Package ‘ManifoldOptim’

July 12, 2026

Type Package

Title An R Interface to the 'ROPTLIB' Library for Riemannian Manifold Optimization

Version 1.0.2

Description An R interface to version 0.3 of the 'ROPTLIB' optimization library (see <https://www.math.fsu.edu/~whuang2/> for more information). Optimize real-valued functions over manifolds such as Stiefel, Grassmann, and symmetric positive definite matrices. For details see Martin et al. (2020) <doi:10.18637/jss.v093.i01>. Note that the optional 'ldr' package used in some of this package's examples can be obtained from either the article <doi:10.18637/jss.v061.i03> or from the 'ldr' package <https://cran.r-project.org/package=ldr>.

License GPL (>= 2)

URL https://bitbucket.org/umbc_sdr/manifoldoptim

Depends Rcpp (>= 0.12.0)

Suggests RcppArmadillo

LinkingTo Rcpp, RcppArmadillo

RcppModules ManifoldOptim_module

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation yes

Author Kofi P. Adragni [aut, cph],
Sean R. Martin [aut, cre, cph],
Andrew M. Raim [aut, cph],
Wen Huang [aut, cph]

Maintainer Sean R. Martin <sean.martin@jhuapl.edu>

Repository CRAN

Date/Publication 2026-07-12 18:20:15 UTC

Contents

`Design of C++ code` *Overview of important files.*

Description

Internal design of the ManifoldOptim portion of the embedded C++ code. Most ManifoldOptim users should not need this. ROPTLIB source code is also included in this package, but is not described here; see Huang et al (2016a) for documentation on that portion of the code.

Details

`src/ManifoldOptim/BrockettProblem.cpp`**:** The Brockett problem, written as a module that can be invoked from within the ManifoldOptim package. This serves as an example for package authors who wish to expose modules to their users. Code to invoke this example from outside of the ManifoldOptim package is provided in `inst/examples/brockett/cpp_pkg`.

`src/ManifoldOptim/ManifoldOptim.cpp`**:** Contains the main function `ManifoldOptim` which takes a problem constructed in R, sets it up in ROPTLIB, runs it, and returns the result.

`src/ManifoldOptim/ManifoldOptimModule.cpp`**:** Defines an Rcpp module for ManifoldOptim which exposes C++ classes such as `RProblem`. This module provides the most common means in which R users will interact with ManifoldOptim.

`src/ManifoldOptim/ManifoldFactory.h`**:** The `GetManifold` function constructs a Manifold object based on its name and dimensions. Manifold classes are defined in ROPTLIB.

`src/ManifoldOptim/ProblemAdapter.h`**:** Defines the `ProblemAdapter` class, which takes a `ManifoldOptimProblem`, which is defined in the ManifoldOptim API, and plugs it into the ROPTLIB API as an ROPTLIB `Problem` subclass.

`src/ManifoldOptim/RProblem.h`**:** Defines the `RProblem` class, which allows the objective, gradient, and Hessian functions to be defined in R. When a function in the ROPTLIB library invokes the objective, gradient, or Hessian, this class invokes the appropriate function in R.

`src/ManifoldOptim/SolverFactory.h`**:** The `GetSolver` function constructs a Solver object based on its name, a given `Problem`, an initial value, and an initial Hessian. Solver classes are defined in ROPTLIB.

`src/ManifoldOptim/Util.h`**:** Defines a few utility functions, especially to assist in translating between the ManifoldOptim C++ API and the ROPTLIB API.

`src/ManifoldOptim/VariableFactory.h`**:** The `GetVariable` function returns an optimization variable suitable for a given Manifold, based on its name and dimension. Optimization variables for supported Manifolds are defined in ROPTLIB.

`inst/include/ManifoldOptimException.h`**:** Defines `ManifoldOptimException`, which is a subclass of STL `exception`.

`inst/include/ManifoldOptim.h`**:** For users of the ManifoldOptim C++ API, this is the main header file to include. For an example, see `inst/examples/brockett/cpp_sourceCpp/`.

