. • use_pred_f: Logical. Record predicted objective decrease. • use_pred_f_avg: Logical. Record average predicted decrease. • grad_diff: String. Method for gradient differentiation. • hess_diff: String. Method for Hessian differentiation.
...	Additional arguments passed to objective, gradient, and Hessian functions.

Details

modified_newton is a line search optimization algorithm that utilizes second-order curvature information (the Hessian matrix) to find the minimum of an objective function.

Modified Newton vs. Trust-Region: Unlike the dogleg and double_dogleg functions which use a Trust-Region approach to constrain the step size, this function uses a **Line Search** approach. It first determines the Newton direction (the solution to $H\Delta x = -g$) and then performs a backtracking line search to find a step length α that satisfies the sufficient decrease condition (Armijo condition).

Dynamic Ridge Adjustment: If the Hessian matrix H is not positive definite (making it unsuitable for Cholesky decomposition), the algorithm applies a dynamic ridge adjustment. A diagonal matrix τI is added to the Hessian, where τ is increased until the matrix becomes positive definite. This ensures the search direction always remains a descent direction.

Differentiation Methods: The function allows for independent selection of differentiation methods for the gradient and Hessian:

- forward: Standard forward-difference numerical differentiation.

- `central`: Central-difference (more accurate but slower).
- `complex`: Complex-step differentiation (highly accurate for gradients).
- `richardson`: Richardson extrapolation via the `numDeriv` package.

Value

A list containing optimization results and iteration metadata.

Examples

```
# Simple quadratic function optimization
quad <- function(x) (x[1] - 2)^2 + (x[2] + 1)^2
res <- modified_newton(start = c(0, 0), objective = quad)
print(res$par)
```

`newton_raphson` *Pure Newton-Raphson Optimization*

Description

Implements the standard Newton-Raphson algorithm for non-linear optimization without Hessian modifications or ridge adjustments.

Usage

```
newton_raphson(
  start,
  objective,
  gradient = NULL,
  hessian = NULL,
  control = list(),
  ...
)
```

Arguments

`start`	Numeric vector. Starting values for the optimization parameters.
`objective`	Function. The objective function to minimize.
`gradient`	Function (optional). Gradient of the objective function.
`hessian`	Function (optional). Hessian matrix of the objective function.
`control`	List. Control parameters including convergence flags: • `use_abs_f`: Logical. Use absolute change in objective for convergence. • `use_rel_f`: Logical. Use relative change in objective for convergence. • `use_abs_x`: Logical. Use absolute change in parameters for convergence. • `use_rel_x`: Logical. Use relative change in parameters for convergence.

- use_grad: Logical. Use gradient norm for convergence.
- use_posdef: Logical. Verify positive definiteness at convergence.
- use_pred_f: Logical. Record predicted objective decrease.
- use_pred_f_avg: Logical. Record average predicted decrease.
- diff_method: String. Method for numerical differentiation.

... Additional arguments passed to objective, gradient, and Hessian functions.

Details

newton_raphson provides a classic second-order optimization approach.

Comparison with Modified Newton: Unlike modified_newton, this function does not apply dynamic ridge adjustments (Levenberg-Marquardt style) to the Hessian. If the Hessian is singular or cannot be inverted via solve(), the algorithm will terminate. This "pure" implementation is often preferred in simulation studies where the behavior of the exact Newton step is of interest.

Predicted Decrease: This function explicitly calculates the **Predicted Decrease** ($pred_dec$), which is the expected reduction in the objective function value based on the local quadratic model:

$$m(p) = f + g^T p + \frac{1}{2} p^T H p$$

Stability and Simulations: All return values are explicitly cast to scalars (e.g., as.numeric, as.logical) to ensure stability when the function is called within large-scale simulation loops or packaged into data frames.

Value

A list containing optimization results and iteration metadata.

References

- Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization*. Springer.
- Bollen, K. A. (1989). *Structural Equations with Latent Variables*. Wiley.

Examples

```
# Simple quadratic function optimization
quad <- function(x) (x[1] - 2)^2 + (x[2] + 1)^2
res <- newton_raphson(start = c(0, 0), objective = quad)
print(res$par)
```

Index

bfgs, 2

dogleg, 4
double_dogleg, 5

fast_grad, 7
fast_hess, 8
fast_jac, 8

gauss_newton, 9

is_pd_fast, 10

l_bfgs_b, 11
levenberg_marquardt, 12

modified_newton, 14

newton_raphson, 16