Package 'GALAHAD'

November 7, 2025

Type Package

Title Geometry-Adaptive Lyapunov-Assured Hybrid Optimizer

Version 1.0.0

Author Richard A. Feiss [aut, cre] (ORCID:
<https://orcid.org/0009-0008-0409-6042>)

Maintainer Richard A. Feiss <feiss026@umn.edu>

Description Implements the GALAHAD algorithm (Geometry-Adaptive 'Lyapunov'-Assured Hybrid Optimizer), combining 'Riemannian' metrics, 'Lyapunov' stability checks, and trust-region methods for stable optimization of mixed-geometry parameters. Designed for biological modeling (germination, dose-response, survival) where rates, concentrations, and unconstrained variables coexist. Developed at the Minnesota Center for Prion Research and Outreach (MNPRO), University of Minnesota. Based on Conn et al. (2000) <doi:10.1137/1.9780898719857>, Amari (1998) <doi:10.1162/089976698300017746>, Beck & Teboulle (2003) <doi:10.1016/S0167-6377(02)00231-6>, Nesterov (2017) <https://www.jstor.org/stable/resrep30722>, and Walne et al. (2020) <doi:10.1002/agg2.20098>.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.2.0)

Imports stats

Suggests testthat (>= 3.0.0)

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

BuildResaveData true

Repository CRAN

Date/Publication 2025-11-07 14:10:12 UTC

Contents

GALAHAD *GALAHAD: Geometry-Adaptive Lyapunov-Assured Hybrid Optimizer*

Description

Battle-hardened production optimizer with geometry awareness, Lyapunov stability monitoring, and trust-region safety.

Usage

```
GALAHAD(V, gradV, theta0, parts, control = list(), callback = NULL)
```

Arguments

`V`	Objective function: `function(theta) -> scalar`
`gradV`	Gradient function: `function(theta) -> vector of length p`
`theta0`	Initial parameter vector (numeric, length p)
`parts`	List with geometry partitions: `T`, `P`, `E`. See Details.
`control`	Optional list of control parameters. See Details.
`callback`	Optional progress callback: `function(info)` where info has `iter`, `theta`, `value`, `grad_norm`

Details

Geometry Partitions:

Parameters are divided into three geometric types:

`T` **(log-scale)** Natural gradient on positive reals. Use for scale parameters spanning orders of magnitude (e.g., $\sigma \in (0.01, 100)$).

`P` **(positive orthant)** Entropy mirror descent. Use for positive parameters with moderate range (e.g., $\alpha \in (0.1, 10)$).

`E` **(Euclidean)** Standard gradient descent. Use for unconstrained parameters (e.g., regression coefficients).

Control Parameters:

`max_iter` Maximum iterations (default: 2000)

`tol_g` Gradient tolerance (default: 1e-6)

`tol_x` Step tolerance (default: 1e-9)

`tol_f` Function change tolerance (default: 1e-12)

`delta` Initial trust radius (default: 1.0)

`eta0` Initial step size (default: 1.0)

`V_star` Known minimum (optional, for Polyak steps)

`lambda` L2 regularization weight (default: 0)

Value

List with components:

`theta` Final parameter vector

`value` Final objective value

`grad_inf` Infinity norm of final gradient

`converged` Logical convergence flag

`status` Convergence status string

`reason` Detailed convergence reason

`iterations` Number of iterations performed

`history` data.frame with iteration history

`diagnostics` List with convergence diagnostics and Lyapunov certificates

`certificate` Convergence certificate

References

Conn, A. R., Gould, N. I., & Toint, P. L. (2000). *Trust-region methods*. SIAM.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural computation*, 10(2), 251-276.

Examples

```
# Quadratic objective
p <- 20
Q <- diag(1:p)
theta_true <- rnorm(p)
V <- function(th) 0.5 * sum((th - theta_true) * (Q %*% (th - theta_true)))
gradV <- function(th) Q %*% (th - theta_true)

# Mixed geometry: log-scale, positive, Euclidean
parts <- list(T = 1:5, P = 6:10, E = 11:20)
theta0 <- abs(rnorm(p)) + 0.1

# Set seed for reproducibility (outside the function)
set.seed(42)

# Optimize with progress tracking
result <- GALAHAD(V, gradV, theta0, parts,
  control = list(max_iter = 100, tol_g = 1e-6),
  callback = function(info) {
    if (info$iter %% 10 == 0) {
```

```
      cat(sprintf("Iter %3d: V = %.6f, ||g|| = %.3e\n",
                  info$iter, info$value, info$grad_norm))
    }
  })

print(result$theta)
print(result$diagnostics)
```

Index