`inst/include/ManifoldOptimProblem.h`**:** Defines `ManifoldOptimProblem`, which is the base class for all optimization problems in the ManifoldOptim API. This class facilitates writing problems with Armadillo, which can be instantiated and manipulated in R, and solved through ROPTLIB. This class assumes only that the optimization variable is a one-dimensional vector; the user must reshape it into the appropriate form (e.g. a matrix or list of matrices) when evaluating the objective, gradient, and Hessian functions.

Author(s)

Maintainer: Sean R. Martin <`sean.martin@jhuapl.edu`> [copyright holder]

Authors:

- Kofi P. Adragni [copyright holder]
- Andrew M. Raim <`andrew.raim@gmail.com`> [copyright holder]
- Wen Huang [copyright holder]

References

Wen Huang, P.A. Absil, K.A. Gallivan, Paul Hand (2016a). "ROPTLIB: an object-oriented C++ library for optimization on Riemannian manifolds." Technical Report FSU16-14, Florida State University.

Conrad Sanderson and Ryan Curtin. Armadillo: a template-based C++ library for linear algebra. Journal of Open Source Software, Vol. 1, pp. 26, 2016.

S. Martin, A. Raim, W. Huang, and K. Adragni (2020). "ManifoldOptim: An R Interface to the ROPTLIB Library for Riemannian Manifold Optimization." Journal of Statistical Software, 93(1):1-32.

See Also

Useful links:

- `https://bitbucket.org/umbc_sdr/manifoldoptim`

get.deriv.params *Get parameters to initialize numerical differentiation*

Description

Get parameters to initialize numerical differentiation

Usage

```
get.deriv.params(EpsNumericalGrad = 1e-06, EpsNumericalHessEta = 1e-04)
```

Arguments

`EpsNumericalGrad`
: The "epsilon" used to perturb the objective function when computing numerical gradients

`EpsNumericalHessEta`
: The "epsilon" used to perturb the objective function when computing numerical `HessEta`

Value

List containing input arguments for numerical differentiation

get.manifold.params *Get parameters to initialize manifold*

Description

Get parameters to initialize manifold

Usage

```
get.manifold.params(IsCheckParams = FALSE)
```

Arguments

`IsCheckParams`
: Should internal manifold object check inputs and print summary message before optimization (`TRUE` or `FALSE`)

Value

List containing input arguments for manifold `get.solver.params` *Get parameters to initialize solver*

Description

Get parameters to initialize solver

Usage

```
get.solver.params(
  isconvex = FALSE,
  DEBUG = 0,
  Tolerance = 1e-04,
  Max_Iteration = 1000,
  IsCheckParams = FALSE,
  IsCheckGradHess = FALSE,
  ...
)
```

Arguments

`isconvex`	Indicator for whether the function is convex (`TRUE` or `FALSE`)
`DEBUG`	Verbosity level in {0,1,2,3}. Use 0 for quietest with no messages printed. Use 3 for most verbose,
`Tolerance`	Tolerance used to assess convergence. See Huang et al (2016b) for details on how this is used,
`Max_Iteration`	Maximum iterations to be used by the solver (a non-negative integer),
`IsCheckParams`	Should solver check inputs and print summary message before optimization (`TRUE` or `FALSE`),
`IsCheckGradHess`	Check correctness of the gradient and Hessian functions (`TRUE` or `FALSE`).
`...`	Additional arguments to pass to the solver. These are not validated by the `get.solver.params` function. Users should refer to the C++ library's user manual for available arguments.

Details

Solver-specific parameters may also be added to the object returned from `get.solver.params`, via standard list manipulation. Interested users should refer to Huang et al (2016b) for available options.

Value

List containing input arguments for solver

References

Wen Huang, P.A. Absil, K.A. Gallivan, Paul Hand (2016a). "ROPTLIB: an object-oriented C++ library for optimization on Riemannian manifolds." Technical Report FSU16-14, Florida State University.

Wen Huang, Kyle A. Gallivan, and P.A. Absil (2016b). Riemannian Manifold Optimization Library. URL https://www.math.fsu.edu/~whuang2/pdf/USER_MANUAL_for_2016-04-29.pdf

S. Martin, A. Raim, W. Huang, and K. Adragni (2020). "ManifoldOptim: An R Interface to the ROPTLIB Library for Riemannian Manifold Optimization." Journal of Statistical Software, 93(1):1-32.

`Manifold definitions` *Manifold definitions*

Description

Get definitions for simple manifolds

Usage

```
get.stiefel.defn(n, p, numofmani = 1L, ParamSet = 1L)

get.grassmann.defn(n, p, numofmani = 1L, ParamSet = 1L)

get.spd.defn(n, numofmani = 1L, ParamSet = 1L)

get.sphere.defn(n, numofmani = 1L, ParamSet = 1L)

get.euclidean.defn(n, m, numofmani = 1L, ParamSet = 1L)

get.lowrank.defn(n, m, p, numofmani = 1L, ParamSet = 1L)

get.orthgroup.defn(n, numofmani = 1L, ParamSet = 1L)
```

Arguments

`n`	Dimension for manifold object (see Details)
`p`	Dimension for manifold object (see Details)
`numofmani`	Multiplicity of this space. For example, use `numofmani = 2` if problem requires 2 points from this manifold
`ParamSet`	A positive integer indicating a set of properties for the manifold which can be used by the solver. See Huang et al (2016b) for details.
`m`	Dimension for manifold object (see Details)

Details

The functions define manifolds as follows:

- `get.stiefel.defn`: Stiefel manifold $\{X \in R^{n \times p} : X^T X = I\}$
- `get.grassmann.defn`: Grassmann manifold of p-dimensional subspaces in R^n
- `get.spd.defn`: Manifold of $n \times n$ symmetric positive definite matrices
- `get.sphere.defn`: Manifold of n-dimensional vectors on the unit sphere
- `get.euclidean.defn`: Euclidean $R^{n \times m}$ space
- `get.lowrank.defn`: Low-rank manifold $\{X \in R^{n \times m} : \text{rank}(X) = p\}$
- `get.orthgroup.defn`: Orthonormal group $\{X \in R^{n \times n} : X^T X = I\}$

Value

List containing input arguments and name field denoting the type of manifold

References

Wen Huang, P.A. Absil, K.A. Gallivan, Paul Hand (2016a). "ROPTLIB: an object-oriented C++ library for optimization on Riemannian manifolds." Technical Report FSU16-14, Florida State University.

Wen Huang, Kyle A. Gallivan, and P.A. Absil (2016b). Riemannian Manifold Optimization Library. URL https://www.math.fsu.edu/~whuang2/pdf/USER_MANUAL_for_2016-04-29.pdf

S. Martin, A. Raim, W. Huang, and K. Adragni (2020). "ManifoldOptim: An R Interface to the ROPTLIB Library for Riemannian Manifold Optimization." Journal of Statistical Software, 93(1):1-32.

`manifold.optim` *Manifold optimization*

Description

Optimize a function on a manifold.

Usage

```
manifold.optim(
  prob,
  mani.defn,
  method = "LRBFGS",
  mani.params = get.manifold.params(),
  solver.params = get.solver.params(),
  deriv.params = get.deriv.params(),
  x0 = NULL,
  H0 = NULL,
  has.hhr = FALSE
```

```
)

moptim(
  prob,
  mani.defn,
  method = "LRBFGS",
  mani.params = get.manifold.params(),
  solver.params = get.solver.params(),
  deriv.params = get.deriv.params(),
  x0 = NULL,
  H0 = NULL,
  has.hhr = FALSE
)
```

Arguments

`prob`	A Problem definition
`mani.defn`	Either a Product manifold definition or one of the Manifold definitions
`method`	Name of optimization method. See details.
`mani.params`	Arguments to configure the manifold. Construct with get.manifold.params
`solver.params`	Arguments to configure the solver. Construct with get.solver.params
`deriv.params`	Arguments to configure numerical differentiation for gradient and Hessian, which are used if those functions are not specified. Construct with get.deriv.params
`x0`	Starting point for optimization. A numeric vector whose dimension matches the total dimension of the overall problem
`H0`	Initial value of Hessian. A $d \times d$ matrix, where d is the dimension of `x0`
`has.hhr`	Indicates whether to apply the idea in Huang et al (2015) section 4.1 (`TRUE` or `FALSE`)

Details

`moptim` is an alias for `manifold.optim`.

Currently supported `method` arguments are:

- `"LRBFGS"`: Limited-memory RBFGS,
- `"LRTRSR1"`: Limited-memory RTRSR1,
- `"RBFGS"`: Riemannian BFGS,
- `"RBroydenFamily"`: Riemannian Broyden family,
- `"RCG"`: Riemannian conjugate gradients,
- `"RNewton"`: Riemannian line-search Newton,
- `"RSD"`: Riemannian steepest descent,
- `"RTRNewton"`: Riemannian trust-region Newton,
- `"RTRSD"`: Riemannian trust-region steepest descent,
- `"RTRSR1"`: Riemannian trust-region symmetric rank-one update,
- `"RWRBFGS"`: Riemannian BFGS.

See Huang et al (2016a, 2016b) for details.

Value

xopt	Point returned by the solver
fval	Value of the function evaluated at xopt
normgf	Norm of the final gradient
normgfgf0	Norm of the gradient at final iterate divided by norm of the gradient at initiate iterate
iter	Number of iterations taken by the solver
num.obj.eval	Number of function evaluations
num.grad.eval	Number of gradient evaluations
nR	Number of retraction evaluations
nV	Number of occasions in which vector transport is first computed
nVp	Number of remaining computations of vector transport (excluding count in nV)
nH	Number of actions of Hessian
elapsed	Elapsed time for the solver (in seconds)

References

Wen Huang, P.A. Absil, K.A. Gallivan, Paul Hand (2016a). "ROPTLIB: an object-oriented C++ library for optimization on Riemannian manifolds." Technical Report FSU16-14, Florida State University.

Wen Huang, Kyle A. Gallivan, and P.A. Absil (2016b). Riemannian Manifold Optimization Library. URL https://www.math.fsu.edu/~whuang2/pdf/USER_MANUAL_for_2016-04-29.pdf

Wen Huang, K.A. Gallivan, and P.A. Absil (2015). A Broyden Class of Quasi-Newton Methods for Riemannian Optimization. SIAM Journal on Optimization, 25(3):1660-1685.

S. Martin, A. Raim, W. Huang, and K. Adragni (2020). "ManifoldOptim: An R Interface to the ROPTLIB Library for Riemannian Manifold Optimization." Journal of Statistical Software, 93(1):1-32.

Examples

```
## Not run:
# ----- Example with objective and gradient written in R -----
set.seed(1234)

p <- 5; n <- 150
B <- matrix(rnorm(n*n), nrow=n)
B <- B + t(B)
D <- diag(p:1, p)

tx <- function(x) { matrix(x, n, p) }
f <- function(x) { X <- tx(x); Trace( t(X) %*% B %*% X %*% D ) }
g <- function(x) { X <- tx(x); 2 * B %*% X %*% D }

mod <- Module("ManifoldOptim_module", PACKAGE = "ManifoldOptim")
prob <- new(mod$RProblem, f, g)
```

```
x0 <- as.numeric(orthonorm(matrix(rnorm(n*p), nrow=n, ncol=p)))
mani.params <- get.manifold.params(IsCheckParams = TRUE)
solver.params <- get.solver.params(IsCheckParams = TRUE)
mani.defn <- get.stiefel.defn(n, p)

res <- manifold.optim(prob, mani.defn, method = "RTRSR1",
mani.params = mani.params, solver.params = solver.params, x0 = x0)
print(res)
head(tx(res$xopt))

## End(Not run)
## Not run:
library(ManifoldOptim)
library(RcppArmadillo)

# ----- Example with objective and gradient written in C++ -----
set.seed(1234)

p <- 5; n <- 150
B <- matrix(rnorm(n*n), nrow=n)
B <- B + t(B) # force symmetric
D <- diag(p:1, p)

# The Problem class is written in C++. Get a handle to it and set it up from R
Rcpp::sourceCpp(code = '
//[[Rcpp::depends(RcppArmadillo,ManifoldOptim)]]
#include <RcppArmadillo.h>
#include <ManifoldOptim.h>

using namespace Rcpp;
using namespace arma;

class BrockettProblem : public MatrixManifoldOptimProblem
{
public:
BrockettProblem(const arma::mat& B, const arma::mat& D)
: MatrixManifoldOptimProblem(false, true), m_B(B), m_D(D) { }

virtual ~BrockettProblem() { }

double objFun(const arma::mat& X) const {
return arma::trace(X.t() * m_B * X * m_D);
}

arma::mat gradFun(const arma::mat& X) const {
return 2 * m_B * X * m_D;
}

const arma::mat& GetB() const { return m_B; }
const arma::mat& GetD() const { return m_D; }

private:
```

```
arma::mat m_B;
arma::mat m_D;
};

RCPP_MODULE(Brockett_module) {
class_<BrockettProblem>("BrockettProblem")
.constructor<mat,mat>()
.method("objFun", &BrockettProblem::objFun)
.method("gradFun", &BrockettProblem::gradFun)
.method("GetB", &BrockettProblem::GetB)
.method("GetD", &BrockettProblem::GetD)
;
}
')

prob <- new(BrockettProblem, B, D)
X0 <- orthonorm(matrix(rnorm(n*p), nrow=n, ncol=p))
x0 <- as.numeric(X0)
tx <- function(x) { matrix(x, n, p) }
mani.params <- get.manifold.params(IsCheckParams = TRUE)
solver.params <- get.solver.params(DEBUG = 0, Tolerance = 1e-4,
Max_Iteration = 1000, IsCheckParams = TRUE, IsCheckGradHess = FALSE)
mani.defn <- get.stiefel.defn(n, p)

res <- manifold.optim(prob, mani.defn, method = "RTRSR1",
mani.params = mani.params, solver.params = solver.params, x0 = x0)
print(res)
head(tx(res$xopt))

## End(Not run)
```

`orthonorm` *Orthonormalize the columns of a matrix*

Description

Orthonormalize the columns of a matrix

Usage

```
orthonorm(u)
```

Arguments

u	A matrix

`print.ManifoldOptim` *Print summary from* `manifold.optim` *results*

Description

Print results

Usage

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

Arguments

`x`	A `ManifoldOptim` object output by `manifold.optim`.
`...`	Not currently used

`Problem definition` *Problem definition*

Description

Define a problem for ManifoldOptim to solve.

Details

A problem definition contains an objective function f and a gradient function g. The gradient g is computed as if f is defined on a Euclidean space. If g is not specified it will be computed numerically, which is potentially much slower.

The easiest way to define a problem is completely in R. Example 1 below illustrates how to construct a problem using a given f and g. Example 2 constructs the same problem without providing g. The `Rcpp Module` framework (Eddelbuettel, 2013) creates underlying C++ objects necessary to invoke the `ROPTLIB` library.

The performance of solving an `RProblem` may be too slow for some applications; here, the C++ optimizer calls R functions, which requires some overhead. A faster alternative is to code your problem in C++ directly, and allow it to be manipulated in R. An example is provided in this package, under `tests/brockett/cpp_standalone/`. Example 3 below shows how to instantiate this problem.

Package authors may want to use `ManifoldOptim` within a package to solve a problem written in C++. In this case, the author would probably not want to use `sourceCpp`, but instead have the problem compiled when the package was installed. An example is provided within this package; `tests/brockett/cpp_pkg/driver.R` instantiates the problem defined in:

`src/ManifoldOptim/BrockettProblem.cpp`.

References

Dirk Eddelbuettel. Seamless R and C++ Integration with Rcpp, Chapter 7: Modules, pages 83-102. Springer New York, New York, NY, 2013.

Wen Huang, P.A. Absil, K.A. Gallivan, Paul Hand (2016a). "ROPTLIB: an object-oriented C++ library for optimization on Riemannian manifolds." Technical Report FSU16-14, Florida State University.

S. Martin, A. Raim, W. Huang, and K. Adragni (2020). "ManifoldOptim: An R Interface to the ROPTLIB Library for Riemannian Manifold Optimization." Journal of Statistical Software, 93(1):1-32.

Examples

```
## Not run:
# --- Example 1: Define a problem in R ---
f <- function(x) { ... }
g <- function(x) { ... }
mod <- Module("ManifoldOptim_module", PACKAGE = "ManifoldOptim")
prob <- new(mod$RProblem, f, g)

# --- Example 2: Define a problem in R without specifying gradient ---
f <- function(x) { ... }
mod <- Module("ManifoldOptim_module", PACKAGE = "ManifoldOptim")
prob <- new(mod$RProblem, f)

# --- Example 3: Instantiate a problem written in C++ ---
p <- 5; n <- 150
B <- matrix(rnorm(n*n), nrow=n)
B <- B + t(B) # force symmetric
D <- diag(p:1, p)
Rcpp::sourceCpp("brockett_problem.cpp")
prob <- new(BrockettProblem, B, D)

## End(Not run)
```

`Product manifold definition`

Product manifold definition

Description

Define a product manifold composed of simpler manifolds

Usage

```
get.product.defn(...)
```

Arguments

...	One or more simpler Manifold definitions

Value

List containing manifold definitions for the product manifold

Examples

```
mani.defn1 <- get.product.defn(get.sphere.defn(n=5), get.spd.defn(n=5))
mani.defn2 <- get.product.defn(
    get.stiefel.defn(n=10, p=5),
    get.stiefel.defn(n=7, p=3),
    get.grassmann.defn(n=10, p=5)
)

## Not run:
# --- Estimate jointly: Sigma in SPD manifold and mu in sphere manifold ---
library(mvtnorm)
n <- 400
p <- 3
mu.true <- rep(1/sqrt(p), p)
Sigma.true <- diag(2,p) + 0.1
y <- rmvnorm(n, mean = mu.true, sigma = Sigma.true)

tx <- function(x) {
    idx.mu <- 1:p
    idx.S <- 1:p^2 + p
    mu <- x[idx.mu]
    S <- matrix(x[idx.S], p, p)
    list(mu = mu, Sigma = S)
}
f <- function(x) {
    par <- tx(x)
    -sum(dmvnorm(y, mean = par$mu, sigma = par$Sigma, log = TRUE))
}

mod <- Module("ManifoldOptim_module", PACKAGE = "ManifoldOptim")
prob <- new(mod$RProblem, f)

mu0 <- diag(1, p)[,1]
Sigma0 <- diag(1, p)
x0 <- c(mu0, as.numeric(Sigma0))

mani.defn <- get.product.defn(get.sphere.defn(p), get.spd.defn(p))
mani.params <- get.manifold.params()
solver.params <- get.solver.params(isconvex = TRUE)

res <- manifold.optim(prob, mani.defn, method = "LRBFGS",
    mani.params = mani.params, solver.params = solver.params, x0 = x0)

## End(Not run)
```

`Trace` *Compute the trace of a square matrix*

Description

Compute the trace of a square matrix

Usage

```
Trace(X)
```

Arguments

`X`	A matrix

Index